LECTURE NOTES

ON

Microprocessor and Microcontroller

Prepared by

Dr. M.Vinoth

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

MICROPROCESSORS AND MICROCONTROLLERS

UNIT- I 8085 MICROPROCESSOR

Microprocessor architecture and its operation, memory, I/O devices, 8085 microprocessor – Core architecture - Various registers- Bus Timings, Multiplexing and De-multiplexing of Address Bus, Decoding and Execution, Instruction set – Classification, Instruction Format, Addressing Modes, 8085 Interrupt Process, Hardware and Software Interrupts.

UNIT- II 8086 MICROPROCESSOR

Core Architecture of the 8086 - Memory Segmentation, Minimum mode Operation and Maximum Mode Operation, Instruction Set of the 8086 processor- Classification - Instruction Format Addressing modes, Simple Assembly Language Programs - Arithmetic operations, Data transfer, String Manipulation, Searching and Sorting

UNIT- III I/O INTERFACING

Memory Interfacing and I/O interfacing - Parallel communication interface – Serial Communication interface – D/A and A/D Interface - Timer – Keyboard /display controller – Interrupt controller – DMA controller – Programming and applications Case studies: Traffic Light control, LED display , LCD display, Keyboard display interface and Alarm Controller.

UNIT-IV MICROCONTROLLER

Architecture of 8051 – Special Function Registers (SFRs) - I/O Pins Ports and Circuits – Instruction set- Addressing modes - Assembly language programming - Programming 8051 Timers, Serial Port Programming - Interrupts Programming – LCD & Keyboard Interfacing - ADC, DAC & Sensor Interfacing - External Memory Interface- Stepper Motor and Waveform generation

UNIT-V ADVANCED MICROPROCESSOR & MICROCONTROLLER

Advanced coprocessor Architectures- 286, 486, Pentium -RISC Processors- RISC Vs CISC, RISC properties and evolution- ARM Processor – CPU: programming input and output supervisor mode, exceptions and traps – Co-processors- Memory system mechanisms – CPU performance- CPU power consumption

UNIT-1

Pre - requisite:

- **Basic knowledge of Digital System Design.**
- **To Study the Architecture of 8085 and interrupts**

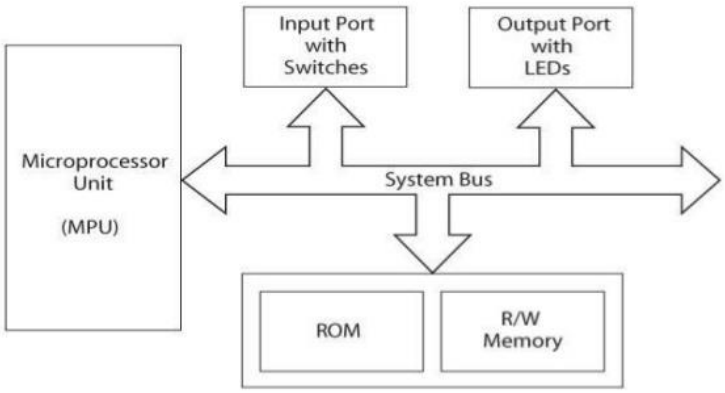
Outcomes

- **Analyze the basic concepts of microprocessor and instructions set**
-

INTRODUCTION TO MICROPROCESSOR AND MICROCOMPUTER

A microprocessor is a programmable electronics chip that has computing and decision making capabilities similar to central processing unit of a computer. Any microprocessor based systems having limited number of resources are called microcomputers. Nowadays, microprocessor can be seen in almost all types of electronics devices like mobile phones, printers, washing machines etc. Microprocessors are also used in advanced applications like radars, satellites and flights. Due to the rapid advancements in electronic industry and large scale integration of devices results in a significant cost reduction and increase application of microprocessors and their derivatives.

Microprocessor-based system



Bit: A bit is a single binary digit.

Word: A word refers to the basic data size or bit size that can be processed by the arithmetic and logic unit of the processor. A 16-bit binary number is called a word in a 16-bit processor.

Bus: A bus is a group of wires/lines that carry similar information.

System Bus: The system bus is a group of wires/lines used for communication between the microprocessor and peripherals.

Memory Word: The number of bits that can be stored in a register or memory element is called a memory word.

Address Bus: It carries the address, which is a unique binary pattern used to identify a memory location or an I/O port. For example, an eight bit address bus has eight lines and thus it can address 28 = 256 different locations. The locations in hexadecimal format can be written as 00H FFH.

Data Bus: The data bus is used to transfer data between memory and processor or between I/O device and processor. For example, an 8-bit processor will generally have an 8-bit data bus and a 16-bit processor will have 16-bit data bus.

Control Bus: The control bus carry control signals, which consists of signals for selection of memory or I/O device from the given address, direction of data transfer and synchronization of data transfer in case of slow devices.

A typical microprocessor consists of arithmetic and logic unit (ALU) in association with control unit to process the instruction execution. Almost all the microprocessors are based on the principle of store-program concept. In store-program concept, programs or instructions are sequentially stored in the memory locations that are to be executed. To do any task using a microprocessor, it is to be programmed by the user. So the programmer must have idea about its internal resources, features and supported instructions. Each microprocessor has a set of instructions, a list which is provided by the microprocessor manufacturer. The instruction set of a microprocessor is provided in two forms: binary machine code and mnemonics.

Microprocessor communicates and operates in binary numbers 0 and 1. The set of instructions in the form of binary patterns is called a machine language and it is difficult for us to understand. Therefore, the binary patterns are given abbreviated names, called mnemonics, which forms the assembly language. The conversion of assembly-level language into binary machine-level language is done by using an application called assembler.

Technology Used:

- The semiconductor manufacturing technologies used for chips are:
- Transistor-Transistor Logic (TTL)
- Emitter Coupled Logic (ECL)
- Complementary Metal-Oxide Semiconductor (CMOS)

Classification of Microprocessors:

Based on their specification, application and architecture microprocessors are classified.

Based on size of data bus:

- 4-bit microprocessor
- 8-bit microprocessor
- 16-bit microprocessor
- 32-bit microprocessor

Based on application:

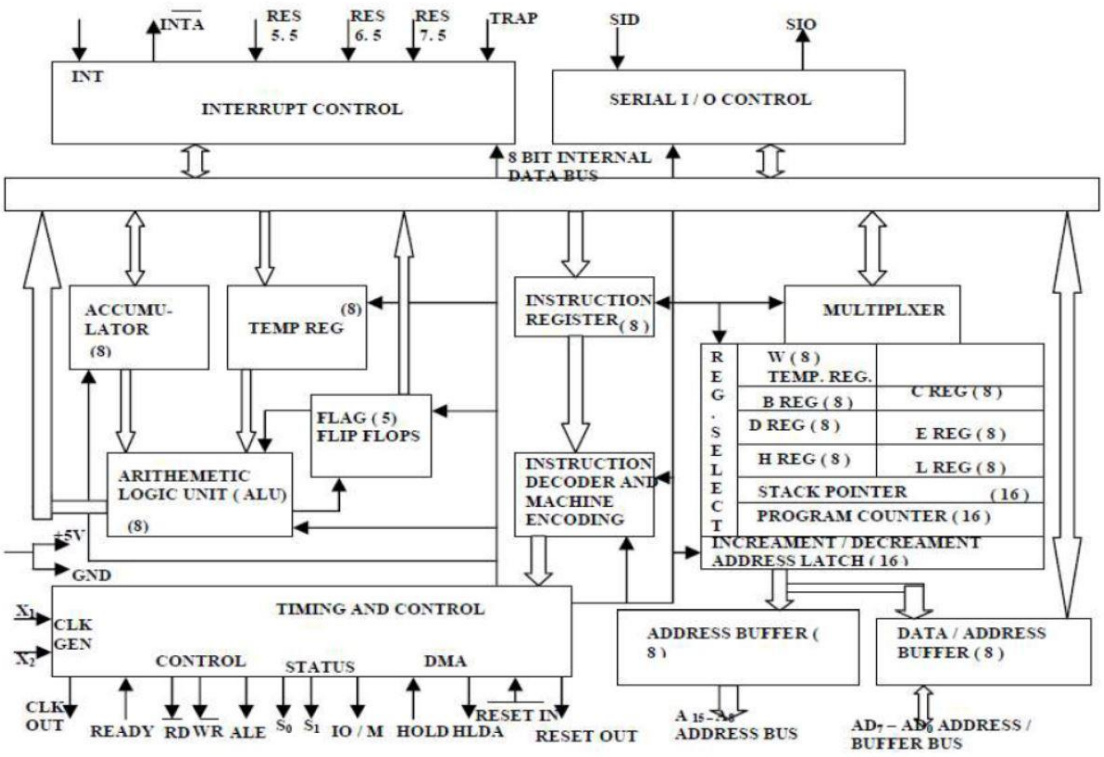
- General-purpose microprocessor- used in general computer system and can be used by programmer for any application. Examples, 8085 to Intel Pentium.
- Microcontroller- microprocessor with built-in memory and ports and can be programmed for any generic control application. Example, 8051.
- Special-purpose processors- designed to handle special functions required for an application. Examples, digital signal processors and application-specific integrated circuit (ASIC) chips.

Based on architecture:

- Reduced Instruction Set Computer (RISC) processors
- Complex Instruction Set Computer (CISC) processors

8085 MICROPROCESSOR ARCHITECTURE

The 8085 microprocessor is an 8-bit processor available as a 40-pin IC package and uses +5 V for power. It can run at a maximum frequency of 3 MHz. Its data bus width is 8-bit and address bus width is 16-bit, thus it can address 216 = 64 KB of memory. The internal architecture of 8085.



Arithmetic and Logic Unit

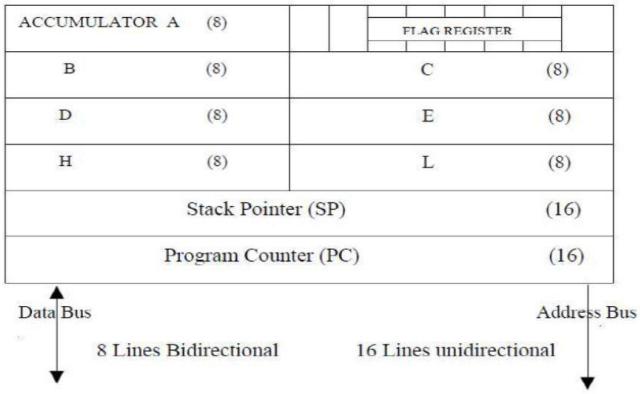
The ALU performs the actual numerical and logical operations such as Addition (ADD), Subtraction (SUB), AND, OR etc. It uses data from memory and from Accumulator to perform operations. The results of the arithmetic and logical operations are stored in the accumulator.

Registers

The 8085 includes six registers, one accumulator and one flag register, In addition, it has two 16-bit registers: stack pointer and program counter. The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H and L. they can be combined as register pairs - BC, DE and HL to perform some 16-bit operations. The programmer can use these registers to store or copy data into the register by using data copy instructions.

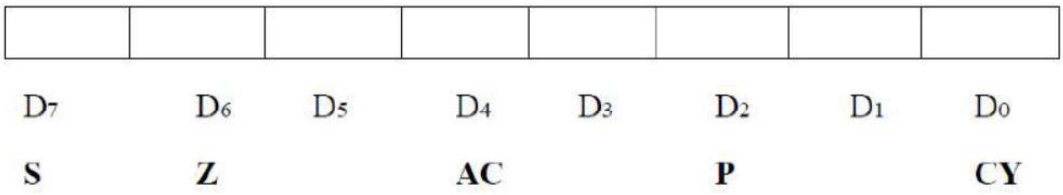
Accumulator

The accumulator is an 8-bit register that is a part of ALU. This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.



Flag register

The ALU includes five flip-flops, which are set or reset after an operation according to data condition of the result in the accumulator and other registers. They are called Zero (Z), Carry (CY), Sign (S), Parity (P) and Auxiliary Carry (AC) flags.The microprocessor uses these flags to test data conditions.



For example, after an addition of two numbers, if the result in the accumulator is larger than 8-bit, the flip-flop uses to indicate a carry by setting CY flag to 1. When an arithmetic operation results in zero, Z flag is set to 1. The S flag is just a copy of the bit D7 of the accumulator. A negative number has a 1 in bit D7 and a positive number has a 0 in 2's complement representation. The AC flag is set to 1, when a carry result from bit D3 and passes to bit D4. The P flag is set to 1, when the result in accumulator contains even number of 1s.

Program Counter (PC)

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte is being fetched, the program counter is automatically incremented by one to point to the next memory location.

Stack Pointer (SP)

The stack pointer is also a 16-bit register, used as a memory pointer. It points to a memory location in R/W memory, called stack. The beginning of the stack is defined by loading 16- bit address in the stack pointer.

Instruction Register/Decoder

It is an 8-bit register that temporarily stores the current instruction of a program. Latest instruction sent here from memory prior to execution. Decoder then takes instruction and decodes or interprets the instruction. Decoded instruction then passed to next stage.

Control Unit

Generates signals on data bus, address bus and control bus within microprocessor to carry out the instruction, which has been decoded. Typical buses and their timing are described as follows:

Data Bus: Data bus carries data in binary form between microprocessor and other external units such as memory. It is used to transmit data i.e. information, results of arithmetic etc between memory and the microprocessor. Data bus is bidirectional in nature. The data bus width of 8085 microprocessor is 8-bit i.e. 28 combination of binary digits and are typically identified as D0 D7. Thus size of the data bus determines what arithmetic can be done. If only 8-bit wide then largest number is 11111111 (255 in decimal). Therefore, larger numbers have to be broken down into chunks of 255. This slows microprocessor.

Address Bus: The address bus carries addresses and is one way bus from microprocessor to the memory or other devices. 8085 microprocessor contain 16-bit address bus and are generally identified as A0 - A15. The higher order address lines (A8 A15) are unidirectional and the lower order lines (A0 A7) are multiplexed (time-shared) with the eight data bits (D0 D7) and hence, they are bidirectional.

Control Bus: Control bus are various lines which have specific functions for coordinating and controlling microprocessor operations. The control bus carries control signals partly unidirectional and partly bidirectional. The following control and status signals are used by 8085 processor:

- ALE (output): Address Latch Enable is a pulse that is provided when an address appears on the AD0 AD7 lines, after which it becomes 0.
- RD (active low output): The Read signal indicates that data are being read from the selected I/O or memory device and that they are available on the data bus.
- WR (active low output): The Write signal indicates that data on the data bus are to be written into a selected memory or I/O location.
- IO/M(output): It is a signal that distinguished between a memory operation and an I/O operation. When IO/M= 0 it is a memory operation and IO/M= 1 it is an I/O operation.
- S1 and S0 (output): These are status signals used to specify the type of operation being performed

S1	S0	States
0	0	Halt
0	1	Write
1	0	Read
1	1	Fetch

The microprocessor performs primarily four operations:

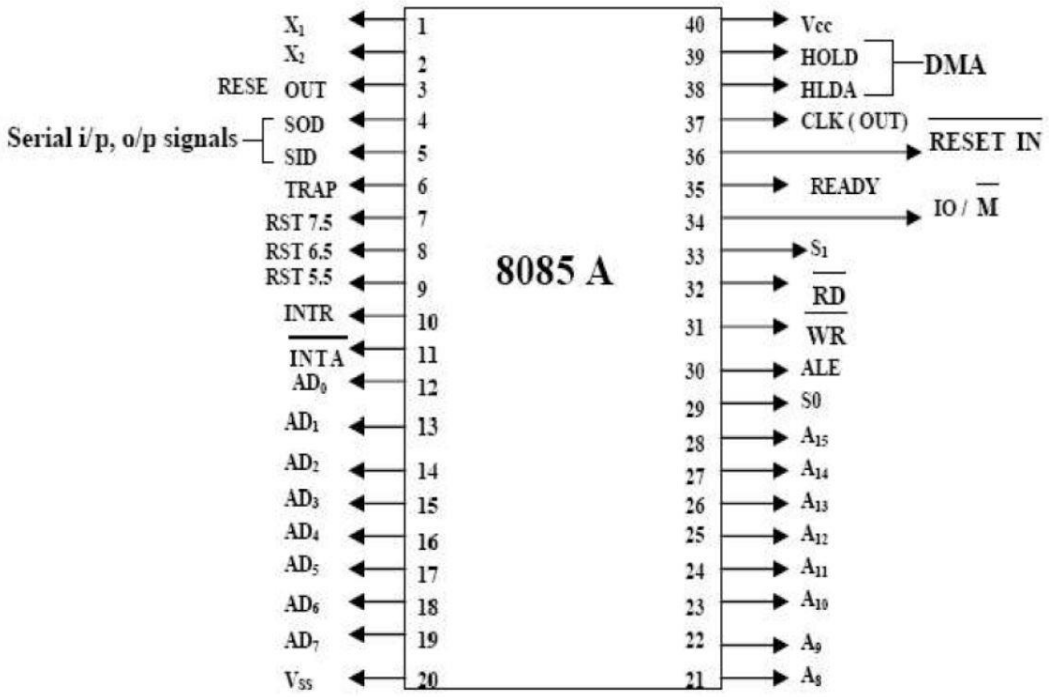
I. Memory Read: Reads data (or instruction) from memory.
II. Memory Write: Writes data (or instruction) into memory.
III. I/O Read: Accepts data from input device.
IV. I/O Write: Sends data to output device.

8085 PIN DESCRIPTION

- It is a 8-bit microprocessor
- Manufactured with N-MOS technology
- 40 pin IC package
- It has 16-bit address bus and thus has 216 = 64 KB addressing capability.
- Operate with 3 MHz single-phase clock
- +5 V single power supply

The logic pin layout and signal groups of the 8085nmicroprocessor are shown in Fig. 6. All the signals are classified into six groups:

- Address bus
- Data bus
- Control & status signals
- Power supply and frequency signals
- Externally initiated signals
- Serial I/O signals



Address and Data Buses:

A8 A15 (output, 3-state): Most significant eight bits of memory addresses and the eight bits of the I/O addresses. These lines enter into tri-state high impedance state during HOLD and HALT modes.

AD0 AD7 (input/output, 3-state): Lower significant bits of memory addresses and the eight bits of the I/O addresses during first clock cycle. Behaves as data bus during third and fourth clock cycle. These lines enter into tri-state high impedance state during HOLD and HALT modes.

Control & Status Signals:

- ALE: Address latch enable
- RD : Read control signal.
- WR : Write control signal.
- IO/M, S1 and S0 : Status signals.

Power Supply & Clock Frequency:

- Vcc: +5 V power supply
- Vss: Ground reference
- X1, X2: A crystal having frequency of 6 MHz is connected at these two pins
- CLK: Clock output

Externally Initiated and Interrupt Signals:

- RESET IN: When the signal on this pin is low, the PC is set to 0, the buses are tristated and the processor is reset.

- RESET OUT: This signal indicates that the processor is being reset. The signal can be used to reset other devices.
- READY: When this signal is low, the processor waits for an integral number of clock cycles until it goes high.
- HOLD: This signal indicates that a peripheral like DMA (direct memory access) controller is requesting the use of address and data bus.
- HLDA: This signal acknowledges the HOLD request.
- INTR: Interrupt request is a general-purpose interrupt.
- INTA : This is used to acknowledge an interrupt.
- RST 7.5, RST 6.5, RST 5,5 restart interrupt: These are vectored interrupts and have highest priority than INTR interrupt.
- TRAP: This is a non-maskable interrupt and has the highest priority.

Serial I/O Signals:

- SID: Serial input signal. Bit on this line is loaded to D7 bit of register A using RIM
- instruction.
- SOD: Serial output signal. Output SOD is set or reset by using SIM instruction.

Addressing Modes in Instructions:

The process of specifying the data to be operated on by the instruction is called addressing. The various formats for specifying operands are called addressing modes. The 8085 has the following five types of addressing:

- Immediate addressing
- Memory direct addressing
- Register direct addressing
- Indirect addressing
- Implicit addressing

Immediate Addressing:

In this mode, the operand given in the instruction - a byte or word transfers to the destination register or memory location.

Ex: MVI A, 9AH
The operand is a part of the instruction.
The operand is stored in the register mentioned in the instruction.

Memory Direct Addressing:

Memory direct addressing moves a byte or word between a memory location and register. The memory location address is given in the instruction.

Ex: LDA 850FH
This instruction is used to load the content of memory address 850FH in the accumulator.

Register Direct Addressing:

Register direct addressing transfer a copy of a byte or word from source register to destination register.

Ex: MOV B, C
It copies the content of register C to register B.

Indirect Addressing:

Indirect addressing transfers a byte or word between a register and a memory location.

Ex: MOV A, M
Here the data is in the memory location pointed to by the contents of HL pair. The data is moved to the accumulator.

Implicit Addressing

In this addressing mode the data itself specifies the data to be operated upon.

Ex: CMA
The instruction complements the content of the accumulator. No specific data or operand is mentioned in the instruction.

INSTRUCTION SET OF 8085

Based on the design of the ALU and decoding unit, the microprocessor manufacturer provides instruction set for every microprocessor. The instruction set consists of both machine code and mnemonics.

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions that a microprocessor supports is called instruction set. Microprocessor instructions can be classified based on the parameters such functionality, length and operand addressing.

Data transfer operations:

This group of instructions copies data from source to destination. The content of the source is not altered.

DATA TRANSFER INSTRUCTIONS

Opcode	Operand	Description
Copy from source to destination		
MOV	Rd, Rs M, Rs Rd, M	This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. Example: MOV B, C or MOV B, M
Move immediate 8-bit		
MVI	Rd, data M, data	The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: MVI B, 57H or MVI M, 57H
Load accumulator		
LDA	16-bit address	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. Example: LDA 2034H
Load accumulator indirect		
LDAX	B/D Reg. pair	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. Example: LDAX B
Load register pair immediate		
LXI	Reg. pair, 16-bit data	The instruction loads 16-bit data in the register pair designated in the operand. Example: LXI H, 2034H or LXI H, XYZ
Load H and L registers direct		
LHLD	16-bit address	The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered. Example: LHLD 2040H

Store accumulator direct

STA 16-bit address

The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address.
Example: STA 4350H

Store accumulator indirect

STAX Reg. pair

The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered.
Example: STAX B

Store H and L registers direct

SHLD 16-bit address

The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address.
Example: SHLD 2470H

Exchange H and L with D and E

XCHG none

The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E.
Example: XCHG

Copy H and L registers to the stack pointer

SPHL none

The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered.
Example: SPHL

Exchange H and L with top of stack

XTHL none

The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1); however, the contents of the stack pointer register are not altered.
Example: XTHL

Arithmetic operations:

Instructions of this group perform operations like addition, subtraction, increment & decrement. One of the data used in arithmetic operation is stored in accumulator and the result is also stored in accumulator.

Opcode	Operand	Description
Add register or memory to accumulator		
ADD	R M	The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADD B or ADD M
Add register to accumulator with carry		
ADC	R M	The contents of the operand (register or memory) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADC B or ADC M
Add immediate to accumulator		
ADI	8-bit data	The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ADI 45H
Add immediate to accumulator with carry		
ACI	8-bit data	The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ACI 45H
Add register pair to H and L registers		
DAD	Reg. pair	The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. Example: DAD H

Subtract register or memory from accumulator

SUB	R M	The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. Example: SUB B or SUB M

Subtract source and borrow from accumulator

SBB	R M	The contents of the operand (register or memory) and the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. Example: SBB B or SBB M

Subtract immediate from accumulator

SUI	8-bit data	The 8-bit data (operand) is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction. Example: SUI 45H

Subtract immediate from accumulator with borrow

SBI	8-bit data	The 8-bit data (operand) and the Borrow flag are subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtracion. Example: SBI 45H

Increment register or memory by 1

INR	R M	The contents of the designated register or memory) are incremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: INR B or INR M

Increment register pair by 1

INX	R	The contents of the designated register pair are incremented by 1 and the result is stored in the same place. Example: INX H

Decrement register or memory by 1

DCR	R M	The contents of the designated register or memory are decremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: DCR B or DCR M

Decrement register pair by 1

DCX	R	The contents of the designated register pair are decremented by 1 and the result is stored in the same place. Example: DCX H

Decimal adjust accumulator

DAA	none	The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation. If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits. If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits. Example: DAA

Logical operations:

Logical operations include AND, OR, EXOR, NOT. The operations like AND, OR and EXOR uses two operands, one is stored in accumulator and other can be any register or memory location. The result is stored in accumulator. NOT operation requires single operand, which is stored in accumulator.

Logical AND immediate with accumulator

ANI	8-bit data	The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. Example: ANI 86H

Exclusive OR register or memory with accumulator

XRA	R M	The contents of the accumulator are Exclusive ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: XRA B or XRA M

Exclusive OR immediate with accumulator

XRI	8-bit data	The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: XRI 86H

Logical OR register or memory with accumulaotr

ORA	R M	The contents of the accumulator are logically ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: ORA B or ORA M

Logical OR immediate with accumulator

ORI	8-bit data	The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: ORI 86H

Rotate accumulator left

RLC	none	Each binary bit of the accumulator is rotated left by one position. Bit D_7 is placed in the position of D_0 as well as in the Carry flag. CY is modified according to bit D_7. S, Z, P, AC are not affected. Example: RLC

Rotate accumulator right

RRC	none	Each binary bit of the accumulator is rotated right by one position. Bit D_0 is placed in the position of D_7 as well as in the Carry flag. CY is modified according to bit D_0. S, Z, P, AC are not affected.

Rotate accumulator left through carry

RAL	none	Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D_7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D_0. CY is modified according to bit D_7. S, Z, P, AC are not affected. Example: RAL

Rotate accumulator right through carry

RAR	none	Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D_0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D_7. CY is modified according to bit D_0. S, Z, P, AC are not affected. Example: RAR

Complement accumulator

CMA	none	The contents of the accumulator are complemented. No flags are affected. Example: CMA

Complement carry

CMC	none	The Carry flag is complemented. No other flags are affected. Example: CMC

Set Carry

STC	none	The Carry flag is set to 1. No other flags are affected. Example: STC

Compare register or memory with accumulator

CMP	R
	M
		The contents of the operand (register or memory) are compared with the contents of the accumulator. Both contents are preserved . The result of the comparison is shown by setting the flags of the PSW as follows:
		if (A) < (reg/mem): carry flag is set
		if (A) = (reg/mem): zero flag is set
		if (A) > (reg/mem): carry and zero flags are reset
		Example: CMP B or CMP M

Compare immediate with accumulator

CPI	8-bit data	The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:
		if (A) < data: carry flag is set
		if (A) = data: zero flag is set
		if (A) > data: carry and zero flags are reset
		Example: CPI 89H

Branching operations:

Instructions in this group can be used to transfer program sequence from one memory location to another either conditionally or unconditionally.

Opcode	Operand	Description
Jump unconditionally		
JMP	16-bit address	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Example: JMP 2034H or JMP XYZ

Jump conditionally

Operand: 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below.
Example: JZ 2034H or JZ XYZ

Opcode	Description	Flag Status
JC	Jump on Carry	CY = 1
JNC	Jump on no Carry	CY = 0
JP	Jump on positive	S = 0
JM	Jump on minus	S = 1
JZ	Jump on zero	Z = 1
JNZ	Jump on no zero	Z = 0
JPE	Jump on parity even	P = 1
JPO	Jump on parity odd	P = 0

Return from subroutine unconditionally		
RET	none	The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address. Example: RET

Unconditional subroutine call

CALL 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack.
Example: CALL 2034H or CALL XYZ

Call conditionally

Operand: 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below. Before the transfer, the address of the next instruction after the call (the contents of the program counter) is pushed onto the stack.
Example: CZ 2034H or CZ XYZ

Opcode	Description	Flag Status
CC	Call on Carry	CY = 1
CNC	Call on no Carry	CY = 0
CP	Call on positive	S = 0
CM	Call on minus	S = 1
CZ	Call on zero	Z = 1
CNZ	Call on no zero	Z = 0
CPE	Call on parity even	P = 1
CPO	Call on parity odd	P = 0

Return from subroutine conditionally

Operand: none

The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW as described below. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.
Example: RZ

Opcode	Description	Flag Status
RC	Return on Carry	CY = 1
RNC	Return on no Carry	CY = 0
RP	Return on positive	S = 0
RM	Return on minus	S = 1
RZ	Return on zero	Z = 1
RNZ	Return on no zero	Z = 0
RPE	Return on parity even	P = 1
RPO	Return on parity odd	P = 0

Machine control operations:

Instruction in this group control execution of other instructions and control operations like interrupt, halt etc.

Opcode	Operand	Description
No operation		
NOP	none	No operation is performed. The instruction is fetched and decoded. However no operation is executed. Example: NOP
Halt and enter wait state		
HLT	none	The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state. Example: HLT
Disable interrupts		
DI	none	The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected. Example: DI
Enable interrupts		
EI	none	The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to reenable the interrupts (except TRAP). Example: EI

INSTRUCTION EXECUTION AND TIMING DIAGRAM

Each instruction in 8085 microprocessor consists of two part- operation code (opcode) and operand. The opcode is a command such as ADD and the operand is an object to be operated on, such as a byte or the content of a register.

Instruction Cycle: The time taken by the processor to complete the execution of an instruction. An instruction cycle consists of one to six machine cycles.

Machine Cycle: The time required to complete one operation; accessing either the memory or I/O device. A machine cycle consists of three to six T-states.

T-State: Time corresponding to one clock period. It is the basic unit to calculate execution of instructions or programs in a processor.

To execute a program, 8085 performs various operations as:

- Opcode fetch
- Operand fetch
- Memory read/write
- I/O read/write

External communication functions are:

- Memory read/write
- I/O read/write
- Interrupt request acknowledge

Opcode Fetch Machine Cycle:

T1 clock cycle

- The content of PC is placed in the address bus; AD0 - AD7 lines contains lower bit address and A8 A15 contains higher bit address.
- IO/M signal is low indicating that a memory location is being accessed. S1 and S0 also changed to the levels as indicated.
- ALE is high, indicates that multiplexed AD0 AD7 act as lower order bus.

T2 clock cycle

- Multiplexed address bus is now changed to data bus.
- The RD signal is made low by the processor. This signal makes the memory device load the data bus with the contents of the location addressed by the processor.

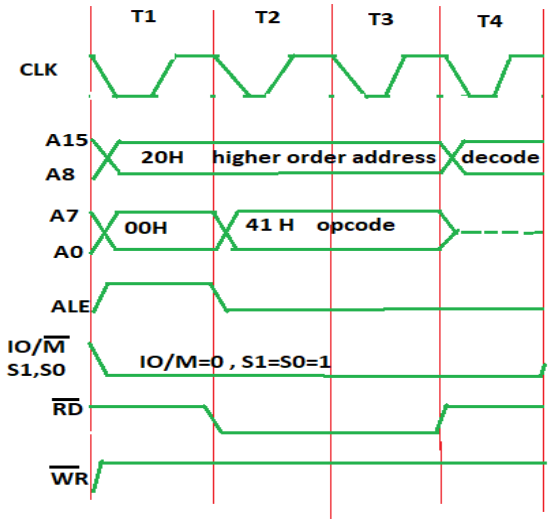
T3 clock cycle

- The opcode available on the data bus is read by the processor and moved to the instruction register.
- The RD signal is deactivated by making it logic 1.

T4 clock cycle

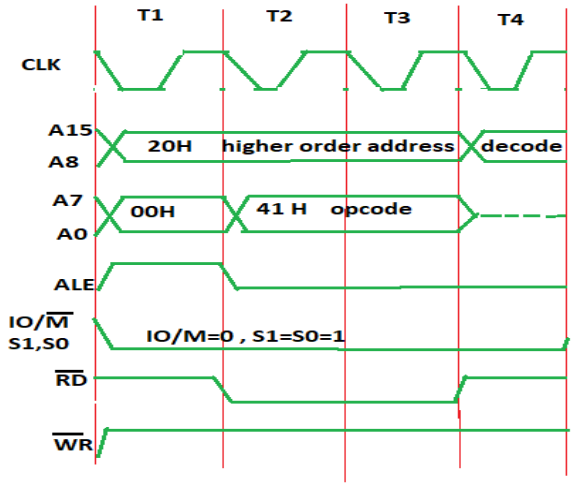
- The processor decode the instruction in the instruction register and generate the
- necessary control signals to execute the instruction. Based on the instruction further operations such as fetching, writing into memory etc takes place.

Timing diagram for opcode fetch cycle



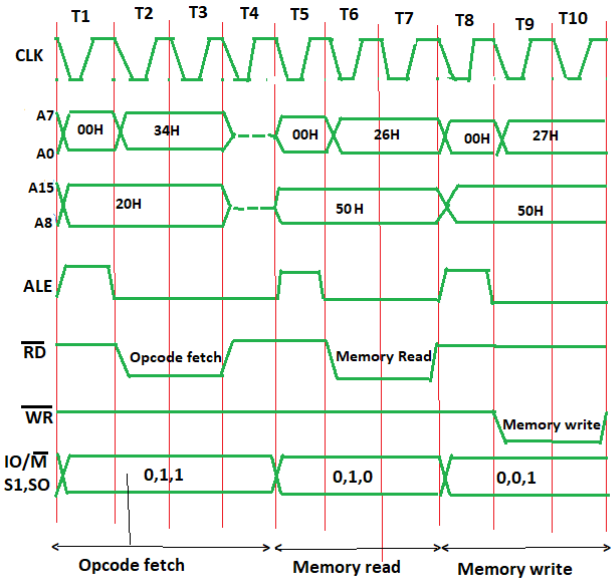
Memory Read Machine Cycle:

The memory read cycle is executed by the processor to read a data byte from memory. The machine cycle is exactly same to opcode fetch except: a) It has three T-states b) The S0 signal is set to 0. The timing diagram of this cycle is given in



Memory Write Machine Cycle:

The memory write cycle is executed by the processor to write a data byte in a memory location. The processor takes three T-states and WR signal is made low. The timing diagram of this cycle is given in



I/O Read Cycle:

The I/O read cycle is executed by the processor to read a data byte from I/O port or from peripheral, which is I/O mapped in the system. The 8-bit port address is placed both in the lower and higher order address bus. The processor takes three T-states to execute this machine cycle.

8085 INTERRUPTS

Interrupt Structure:

Interrupt is the mechanism by which the processor is made to transfer control from its current program execution to another program having higher priority. The interrupt signal may be given to the processor by any external peripheral device.

The program or the routine that is executed upon interrupt is called interrupt service routine (ISR). After execution of ISR, the processor must return to the interrupted program. Key features in the interrupt structure of any microprocessor are as follows:

Number and types of interrupt signals available.

- The address of the memory where the ISR is located for a particular interrupt signal. This address is called interrupt vector address (IVA).
- Masking and unmasking feature of the interrupt signals.
- Priority among the interrupts.
- Timing of the interrupt signals.
- Handling and storing of information about the interrupt program.

Types of Interrupts:

i. Vectored and Non-Vectored Interrupts

Vectored interrupts require the IVA to be supplied by the external device that gives the interrupt signal. This technique is vectoring, is implemented in number of ways. Non-vectored interrupts have fixed IVA for ISRs of different interrupt signals.

ii. Maskable and Non-Maskable Interrupts

Maskable interrupts are interrupts that can be blocked. Masking can be done by software or hardware means. Non-maskable interrupts are interrupts that are always recognized; the corresponding ISRs are executed.

iii. Software and Hardware Interrupts

Software interrupts are special instructions, after execution transfer the control to predefined ISR. Hardware interrupts are signals given to the processor, for recognition as an interrupt and execution of the corresponding ISR.

Interrupt Handling Procedure:

The following sequence of operations takes place when an interrupt signal is recognized:

- Save the PC content and information about current state (flags, registers etc) in the stack.
- Load PC with the beginning address of an ISR and start to execute it.
- Finish ISR when the return instruction is executed.
- Return to the point in the interrupted program where execution was interrupted.

Software Interrupts:

8085 instruction set includes eight software interrupt instructions called Restart (RST) instructions. These are one byte instructions that make the processor execute a subroutine at predefined locations. Instructions and their vector addresses are given in Table

Instruction	Machine hex code	Interrupt Vector Address
RST 0	C7	0000H
RST 1	CF	0008H
RST 2	D7	0010H
RST 3	DF	0018H
RST 4	E7	0020H
RST 5	EF	0028H
RST 6	F7	0030H
RST 7	FF	0032H

The software interrupts can be treated as CALL instructions with default call locations. The concept of priority does not apply to software interrupts as they are inserted into the program as instructions by the programmer and executed by the processor when the respective program lines are read.

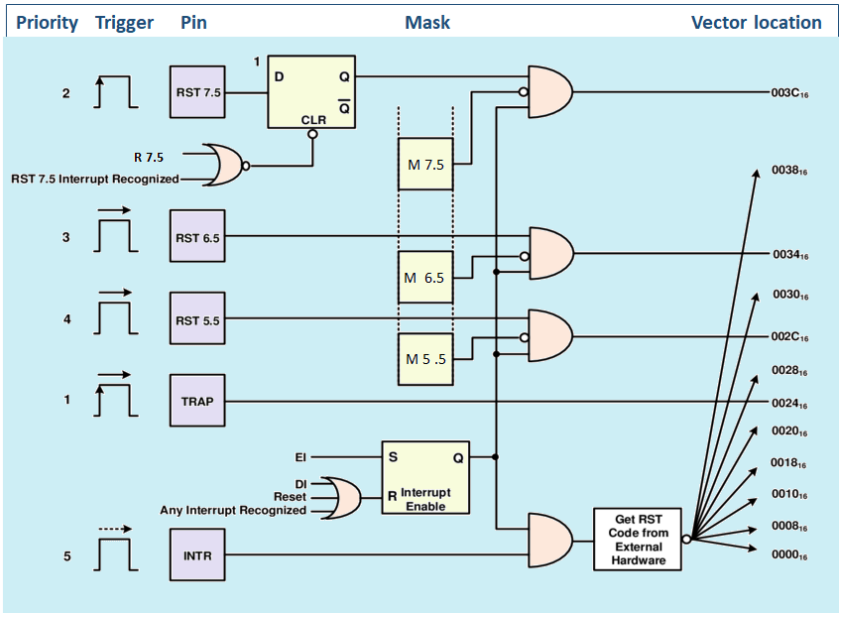
Hardware Interrupts and Priorities:

8085 have five hardware interrupts INTR, RST 5.5, RST 6.5, RST 7.5 and TRAP. Their IVA and priorities are given in Table

Interrupt	Interrupt vector address	Maskable or non-maskable	Edge or level triggered	priority
TRAP	0024H	Non-makable	Level	1
RST 7.5	003CH	Maskable	Rising edge	2
RST 6.5	0034H	Maskable	Level	3
RST 5.5	002CH	Maskable	Level	4
INTR	Decided by hardware	Maskable	Level	5

Masking of Interrupts:

Masking can be done for four hardware interrupts INTR, RST 5.5, RST 6.5, and RST 7.5. The masking of 8085 interrupts is done at different levels. Fig. 13 shows the organization of hardware interrupts in the 8085.

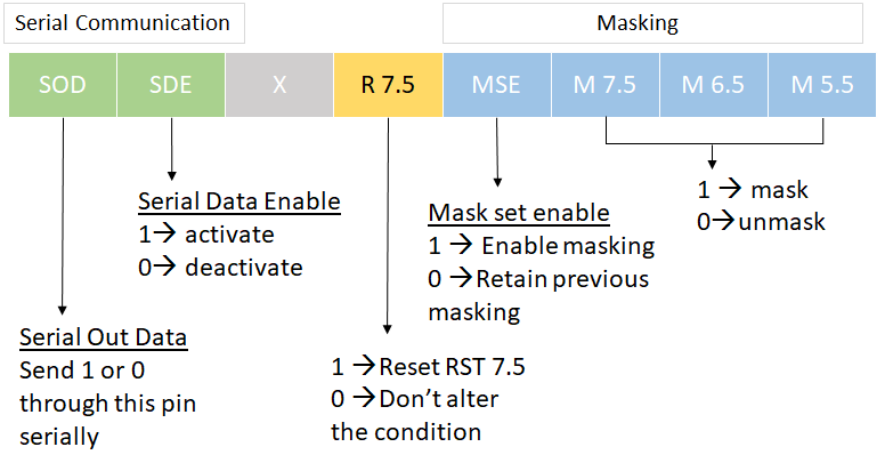


- The maskable interrupts are by default masked by the Reset signal. So no interrupt is recognized by the hardware reset.
- The interrupts can be enabled by the EI instruction.
- The three RST interrupts can be selectively masked by loading the appropriate word in the accumulator and executing SIM instruction. This is called software masking.
- All maskable interrupts are disabled whenever an interrupt is recognized.
- All maskable interrupts can be disabled by executing the DI instruction. RST 7.5 alone has a flip-flop to recognize edge transition. The DI instruction reset interrupt enable flip-flop in the processor and the interrupts are disabled. To enable interrupts, EI instruction has to be executed.

RST 7.5 alone has a flip-flop to recognize edge transition. The DI instruction reset interrupt enable flip-flop in the processor and the interrupts are disabled. To enable interrupts, EI instruction has to be executed.

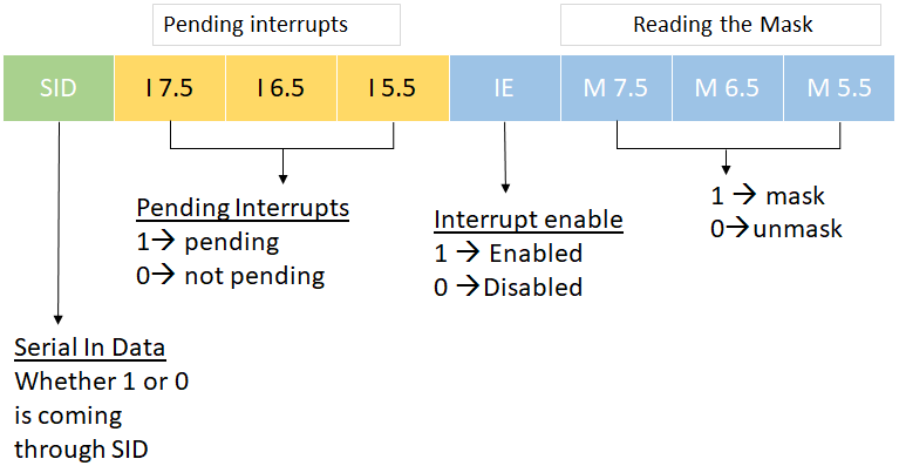
SIM Instruction:

The SIM instruction is used to mask or unmask RST hardware interrupts. When executed, the SIM instruction reads the content of accumulator and accordingly mask or unmask the interrupts.



RIM Instruction:

RIM instruction is used to read the status of the interrupt mask bits. When RIM instruction is executed, the accumulator is loaded with the current status of the interrupt masks and the pending interrupts.



UNIT-II

Pre - requisite:

- **To Study the Architecture of 8086 and configurations**

Outcomes

- **Analyze the basic concepts of 8086bmicroprocessor and instructions set**
- **To learn the design aspects of I/O and minimum , maximum mode circuits.**

INTRODUCTION 8086 MICROPROCESSOR

It is a semiconductor device consisting of electronic logic circuits manufactured by using either a Large scale (LSI) or Very Large Scale (VLSI) Integration Technique. It includes the ALU, register arrays and control circuits on a single chip. The microprocessor has a set of instructions, designed internally, to manipulate data and communicate with peripherals.

The era microprocessors in the year 1971, the Intel introduced the first 4-bit microprocessor is 4004. Using this the first portable calculator is designed. The 16-bit Microprocessor families are designed primarily to complete with microcomputers and are oriented towards high-level languages. They have powerful instruction sets and capable of addressing megabytes of memory. The era of 16-bit Microprocessors began in 1974 with the introduction of PACE chip by National Semiconductor. The Texas Instruments TMS9900 was introduced in the year 1976. The Intel 8086 commercially available in the year 1978, Zilog Z800 in the year 1979, The Motorola MC68000 in the year 1980. The 16-bit Microprocessors are available in different pin packages. Ex: Intel 8086/8088 40 pin package Zilog Z8001 40 pin package, Digital equipment LSI-II 40 pin package, Motorola MC68000 64 pin package National Semiconductor NS16000 48 pin package.

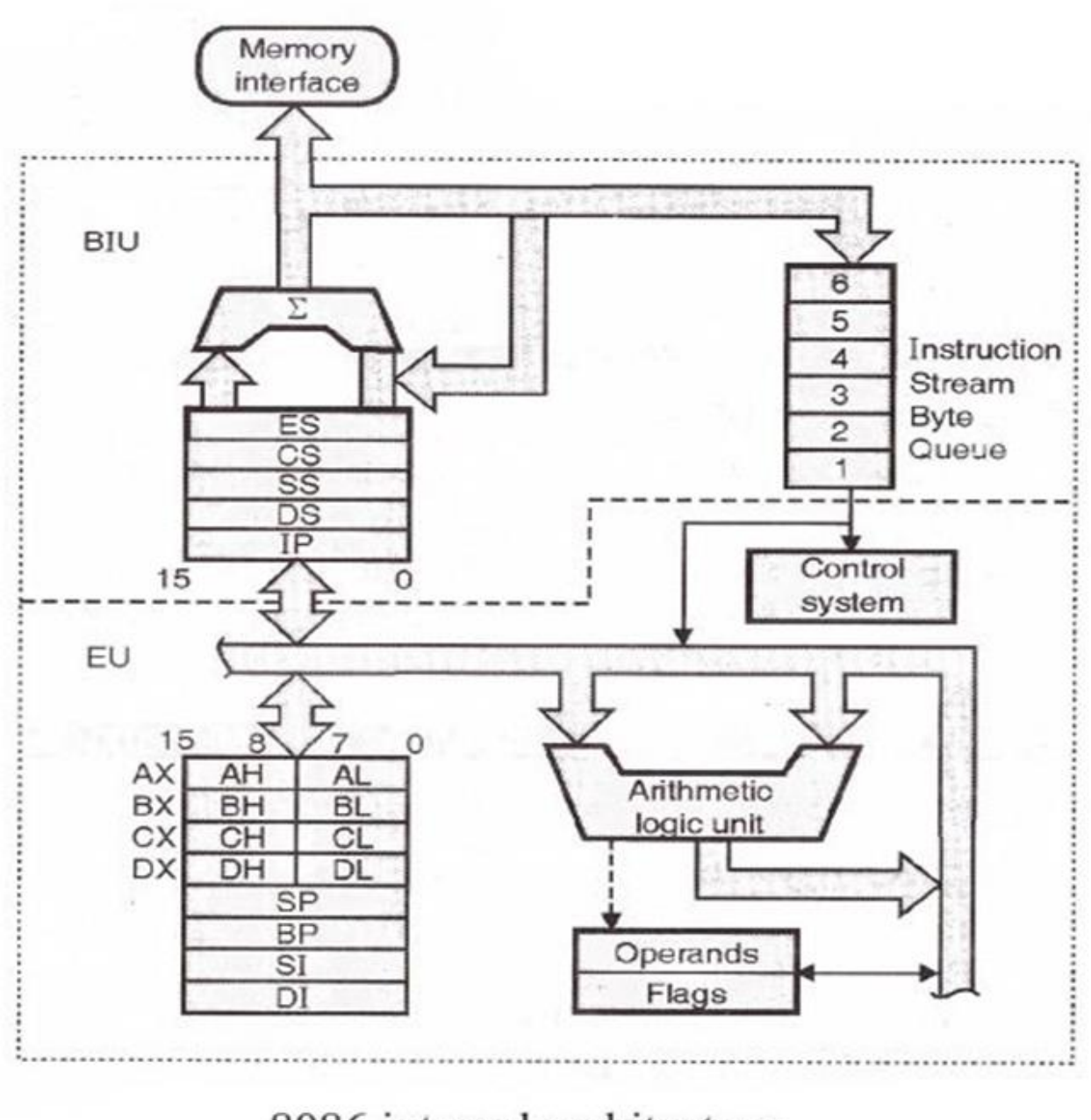
The primary objectives of this 16-bit Microprocessor can be summarized as follows.

1. Increase memory addressing capability
2. Increase execution speed
3. Provide a powerful instruction set
4. Facilitate programming in high-level languages.

Microprocessor Architecture:

The 8086 CPU is divided into two independent functional parts, the Bus interface unit (BIU) and execution unit (EU).
The Bus Interface Unit contains Bus Interface Logic, Segment registers, Memory addressing logic and a Six byte instruction object code queue. The BIU sends out address, fetches the instructions from memory, read data from ports and memory, and writes the data to ports and memory.
The execution unit: contains the Data and Address registers, the Arithmetic and Logic Unit, the Control Unit and flags. tells the BIU where to fetch instructions or data from, decodes instructions and executes instruction. The EU contains control circuitry which directs internal operations. A decoder in the EU translates instructions fetched from memory into a series of actions which the EU carries out. The EU is has a 16-bit ALU which can add, subtract, AND, OR, XOR, increment, decrement, complement or shift binary numbers. The EU is decoding an instruction or executing an instruction which does not require use of the buses.

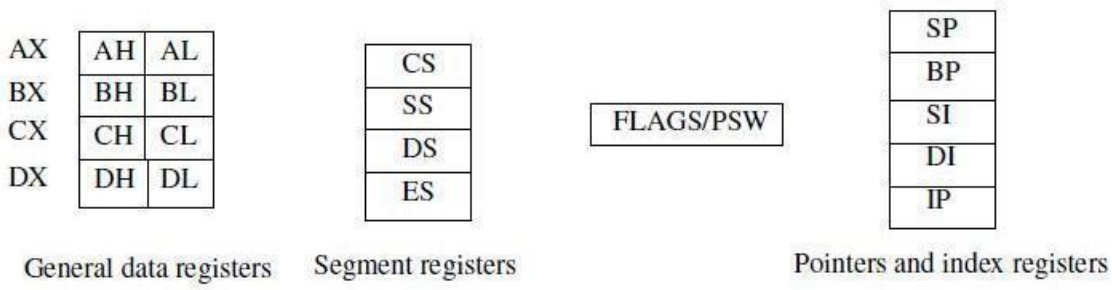
cx



8086 internal architecture

Register organization of 8086:

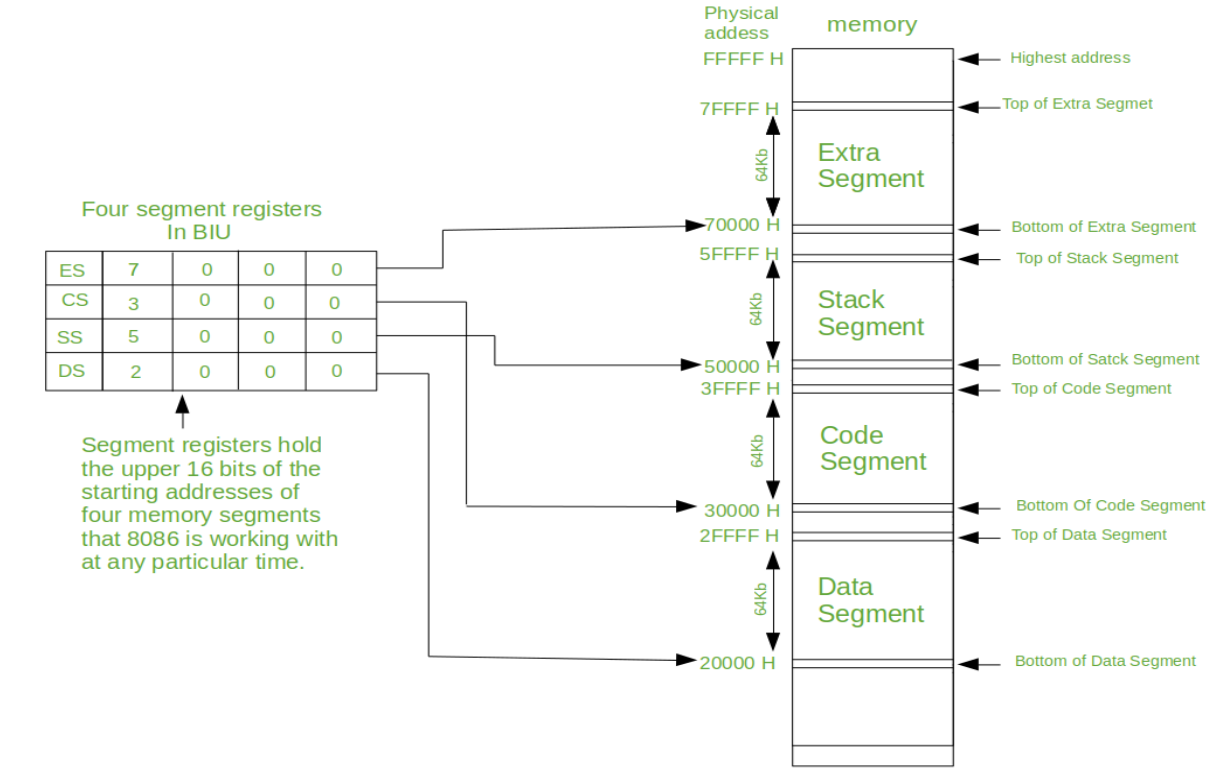
All the registers of 8086 are 16-bit registers. The general purpose registers, can be used either 8-bit registers or 16-bit registers used for holding the data, variables and intermediate results temporarily or for other purpose like counter or for storing offset address for some particular addressing modes etc. The special purpose registers are used as segment registers, pointers, index registers or as offset storage registers for particular addressing modes.



AX Register: Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16- bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations, rotate and string manipulation.
BX Register: This register is mainly used as a **base register**. It holds the starting base location of a memory region within a data segment. It is used as offset storage for forming physical address in case of certain addressing mode.
CX Register: It is used as default counter - **count register** in case of string and loop instructions.
D**X Register: Data register** can be used as a port number in I/O operations and implicit operand or destination in case of few instructions. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

Segment registers:

Memory Segmentation



1Mbyte memory is divided into 16 logical segments. The complete 1Mbyte memory segmentation is as shown in fig 1.4. Each segment contains 64Kbyte of memory. There are four segment registers.
Code segment (CS) is a 16-bit register containing address of 64 KB segment .The CS register is automatically updated during far jump, far call and far return instructions. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.

Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction. It is used for addressing stack segment of memory. The stack segment is that segment of memory, which is used to store stack data.

Data segment (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions. It points to the data segment memory where the data is resided.

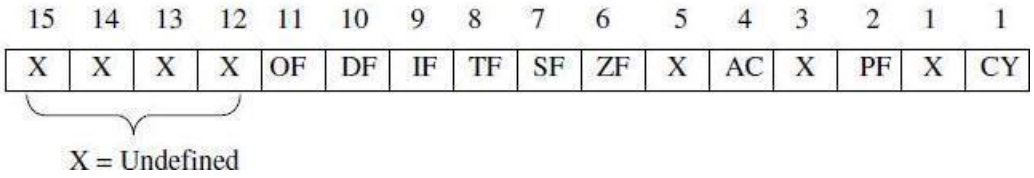
Extra segment (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions. It also refers to segment which essentially is another data segment of the memory.

Pointers and index registers.

The pointers contain within the particular segments. The pointers IP, BP, SP usually contain offsets within the code, data and stack segments respectively
Stack Pointer (SP) is a 16-bit register pointing to program stack in stack segment.
Base Pointer (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.
Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation instructions.

Destination Index (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

Flag Register:



Flags Register determines the current state of the processor. They are modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program.
8086 has 9 active flags and they are divided into two categories:
1. Conditional Flags
2. Control Flags

Conditional Flags

Carry Flag (CY): This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple-precision arithmetic.
Auxiliary Flag (AC): If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. $D_0 – D_3$) to upper nibble (i.e. $D_4 – D_7$), the AC flag is set i.e. carry given by D_3 bit to D_4 is AC flag. This is not a general-purpose flag, it is used internally by the Processor to perform Binary to BCD conversion.
Parity Flag (PF): This flag is used to indicate the parity of result. If lower order 8- bits of the result contains even number of 1‘s, the Parity Flag is set and for odd number of 1‘s, the Parity flag is reset.
Zero Flag (ZF): It is set; if the result of arithmetic or logical operation is zero else it is reset.
Sign Flag (SF): In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set.

Control Flags

Control flags are set or reset deliberately to control the operations of the execution unit. Control flags are as follows:

Trap Flag (TF): It is used for single step control. It allows user to execute one instruction of a program at a time for debugging. When trap flag is set, program can be run in single step mode.
Interrupt Flag (IF): It is an interrupt enable/disable flag. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled. It can be set by executing instruction sit and can be cleared by executing CLI instruction.
Direction Flag (DF): It is used in string operation. If it is set, string bytes are accessed from higher memory address to lower memory address. When it is reset, the string bytes are accessed from lower memory address to higher memory address.

Addressing Modes

The 8086 has 12 addressing modes can be classified into five groups.

- Addressing modes for accessing immediate and register data (register and immediate modes).
- Addressing modes for accessing data in memory (memory modes)
- Addressing modes for accessing I/O ports (I/O modes)
- Relative addressing mode
- Implied addressing mode

Immediate addressing mode:

- In this mode, 8 or 16 bit data can be specified as part of the instruction - OP Code Immediate Operand
- Example 1: MOV CL, 03 H:Moves the 8 bit data 03 H into CL
- Example 2: MOV DX, 0525 H: Moves the 16 bit data 0525 H into DX
- In the above two examples, the source operand is in immediate mode and the destination operand is in register mode. A constant such as ―VALUE‖ can be defined by the assembler EQUATE directive such as VALUE EQU 35H
- Example: MOV BH, VALUE Used to load 35 H into BH

Direct addressing mode:
In the direct addressing mode, a 16-bit memory address (offset) is directly specified in the instruction as a part of it.

Example: MOV AX,[5000H]
Here, data resides in a memory location in the data segment, whose effective address may be computed using 5000H as the offset address and content of DS as segment address. The effective address, hare, is 10H*DS+5000H

Register addressing mode:
In register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers, except IP, may be used in this
mode.

Example: MOV AX,BX

Register Indirect addressing mode:

Sometimes, the address of the memory location which contains data or operand is determined in an indirect way, using the offset registers. This mode of addressing is known as register indirect mode. In this addressing mode, the offset address of data is in either BX or SI or DI register. The default segment is either DS or ES. The data is supposed to be available at the address pointed to by the content of any of the above registers in the default data segment.

Example: MOV AX,[BX]
Here, data is present in a memory location in DS whose offset address is in BX. The effective address of the data is given as 10H*DS+[BX].

Indexed addressing mode:

In this addressing mode, offset of the operand is stored in one of the index registers. DS and ES are the default segments for index register SI and DI respectively. This mode is a special case of the above discussed register indirect addressing
mode.

Example: MOV AX,[SI]
Here, data is available at an offset address stored in SI in DS. The effective address, in this case, is computed as 10H*DS+[SI].

Register Relative addressing mode:

In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI and DI in the default (either DS or ES) segment. The example given below explains this mode.

Example: MOV AX,[BX]50H
Here, the effective address is given as 10H*DS+50H+[BX].

Based Indexed addressing mode:

The effective address of data is formed, in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

Example: MOV AX,[SI] [BX]
Here, BX is the base register and SI is the index register. The effective address is computed as 10H*DS+[BX]+[SI].

Relative Based Indexed addressing mode:

The effective address is formed by adding an 8 or 16-bit displacement with the sum of contents of any one of the base registers (BX or BP) and any one of the index registers, in a default segment.

Example: MOV AX,[SI] [BX]
Here, 50H is an immediate displacement, BX is a base register and SI is an index register. The effective address of data is computed as 160H*DS+[BX]+[SI]+50H.

Instruction Set of the 8086 processor

The 8086/8088 instructions are categorized into the following main types. This section explains the function of each of the instructions with suitable examples wherever necessary.

Data Copy/Transfer Instructions

This types of instructions are used to transfer data from source operand to destination operand. All the store, move, load, exchange, input and output instructions belong to this category.

MOV: Move This data transfer instruction transfers data from one register/memory location to another register/memory location. The source may be any one of the segment registers or other general or special purpose registers or a memory location and , another register or memory location may act as destination.

MOV AX, 5000H
MOV DS, AX
It may be noted, here, the both source and destination operands cannot be memory locations (Except for string instructions). Other MOV instruction example is given below with the corresponding addressing modes.
MOV AX, BX; Register

IN: Input the port This instruction is used for reading an input port. The address of the input port may be specified in the instruction directly or indirectly. AL and AX are the allowed destinations for 8 and 16-bit input operations. DX is the only register (implicit) which is allowed to carry the port address.

Example
1. IN AL, 0300H ; This instruction reads data from an 8-bitport whose address is 0300H and stores it in AL.

OUT: Output to the Port This instruction is used for writing to an output port. The address of the output port may be specified in the instruction directly or implicitly in DX. Contents of AX or AL are transferred to a directly or indirectly addressed port after execution of this instruction. The data to an odd addressed port is transferred on D8-D15 while that to an even addressed port is transferred port is transferred on D0-D7. The registers AL and AX are the allowed source operands for 8-bit and 16-bit operations respectively.

Example
1. OUT 0300H, AL ; This sends data available in AL to a port whose address is 0300H.

Arithmetic and Logical Instructions All the instructions performing arithmetic, logical, increment, decrement, compare and scan instructions belong to this category.

ADD: Add This instruction adds an immediate data or contents of a memory location specified in the instruction or a register (source) to the contents of another register (destination) or memory location. The result is in the destination operand. However, both the source and destination operands cannot be memory operands. The examples of this instruction is given along with the corresponding modes.

Example
1. ADD AX, 0100hH Immediate

INC: Increment This instruction increments the contents of the specified register or memory location by 1. All the condition code flags are affected except the carry flag CF. This instruction adds 1 to the contents of the operand. Immediate data cannot be operand of this instruction. The examples of this instruction is as follows:

Example
1. INC AX Register

DEC: Decrement The decrement instruction subtracts 1 from the contents of the specified register of memory location. The examples of this instruction is as *follows:*

Example
1. DEC AX Register

SUB: Subtract The subtract instruction subtracts the source operand from the destination operand and the result is left in the destination operand. The examples of this instruction along with the addressing modes are as follows:

Example
1. SUB 0100H Immediate [destination AX]

MUL: Unsigned Multiplication Byte or Word This instruction multiplies an unsigned byte or would by the contents of AL. The unsigned byte or word may be in any one of the general purpose registers or memory locations. The most significant word of the result is stored in DX, while the least significant word of the result is stored in AX. All the flags are modified depending upon the result. The example instructions are as shown. Immediate operand is not allowed in this instruction. If the most significant byte or word or the result is '0' CF and OF both will be set.

Example
1. MUL BH ; (AX) (AL) x (BH)
2. MUL CX ; (DX) (AX) (AX) x (CX)

DIV: Unsigned Division This instruction performs unsigned division. It division an unsigned word or double word by a 16-bit or 8-bit operand. The dividend must be in AX for 16-bit operation and divisor may be specified using any one of the addressing modes except immediate. The result ill be in AL (quotient) while AH will contain the remainder. If the result is too big to fit in AL, type 0 (divide by zero) interrupt is generated. In case of double would divided (32-bit), the higher word should be in DX and lower would should be in AX. The divisor may be specified as already explained. The quotient and the remainder, in this case, will be in AX and DX respectively. The instruction does not affect any **flag.**

Logical Instructions

These type of instructions are used for carrying out the bit by bit shift, rotate, or basic logical operations. All the condition code flags are affected upon the result. Basic logical operations available with 8086 instruction set are AND, OR, NOT, and XOR. The instruction for each of these operations are discussed as follows.

AND: Logical AND This instruction bit by bit ANDs the source operand that may be an immediate, a register or a memory location to the destination operand that may be a register or a memory location. The result is stored in the destination operand. At least one of the operands should be a register or a memory operand. Both the operands cannot be memory locations or immediate operands. An immediate operand cannot be a destination operand.
The example of this instruction is as follows:

Example
1. AND AX, 0008H

OR: Logical OR The OR instruction carries out the OR operation in the same way as described in case of the AND operation. The limitations on source and destination operands are also the same as in case of AND operation. The example are as follows:

Example

1. OR AX, 0098H
2. OR AX, BX

XOR: Logical Exclusive OR The XOR operation is again carried out in a similar way to the operation. The constraints on the operands are also similar. The XOR operation gives a high output, when the 2 input bits are dissimilar. Otherwise, the output is zero. The example instructions is as follows:

Shift and Rotate Instructions There instructions involve the bitwise shifting rotating in either direction with or without a count in CX.

SHIFT INSTRUCTIONS:

The four shift instructions of the 8086 can perform two basic types of shift operations; the **logical shift and the arithmetic shift.** Moreover, each of these operations can be performed to the right or to the left.
The shift instructions are **shift logical left (SHL), shift arithmetic left (SAL), shift logical right (SHR) and shift arithmetic right (SAR).** These instructions are used to align data, to isolate bits of a byte or word so that it can be tested, and to perform simple multiply and divide computations.
SAL / SHL Instruction :
SAL destination, count
SHL destination , count

SAL and SHL are two mnemonics for the same instruction. This instruction shifts each bit in the specified destination to the left and 0 is stored at LSB position. The MSB is shifted into the carry flag. The destination can be a byte or a word. It can be in a register or in a memory location. The number of shifts are indicated by count. But if the number of shifts required is one, you can place 1 in the count position. If number of shifts are greater than 1 then shift count must be loaded in CL register and CL must be placed in the count position of the instruction.

ROL Instruction : This instruction rotates all bits in a specified byte or word to the left some number of bit positions. MSB is placed as a new LSB and a new CF. The destination can be byte or a word. It can be in a register or in a memory location. The number of shifts are indicated by count. If number of shifts required is one you can place 1 in the count position. If number of shifts are greater than 1 then shift count must be loaded in CL register and CL must be placed in the count position of the instruction.

String Manipulation Instructions

A series of data bytes or words available in memory at consecutive locations, to be referred to collectively of individually, are called as byte strings or word strings. For referring to a string, two parameters are required, (a) starting or end address of the string and (b) length of the string. The length of a string is usually stored as count in CX register.

REP / REPE / REPZ / REPNE / REPNZ instructions: REP is a prefix which is written

before one of the string instructions. These instructions repeat until specified condition exists.

MOVSB / MOVSW INSTRUCTION
These instructions are used to move one byte or one word from one location to another location. MOVSB is used to move byte strings and MOVSW is used to move word strings.

Example : Assume STRING1 is a string of bytes. To copy STRING1 (forward) to STRING2, the following statements are used.

```
LEA SI , STRING1 ;copy the offset of the STRING1 to SI
LEA DI , STRING2 ;copy the offset of the STRING2 to DI
MOV CX,LENGTH STRING1 ; copy the length of the STRING1 to CX
CLD ; clear direction flag for forward move
REP MOVSB
```

CMPSB / CMPSW INSTRUCTIONS:

These instructions are used to compare one byte or one word from one location to another location. CMPSB is used to compare byte strings and CMPSW is used to compare word strings. The CMPS instruction can be used with REP , REPE or REPNE prefix to compare all the elements of a string.

Branch Instructions There instructions have REP prefix with CX used as count register, they can be used to implement unconditional and conditional loops. The LOOP, LOOPNZ and LOOPZ instructions belong to this category. These are useful to implement different loop structure.

Unconditional Control Transfer (Branch) Instruction: In case of unconditional control transfer instructions, the execution control is transferred to the specified location independent of any status or conditional. The CS and IP are unconditionally modified to the new CS and IP.
JMP: Unconditional Jump This instruction unconditionally transfers the control of execution to the specified address using an 8-bit or 16-bit displacement (intrasegment relative, short or long) or CS: IP (intersegment direct far). No flags are affected by this instruction. Corresponding to the three methods of specifying jump addresses

Conditional Control Transfer (Branch) Instructions: In the conditional control transfer instructions, the control is transferred to the specified location provided the result of the previous operation satisfies a particular condition, otherwise, the execution continues in normal flow sequence. The results of the previous operations are replicated by condition code flags.

JCXZ ‘Label’ Transfer execution control to address ‘Label’, if CX=0.

Loop Instructions If these instructions have REP prefix with CX used as count register, they can be used to implement unconditional and conditional loops. The LOOP, LOOP, LOOPNZ and LOOPZ instructions belong to this category. These are useful to implement different loop structures.

LOOPZ / LOOPE Label Loop through a sequence of instructions
from 'Label' while ZE=0 and CX = 0.
LOOPZ / LOOPENE Label Loop through a sequence of instructions
from 'Label' while ZE=0 and CX = 0.

Machine Control Instructions There instructions control the machine status.
NOP, HLT, WAIT and LOCK instructions belong to this class.

WAIT - Wait for Test input pin to go low
HIT - Halt the processor
NOP - No operation
After executing the HLT instruction, the processor enters the halt state. The two ways to pull it out of the halt state are to reset the processor or to interrupt it. When NOP instruction

Flag Manipulation Instructions All the instructions which directly affect the flag register, come under this group of instructions. Instructions like CLD, STD, CLI, STI, etc. belong to this category of instructions.

CLC - Clear carry flag
STC - Set carry flag
CLD - Clear direction flag
STD - Set direction flag

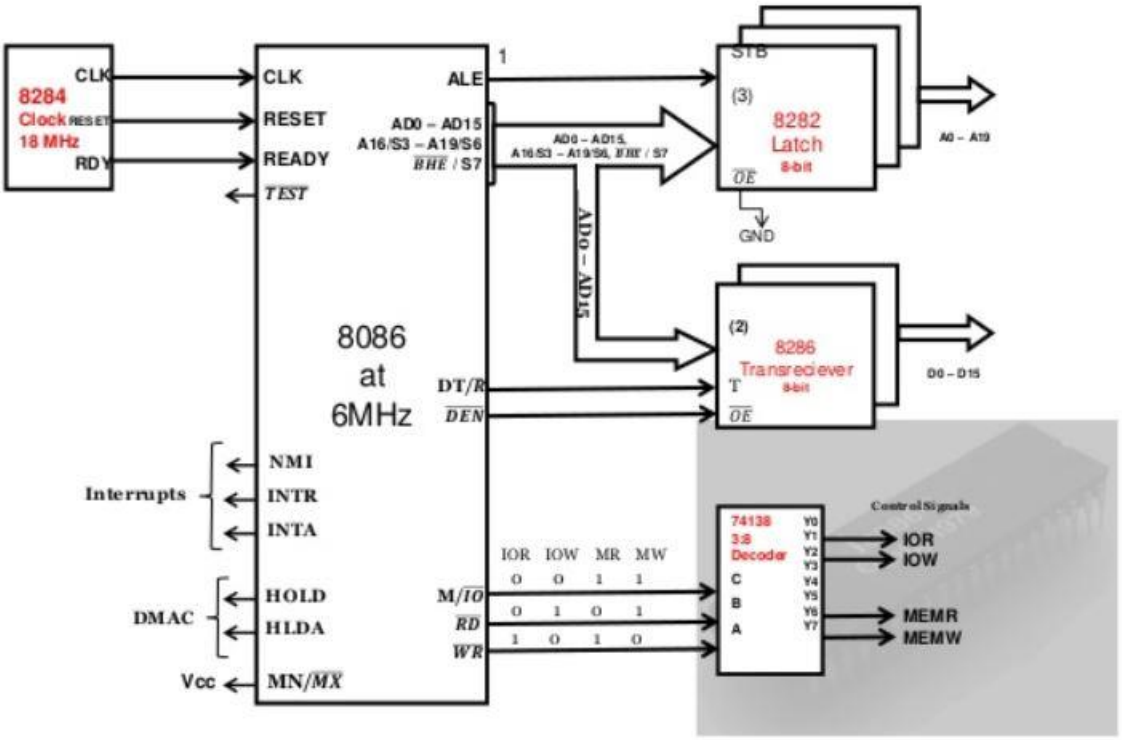
These instruction modify the carry(CF), direction(DF) and interrupt(IF) flags directly. The DF and IF, which may be modified using the flag manipulation instructions, further control the processor operation; like interrupt responses and auto increment or auto decrement modes.

Minimum mode Operation and Maximum Mode Operation

In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX* pin to logic1. In this mode, all the control signals are given out by the microprocessor chip itself. There is a single microprocessor in the minimum mode system. The remaining components in the system are latches, transceivers, clock generator, memory and I/O devices.

The opcode fetch and read cycles are similar. Hence the timing diagram can be categorized in two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle. The read cycle timing diagram. The read cycle begins in T1 with the assertion of the address latch enable (ALE) signal and also M/IO* signal. During the negative going edge of this signal, the valid address is latched on the local bus. The BHE* and A0 signals address low, high or both bytes. From Tl to T4, the M/IO* signal indicates a memory or I/O operation. At T2 the address is removed from the local bus and is sent to the output. The bus is then tristated. The read (RD*) control signal is also activated in T2 .The read (RD) signal causes the addressed device to enable its data bus drivers. After RD* goes low, the valid data is available on the data bus. The addressed

device will drive the READY line high, when the processor returns the read signal to high level, the addressed device will again tristate its bus drivers.

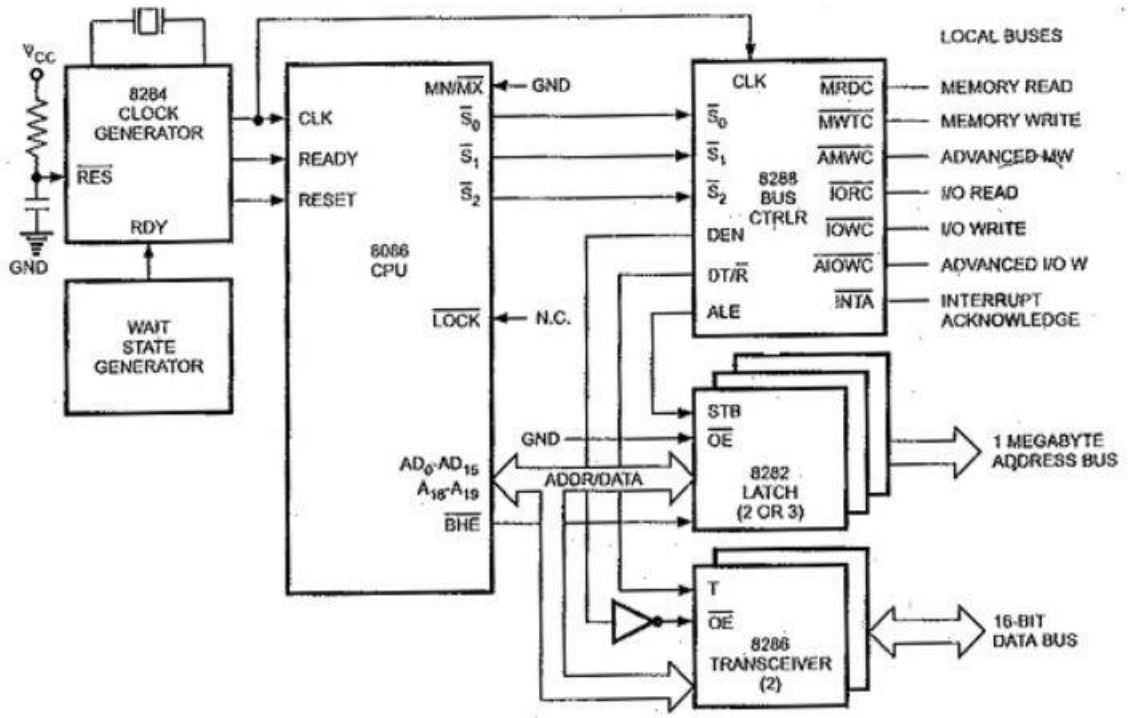


A write cycle also begins with the assertion of ALE and the emission of the address. The M/IO* signal is again asserted to indicate a memory or I/O operation. In T2 after sending the address in Tl the processor sends the data to be written to the addressed location. The data remains on the bus until middle of T4 state. The WR* becomes active at the beginning ofT2 (unlike RD* is somewhat delayed in T2 to provide time for floating). The BHE* and A0 signals are used to select the proper byte or bytes of memory or I/O word to be read or written. The M/IO*, RD* and WR* signals indicate the types of data transfer as specified in Table

In the maximum mode, the 8086 is operated by strapping the MN/MX* pin to ground. In this mode, the processor derives the status signals S2*, S1* and S0*. Another chip called bus controller derives the control signals using this status information. In the maximum mode, there may be more than one microprocessor in the system configuration.

The basic functions of the bus controller chip IC8288, is to derive control signals like RD* and WR* (for memory and I/O devices), DEN*, DT/R*, ALE, etc. using the information made available by the processor on the status lines. The bus controller chip has input lines S2*, S1* and S0* and CLK. These inputs to 8288 are driven by the CPU. It derives the outputs ALE, DEN*, DT/R*, MWTC*, AMWC*,

IORC*, IOWC* and AIOWC*. The AEN*, IOB and CEN pins are especially useful for multiprocessor systems. AEN* and IOB are generally grounded. CEN pin is usually tied to +5V.



The significance of the MCE/PDEN* output depends upon the status of the IOB pin. If IOB is grounded, it acts as master cascade enable to control cascaded 8259A; else it acts as peripheral data enable used in the multiple bus configurations. INTA* pin is used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device. IORC*, IOWC* are I/O read command and I/O write command signals respectively.

These signals enable an IO interface to read or write the data from or to the addressed port. The MRDC*, MWTC* are memory read command and memory write command signals respectively and may be used as memory read and write signals. All these command signals instruct the memory to accept or send data from or to the bus.

For both of these write command signals, the advanced signals namely AIOWC* and AMWTC* are available. They also serve the same purpose, but are activated one clock cycle earlier than the IOWC* and MWTC* signals, respectively. The maximum mode system timing diagrams are also divided in two portions as read (input) and write (output) timing diagrams. The address/data and address/status timings are similar to the minimum mode. ALE is asserted in T1, just like minimum mode. The only difference lies in the status signals used and the available control and advanced command signals.

ASSEMBLY LANGUAGE PROGRAMMING

ALP for addition of two 8-bit numbers

```
MOV DS, AX
MOV AL, VAR1
MOV BL, VAR2
ADD AL, BL
MOV RES, AL
MOV AH, 4CH
INT 21H
CODE ENDS
END START
```

ALP for Subtraction of two 8-bit numbers

```
START: MOV AX,DATA
MOV DS,AX
MOV AL,VAR1
MOV BL,VAR2
SUB AL,BL
MOV RES,AL
MOV AH,4CH
INT 21H
CODE ENDS
```

ALP for Multiplication of two 8-bit numbers

```
START: MOV AX, DATA MOV DS, AX
MOV AL, VAR1
MOV BL, VAR2
MUL BL
MOV RES, AX MOV AH, 4CH INT 21H
CODE ENDS
END START
```

ALP for division of 16-bit number with 8-bit number

```
START: MOV AX, DATA MOV DS, AX
MOV AX, VAR1
DIV VAR2
MOV QUO, AL
MOV REM, AH
MOV AH, 4CH
INT 21H
CODE ENDS
END START
```

UNIT-III

Pre - requisite:

- **To Study about communication and bus interfacing.**

Outcomes

- **Design interfacing peripherals like, I/O, A/D, D/A, timer etc.**

Memory Devices and Interfacing

Any application of a microprocessor based system requires the transfer of data between external circuitry to the microprocessor and microprocessor to the external circuitry. Most of the peripheral devices are designed and interfaced with a CPU either to enable it to communicate with the user or an external process and to ease the circuit operations so that the microprocessor works more efficiently.
The use of peripheral integrated devices simplifies both the hardware circuits and software considerable. The following are the devices used in interfacing of Memory and General I/O devices

• 74LS138 (Decoder / Demultiplexer).
• 74LS373 / 74LS374 3-STATE Octal D-Type Transparent Latches.
• 74LS245 Octal Bus Traniver: 3-State.

74LS138 (Decoder / Demultiplexer)

The LS138 is a high speed 1-of-8 Decoder/ Demultiplexer fabricated with the low power Schottky barrier diode process. The decoder accepts three binary weighted inputs (A0, A1, A2) and when enabled provides eight mutually exclusive active LOW Outputs (O0–O7).
The LS138 can be used as an 8-output demultiplexer by using one of the active LOW Enable inputs as the data input and the other Enable inputs as strobes. The Enable inputs which are not used must be permanently tied to their appropriate active HIGH or
active LOW state.

74LS373 / 74LS374 3-STATE Octal D-Type Transparent Latches and Edge-Triggered Flip-Flops

These 8-bit registers feature totem-pole 3-STATE outputs designed specifically for implementing buffer registers, I/O ports, bidirectional bus drivers, and working registers. The eight latches of the 74LS373 are transparent D type latches meaning that while the enable (G) is HIGH the Q outputs will follow the data (D) inputs.
When the enable is taken LOW the output will be latched at the level of the data that was
set up. The eight flip-flops of the 74LS374 are edge-triggered D-type flip flops. On the positive transition of the clock, the Q outputs will be set to the logic states that were set up at the D inputs.

Main Features

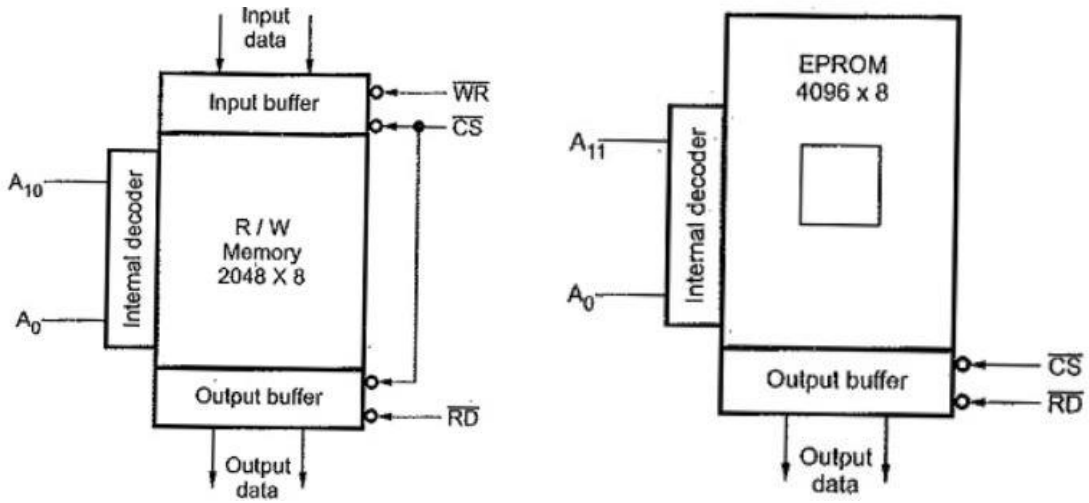
• Choice of 8 latches or 8 D-type flip-flops in a single package
• 3-STATE bus-driving outputs
• Full parallel-access for loading
• Buffered control inputs
• P-N-P inputs reduce D-C loading on data lines

74LS245 Octal Bus Traniver: 3-State

The 74LS245 is a high-speed Si-gate CMOS device. The 74LS245 is an octal traniver featuring non-inverting 3-state bus compatible outputs in both send and receive directions. The 74LS245 features an Output Enable (OE) input for easy cascading and a send/receive
(DIR) input for direction control. OE controls the outputs so that the buses are effectively isolated. All inputs have a Schmitt-trigger action.
These octal bus tranivers are designed for asynchronous two-way communication between data buses.
The 74LS245 is a high-speed Si-gate CMOS device. The 74LS245 is an octal traniver featuring non-inverting 3-state bus compatible outputs in both send and receive directions.
The 74LS245 features an Output Enable (OE) input for easy cascading and a send/receive (DIR) input for direction control. OE controls the outputs so that the buses are effectively isolated. All inputs have a Schmitt-trigger action. These octal bus tranivers are designed for asynchronous two-way communication between data buses.

Memory Devices And Interfacing

The memory interfacing circuit is used to access memory quit frequently to read instruction codes and data stored in the memory. The read / write operations are monitored by control
signals. Semiconductor memories are of two types. Viz. RAM (Random Access Memory) and ROM (Read Only Memory) The Semiconductor RAM‘s are broadly two types- static Ram and dynamic RAM



Memory structure and its requirements

The read / write memories consist of an array of registers in which each register has unique address. The size of memory is N * M as shown in figure.
Where N is number of register and M is the word length, in number of bits. As shown in figure(a) memory chip has 12 address lines A**o**–A**11**, one chip select (CS), and two control lines, Read (RD) to enable output buffer and Write (WR) to enable the input buffer.
The internal decoder is used to decoder the address lines. Figure(b) shows the logic diagram of a typical EPROM (Erasable Programmable Read-Only Memory) with 4096 (4K) register. It has 12 address lines A**o** **–** A**11,** one chip select (CS)**,** one read control signal. Since EPROM does not require the (WR) signal.

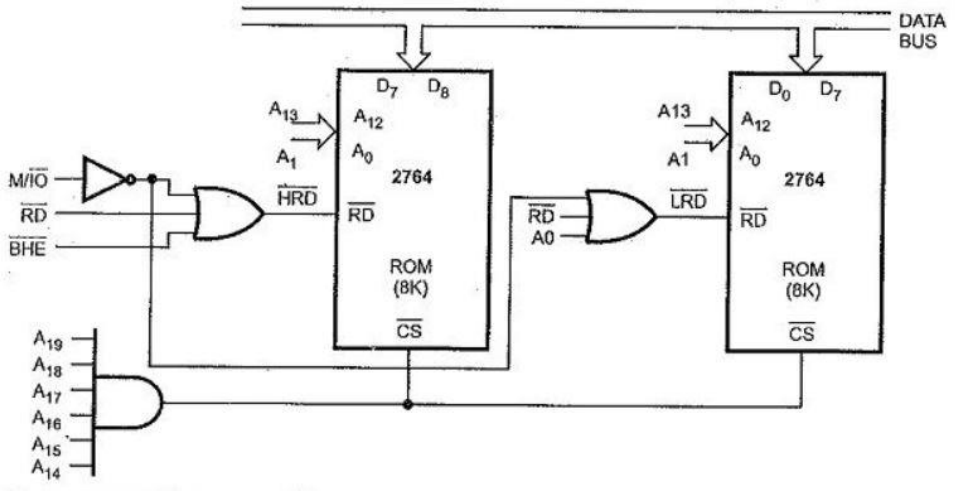
EPROM (or EPROMs) is used as a program memory and RAM (or RAMs) as a data memory. When both, EPROM and RAM are used, the total address space 1 Mbytes is shared by them.

Address Decoding Techniques

- Absolute decoding
- Linear decoding
- Block decoding

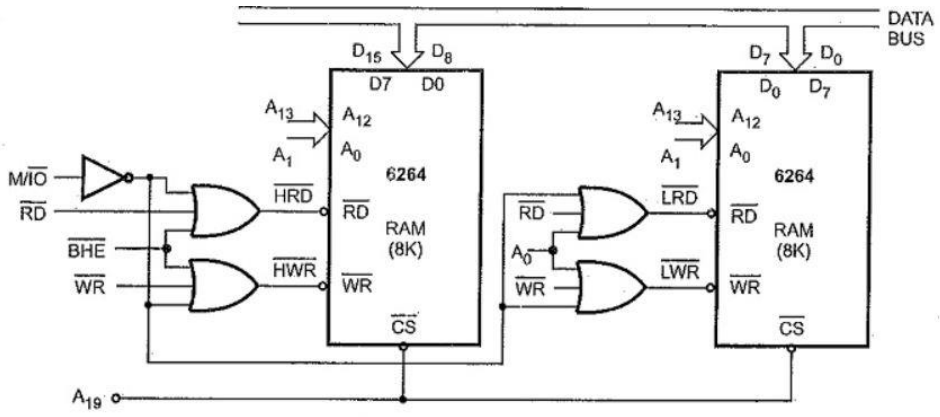
Absolute Decoding:

In the absolute decoding technique the memory chip is selected only for the specified logic level on the address lines: no other logic levels can select the chip. Below figure the memory interface with absolute decoding. Two 8K EPROMs (2764) are used to provide even and odd memory banks. Control signals BHE and Ao are use to enable output of odd and even memory banks respectively. As each memory chip has 8K memory locations, thirteen address lines are required to address each locations, independently. All remaining address lines are used to generate an unique chip select signal. This address technique is normally used in large memory systems



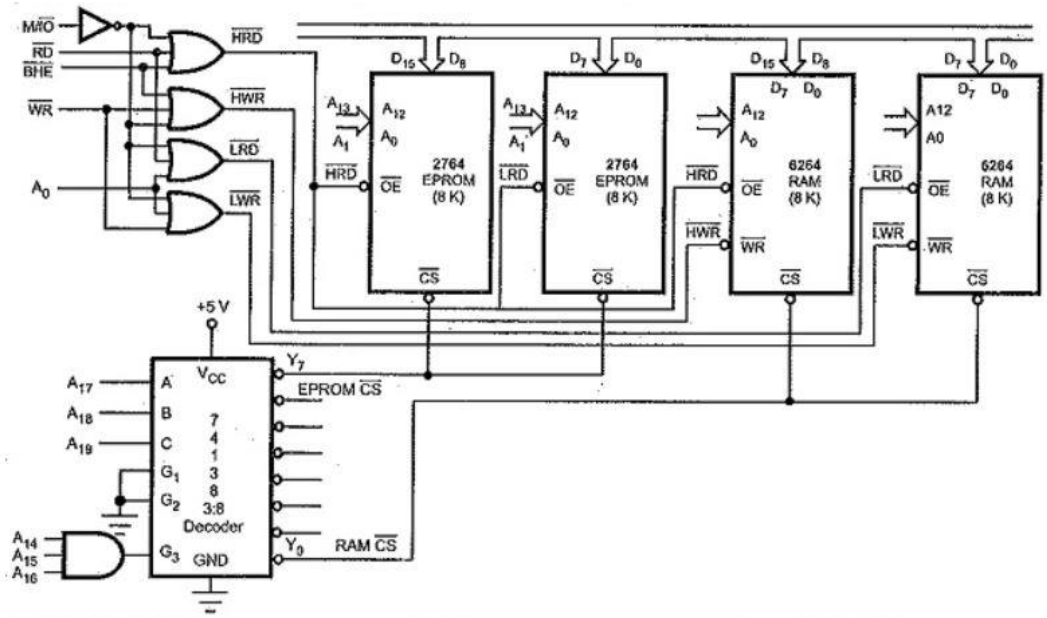
Linear decoding

In small system hardware for the decoding logic can be eliminated by using only required number of addressing lines (not all). Other lines are simple ignored. This technique is referred as linear decoding or partial decoding. Control signals BHE and A**o** are used to enable odd and even memory banks, respectively. Figure shows the addressing of 16K RAM (6264) with linear decoding. The address line A**19** is used to select the RAM chips. When A**19** is low, chip is selected, otherwise it is disabled. The status of A**14** to A**18** does not affect the chip selection logic. This gives you multiple addresses (shadow addresses). This technique reduces the cost of decoding circuit, but it gas drawback of multiple addresses



Block Decoding:

In a microcomputer system the memory array is often consists of several blocks of memory chips. Each block of memory requires decoding circuit. To avoid separate decoding for each memory block special decoder IC is used to generate chip select signal for each block.



Static Memory Interfacing

The general procedure of static memory interfacing with 8086 as follows:

1. Arrange the available memory chips so as to obtain 16-bit data bus width. The upper 8-bit bank is called _odd address memory bank‘ and the lower 8-bit bank is called _even address memory bank‘.
2. Connect available memory address lines of memory chips with those of the microprocessor and also connect the memory RD and WR inputs to the corresponding processor control signals. Connect the 16-bit data bus of the memory bank with that of the microprocessor 8086.
3. The remaining address lines of the microprocessor, BHE and Ao are used for decoding the required chip select signals for the odd and even memory banks. The CS of memory is derived from the output of the decoding circuit.

4. As a good and efficient interfacing practice, the address map of the system should be continuous as far as possible

Dynamic RAM Interfacing

The basic Dynamic RAM cell uses a capacitor to store the charge as a representation of data. This capacitor is manufactured as a diode that is reverse-biased so that the storage capacitance comes into the picture. This storage capacitance is utilized for storing the charge representation of data but the reverse-biased diode has a leakage current that tends to discharge the capacitor giving rise to the possibility of data loss.

To avoid this possible data loss, the data stored in a dynamic RAM cell must be refreshed after a fixed time interval regularly. The process of refreshing the data in the RAM is known as refresh cycle. This activity is similar to reading the data from each cell of the memory, independent of the requirement of microprocessor, regularly. During this refresh period all other operations (accesses) related to the memory subsystem are suspended.

The advantages of dynamic RAM. Like low power consumption, higher packaging density and low cost, most of the advanced computer systems are designed using dynamic RAMs. Also the refresh mechanism and the additional hardware required makes the interfacing hardware, in case of dynamic RAM, more complicated, as compared to static RAM interfacing circuit.

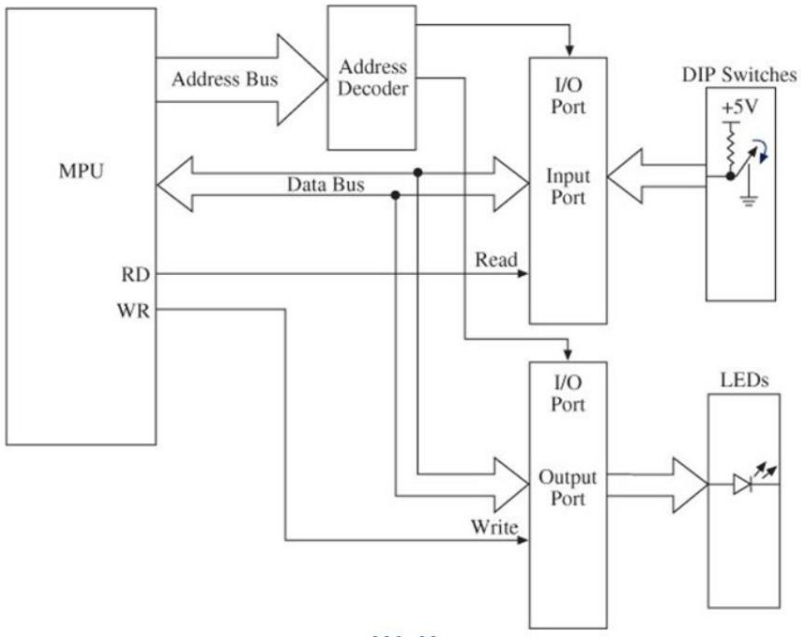
Interfacing I/O Ports

I/O ports or input/output ports are the devices through which the microprocessor communicates with other devices or external data sources/destinations. Input activity, as one may expect, is the activity that enables the microprocessor to read data from external devices, for example keyboard, joysticks, mouser etc. the devices are known as input devices as they feed data into a microprocessor system. Output activity transfers data from the microprocessor top the external devices, for example CRT display, 7-segment displays, printer, etc, the devices that accept the data from a microprocessor system are called output devices.

Steps in Interfacing an I/O Device

The following steps are performed to interface a general I/O device with a CPU:

1. Connect the data bus of the microprocessor system with the data bus of the I/O port.
2. Derive a device address pulse by decoding the required address of the device and use it as the chip select of the device.
3. Use a suitable control signal, i.e. IORD and /or IOWR to carry out device operations, i.e. connect IORD to RD input of the device if it is an input devise, otherwise connect IOWR to WR input of the device. In some cases the RD or WR control signals are combined with the device address pulse to generate the device select pulse.



I/O Interfacing Techniques

Input/output devices can be interfaced with microprocessor systems in two ways:

1. I/O mapped I/O
2. Memory mapped I/O

1. I/O mapped I/O:

8086 has special instructions IN and OUT to transfer data through the input/output ports in I/O mapped I/O system. The IN instruction copies data from a port to the Accumulator. If an 8-bit port is read data will go to AL and if 16-bit port is read the data will go to AX. The OUT instruction copies a byte from AL or a word from AX to the specified port. The M/IO signal is always low when 8086 is executing these instructions. In this address of I/O device is 8-bit or 16-bit. It is 8-bit for Direct addressing and 16-bit for Indirect addressing.

2. Memory mapped I/O

In this type of I/O interfacing, the 8086 uses 20 address lines to identify an I/O device. The I/O device is connected as if it is a memory device. The 8086 uses same control signals and instructions to access I/O as those of memory, here RD and WR signals are activated indicating memory bus cycle.

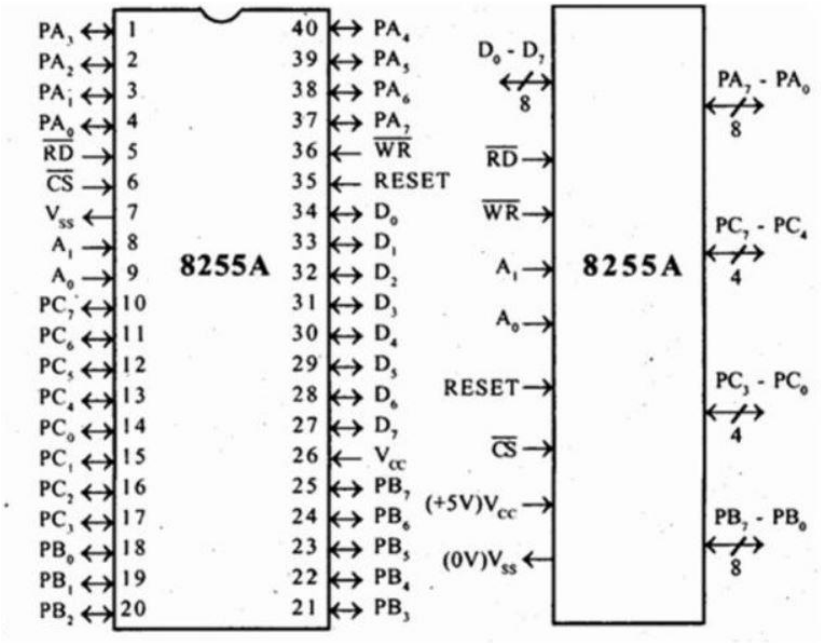
Parallel Communication Interface: 8255 Programmable Peripheral Interface and Interfacing

The 8255 is a widely used, programmable parallel I/O device. It can be programmed to transfer data under data under various conditions, from simple I/O to interrupt I/O. It is flexible, versatile and economical (when multiple I/O ports are required). It is an important general purpose I/O device that can be used with almost any microprocessor.

The 8255 has 24 I/O pins that can be grouped primarily into two 8 bit parallel ports: A and B, with the remaining 8 bits as Port C. The 8 bits of port C can be used as individual bits or be grouped into two 4 bit ports: CUpper (CU) and CLower (CL). The functions of these ports are defined by writing a control word in the control register.

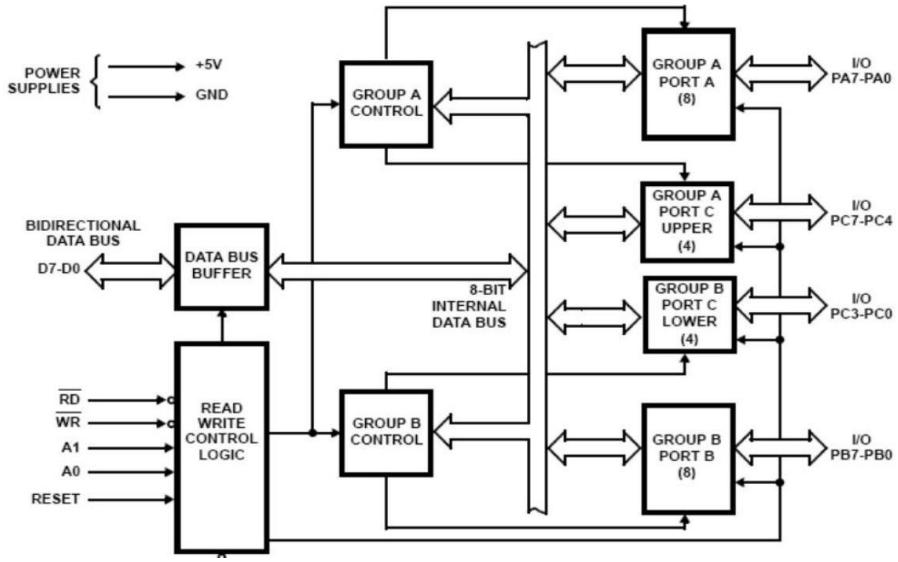
8255 can be used in two modes: Bit set/Reset (BSR) mode and I/O mode. The BSR mode is used to set or reset the bits in port C. The I/O mode is further divided into 3 modes: mode 0, mode 1 and mode 2. In mode 0, all ports function as simple I/O ports.

Mode 1 is a handshake mode whereby Port A and/or Port B use bits from Port C as handshake signals. In the handshake mode, two types of I/O data transfer can be implemented: status check and interrupt. In mode 2, Port A can be set up for bidirectional data transfer using handshake signals from Port C, and Port B can be set up either in mode 0 or mode 1.



RD: (Read): This signal enables the Read operation. When the signal is low, microprocessor reads data from a selected I/O port of 8255.
WR: (Write): This control signal enables the write operation.
RESET (**Reset**): It clears the control registers and sets all ports in input mode. **CS,A0, A1:** These are device select signals. is connected to a decoded address and A0, A1 are connected to A0, A1 of microprocessor.

Block diagram of 8255



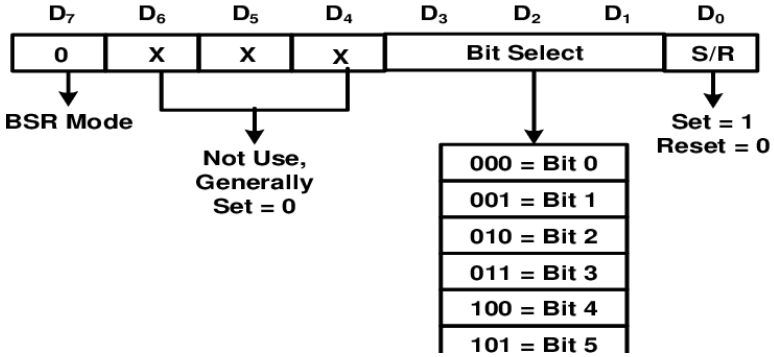
I/O Modes of 8255

Mode 0: Simple Input or Output

In this mode, Port A and Port B are used as two simple 8-bit I/O ports and Port C as two4- bit I/O ports. Each port (or half-port, in case of Port C) can be programmed to function as simply an input port or an output port. The input/output features in mode 0 are: Outputs are latched, Inputs are not latched. Ports do not have handshake or interrupt capability.

Mode 1: Input or Output with handshake

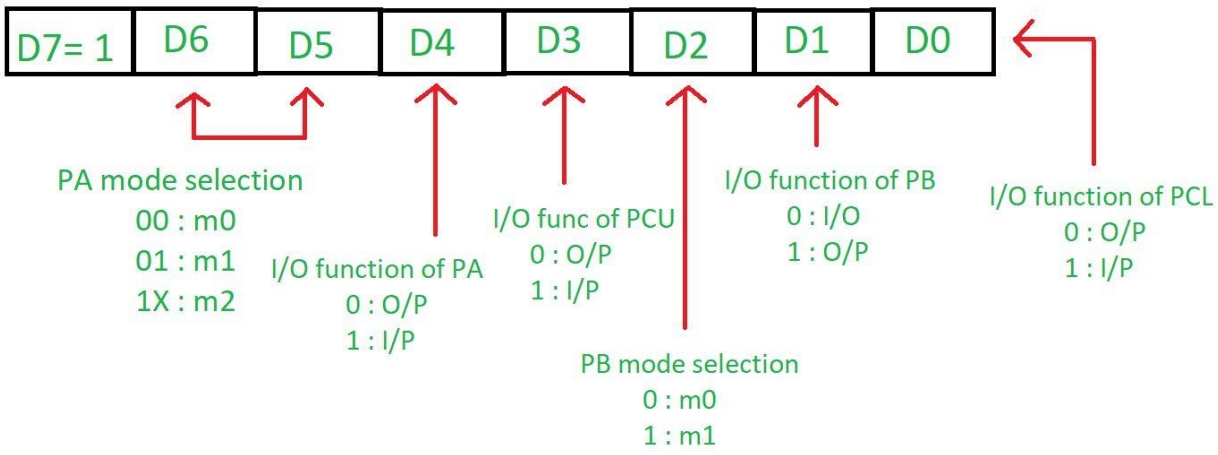
In mode 1, handshake signals are exchanged between the microprocessor and peripherals prior to data transfer. The ports (A and B) function as 8-bit I/O ports. They can be configured either as input or output ports. Each port (Port A and Port B) uses 3 lines from port C as handshake signals. The remaining two lines of port C can be used for simple I/O functions. Input and output data are latched and Interrupt logic is supported.



STB **Strobe Input**): This signal (active low) is generated by a peripheral device that it has transmitted a byte of data. The 8255, in response to, generates IBF and INTR.

IBF (**Input buffer full):** This signal is an acknowledgement by the 8255 to indicate that the input latch has received the data byte. This is reset when the microprocessor reads the data. **INTR (Interrupt Request):** This is an output signal that may be used to interrupt the microprocessor. This signal is generated if , IBF and INTE are all at logic 1.

INTE (Interrupt Enable): This is an internal flip-flop to a port and needs to be set to generate the INTR signal. The two flip-flops INTEA and INTEB are set /reset using the BSR mode. The INTEA is enabled or disabled through PC4, and INTEB is enabled or disabled through PC2.



(Output Buffer Full): This is an output signal that goes low when the microprocessor writes data into the output latch of the 8255. This signal indicates to an output peripheral that new data is ready to be read. It goes high again after the 8255 receives a signal from the peripheral.
(**Acknowledge):** This is an input signal from a peripheral that must output a low when the peripheral receives the data from the 8255 ports.

INTR (Interrupt Request): This is an output signal, and it is set by the rising edge of the signal. This signal can be used to interrupt the microprocessor to request the next data byte for output. The INTR is set and INTE are all one and reset by the rising edge of . .
INTE (Interrupt Enable): This is an internal flip-flop to a port and needs to be set to generate the INTR signal. The two flip-flops INTEA and INTEB are set /reset using the BSR mode. The INTEA signal can be enabled or disabled through PC6, and INTEB is enabled or disabled through PC2.

Mode 2: Bidirectional Data Transfer
OBF This mode is used primarily in applications such as data transfer between the two computers or floppy disk controller interface. Port A can be configured as the bidirectional port and Port B either in mode 0 or mode 1. Port A uses five signals from Port C as handshake signals for data transfer. The remaining three lines from Port C can be used
either as simple I/O or as handshake signals for Port B.

Serial Communication: Using 8251

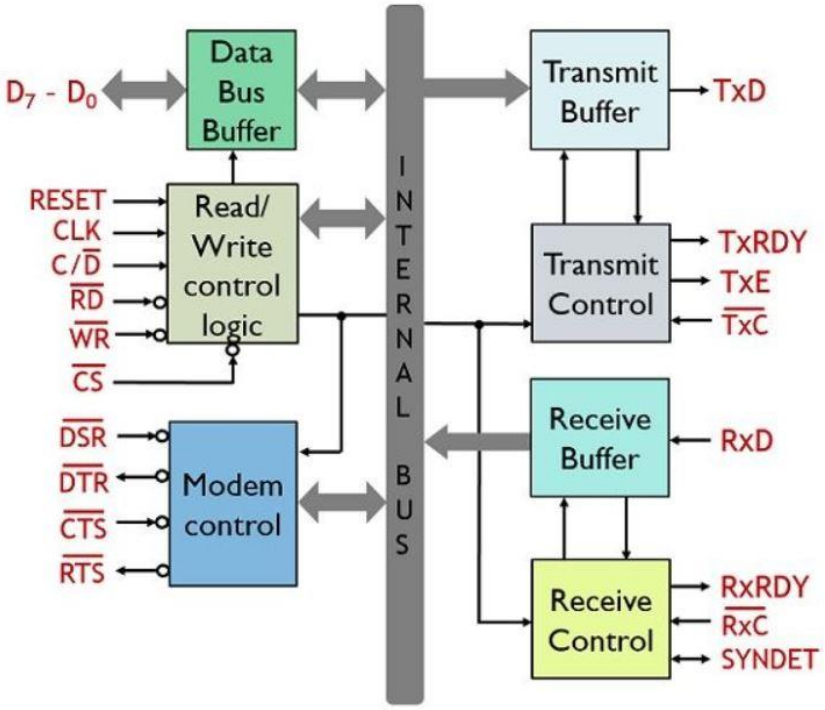
8251 is a Universal Synchronous and Asynchronous Receiver and Transmitter
compatible with Intel‘s processors. This chip converts the parallel data into a serial stream of bits suitable for serial transmission. It is also able to receive a serial stream of bits and convert it into parallel data bytes to be read by a microprocessor.
Basic Modes of data transmission
a) Simplex
b) Duplex
c) Half Duplex
a) Simplex mode
Data is transmitted only in one direction over a single communication channel. For example, the processor may transmit data for a CRT display unit in this mode.

b) Duplex Mode

In duplex mode, data may be transferred between two transceivers in both directions simultaneously.

c) Half Duplex mode

In this mode, data transmission may take place in either direction, but at a time data may be transmitted only in one direction. A computer may communicate with a terminal in this mode. It is not possible to transmit data from the computer to the terminal and terminal to computer simultaneously.



Serial communication interface 8251

The data buffer interfaces the internal bus of the circuit with the system bus. The read / write control logic controls the operation of the peripheral depending upon the operations initiated by the CPU decides whether the address on internal data bus is control address / data address. The modem control unit handles the modem handshake signals to coordinate the communication between modem and USART.

The transmit control unit transmits the data byte received by the data buffer from the CPU for serial communication. The transmission rate is controlled by the input frequency. Transmit control unit also derives two transmitter status signals namely TXRDY and TXEMPTY which may be used by the CPU for handshaking.

The transmit buffer is a parallel to serial converter that receives a parallel byte for conversion into a serial signal for further transmission. The receive control unit decides the receiver frequency as controlled by the RXC input frequency. The receive control unit generates a receiver ready (RXRDY) signal that may be used by the CPU for handshaking. This unit also detects a break in the data string while the 8251 is in asynchronous mode. In synchronous mode, the 8251 detects SYNC characters using SYNDET/BD pin.

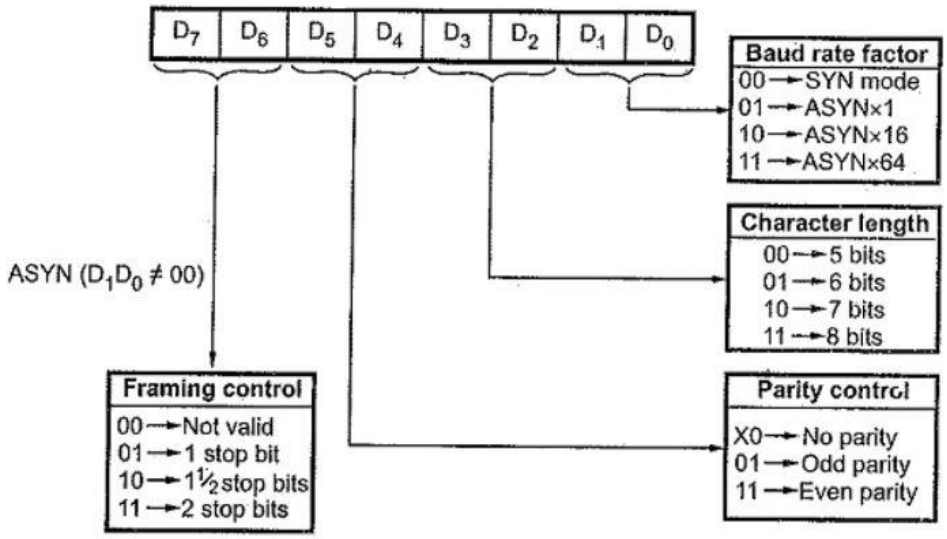
Signal Description of 8251

D_0 – D_7: This is an 8-bit data bus used to read or write status, command word or data from or to the 8251A.
C / D: (Control Word/Data): This input pin, together with RD and WR inputs, informs the 8251A that the word on the data bus is either a data or control word/status information. If this pin is 1, control / status is on the bus, otherwise data is on the bus.
RD: This active-low input to 8251A is used to inform it that the CPU is reading either data or status information from its internal registers. This active-low input to 8251A is used to inform it that the CPU is writing data or control word to 8251A
WR: This is an active-low chip select input of 825lA. If it is high, no read or write operation can be carried out on 8251. The data bus is tristated if this pin is high.
CLK: This input is used to generate internal device timings and is normally connected to clock generator output. This input frequency should be at least 30 times greater than the receiver or transmitter data bit transfer rate.
RESET: A high on this input forces the 8251A into an idle state. The device will remain idle till this input signal again goes low and a new set of control word is written into it. The minimum required reset pulse width is 6 clock states, for the proper reset operation.
TXC (Transmitter Clock Input): This transmitter clock input controls the rate at which the character is to be transmitted. The serial data is shifted out on the successive negative edge of the TXC.
TXD (Transmitted Data Output): This output pin carries serial stream of the transmitted data bits along with other information like start bit, stop bits and parity bit, etc.
RXC **(Receiver Clock Input):** This receiver clock input pin controls the rate at which the character is to be received.
RXD (Receive Data Input): This input pin of 8251A receives a composite stream of the data to be received by 8251 A.
RXRDY (Receiver Ready Output): This output indicates that the 8251A contains a character to be read by the CPU.
TXRDY - Transmitter Ready: This output signal indicates to the CPU that the internal circuit of the transmitter is ready to accept a new character for transmission from the CPU. **DSR - Data Set Ready:** This is normally used to check if data set is ready when communicating with a modem.
DTR - Data Terminal Ready: This is used to indicate that the device is ready to accept data when the 8251 is communicating with a modem.
RTS - Request to Send Data: This signal is used to communicate with a modem.
TXE- Transmitter Empty: The TXE signal can be used to indicate the end of a transmission mode.

Operating Modes of 8251
1. Asynchronous mode
2. Synchronous mode

Asynchronous Mode (Transmission)
When a data character is sent to 8251A by the CPU, it adds start bits prior to the serial data bits, followed by optional parity bit and stop bits using the asynchronous mode instruction control word format. This sequence is then transmitted using TXD output pin on the falling edge of TXC.
Asynchronous Mode (Receive)
A falling edge on RXD input line marks a start bit. The receiver requires only one stop bit to mark end of the data bit string, regardless of the stop bit programmed at the transmitting end. The 8-bit character is then loaded into the into parallel I/O buffer of 8251.

RXRDY pin is raised high to indicate to the CPU that a character is ready for it. If the previous character has not been read by the CPU, the new character replaces it, and the overrun flag is set indicating that the previous character is lost.

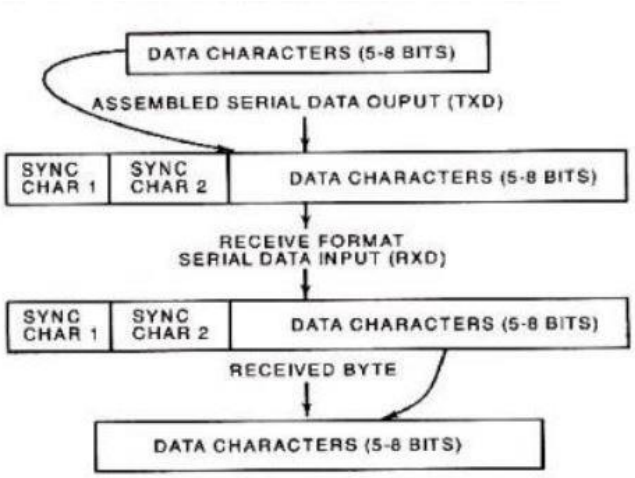


Synchronous Mode (Transmission)

The TXD output is high until the CPU sends a character to 8251 which usually is a SYNC character. When CTS line goes low, the first character is serially transmitted out. Characters are shifted out on the falling edge of TXC .Data is shifted out at the same rate as TXC **,** over TXD output line. If the CPU buffer becomes empty, the SYNC character or characters are inserted in the data stream over TXD output.

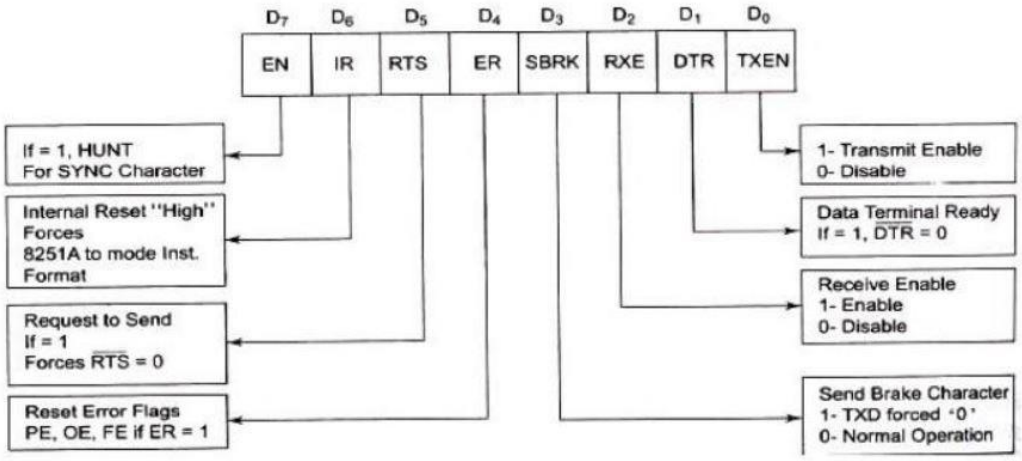
Synchronous Mode (Receiver)

In this mode, the character synchronization can be achieved internally or externally. The data on RXD pin is sampled on rising edge of the RXC. The content of the receiver buffer is compared with the first SYNC character at every edge until it matches. If 8251 is programmed for two SYNC characters, the subsequent received character is also checked. When the characters match, the hunting stops. The SYNDET pin set high and is reset automatically by a status read operation. In the external SYNC mode, the synchronization is achieved by applying a high level on the SYNDET input pin that forces 8251 out of HUNT mode. The high level can be removed after one RXC cycle. The parity and overrun error both are checked in the same way as in asynchronous mode.



Command Instruction Definition

The command instruction controls the actual operations of the selected format like enable transmit/receive, error reset and modem control. A reset operation returns 8251 back to mode instruction format.



D/A And A/D Interface:

The function of an A/D converter is to produce a digital word which represents the magnitude of some analog voltage or current.
The resolution of an A/D converter refers to the number of bits in the output binary word. An 8-bit converter for example has a resolution of 1 part in 256. Accuracy and linearity specifications have the same meaning for an A/D converter as they do for a D/A converter. Another important specification for an ADC is its conversion time. - the time it takes the converter to produce a valid output binary code for an applied input voltage. When we refer to a converter as high speed, it has a short conversion time.

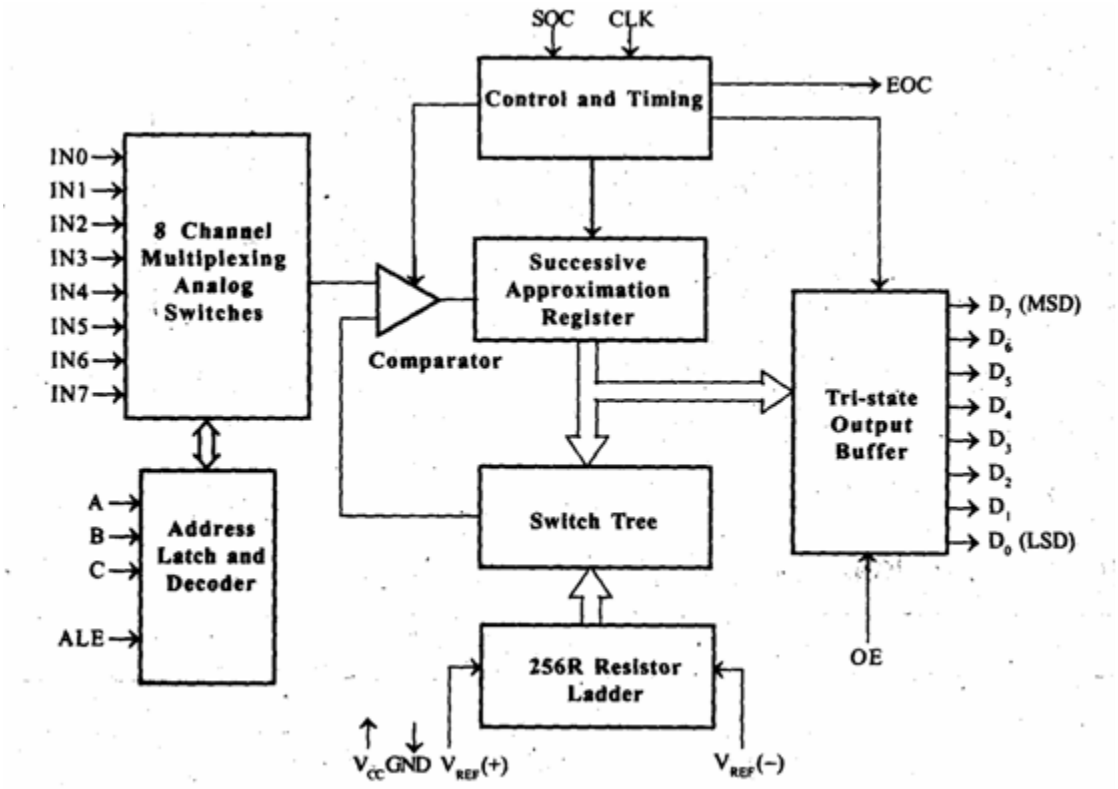
The analog to digital converter is treated as an input device by the microprocessor that sends an initialising signal to the ADC to start the analog to digital data conversation process. The start of conversion signal is a pulse of a specific duration. The process of analog to digital conversion is a slow process, and the microprocessor has to wait for the digital data till the conversion is over.

After the conversion is over, the ADC sends end of conversion (EOC) signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC. These tasks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports. The time taken by the ADC from the active edge of SOC pulse (the edge at which the conversion process actually starts) till the active edge of EOC signal is called as the conversion delay of the ADC- the time taken by the converter to calculate the equivalent digital data output from the instant of the start of conversion is called conversion delay. It may range anywhere from a few microseconds in case of fast ADCs to even a few hundred milliseconds in case of slow ADCs. A number of ADCs are available in the market, the selection of ADC for a particular application is done, keeping in mind the required speed, resolution range of operation, power supply requirements, sample and hold device requirements and the cost factors are considered.
The available ADCs in the market use different conversion techniques for the conversion of analog signals to digital signals. Parallel converter or flash converter, Successive approximation and

dual slope integration
A general algorithm for ADC interfacing contains the following steps.
1. Ensure the stability of analog input, applied to the ADC.
2. Issue start of conversion (SOC) pulse to ADC.
3. Read end of conversion (EOC) signal to mark the end of conversion process.
4. Read digital data output of the ADC as equivalent digital output.
It may be noted that analog input voltage must be constant at the input of the ADC right from the start of conversion till the end of conversion to get correct results. This may be ensured by a sample and hold circuit which samples the analog signal and holds it constant for a specified time duration. The microprocessor may issue a hold signal to the sample and Hold circuit. If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct. If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct.

ADC 0808/0809

The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, *successive approximation converters.* Successive approximation technique is one of the fast techniques for analog to digital conversion. The conversion delay is 100 μs at a clock frequency of 640 kHz, which is quite low as compared to other converters. These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits. These converters internally have a 3:8 analog multiplexer so that at a time eight different analog inputs can be connected to the chips. Out of these eight inputs only one can be selected for conversion by using address lines ADD A, ADD B and ADD C, as shown. Using these address inputs, multichannel data acquisition systems can be designed using a single ADC. The CPU may drive these lines using output port lines in case of multichannel applications. In case of single input applications, these may be hard wired to select the proper input.



INTERFACING DIGITAL TO ANALOG ONVERTERS:

The digital to analog converters convert binary numbers into their analog equivalent voltages or currents. Several techniques are employed for digital to analog conversion.

Weighted resistor network

R-2R ladder network

Current output D/A converter

Applications in areas like digitally controlled gains, motor speed control, programmable gain amplifiers, digital voltmeters, panel meters, etc. In a compact disk audio player for example a 14-or16-bit D/A converter is used to convert the binary data read off the disk by a laser to an analog audio signal.Most speech synthesizer integrated circuits contain a D/A converter to convert stored binary data words into analog audio signals.

Characteristics:

1. Resolution: It is a change in analog output for one LSB change in digital input.

It is given by(1/2n)*Vref.

If n=8 (i.e.8-bit DAC) 1/256*5V=39.06mV

2. Settling time: It is the time required for the DAC to settle for a full scale code change.

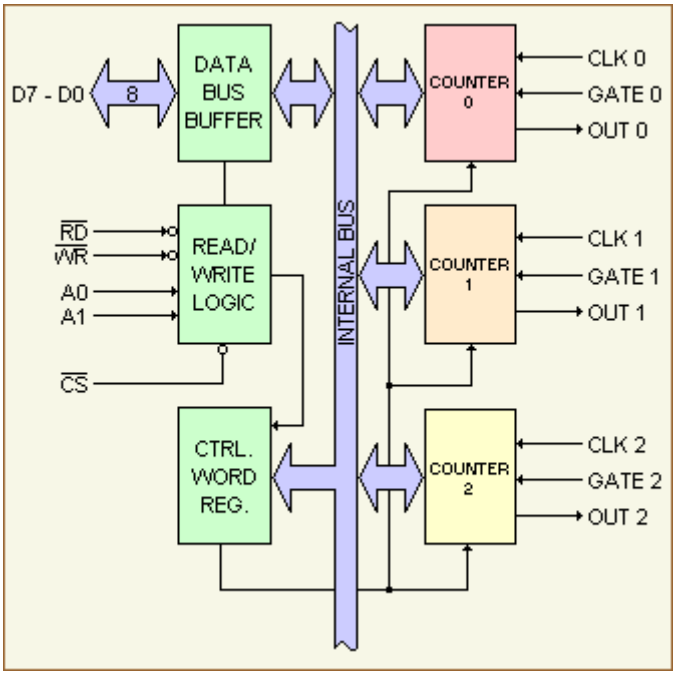
Programmable timer device 8253

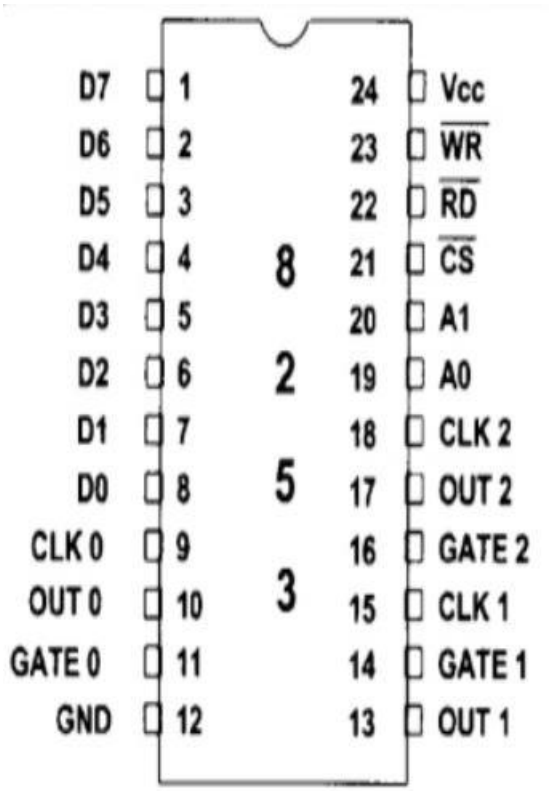
Intel‘s programmable counter/timer device (8253) facilitates the generation of accurate time delays. When 8253 is used as timing and delay generation peripheral, the microprocessor becomes free from the tasks related to the counting process and execute the programs in memory, while the timer device may perform the counting tasks. This minimizes the software overhead on the microprocessor.

Architecture and Signal Descriptions

The programmable timer device 8253 contains three independent 16-bit counters, each with a maximum count rate of 2.6 MHz to generate three totally independent delays or maintain three independent counters simultaneously. All the three counters may be independently controlled by programming the three internal command word registers.

The 8-bit, bidirectional data buffer interfaces internal circuit of 8253 to microprocessor systems bus. Data is transmitted or received by the buffer upon the execution of IN or OUT instruction. The read/write logic controls the direction of the data buffer depending upon whether it is a read or a write operation. It may be noted that IN instruction reads data while OUT instruction writes data to a peripheral





The three counters all 16-bit presettable, down counters, able to operate either in BCD or in hexadecimal mode. The mode control word register contains the information that can be used for writing or reading the count value into or from the respective count register using the OUT and IN instructions. The specialty of the 8253 counters is that they can be easily read on line without disturbing the clock input to the counter. This facility is called as "on the fly" reading of counters, and is invoked using a mode control word.

CS	RD	WR	A_1	A_0	Selected Operations
0	1	0	0	0	Write Counter 0
0	1	0	0	1	Write Counter 1
0	1	0	1	0	Write Counter 2
0	1	0	1	1	Write control Word
0	0	1	0	0	Read Counter 0
0	0	1	0	1	Read Counter 1
0	0	1	1	0	Read Counter 2
0	0	1	1	1	No Operation
0	1	1	X	X	No Operation
1	X	X	X	X	Disabled

A0, Al pins are the address input pins and are required internally for addressing the mode control word registers and the three counter registers. A low on CS line enables the 8253. No operation will be performed by 8253 till it is enabled.

A control word register accepts the 8-bit control word written by the microprocessor and stores it for controlling the complete operation of the specific counter. It may be noted that, the control word register can only be written and cannot be read as it is obvious from Table
The CLK, GATE and OUT pins are available for each of the three timer channels. Their functions will be clear when we study the different operating modes of 8253.

Control Word Register
The 8253 can operate in anyone of the six different modes. A control word must be written in the respective control word register by the microprocessor to initialize each of the counters of 8253 to decide its operating mode. All the counters can operate in anyone of the modes or they may be even in different modes of operation, at a time.
The control word format is presented, along with the definition of each bit, while writing a count in the counter, it should be noted that, the count is written in the counter only after the data is put on the data bus and a falling edge appears at the clock pin of the peripheral thereafter. Any reading operation of the counter, before the falling edge appears may result in garbage data.

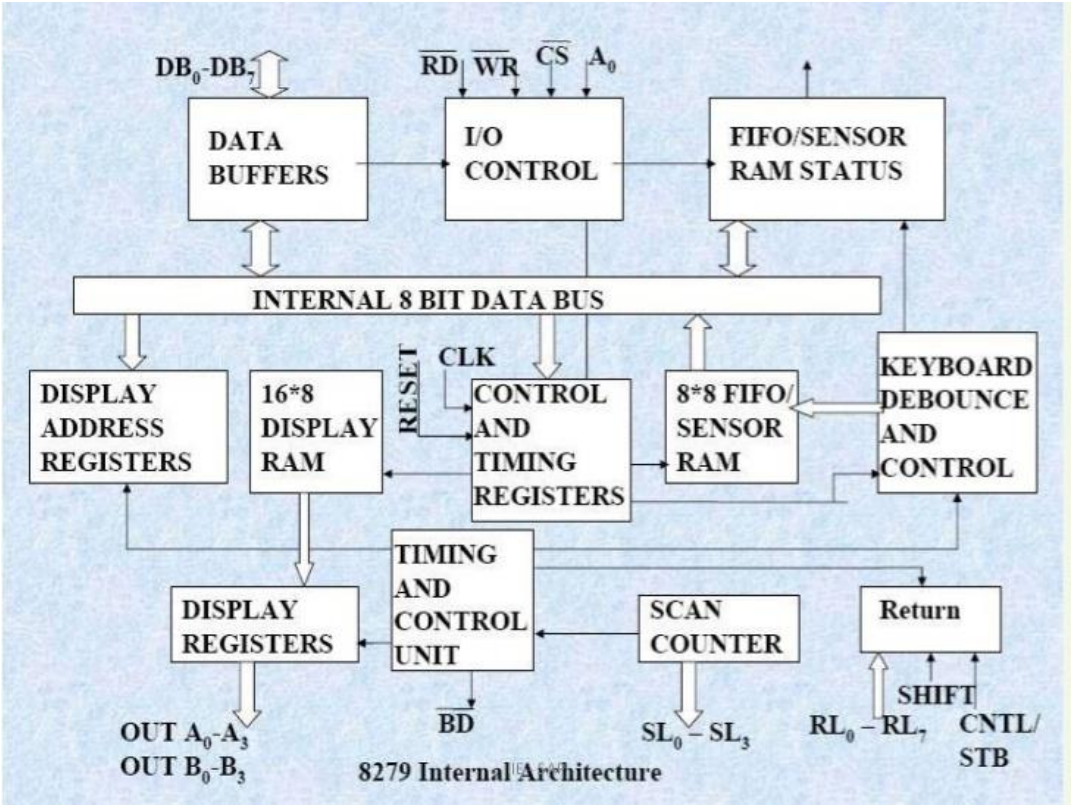
8279 Programmable Keyboard/Display Controller

Intel‘s 8279 is a general purpose Keyboard Display controller that simultaneously drives the display of a system and interfaces a Keyboard with the CPU. The Keyboard Display interface scans the Keyboard to identify if any key has been pressed and sends the code of the pressed key to the CPU. It also transmits the data received from the CPU, to the display device. Both of these functions are performed by the controller in repetitive fashion without involving the CPU. The Keyboard is interfaced either in the interrupt or the polled mode. In the interrupt mode, the processor is requested service only if any key is pressed, otherwise the CPU can proceed with its main task.

In the polled mode, the CPU periodically reads an internal flag of 8279 to check for a key pressure. The Keyboard section can interface an array of a maximum of 64 keys with the CPU. The Keyboard entries (key codes) are debounced and stored in an 8-byte FIFO RAM, that is further accessed by the CPU to read the key codes. If more than eight characters are entered in the FIFO (i.e. more that eight keys are pressed), before any FIFO read operation, the overrun status is set. If a FIFO contains a valid key entry, the CPU is interrupted (in interrupt mode) or the CPU checks the status (in polling) to read the entry. Once the CPU reads a key entry, the FIFO is updated, i.e. the key entry is pushed out of the FIFO to generate space for new entries. The 8279 normally provides a maximum of sixteen 7-seg display interface with CPU It contains a 16-byte display RAM that can be used either as an integrated block of 16x8-bits or two 16x4-bit block of RAM. The data entry to RAM block is controlled by CPU using the command words of the 8279.

Architecture and Signal Descriptions of 8279
The Keyboard display controller chip 8279 provides
1. A set of four scan lines and eight return lines for interfacing keyboards.
2. A set of eight output lines for interfacing display.

I/O Control and Data Buffer
The I/O control section controls the flow of data to/from the 8279. The data buffer interface the external bus of the system with internal bus of 8279 the I/O section is enabled only if D The pin A**o**, RD and WR select the command, status or data read/write operations carried out by the CPU with 8279.



Control and Timing Register and Timing Control

These registers store the keyboard and display modes and other operating conditions programmed by CPU. The registers are written with A**o**=1 and WR =0. The timing and control unit controls the basic timings for the operation of the circuit. Scan Counter divide down the operating frequency of 8279 to derive scan keyboard and scan display frequencies.

Scan Counter

The Scan Counter has two modes to scan the key matrix and refresh the display. In the Encoded mode, the counter provides a binary count that is to be externally decoded to provide the scan lines for keyboard and display (four externally decoded scan lines may drive up to 16 displays). In the decoded scan mode, the counter internally decodes the least significant 2 bits and provides a decoded 1 out of 4 scan on SL**0**-SL**3** (four internally
decoded scan lines may drive up to 4 Displays). The Keyboard and Display both are in the same mode at a time.

Return Buffers and Keyboard Debounce and Control

This section scans for a Key closure row-wise. If it is detected, the Keyboard debounce unit debounces the key entry (i.e. wait for 10 ms). After the debounce period, if the key continues to be detected. The code of the Key is directly transferred to the sensor RAM along with SHIFT and CONTROL key status.

FIFO/Sensor RAM and Status Logic

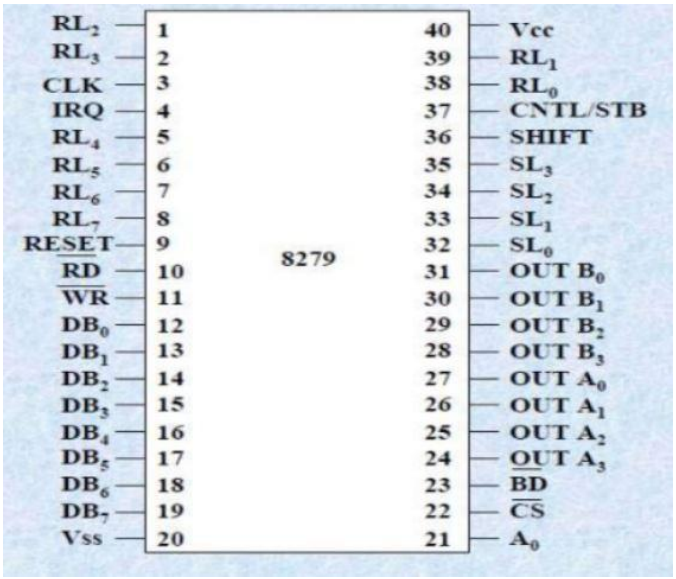
In Keyboard or strobed input mode, this block acts as 8-byte first-in-first-out (FIFO) RAM. Each key code of the pressed key is entered in the order of the entry, and in the meantime, read by the CPU, till the RAM becomes empty. The status logic generates an interrupt request after each FIFO read operation till the FIFO is empty.

In scanned sensor matrix mode, this unit acts as sensor RAM. Each row of the sensor RAM is loaded with the status of the corresponding row of sensors in the matrix. If a sensor changes its state, the IRQ line goes high to interrupt the CPU.

Display Address Registers and Display RAM.

The Display address registers hold the addresses of the word currently being written or read by the CPU to or from the display RAM. The contents of the registers are automatically updated by 8279 to accept the next data entry by CPU. The 16-byte display RAM contains the 16-byte of data to be displayed on the sixteen 7-seg displays in the encoded scan mode

Pin Diagram of 8279



DB0 - DB7:

These are bidirectional data bus lines. The data and command words to and from the CPU are transferred on these lines.

CLK:

This is a clock input used to generate internal timings required by 8279.

RESET:

This pin is used to reset 8279. A high on this line resets 8279. After resetting 8279, its in sixteen 8-bit display, left entry encoded scan, 2-key lock out mode. The clock prescaler is set to 31.

CS chip select:

A low on this line enables 8279 for normal read or write operations. Otherwise this pin should be high.

Ao:

A high on the A**o** line indicates the transfer of a command or status information. A low on this line indicates the transfer of data. This is used to select one of the internal registers of 8279.

RD, WR:

(Input/Output) READ/WRITE input pins enable the data buffer to receive or send data over the data bus.

IRQ:

This interrupt output line goes high when there is data in the FIFO sensor RAM. The interrupt line goes low with each FIFO RAM read operation. However, if the FIFO RAM further contains any Key-code entry to be read by the CPU, this pin again goes high to generate an interrupt to the CPU.

Vss, Vcc:
These are the ground and power supply lines for the circuit.
SL0-SL3 – Scan Lines:
These lines are used to scan the keyboard matrix and display digits. These lines can be programmed as encoded or decoded, using the mode control register.
RL0-RL7 – Return Lines:
These are the input lines which are connected to one terminal of keys, while the other terminal of the keys are connected to the decoded scan lines. These are normally high, but pulled low when a key is pressed.
SHIFT:
The status of the Shift input line is stored along with each key code in FIFO in the scanned keyboard mode. Till it is pulled low with a key closure it is pulled up internally to keep it high.
CNTL/STB-CONTROL/STROBED I/P Mode:
In the Keyboard mode, this line is used as a control input and stored in FIFO on a key closure. The line is a strobe line that enters the data into FIFO RAM, in the strobed input mode. It has an internal pull up. The line is pulled down with a Key closure.
BD – Blank Display:
This output pin is used to blank the display during digit switching or by a blanking command.
OUTA0 – OUTA3 and OUTB0 – OUTB3:
These are the output ports for two 16x4 (or one 16 x 8) internal display refresh registers. The data from these lines is synchronized with the scan lines to scan the display and keyboard. The two 4-bit ports may also be used as one 8-bit port.
Modes of Operation of 8279
The Modes of operation of 8279 are
i. Input (Keyboard) modes ii. Output (Display) modes **Input (Keyboard) modes:**
8279 provides three input modes, they are:
1. Scanned Keyboard Mode:
This mode allows a key matrix to be interfaced using either encoded or decoded scans. In the encoded scan, an 8 x 8 keyboard or in decoded scan , a 4 x 8 Keyboard can be interfaced. The code of key pressed with SHIFT and CONTROL status is stored into the FIFO RAM.
2. Scanned Sensor Matrix:
In this mode, a sensor array can be interfaced with 8279 using either encoder or decoder scans. With encoder scan 8 x 8 sensor matrix or with decoder scan 4 x 8 sensor matrix can be interfaced. The sensor codes are stored in the CPU addressable sensor RAM.
3. Strobed Input: In this mode, if the control line goes low, the data on return lines, is stored in the FIFO byte by byte.
Output (Display) Modes:
8279 provides two output modes for selecting the display options.
1. Display Scan:
In this mode, 8279 provides 8 or 16 character multiplexed displays those can be organized as dual 4-bit or single 8-bit display units.
2. Display Entry:
The Display data is entered for display either from the right side or from the left side.

Details of Modes of Operation

Keyboard Modes

1. Scanned Keyboard Mode with 2 Key Lockout

In this mode of operation, when a key is pressed, a debounce logic comes into operation. The Key code of the identified key is entered into the FIFO with SHIFT and CNTL status, provided the FIFO is not full.

2. Scanned Keyboard with N-key Rollover

In this mode, each key depression is treated independently. When a key is pressed, the debounce circuit waits for 2 keyboard scans and then checks whether the key is still depressed. If it is still depressed, the code is entered in FIFO RAM. Any number of keys can be pressed simultaneously and recognized in the order, the Keyboard scan record them.

3. Scanned Keyboard Special Error Mode

This mode is valid only under the N-Key rollover mode. This mode is programmed using *end interrupt/error mode set* command. If during a single debounce period (two Keyboard scan) two keys are found pressed, this is considered a simultaneous depression and an error
flag is set. This flag, if set, prevents further writing in FIFO but allows generation of further interrupts to the CPU for FIFO read.

3. Sensor Matrix Mode

In the Sensor Matrix mode, the debounce logic is inhibited the 8-byte memory matrix. The status of the sensor switch matrix is fed directly to sensor RAM matrix Thus the sensor RAM bits contains the row-wise and column-wise status of the sensors in the sensor matrix.

Display Modes

There are various options of data display The first one is known as left entry mode or type writer mode. Since in a type writer the first character typed appears at the left-most position, while the subsequent characters appears successively to the right of the first one. The other display format is known as right entry mode, or calculator mode, since the calculator the first character entered appears to the right-most position and this character is shifted one position left when the next character is entered.

1. Left Entry Mode

In the Left entry mode, the data is entered from the left side of the display unit. Address 0 of the display RAM contains the leftmost display character and address 15 of the RAM contains the rightmost display character.

2. Right Entry Mode

In the right entry mode, the first entry to be displayed is entered on the rightmost display. The next entry is also placed in the right most display but after the previous display is shifted left by one display position.

Command Words of 8279

All the Command words or status words are written or read with Ao = 1 and CS = 0 to or from 8279.

a. Keyboard Display mode set

The format of the command word to select different modes of operation of 8279 is given below with its bit definitions.

b. Programmable Clock

The clock for operation of 8279 is obtained by dividing the external clock input signal by a programmable constant called prescaler.

PPPPP is a 5-bit binary constant. The input frequency is divided by a decimal constant ranging from 2 to 31, decided by the bits of an internal prescalar, PPPPP.

c. Read FIFO/Sensor RAM

The format of this command is given as shown below X - don‘t care
AI - Auto increment flag
AAA - Address pointer to 8 bit FIFO RAM
This word is written to set up 8279 for reading FIFO/Sensor RAM. In scanned keyboard mode, AI and AAA bits are of no use. The 8279 will automatically drive data bus for each subsequent read, in the same sequence, in which the data was entered.

d. Read Display RAM

This command enables a programmer to read the display RAM data The CPU writes this command word to 8279 to prepare it for display RAM read operation. AI is auto incremented flag and AAAA, the 4-bit address, points to the 16-byte display RAM that is to be read. If AI = 1, the address will be automatically, incremented after each read or write to the display RAM.

e. Write Display RAM

The format of this command is given as shown below
AI - Auto increment flag
AAAA - 4-bit address for 16-bit display RAM to be written
Other details of this command are similar to the _Read Display RAM
Command.

f. Display Write Inhibit/Blanking

The IW (Inhibit write flag) bits are used to mask the individual nibble Here Do and D2 corresponds to OUTBo – OUTB3 while D**1** and D**3** corresponds to OUTAo-OUTA3 for blanking and masking respectively.

g. Clear Display RAM

The CD**2**, CD**1**, CDo is a selectable blanking code to clear all the rows of the display RAM as given below. The characters A and B represents the output nibbles. CD CD**1** CDo
1 0 x All Zeros (x don‘t care) AB = 00
1 1 0 A**3**-Ao = 2(0010) and B**3**-Bo = 00(0000) 1
1 1 All ones (AB = FF), i.e. clear RAM
Here, CA represents clear All and CF represents Clear FIFO RAM

End Interrupt/Error Mode Set

For the sensor matrix mode, this command lowers the IRQ line and enables further writing into the RAM. Otherwise, if a charge in sensor value is detected, IRQ goes high that inhibits writing in the sensor RAM.

Key-code and status Data Formats

This briefly describes the formats of the Key-code/Sensor data in their respective modes of operation and the FIFO Status Word formats of 8279.

Key-code Data Formats:

a. After a valid Key closure, the key code is entered as a byte code into the FIFO RAM, in the following format, in scanned keyboard mode. The Keycode format contains 3-bit contents of the internal row counter, 3-bit contents of the column counter and status of the SHIFT
b. and CNTL Keys The data format of the Keycode in scanned keyboard mode is given below. In the sensor matrix mode, the data from the return lines is directly entered into an appropriate row of sensor RAM, that identifies the row of the sensor that changes its status. The SHIFT and CNTL Keys are ignored in this mode. RL bits represent the return lines.
c. Rn represents the sensor RAM row number that is equal to the row number of the sensor array in which the status change was detected. Data Format of the sensor code in sensor matrix mode

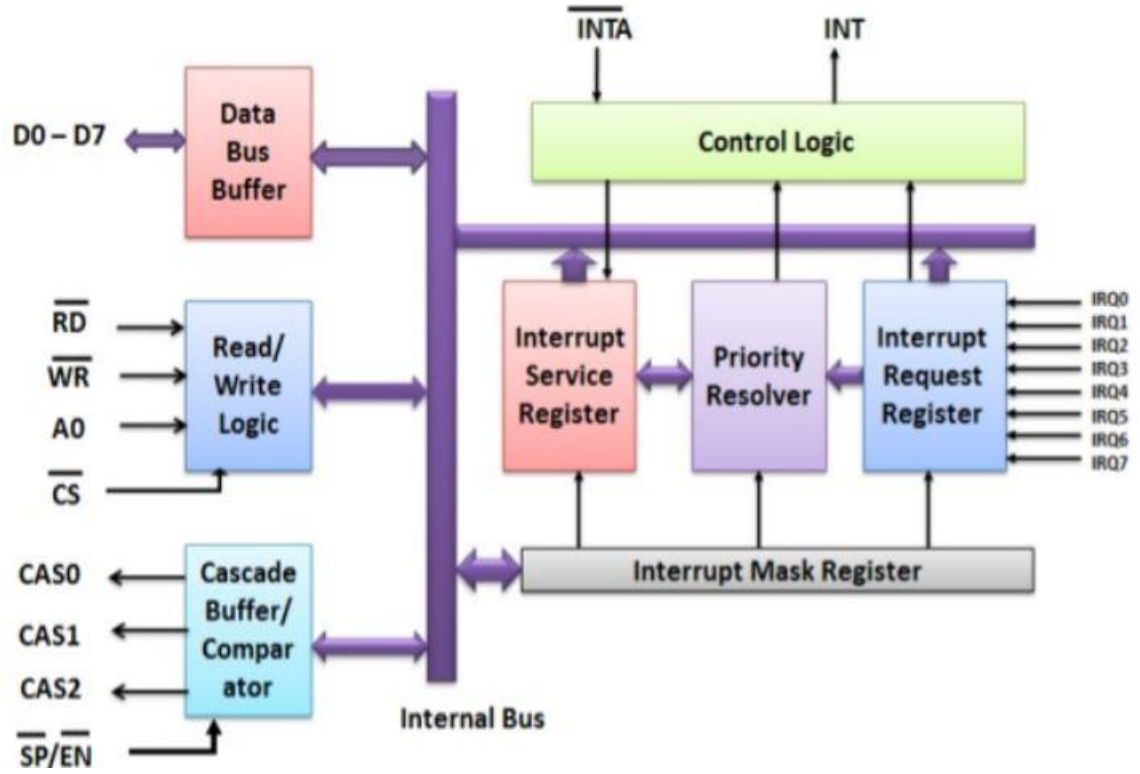
d. **FIFO Status Word:** The FIFO status word is used in keyboard and strobed input mode to indicate the error. Overrun error occurs, when an already full FIFO is attempted an entry, Under run error occurs when an empty FIFO read is attempted. FIFO status word also has a bit to show the unavailability of FIFO RAM because of the ongoing clearing operation.

e. In sensor matrix mode, a bit is reserved to show that at least one sensor closure indication is stored in the RAM, The S/E bit shows the simultaneous multiple closure error in special error mode. In sensor matrix mode, a bit is reserved to show that at least one sensor closure indication is stored in the RAM, The S/E bit shows the simultaneous multiple closure error in special error mode.

Interrupt Controller

The Intel 8259A Programmable Interrupt Controller handles up to eight vectored priority interrupts for the CPU. It is cascadable for up to 64 vectored priority interrupts without additional circuitry. It is packaged in a 28-pin DIP, uses NMOS technology and requires a single a5V supply. Circuitry is static, requiring no clock input. The 8259A is designed to minimize the software and real time overhead in handling multi- level priority interrupts.

It has several modes, permitting optimization for a variety of system requirements. The 8259A is fully upward compatible with the Intel 8259. Software originally written for the 8259 will operate the 8259A in all 8259 equivalent modes (MCS-80/85, Non-Buffered and Edge Triggered).



The microprocessor will be executing its main program and only stop to service peripheral devices when it is told to do so by the device itself. In effect, the method would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is complete, however, the processor would resume exactly where it left off.This method is called Interrupt.

System throughput would drastically increase, and thus more tasks could be assumed by the microcomputer to further enhance its cost effectiveness. The Programmable Interrupt Controller (PIC) functions as an overall manager in an Interrupt-Driven system environment. It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), artains whether the incoming request has a higher priority value than the level currently being serviced, and issues an interrupt to the CPU based on this determination.
Each peripheral device or structure usually has a special program or ``routine'' that is associated with its specific functional or operational requirements; this is referred to as a
``service routine''. The PIC, after issuing an Interrupt to the CPU, must somehow input information into the CPU that can ``point'' the Program Counter to the service routine associated with the requesting device. This ``pointer'' is an address in a vectoring table and will often be referred to, in this document, as vectoring data.

Interrupt request register (IRR) AND in-service register (ISR):

The interrupts at the IR input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service (ISR). The IRR is used to store all the interrupt levels which are requesting service; and the ISR is used to store all the interrupt levels which are being serviced.

Priority resolver

This logic block determines the priorites of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during INTA pulse.

Interrupt mask register (IMR)

The IMR stores the bits which mask the interrupt lines to be masked. The IMR operates on the IRR.Masking of a higher priority input will not affect the interrupt request lines of lower quality.

INT (INTERRUPT)

This output goes directly to the CPU interrupt input. The VOH level on this line is designed to be fully compatible with the 8080A, 8085A and 8086 input levels.

INTA (INTERRUPT ACKNOWLEDGE)

INTA pulses will cause the 8259A to release vectoring information onto the data bus. The format of this data depends on the system mode (mPM) of the 8259A

Data bus buffer

This 3-state, bidirectional 8-bit buffer is used to interface the 8259A to the system Data Bus. Control words and status information are transferred through the Data Bus Buffer.

Read/write control logic

The function of this block is to accept OUTput commands from the CPU. It contains the Initialization Command Word (ICW) registers and Operation Command Word (OCW) registers
which store the various control formats for device operation. This function block also allows the status of the 8259A to be transferred onto the Data Bus.

CS (CHIP SELECT)

A LOW on this input enables the 8259A. No reading or writing of the chip will occur unless the device is selected.

WR (WRITE)
A LOW on this input enables the CPU to write control words (ICWs and OCWs) to the 8259A.

RD (READ)
A LOW on this input enables the 8259A to send the status of the Interrupt Request Register (IRR),In Service Register (ISR), the Interrupt Mask Register (IMR), or the Interrupt level onto the Data Bus.

A0
This input signal is used in conjunction with WR and RD signals to write commands into the various command registers, as well as reading the various status registers of the chip. This line can be directly to one of the address lines.

DMA Controller -DMA Controller 8257

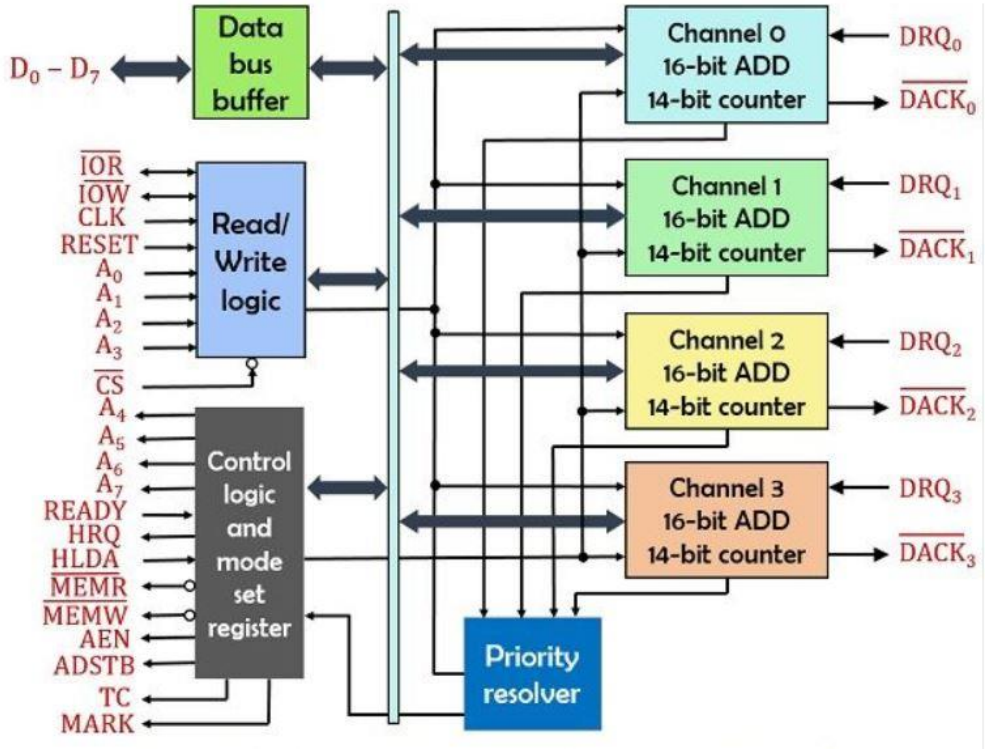
The *Direct Memory Access* or DMA mode of data transfer is the fastest amongst all the modes of data transfer. In this mode, the device may transfer data directly to/from memory without any interference from the CPU. The device requests the CPU (through a DMA controller) to hold its data, address and control bus, so that the device may transfer data directly to/from memory.
The DMA data transfer is initiated only after receiving HLDA signal from the CPU.
Intel's 8257 is a four channel DMA controller designed to be interfaced with their family of microprocessors. The 8257, on behalf of the devices, requests the CPU for bus access using local bus request input i.e. HOLD in minimum mode.

In maximum mode of the microprocessor RQ/GT pin is used as bus request input. On receiving the HLDA signal (in minimum mode) or RQ/GT signal (in maximum mode) from the CPU, the requesting devices gets the access of the bus, and it completes the required number of DMA cycles for the data transfer and then hands over the control of the bus back to the CPU.

Internal Architecture of 8257

The internal architecture of 8257 is shown in figure. The chip support four DMA channels, i.e. four peripheral devices can independently request for DMA data transfer through these channels at a time. The DMA controller has 8-bit internal data buffer, a read/write unit, a control unit, a priority resolving unit along with a set of registers. The 8257 performs the DMA operation over four independent DMA channels. Each of four channels of 8257 has a pair of two 16-bit registers, viz. DMA *address register* and *terminal count register*.

There are two common registers for all the channels, namely, *mode set register* and *status register*. Thus there are a total of ten registers. The CPU selects one of these ten registers using address lines Ao-A3. Table shows how the Ao-A3 bits may be used for selecting one of these registers.



DMA Address Register

Each DMA channel has one DMA address register. The function of this register is to store the address of the starting memory location, which will be accessed by the DMA channel. Thus the starting address of the memory block which will be accessed by the device is first loaded in the DMA address register of the channel. The device that wants to transfer data over a DMA channel, will access the block of the memory with the starting address stored in the DMA Address Register.

Terminal Count Register

Each of the four DMA channels of 8257 has one terminal count register (TC). This 16-bit register issued for artaining that the data transfer through a DMA channel ceases or stops after the required number of DMA cycles. The low order 14-bits of the terminal count register are initialized with the binary equivalent of the number of required DMA cycles minus one.

After each DMA cycle, the terminal count register content will be decremented by one and finally it becomes zero after the required number of DMA cycles are over. The bits

14 and 15 of this register indicate the type of the DMA operation (transfer). If the device wants to write data into the memory, the DMA operation is called DMA write operation. Bit

14 of the register in this case will be set to one and bit 15 will be set to zero.

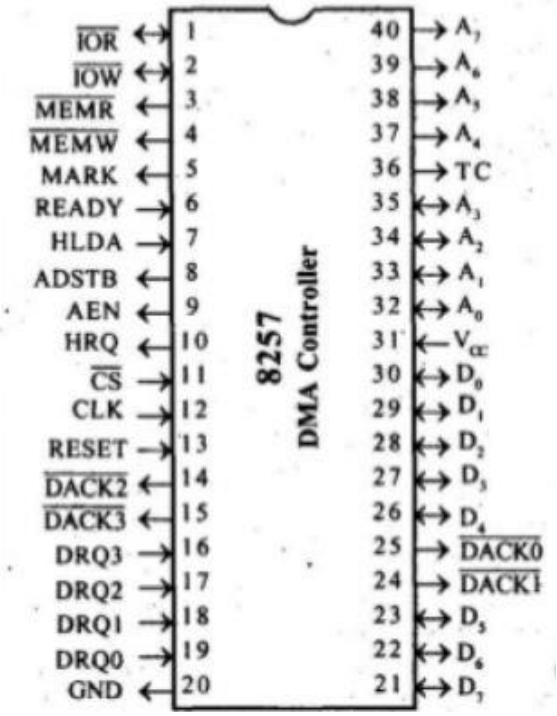
Data Bus Buffer, Read/Write Logic, Control Unit and Priority Resolver

The 8-bit. Tristate, bidirectional buffer interfaces the internal bus of 8257 with the external system bus under the control of various control signals.

In the slave mode, the read/write logic accepts the I/O Read or I/O Write signals, decodes the Ao-A3 lines and either writes the contents of the data bus to the addressed internal register or reads the contents of the selected register depending upon whether IOW or IOR signal is activated.

In master mode, the read/write logic generates the IOR and IOW signals to control the data flow to or from the selected peripheral. The control logic controls the sequences of operations and generates the required control signals like AEN, ADSTB, MEMR, MEMW, TC and MARK along with the address

lines A4-A7, in master mode. The priority resolver resolves the priority of the four DMA channels depending upon whether normal priority or rotating priority is programmed.



DRQo-DRQ3:
These are the four individual channel DMA request inputs, used by the peripheral devices for requesting the DMA services. The DRQo has the highest priority while DRQ**3** has the lowest one, if the fixed priority mode is selected.

DACKo-DACK3:
These are the active-low DMA acknowledge output lines which inform the requesting peripheral that the request has been honoured and the bus is relinquished by the CPU. These lines may act as strobe lines for the requesting devices.

Do-D7:
These are bidirectional, data lines used to interface the system bus with the internal data bus of 8257. These lines carry command words to 8257 and status word from 8257, in slave mode, i.e. under the control of CPU. The data over these lines may be transferred in both the directions. When the 8257 is the bus master (master mode, i.e. not under CPU control), it uses Do-D7 lines to send higher byte of the generated address to the latch. This address is further latched using ADSTB signal. the address is transferred over Do-D7 during the first clock cycle of the DMA cycle. During the rest of the period, data is available on the data bus.

IOR:
This is an active-low bidirectional tristate input line that acts as an input in the slave mode. In slave mode, this input signal is used by the CPU to read internal registers of 8257.this line acts output in master mode. In master mode, this signal is used to read data from a peripheral during a memory write cycle.

IOW:
This is an active low bidirection tristate line that acts as input in slave mode to load the contents of the data bus to the 8-bit mode register or upper/lower byte of a 16-bit DMA address register or terminal count register. In the master mode, it is a control output that loads the data to a peripheral during DMA memory read cycle (write to peripheral).

CLK:
This is a clock frequency input required to derive basic system timings for the internal operation of 8257.
RESET:
This active-high asynchronous input disables all the DMA channels by clearing the mode register and tristates all the control lines.
Ao-A3:
These are the four least significant address lines. In slave mode, they act as input which select one of the registers to be read or written. In the master mode, they are the four least significant memory address output lines generated by 8257.
CS:
This is an active-low chip select line that enables the read/write operations from/to 8257, in slave mode. In the master mode, it is automatically disabled to prevent he chip from getting selected (by CPU) while performing the DMA operation.
A4-A7:
This is the higher nibble of the lower byte address generated by 8257 during the master mode of DMA operation.
READY:
This is an active-high asynchronous input used to stretch memory read and write cycles of 8257 by inserting wait states. This is used while interfacing slower peripherals.
HRQ:
The hold request output requests the access of the system bus. In the non-cascaded 8257 systems, this is connected with HOLD pin of CPU. In the cascade mode, this pin of a slave is connected with a DRQ input line of the master 8257, while that of the master is connected with HOLD input of the CPU.
HLDA:
The CPU drives this input to the DMA controller high, while granting the bus to the device. This pin is connected to the HLDA output of the CPU. This input, if high, indicates to the DMA controller that the bus has been granted to the requesting peripheral by the CPU.
MEMR: This active –low memory read output is used to read data from the addressed memory locations during DMA read cycles.
MEMW:
This active-low three state output is used to write data to the addressed memory location during DMA write operation.
ADST:
This output from 8257 strobes the higher byte of the memory address generated by the DMA controller into the latches.
AEN:
This output is used to disable the system data bus and the control the bus driven by the CPU, this may be used to disable the system address and data bus by using the enable input of the bus drivers to inhibit the non-DMA devices from responding during DMA operations. If the 8257 is I/O mapped, this should be used to disable the other I/O devices, when the DMA controller addresses is on the address bus.
TC:
Terminal count output indicates to the currently selected peripherals that the present DMA cycle is the last for the previously programmed data block. If the TC STOP bit in the mode set register is set, the selected channel will be disabled at the end of the DMA cycle. The TC pin is activated when the 14-bit content of the terminal count register of the selected channel becomes equal to zero. The lower

order 14 bits of the terminal count register are to be programmed with a 14-bit equivalent of (n-1), if n is the desired number of DMA cycles.

MARK:

The modulo 128 mark output indicates to the selected peripheral that the current DMA cycle is the 128_{th} cycle since the previous MARK output. The mark will be activated after each 128 cycles or integral multiples of it from the beginning if the data block (the first DMA cycle), if the total number of the required DMA cycles (n) is completely divisible by 128.

Vcc:

This is a +5v supply pin required for operation of the circuit.

GND:

This is a return line for the supply (ground pin of the IC).

UNIT-IV

Pre - requisite:

- **To Study the Architecture of 8051 and Timer mode**

Outcomes

- **Develop systems using different microcontrollers**

Architecture of 8051

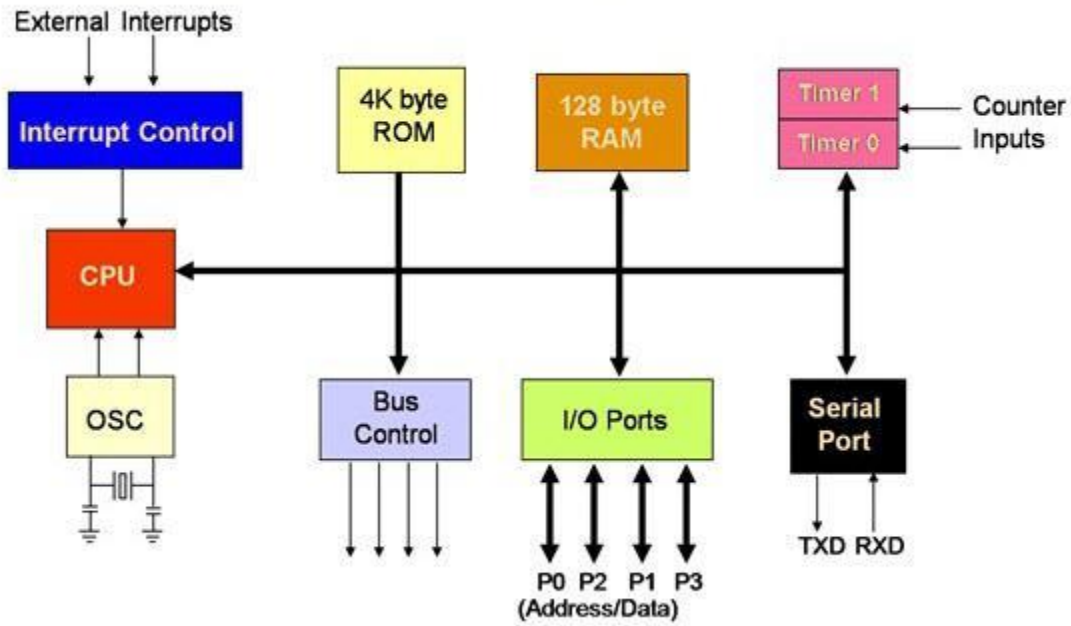
The first microprocessor **4004** was invented by Intel Corporation. **8085** and **8086** microprocessors were also invented by Intel. In 1981, Intel introduced an 8-bit microcontroller called the **8051**. It was referred as **system on a chip** because it had 128 bytes of RAM, 4K byte of on-chip ROM, two timers, one serial port, and 4 ports (8-bit wide), all on a single chip. When it became widely popular, Intel allowed other manufacturers to make and market different flavors of 8051 with its code compatible with 8051. It means that if you write your program for one flavor of 8051, it will run on other flavors too, regardless of the manufacturer. This has led to several versions with different speeds and amounts of on-chip RAM.

8051 - Members

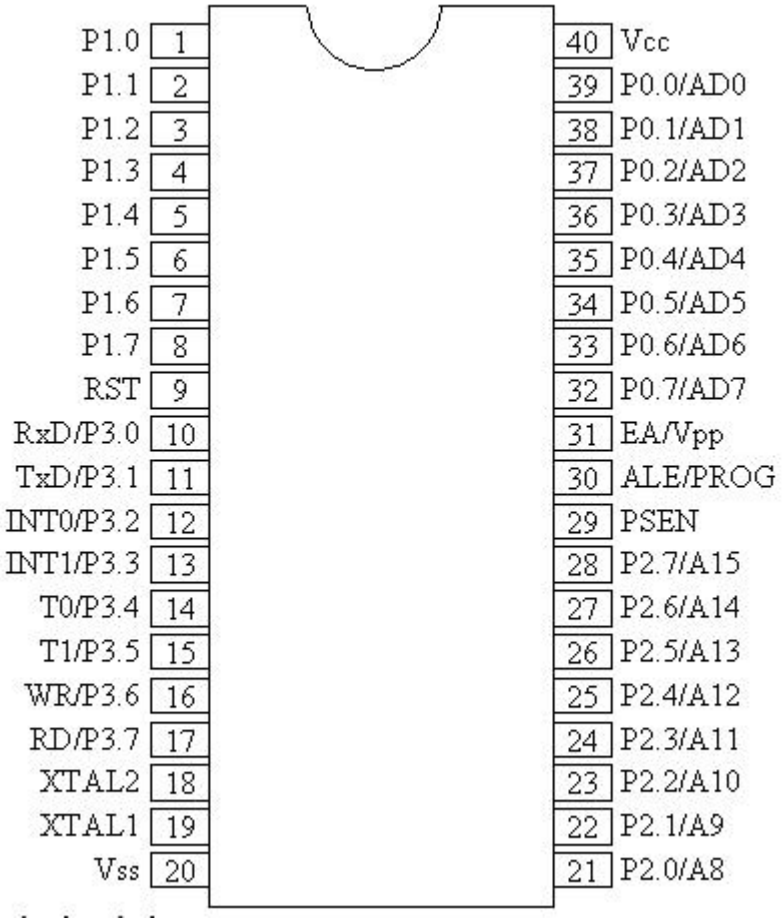
8052 microcontroller − 8052 has all the standard features of the 8051 microcontroller as well as an extra 128 bytes of RAM and an extra timer. It also has 8K bytes of on-chip program ROM instead of 4K bytes.

8031 microcontroller − It is another member of the 8051 family. This chip is often referred to as a ROM-less 8051, since it has 0K byte of on-chip ROM. You must add external ROM to it in order to use it, which contains the program to be fetched and executed. This program can be as large as 64K bytes. But in the process of adding external ROM to the 8031, it lost 2 ports out of 4 ports. To solve this problem, we can add an external I/O to the 8031

Block Diagram of 8051 Microcontroller



In 8051, I/O operations are done using four ports and 40 pins. The following pin diagram shows the details of the 40 pins. I/O operation port reserves 32 pins where each port has 8 pins. The other 8 pins are designated as V_{cc}, GND, XTAL1, XTAL2, RST, EA (bar), ALE/PROG (bar), and PSEN (bar).



I/O Ports and their Functions

The four ports P0, P1, P2, and P3, each use 8 pins, making them 8-bit ports. Upon RESET, all the ports are configured as inputs, ready to be used as input ports. When the first 0 is written to a port, it becomes an output. To reconfigure it as an input, a 1 must be sent to a port.

Port 0 (Pin No 32 – Pin No 39)

It has 8 pins (32 to 39). It can be used for input or output. Unlike P1, P2, and P3 ports, we normally connect P0 to 10K-ohm pull-up resistors to use it as an input or output port being an open drain.

It is also designated as AD0-AD7, allowing it to be used as both address and data. In case of 8031 (i.e. ROMless Chip), when we need to access the external ROM, then P0 will be used for both Address and Data Bus. ALE (Pin no 31) indicates if P0 has address or data. When ALE = 0, it provides data D0-D7, but when ALE = 1, it has address A0-A7. In case no external memory connection is available, P0 must be connected externally to a 10K-ohm pull-up resistor.

Dual Role of Port 0 and Port 2

Dual role of Port 0 − Port 0 is also designated as AD0–AD7, as it can be used for both data and address handling. While connecting an 8051 to external memory, Port 0 can provide both address and data. The 8051 microcontroller then multiplexes the input as address or data in order to save pins.

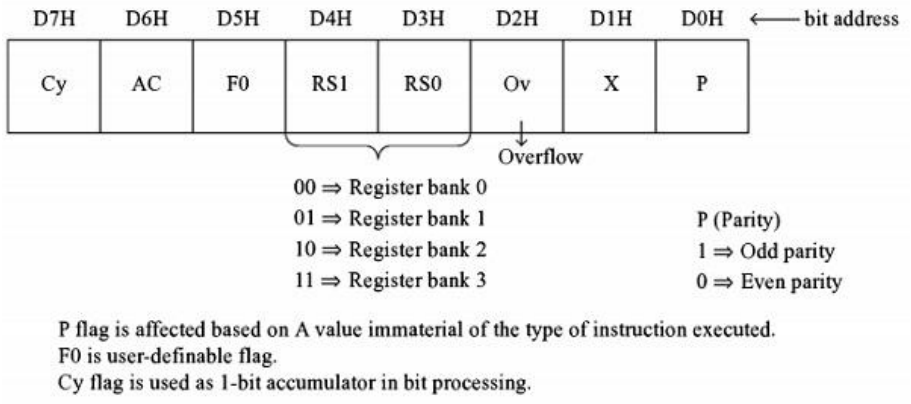
Dual role of Port 2 − Besides working as I/O, Port P2 is also used to provide 16-bit address bus for external memory along with Port 0. Port P2 is also designated as (A8– A15), while Port 0 provides the lower 8-bits via A0–A7. In other words, we can say that when an 8051 is connected to an external memory (ROM) which can be maximum up to 64KB and this is possible by 16 bit address bus because we know 216 = 64KB. Port2 is used for the upper 8-bit of the 16 bits address, and it cannot be used for I/O and this is the way any Program code of external ROM is addressed.

Special Function Registers:

The 8051 is a flexible microcontroller with a relatively large number of modes of operations. Your program may inspect and/or change the operating mode of the 8051 by manipulating the values of the 8051's Special Function Registers (SFRs).

SFRs are accessed as if they were normal Internal RAM. The only difference is that Internal RAM is from address 00h through 7Fh whereas SFR registers exist in the address range of 80h through FFh. Each SFR has an address (80h through FFh) and a name.

The following chart provides a graphical presentation of the 8051's SFRs, their names, and their address. As you can see, although the address range of 80h through FFh offer 128 possible addresses, there are only 21 SFRs in a standard 8051. All other addresses in the SFR range (80h through FFh) are considered invalid. Writing to or reading from these registers may produce undefined values or behavior.



P flag is affected based on A value immaterial of the type of instruction executed.
F0 is user-definable flag.
Cy flag is used as 1-bit accumulator in bit processing.

SFR Types

SFRs related to the I/O ports: The 8051 has four I/O ports of 8 bits, for a total of 32 I/O lines. Whether a given I/O line is high or low and the value read from the line are controlled by the SFRs. The SFRs control the operation or the configuration of some aspect of the 8051. For example, **TCON** controls the timers, SCON controls the serial port, the remaining SFRs, are auxillary SFRs in the sense that they don't directly configure the 8051 but obviously the
8051 cannot operate without them. For example, once the serial port has been configured using **SCON**, the program may read or write to the serial port using the **SBUF** register.

SFR Descriptions

P0 (Port 0, Address 80h, Bit-Addressable): This is input/output port 0. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 0 is pin P0.0, bit 7 is in P0.7. Writing a value of 1 to a bit of this SFR will send a high level on

the corresponding I/O pin whereas a value of 0 will bring it to a low level.

SP (Stack Pointer, Address 81h): This is the stack pointer of the microcontroller. This SFR indicates where the next value to be taken from the stack will be read from in Internal RAM. If you push a value onto the stack, the value will be written to the address of SP + 1. This SFR is modified by all instructions which modify the stack, such as PUSH, POP, LCALL, RET, RETI, and whenever interrupts are provoked by the microcontroller. The Stack Pointer, like all registers except DPTR and PC, may hold an 8-bit (1-byte) value.

When you pop a value off the stack, the 8051 returns the value from the memory location indicated by SP, and then decrements the value of SP.

This order of operation is important. When the 8051 is initialized SP will be initialized to 07h. If you immediately push a value onto the stack, the value will be stored in Internal RAM address 08h.

First the 8051 will increment the value of SP (from 07h to 08h) and then will store the pushed value at that memory address (08h). It is also used intrinsically whenever an interrupt is triggered .

DPL/DPH (Data Pointer Low/High, Addresses 82h/83h): The SFRs DPL and DPH work together to represent a 16-bit value called the *Data Pointer*. The data pointer is used in operations regarding external RAM and some instructions involving code memory. Since it is an unsigned two-byte integer value, it can represent values from 0000h to FFFFh (0 through 65,535 decimal).

PCON (Power Control, Addresses 87h): The Power Control SFR is used to control the 8051's power control modes. Certain operation modes of the 8051 allow the 8051 to go into a type of "sleep" mode which requires much less power. These modes of operation are controlled through PCON. Additionally, one of the bits in PCON is used to double the effective baud rate of the 8051's serial port.

TCON (Timer Control, Addresses 88h, Bit-Addressable): The Timer Control SFR is used to configure and modify the way in which the 8051's two timers operate. This SFR controls whether each of the two timers is running or stopped and contains a flag to indicate that each timer has overflowed. Additionally, some non-timer related bits are located in the TCON SFR. These bits are used to configure the way in which the external interrupts are activated and also contain the external interrupt flags which are set when an external interrupt has occurred.

TMOD (Timer Mode, Addresses 89h): The Timer Mode SFR is used to configure the mode of operation of each of the two timers. Using this SFR your program may configure each timer to be a 16-bit timer, an 8-bit autoreload timer, a 13-bit timer, or two separate timers. Additionally, you may configure the timers to only count when an external pin is activated or to count "events" that are indicated on an external pin.

TL0/TH0 (Timer 0 Low/High, Addresses 8Ah/8Bh): These two SFRs, taken together, represent timer 0. Their exact behavior depends on how the timer is configured in the TMOD SFR; however, these timers always count up. What is configurable is how and when they increment in value.

TL1/TH1 (Timer 1 Low/High, Addresses 8Ch/8Dh): These two SFRs, taken together, represent timer 1. Their exact behavior depends on how the timer is configured in the TMOD SFR; however, these timers always count up. What is configurable is how and when they increment in value.

P1 (Port 1, Address 90h, Bit-Addressable): This is input/output port 1. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 1 is pin P1.0, bit 7 is pin P1.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to a low level.

SCON (Serial Control, Addresses 98h, Bit-Addressable): The Serial Control SFR is used to configure the behavior of the 8051's on-board serial port. This SFR controls the baud rate of the serial port, whether the serial port is activated to receive data, and also contains flags that are set when a byte is successfully sent or received.

SBUF (Serial Control, Addresses 99h): The Serial Buffer SFR is used to send and receive data via the on-board serial port. Any value written to SBUF will be sent out the serial port's TXD pin. Any value which the 8051 receives via the serial port's RXD pin will be delivered to the user program via SBUF. In other words, SBUF serves as the output port when written to and as an input port when read from.

P2 (Port 2, Address A0h, Bit-Addressable): This is input/output port 2. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 2 is pin P2.0, bit 7 is pin P2.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to a low level.

IE (Interrupt Enable, Addresses A8h): The Interrupt Enable SFR is used to enable and disable specific interrupts. The low 7 bits of the SFR are used to enable/disable the specific interrupts, where as the highest bit is used to enable or disable ALL interrupts. Thus, if the high bit of IE is 0 all interrupts are disabled regardless of whether an individual interrupt is enabled by setting a lower bit.

P3 (Port 3, Address B0h, Bit-Addressable): This is input/output port 3. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 3 is pin P3.0, bit 7 is pin P3.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to a low level.

IP (Interrupt Priority, Addresses B8h, Bit-Addressable): The Interrupt Priority SFR is used to specify the relative priority of each interrupt. On the 8051, an interrupt may either be of low (0) priority or high (1) priority. An interrupt may only interrupt interrupts of lower priority. For example, if we configure the 8051 so that all interrupts are of low priority except the serial interrupt, the serial interrupt will always be able to interrupt the system, even if another interrupt is currently executing. However, if a serial interrupt is executing no other interrupt will be able to interrupt the serial interrupt routine since the serial interrupt routine has the highest priority.

PSW (Program Status Word, Addresses D0h, Bit-Addressable): The Program Status Word is used to store a number of important bits that are set and cleared by 8051 instructions. The PSW SFR contains the carry flag, the auxiliary carry flag, the overflow flag, and the parity flag. Additionally, the PSW register contains the register bank select flags which are used to select which of the "R" register banks
are currently selected.

ACC (Accumulator, Addresses E0h, Bit-Addressable): The Accumulator is one of the most used SFRs on the 8051 since it is involved in so many instructions. The Accumulator resides as an SFR at E0h, which means the instruction **MOV A,#20h** is really the same as **MOV E0h,#20h**. first method requires two bytes whereas the second option requires three bytes.
It can hold an 8-bit (1-byte) value and More than half of the 8051‘s 255 instructions manipulate or use the accumulator in some way.
For example, if you want to add the number 10 and 20, the resulting 30 will be store in the Accumulator. Once you have a value in the Accumulator you may continue processing the value or you may store it in another register or in memory.

B (B Register, Addresses F0h, Bit-Addressable): The "B" register is used in two instructions: the multiply and divide operations. The B register is also commonly used by programmers as an auxiliary register to temporarily store values. Thus, if you want to quickly and easily multiply or divide A by another number, you may store the other number in "B" and make use of these two instruction Aside from the MUL and DIV instructions, the "B" register is often used as yet another temporary storage register much like a ninth "R" register

Program Counter

The Program Counter is a 16- or 32-bit register which contains the address of the next instruction to be executed. The PC automatically increments to the next sequential memory location every time an instruction is fetched. Branch, jump, and interrupt operations load the Program Counter with an address other than the next sequential location.

Activating a power-on reset will cause all values in the register to be lost. It means the value of the PC (program counter) is 0 upon reset, forcing the CPU to fetch the first opcode from the ROM memory location 0000. It means we must place the first byte of upcode in ROM location 0000 because that is where the CPU expects to find the first instruction

Reset Vector
The significance of the reset vector is that it points the processor to the memory address which contains the firmware's first instruction. Without the Reset Vector, the processor would not know where to begin execution. Upon reset, the processor loads the Program Counter (PC) with the reset vector value from a predefined memory location. On CPU08 architecture, this is at location $FFFE:$FFFF.

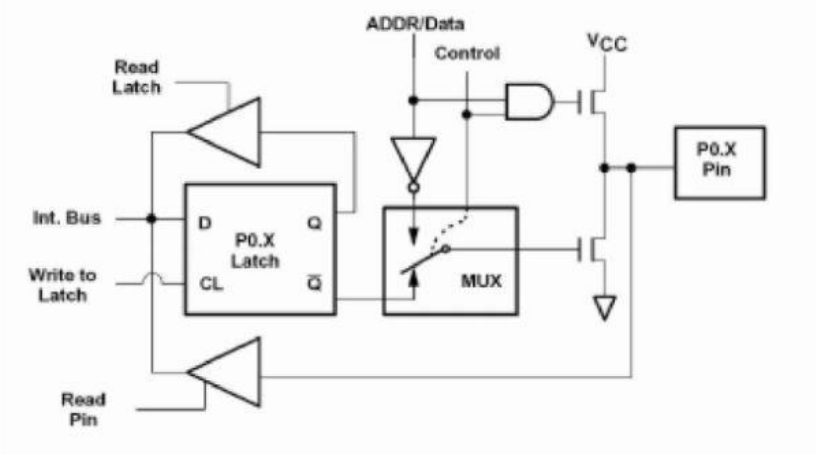
When the reset vector is not necessary, developers normally take it for granted and don’t program into the final image. As a result, the processor doesn't start up on the final product. It is a common mistake that takes place during the debug phase.

Stack Pointer
Stack is implemented in RAM and a CPU register is used to access it called SP (Stack Pointer) register. SP register is an 8-bit register and can address memory addresses of range 00h to FFh. Initially, the SP register contains value 07 to point to location 08 as the first location being used for the stack by the 8051.

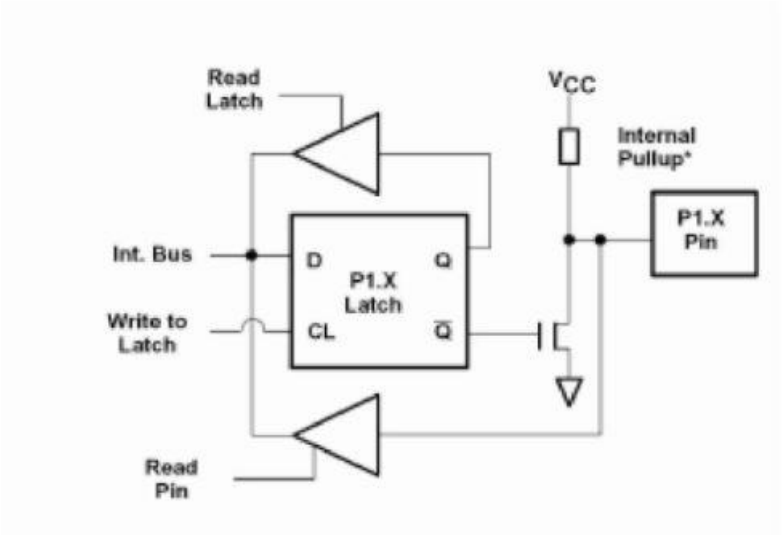
When the content of a CPU register is stored in a stack, it is called a PUSH operation. When the content of a stack is stored in a CPU register, it is called a POP operation. In other words, a register is pushed onto the stack to save it and popped off the stack to retrieve it.

I/O ports and circuits

Each port of 8051 has bidirectional capability. Port 0 is called 'true bidirectional port' as it floats (tristated) when configured as input. Port-1, 2, 3 are called 'quasi bidirectional port'. Port-0 Pin Structure Port -0 has 8 pins (P0.0-P0.7).

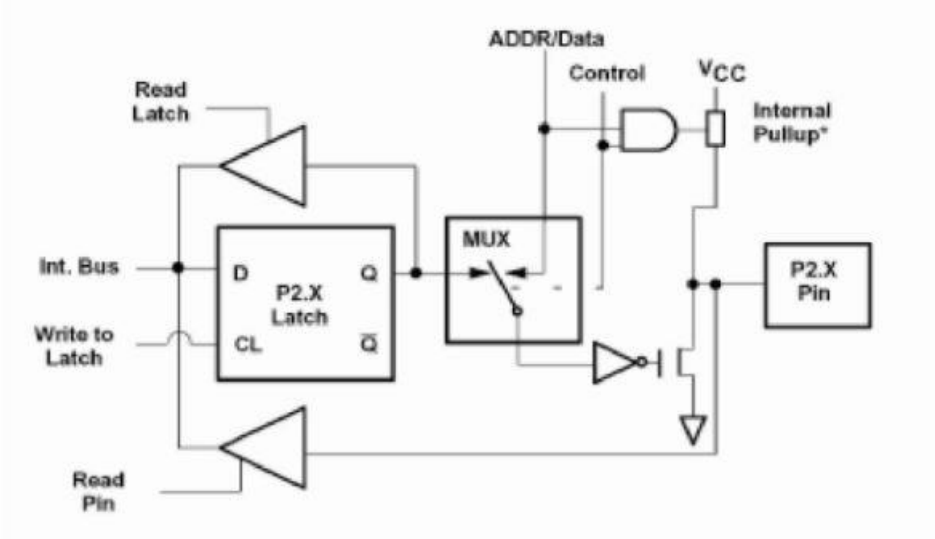


Port-0 can be configured as a normal bidirectional I/O port or it can be used for address/data interfacing for accessing external memory. When control is '1', the port is used for address/data interfacing. When the control is '0', the port can be used as a normal bidirectional I/O port. Let us assume that control is '0'. When the port is used as an input port, '1' is written to the latch. In this situation both the output MOSFETs are 'off'. Hence the output pin floats. This high impedance pin can be pulled up or low by an external source. When the port is used as an output port, a '1' written to the latch again turns 'off' both the output MOSFETs and causes the output pin to float. An external pull-up is required to output a
'1'. But when '0' is written to the latch, the pin is pulled down by the lower MOSFET. Hence the output becomes zero.
When the control is '1', address/data bus controls the output driver MOSFETs. If the address/data bus (internal) is '0', the upper MOSFET is 'off' and the lower MOSFET is
'on'. The output becomes '0'. If the address/data bus is '1', the upper transistor is 'on' and the lower transistor is 'off'. Hence the output is '1'. Hence for normal address/data
interfacing (for external memory access) no pull-up resistors are required.
Port-0 latch is written to with 1's when used for external memory access.
Port-1 Pin Structure Port-1 has 8 pins (P1.1-P1.7)



Port-1 does not have any alternate function i.e. it is dedicated solely for I/O interfacing. When used as output port, the pin is pulled up or down through internal pull-up. To use port-1 as input port, '1' has to be written to the latch. In this input mode when '1' is written to the pin by the external device then it read fine. But when '0' is written to the pin by the external device then the external source must sink current due to internal pull-up. If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading.

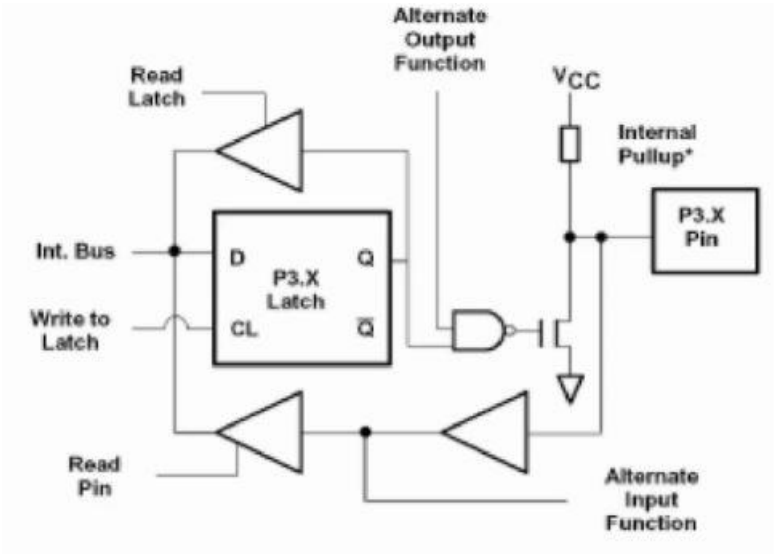
PORT 2 Pin Structure Port-2 has 8-pins (P2.0-P2.7) .



Port-2 is used for higher external address byte or a normal input/output port. The I/O operation is similar to Port-1. Port-2 latch remains stable when Port-2 pin are used for external memory access. Here again due to internal pull-up there is limited current driving capability.

PORT 3 Pin Structure

Port-3 has 8 pin (P3.0-P3.7). Port-3 pins have alternate functions



Each pin of Port-3 can be individually programmed for I/O operation or for alternate function. The alternate function can be activated only if the corresponding latch has been written to '1'. To use the port as input port, '1' should be written to the latch. This port also has internal pull-up and limited current driving capability.

8051 Addressing Modes

8051 has four addressing modes.

1. Immediate Addressing: Data is immediately available in the instruction.
For example -
ADD A, #77; Adds 77 (decimal) to A and stores in A
ADD A, #4DH; Adds 4D (hexadecimal) to A and stores in A MOV DPTR, #1000H; Moves 1000 (hexadecimal) to data pointer

2. Bank Addressing or Register Addressing: This way of addressing accesses the bytes in the current register bank. Data is available in the register specified in the instruction. The register bank is decided by 2 bits of Processor Status Word (PSW). For example-
ADD A, R0; Adds content of R0 to A and stores in A

3. Direct Addressing:
The address of the data is available in the instruction. For example - MOV A, 088H; Moves content of SFR TCON (address 088H)to A

4. Register Indirect Addressing:
The address of data is available in the R0 or R1 registers as specified in the instruction. For example - MOV A, @R0 moves content of address pointed by R0 to A .

5. External Data Addressing:
Pointer used for external data addressing can be either R0/R1 (256 byte access) or DPTR (64kbyte access).

For example - MOVX A, @R0; Moves content of 8-bit address pointed by R0 to A
MOVX A, @DPTR; Moves content of 16-bit address pointed by DPTR to A

6. External Code Addressing:
Sometimes we may want to store non-volatile data into the ROM e.g. look-up tables. Such data may require reading the code memory. This may be done as follows -
MOVC A, @A+DPTR; Moves content of address pointed by A+DPTR to A MOVC A, @A+PC; Moves content of address pointed by A+PC to A

Reference 8051 / 8052 Instruction Set

ACALL

Operation: ACALL
Function: Absolute Call Within 2K Block
Syntax: ACALL code address

Instructions	OpCode	Bytes	Cycles	Flags
ACALL page0	0x11	2	2	None
ACALL page1	0x31	2	2	None
ACALL page2	0x51	2	2	None
ACALL page3	0x71	2	2	None
ACALL page4	0x91	2	2	None
ACALL page5	0xB1	2	2	None
ACALL page6	0xD1	2	2	None
ACALL page7	0xF1	2	2	None

Description: ACALL unconditionally calls a subroutine at the indicated code address. ACALL pushes the address of the instruction that follows ACALL onto the stack, least-significant-byte first, most-significant-byte second. The Program Counter is then updated so that program execution continues at the indicated address.

The new value for the Program Counter is calculated by replacing the least-significant-byte of the Program Counter with the second byte of the ACALL instruction, and replacing bits 0-2 of the most-significant-byte of the Program Counter with 3 bits that indicate the page. Bits 3-7 of the most-significant-byte of the Program Counter remain unchanged.

Since only 11 bits of the Program Counter are affected by ACALL, calls may only be made to routines located within the same 2k block as the first byte that follows ACALL.

See Also: LCALL, RET, Instruction Set

ADD, ADDC

Operation: ADD, ADDC
Function: Add Accumulator, Add Accumulator With Carry
Syntax: ADD A,operand
ADDC A,operand

Instructions	OpCode	Bytes	Cycles	Flags
ADD A,#data	0x24	2	1	C, AC, OV
ADD A,iram addr	0x25	2	1	C, AC, OV
ADD A,@R0	0x26	1	1	C, AC, OV
ADD A,@R1	0x27	1	1	C, AC, OV
ADD A,R0	0x28	1	1	C, AC, OV
ADD A,R1	0x29	1	1	C, AC, OV
ADD A,R2	0x2A	1	1	C, AC, OV
ADD A,R3	0x2B	1	1	C, AC, OV
ADD A,R4	0x2C	1	1	C, AC, OV
ADD A,R5	0x2D	1	1	C, AC, OV
ADD A,R6	0x2E	1	1	C, AC, OV
ADD A,R7	0x2F	1	1	C, AC, OV

Instructions	OpCode	Bytes	Cycles	Flags
ADDC A,#data	0x34	2	1	C, AC, OV
ADDC A,iram addr	0x35	2	1	C, AC, OV
ADDC A,@R0	0x36	1	1	C, AC, OV
ADDC A,@R1	0x37	1	1	C, AC, OV
ADDC A,R0	0x38	1	1	C, AC, OV
ADDC A,R1	0x39	1	1	C, AC, OV
ADDC A,R2	0x3A	1	1	C, AC, OV
ADDC A,R3	0x3B	1	1	C, AC, OV
ADDC A,R4	0x3C	1	1	C, AC, OV

ADDC A,R5	0x3D	1	1	C, AC, OV
ADDC A,R6	0x3E	1	1	C, AC, OV
ADDC A,R7	0x3F	1	1	C, AC, OV

Description: Description: ADD and ADDC both add the value operand to the value of the Accumulator, leaving the resulting value in the Accumulator. The value operand is not affected. ADD and ADDC function identically except that ADDC adds the value of operand as well as the value of the Carry flag whereas ADD does not add the Carry flag to the result.

The **Carry bit (C)** is set if there is a carry-out of bit 7. In other words, if the unsigned summed value of the Accumulator, operand and (in the case of ADDC) the Carry flag exceeds 255 Carry is set. Otherwise, the Carry bit is cleared.

The **Auxiliary Carry (AC)** bit is set if there is a carry-out of bit 3. In other words, if the unsigned summed value of the low nibble of the Accumulator, operand and (in the case of ADDC) the Carry flag exceeds 15 the Auxiliary Carry flag is set. Otherwise, the Auxiliary Carry flag is cleared.

The **Overflow (OV)** bit is set if there is a carry-out of bit 6 or out of bit 7, but not both. In other words, if the addition of the Accumulator, operand and (in the case of ADDC) the Carry flag treated as signed values results in a value that is out of the range of a signed byte (-128 through +127) the Overflow flag is set. Otherwise, the Overflow flag is cleared.

See Also: SUBB, DA, INC, DEC, Instruction Set

AJMP

Operation: AJMP
Function: Absolute Jump Within 2K Block
Syntax: AJMP code address

Instructions	OpCode	Bytes	Cycles	Flags
AJMP page0	0x01	2	2	None
AJMP page1	0x21	2	2	None
AJMP page2	0x41	2	2	None
AJMP page3	0x61	2	2	None
AJMP page4	0x81	2	2	None

AJMP page5	0xA1	2	2	None
AJMP page6	0xC1	2	2	None
AJMP page7	0xE1	2	2	None

Description: AJMP unconditionally jumps to the indicated code address. The new value for the Program Counter is calculated by replacing the least-significant-byte of the Program Counter with the second byte of the AJMP instruction, and replacing bits 0-2 of the most-significant-byte of the Program Counter with 3 bits that indicate the page of the byte following the AJMP instruction. Bits 3-7 of the most-significant-byte of the Program Counter remain unchanged.

Since only 11 bits of the Program Counter are affected by AJMP, jumps may only be made to code located within the same 2k block as the first byte that follows AJMP.

See Also: LJMP, SJMP, Instruction Set

ANL

Operation: ANL
Function: Bitwise AND
Syntax: ANL operand1, operand2

Instructions	OpCode	Bytes	Cycles	Flags
ANL iram addr,A	0x52	2	1	None
ANL iram addr,#data	0x53	3	2	None
ANL A,#data	0x54	2	1	None
ANL A,iram addr	0x55	2	1	None
ANL A,@R0	0x56	1	1	None
ANL A,@R1	0x57	1	1	None
ANL A,R0	0x58	1	1	None
ANL A,R1	0x59	1	1	None
ANL A,R2	0x5A	1	1	None
ANL A,R3	0x5B	1	1	None

ANL A,R4	0x5C	1	1	None
ANL A,R5	0x5D	1	1	None
ANL A,R6	0x5E	1	1	None
ANL A,R7	0x5F	1	1	None
ANL C,bit addr	0x82	2	1	C
ANL C,/bit addr	0xB0	2	1	C

Description: ANL does a bitwise "AND" operation between operand1 and operand2, leaving the resulting value in operand1. The value of operand2 is not affected. A logical "AND" compares the bits of each operand and sets the corresponding bit in the resulting byte only if the bit was set in both of the original operands, otherwise the resulting bit is cleared.

See Also: ORL, XRL, Instruction Set

CJNE

Operation: CJNE
Function: Compare and Jump If Not Equal
Syntax: CJNE operand1,operand2,reladdr

Instructions	OpCode	Bytes	Cycles	Flags
CJNE A,#data,reladdr	0xB4	3	2	C
CJNE A,iram addr,reladdr	0xB5	3	2	C
CJNE @R0,#data,reladdr	0xB6	3	2	C
CJNE @R1,#data,reladdr	0xB7	3	2	C
CJNE R0,#data,reladdr	0xB8	3	2	C
CJNE R1,#data,reladdr	0xB9	3	2	C
CJNE R2,#data,reladdr	0xBA	3	2	C
CJNE R3,#data,reladdr	0xBB	3	2	C
CJNE R4,#data,reladdr	0xBC	3	2	C

CJNE R5,#data,reladdr	0xBD	3	2	C
CJNE R6,#data,reladdr	0xBE	3	2	C
CJNE R7,#data,reladdr	0xBF	3	2	C

Description: CJNE compares the value of operand1 and operand2 and branches to the indicated relative address if operand1 and operand2 are not equal. If the two operands are equal program flow continues with the instruction following the CJNE instruction.

The **Carry bit (C)** is set if operand1 is less than operand2, otherwise it is cleared.

See Also: DJNZ, Instruction Set

CLR

Operation: CLR
Function: Clear Register
Syntax: CLR register

Instructions	OpCode	Bytes	Cycles	Flags
CLR bit addr	0xC2	2	1	None
CLR C	0xC3	1	1	C
CLR A	0xE4	1	1	None

Description: CLR clears (sets to 0) all the bit(s) of the indicated register. If the register is a bit (including the carry bit), only the specified bit is affected. Clearing the Accumulator sets the Accumulator's value to 0.

See Also: SETB, Instruction Set

CPL

Operation: CPL
Function: Complement Register
Syntax: CPL operand

Instructions	OpCode	Bytes	Cycles	Flags
CPL A	0xF4	1	1	None
CPL C	0xB3	1	1	C
CPL bit addr	0xB2	2	1	None

Description: CPL complements operand, leaving the result in operand. If operand is a single bit then the state of the bit will be reversed. If operand is the Accumulator then all the bits in the Accumulator will be reversed. This can be thought of as "Accumulator Logical Exclusive OR 255" or as "255-Accumulator." If the operand refers to a bit of an output Port, the value that will be complemented is based on the last value written to that bit, not the last value read from it.

See Also: CLR, SETB, Instruction Set

DA

Operation: DA
Function: Decimal Adjust Accumulator
Syntax: DA A

Instructions	OpCode	Bytes	Cycles	Flags
DA	0xD4	1	1	C

Description: DA adjusts the contents of the Accumulator to correspond to a BCD (Binary Coded Decimal) number after two BCD numbers have been added by the ADD or ADDC instruction. If the carry bit is set or if the value of bits 0-3 exceed 9, 0x06 is added to the accumulator. If the carry bit was set when the instruction began, or if 0x06 was added to the accumulator in the first step, 0x60 is added to the accumulator.

The **Carry bit (C)** is set if the resulting value is greater than 0x99, otherwise it is cleared.

See Also: ADD, ADDC, Instruction Set

DEC

Operation: DEC
Function: Decrement Register
Syntax: DEC register

Instructions	OpCode	Bytes	Cycles	Flags
DEC A	0x14	1	1	None
DEC iram addr	0x15	2	1	None
DEC @R0	0x16	1	1	None
DEC @R1	0x17	1	1	None
DEC R0	0x18	1	1	None
DEC R1	0x19	1	1	None
DEC R2	0x1A	1	1	None
DEC R3	0x1B	1	1	None
DEC R4	0x1C	1	1	None
DEC R5	0x1D	1	1	None
DEC R6	0x1E	1	1	None
DEC R7	0x1F	1	1	None

Description: DEC decrements the value of register by 1. If the initial value of register is 0, decrementing the value will cause it to reset to 255 (0xFF Hex). Note: The Carry Flag is NOT set when the value "rolls over" from 0 to 255.

See Also: INC, SUBB, Instruction Set

DIV

Operation: DIV
Function: Divide Accumulator by B
Syntax: DIV AB

Instructions	OpCode	Bytes	Cycles	Flags
DIV AB	0x84	1	1	C, OV

Description: Divides the unsigned value of the Accumulator by the unsigned value of the "B" register. The resulting quotient is placed in the Accumulator and the remainder is placed in the "B" register.

The **Carry flag (C)** is always cleared.

The **Overflow flag (OV)** is set if division by 0 was attempted, otherwise it is cleared.

See Also: MUL AB, Instruction Set

DJNZ

Operation: DJNZ
Function: Decrement and Jump if Not Zero
Syntax: DJNZ register,reladdr

Instructions	OpCode	Bytes	Cycles	Flags
DJNZ iram addr,reladdr	0xD5	3	2	None
DJNZ R0,reladdr	0xD8	2	2	None
DJNZ R1,reladdr	0xD9	2	2	None
DJNZ R2,reladdr	0xDA	2	2	None
DJNZ R3,reladdr	0xDB	2	2	None
DJNZ R4,reladdr	0xDC	2	2	None
DJNZ R5,reladdr	0xDD	2	2	None
DJNZ R6,reladdr	0xDE	2	2	None
DJNZ R7,reladdr	0xDF	2	2	None

Description: DJNZ decrements the value of register by 1. If the initial value of register is 0, decrementing the value will cause it to reset to 255 (0xFF Hex). If the new value of register is not 0 the program will branch to the address indicated by relative addr. If the new value of register is 0 program flow continues with the instruction following the DJNZ instruction.

See Also: DEC, JZ, JNZ, Instruction Set

INC

Operation: INC
Function: Increment Register
Syntax: INC register

Instructions	OpCode	Bytes	Cycles	Flags
INC A	0x04	1	1	None
INC iram addr	0x05	2	1	None
INC @R0	0x06	1	1	None
INC @R1	0x07	1	1	None
INC R0	0x08	1	1	None
INC R1	0x09	1	1	None
INC R2	0x0A	1	1	None
INC R3	0x0B	1	1	None
INC R4	0x0C	1	1	None
INC R5	0x0D	1	1	None
INC R6	0x0E	1	1	None
INC R7	0x0F	1	1	None
INC DPTR	0xA3	1	2	None

Description: INC increments the value of register by 1. If the initial value of register is 255 (0xFF Hex), incrementing the value will cause it to reset to 0. Note: The Carry Flag is NOT set when the value "rolls over" from 255 to 0.

In the case of "INC DPTR", the value two-byte unsigned integer value of DPTR is incremented. If the initial value of DPTR is 65535 (0xFFFF Hex), incrementing the value will cause it to reset to 0. Again, the Carry Flag is NOT set when the value of DPTR "rolls over" from 65535 to 0.

See Also: ADD, ADDC, DEC, Instruction Set

JB

Operation: JB
Function: Jump if Bit Set
Syntax: JB bit addr, reladdr

Instructions	OpCode	Bytes	Cycles	Flags
JB bit addr,reladdr	0x20	3	2	None

Description: JB branches to the address indicated by reladdr if the bit indicated by bit addr is set. If the bit is not set program execution continues with the instruction following the JB instruction.

See Also: JBC, JNB. Instruction Set

JBC

Operation: JBC
Function: Jump if Bit Set and Clear Bit
Syntax: JB bit addr, reladdr

Instructions	OpCode	Bytes	Cycles	Flags
JBC bit addr,reladdr	0x10	3	2	None

Description: JBC will branch to the address indicated by reladdr if the bit indicated by bit addr is set. Before branching to reladdr the instruction will clear the indicated bit. If the bit is not set program execution continues with the instruction following the JBC instruction.

See Also: JB, JNB, Instruction Set

JC

Operation: JC
Function: Jump if Carry Set
Syntax: JC reladdr

Instructions	OpCode	Bytes	Cycles	Flags
JC reladdr	0x40	2	2	None

Description: JC will branch to the address indicated by reladdr if the Carry Bit is set. If the Carry Bit is not set program execution continues with the instruction following the JC instruction.

See Also: JNC, Instruction Set

JMP

Operation: JMP
Function: Jump to Data Pointer + Accumulator
Syntax: JMP @A+DPTR

Instructions	OpCode	Bytes	Cycles	Flags
JMP @A+DPTR	0x73	1	2	None

Description: JMP jumps unconditionally to the address represented by the sum of the value of DPTR and the value of the Accumulator.

See Also: LJMP, AJMP, SJMP, Instruction Set

JNP

Operation: JNB
Function: Jump if Bit Not Set
Syntax: JNB bit addr,reladdr

Instructions	OpCode	Bytes	Cycles	Flags
JNB bit addr,reladdr	0x30	3	2	None

Description: JNB will branch to the address indicated by reladdress if the indicated bit is not set. If the bit is set program execution continues with the instruction following the JNB instruction.

See Also: JB, JBC, Instruction Set

JNC

Operation: JNC
Function: Jump if Carry Not Set
Syntax: JNC reladdr

Instructions	OpCode	Bytes	Cycles	Flags
JNC reladdr	0x50	2	2	None

Description: JNC branches to the address indicated by reladdr if the carry bit is not set. If the carry bit is set program execution continues with the instruction following the JNB instruction.

See Also: JC, Instruction Set

JNZ

Operation: JNZ
Function: Jump if Accumulator Not Zero
Syntax: JNZ reladdr

Instructions	OpCode	Bytes	Cycles	Flags
JNZ reladdr	0x70	2	2	None

Description: JNZ will branch to the address indicated by reladdr if the Accumulator contains any value except 0. If the value of the Accumulator is zero program execution continues with the instruction following the JNZ instruction.

See Also: JZ, Instruction Set

JZ

Operation: JZ
Function: Jump if Accumulator Zero
Syntax: JNZ reladdr

Instructions	OpCode	Bytes	Cycles	Flags
JZ reladdr	0x60	2	2	None

Description: JZ branches to the address indicated by reladdr if the Accumulator contains the value 0. If the value of the Accumulator is non-zero program execution continues with the instruction following the JNZ instruction.

See Also: JNZ, Instruction Set

LCALL

Operation: LCALL
Function: Long Call
Syntax: LCALL code addr

Instructions	OpCode	Bytes	Cycles	Flags
LCALL code addr	0x12	3	2	None

Description: LCALL calls a program subroutine. LCALL increments the program counter by 3 (to point to the instruction following LCALL) and pushes that value onto the stack (low byte first, high byte second). The Program Counter is then set to the 16-bit value which follows the LCALL opcode, causing program execution to continue at that address.

See Also: ACALL, RET, Instruction Set

LJMP

Operation: LJMP
Function: Long Jump
Syntax: LJMP code addr

Instructions	OpCode	Bytes	Cycles	Flags
LJMP code addr	0x02	3	2	None

Description: LJMP jumps unconditionally to the specified code addr.

See Also: AJMP, SJMP, JMP, Instruction Set

MOV

Operation: MOV
Function: Move Memory
Syntax: MOV operand1,operand2

Instructions	OpCode	Bytes	Cycles	Flags
MOV @R0,#data	0x76	2	1	None
MOV @R1,#data	0x77	2	1	None
MOV @R0,A	0xF6	1	1	None
MOV @R1,A	0xF7	1	1	None

MOV @R0,iram addr	0xA6	2	2	None
MOV @R1,iram addr	0xA7	2	2	None
MOV A,#data	0x74	2	1	None
MOV A,@R0	0xE6	1	1	None
MOV A,@R1	0xE7	1	1	None
MOV A,R0	0xE8	1	1	None
MOV A,R1	0xE9	1	1	None
MOV A,R2	0xEA	1	1	None
MOV A,R3	0xEB	1	1	None
MOV A,R4	0xEC	1	1	None
MOV A,R5	0xED	1	1	None
MOV A,R6	0xEE	1	1	None
MOV A,R7	0xEF	1	1	None
MOV A,iram addr	0xE5	2	1	None
MOV C,bit addr	0xA2	2	1	C
MOV DPTR,#data16	0x90	3	2	None
MOV R0,#data	0x78	2	1	None
MOV R1,#data	0x79	2	1	None
MOV R2,#data	0x7A	2	1	None
MOV R3,#data	0x7B	2	1	None
MOV R4,#data	0x7C	2	1	None
MOV R5,#data	0x7D	2	1	None
MOV R6,#data	0x7E	2	1	None
MOV R7,#data	0x7F	2	1	None
MOV R0,A	0xF8	1	1	None
MOV R1,A	0xF9	1	1	None
MOV R2,A	0xFA	1	1	None
MOV R3,A	0xFB	1	1	None

MOV R4,A	0xFC	1	1	None
MOV R5,A	0xFD	1	1	None
MOV R6,A	0xFE	1	1	None
MOV R7,A	0xFF	1	1	None
MOV R0,iram addr	0xA8	2	2	None
MOV R1,iram addr	0xA9	2	2	None
MOV R2,iram addr	0xAA	2	2	None
MOV R3,iram addr	0xAB	2	2	None
MOV R4,iram addr	0xAC	2	2	None
MOV R5,iram addr	0xAD	2	2	None
MOV R6,iram addr	0xAE	2	2	None
MOV R7,iram addr	0xAF	2	2	None
MOV bit addr,C	0x92	2	2	None
MOV iram addr,#data	0x75	3	2	None
MOV iram addr,@R0	0x86	2	2	None
MOV iram addr,@R1	0x87	2	2	None
MOV iram addr,R0	0x88	2	2	None
MOV iram addr,R1	0x89	2	2	None
MOV iram addr,R2	0x8A	2	2	None
MOV iram addr,R3	0x8B	2	2	None
MOV iram addr,R4	0x8C	2	2	None
MOV iram addr,R5	0x8D	2	2	None
MOV iram addr,R6	0x8E	2	2	None
MOV iram addr,R7	0x8F	2	2	None
MOV iram addr,A	0xF5	2	1	None
MOV iram addr,iram addr	0x85	3	2	None

Description: MOV copies the value of operand2 into operand1. The value of operand2 is not affected. Both operand1 and operand2 must be in Internal RAM. No flags are affected unless the instruction is moving the value of a bit into the carry bit in which case the carry bit is affected or unless the instruction is moving a value into the PSW register (which contains all the program flags).

** Note: In the case of "MOV iram addr,iram addr", the operand bytes of the instruction are stored in reverse order. That is, the instruction consisting of the bytes 0x85, 0x20, 0x50 means "Move the contents of Internal RAM location 0x20 to Internal RAM location 0x50" whereas the opposite would be generally presumed.

See Also: MOVC, MOVX, XCH, XCHD, PUSH, POP, Instruction Set

MOVC

Operation: MOVC
Function: Move Code Byte to Accumulator
Syntax: MOVC A,@A+register

Instructions	OpCode	Bytes	Cycles	Flags
MOVC A,@A+DPTR	0x93	1	2	None
MOVC A,@A+PC	0x83	1	1	None

Description: MOVC moves a byte from Code Memory into the Accumulator. The Code Memory address from which the byte will be moved is calculated by summing the value of the Accumulator with either DPTR or the Program Counter (PC). In the case of the Program Counter, PC is first incremented by 1 before being summed with the Accumulator.

See Also: MOV, MOVX, Instruction Set

MOVX

Operation: MOVX
Function: Move Data To/From External Memory (XRAM)
Syntax: MOVX operand1,operand2

Instructions	OpCode	Bytes	Cycles	Flags
MOVX @DPTR,A	0xF0	1	2	None
MOVX @R0,A	0xF2	1	2	None
MOVX @R1,A	0xF3	1	2	None
MOVX A,@DPTR	0xE0	1	2	None
MOVX A,@R0	0xE2	1	2	None
MOVX A,@R1	0xE3	1	2	None

Description: MOVX moves a byte to or from External Memory into or from the Accumulator.

If operand1 is @DPTR, the Accumulator is moved to the 16-bit External Memory address indicated by DPTR. This instruction uses both P0 (port 0) and P2 (port 2) to output the 16-bit address and data. If operand2 is DPTR then the byte is moved from External Memory into the Accumulator.

If operand1 is @R0 or @R1, the Accumulator is moved to the 8-bit External Memory address indicated by the specified Register. This instruction uses only P0 (port 0) to output the 8-bit address and data. P2 (port 2) is not affected. If operand2 is @R0 or @R1 then the byte is moved from External Memory into the Accumulator.

See Also: MOV, MOVC, Instruction Set

MUL

Operation: MUL
Function: Multiply Accumulator by B
Syntax: MUL AB

Instructions	OpCode	Bytes	Cycles	Flags
MUL AB	0xA4	1	4	C, OV

Description: Multiples the unsigned value of the Accumulator by the unsigned value of the "B" register. The least significant byte of the result is placed in the Accumulator and the most-significant-byte is placed in the "B" register.

The **Carry Flag (C)** is always cleared.

The **Overflow Flag (OV)** is set if the result is greater than 255 (if the most-significant byte is not zero), otherwise it is cleared.

See Also: DIV, Instruction Set

NOP

Operation: NOP
Function: None, waste time
Syntax: No Operation

Instructions	OpCode	Bytes	Cycles	Flags
NOP	0x00	1	1	None

Description: NOP, as it's name suggests, causes No Operation to take place for one machine cycle. NOP is generally used only for timing purposes. Absolutely no flags or registers are affected.

See Also: Instruction Set

ORL

Operation: ORL
Function: Bitwise OR
Syntax: ORL operand1,operand2

Instructions	OpCode	Bytes	Cycles	Flags
ORL iram addr,A	0x42	2	1	None
ORL iram addr,#data	0x43	3	2	None
ORL A,#data	0x44	2	1	None
ORL A,iram addr	0x45	2	1	None
ORL A,@R0	0x46	1	1	None
ORL A,@R1	0x47	1	1	None
ORL A,R0	0x48	1	1	None

ORL A,R1	0x49	1	1	None
ORL A,R2	0x4A	1	1	None
ORL A,R3	0x4B	1	1	None
ORL A,R4	0x4C	1	1	None
ORL A,R5	0x4D	1	1	None
ORL A,R6	0x4E	1	1	None
ORL A,R7	0x4F	1	1	None
ORL C,bit addr	0x72	2	2	C
ORL C,/bit addr	0xA0	2	1	C

Description: ORL does a bitwise "OR" operation between operand1 and operand2, leaving the resulting value in operand1. The value of operand2 is not affected. A logical "OR" compares the bits of each operand and sets the corresponding bit in the resulting byte if the bit was set in either of the original operands, otherwise the resulting bit is cleared.

See Also: ANL, XRL, Instruction Set

POP

Operation: POP
Function: Pop Value From Stack
Syntax: POP

Instructions	OpCode	Bytes	Cycles	Flags
POP iram addr	0xD0	2	2	None

Description: POP "pops" the last value placed on the stack into the iram addr specified. In other words, POP will load iram addr with the value of the Internal RAM address pointed to by the current Stack Pointer. The stack pointer is then decremented by 1.

See Also: PUSH, Instruction Set

PUSH

Operation: PUSH
Function: Push Value Onto Stack

Syntax: PUSH

Instructions	OpCode	Bytes	Cycles	Flags
PUSH iram addr	0xC0	2	2	None

Description: PUSH "pushes" the value of the specified iram addr onto the stack. PUSH first increments the value of the Stack Pointer by 1, then takes the value stored in iram addr and stores it in Internal RAM at the location pointed to by the incremented Stack Pointer.

See Also: POP, Instruction Set

RET

Operation: RET
Function: Return From Subroutine
Syntax: RET

Instructions	OpCode	Bytes	Cycles	Flags
RET	0x22	1	2	None

Description: RET is used to return from a subroutine previously called by LCALL or ACALL. Program execution continues at the address that is calculated by popping the topmost 2 bytes off the stack. The most-significant-byte is popped off the stack first, followed by the least-significant-byte.

See Also: LCALL, ACALL, RETI, Instruction Set

RETI

Operation: RETI
Function: Return From Interrupt
Syntax: RETI

Instructions	OpCode	Bytes	Cycles	Flags
RETI	0x32	1	2	None

Description: RETI is used to return from an interrupt service routine. RETI first enables interrupts of equal and lower priorities to the interrupt that is terminating. Program execution continues at the address that is calculated by popping the topmost 2 bytes off the stack. The most-significant-byte is popped off the stack first, followed by the least-significant-byte.

RETI functions identically to RET if it is executed outside of an interrupt service routine.

See Also: RET, Instruction Set

RL

Operation: RL
Function: Rotate Accumulator Left
Syntax: RL A

Instructions	OpCode	Bytes	Cycles	Flags
RL A	0x23	1	1	C

Description: Shifts the bits of the Accumulator to the left. The left-most bit (bit 7) of the Accumulator is loaded into bit 0.

See Also: RLC, RR, RRC, Instruction Set

RLC

Operation: RLC
Function: Rotate Accumulator Left Through Carry
Syntax: RLC A

Instructions	OpCode	Bytes	Cycles	Flags
RLC A	0x33	1	1	C

Description: Shifts the bits of the Accumulator to the left. The left-most bit (bit 7) of the Accumulator is loaded into the Carry Flag, and the original Carry Flag is loaded into bit 0 of the Accumulator. This function can be used to quickly multiply a byte by 2.

See Also: RL, RR, RRC, Instruction Set

RR

Operation: RR
Function: Rotate Accumulator Right
Syntax: RR A

Instructions	OpCode	Bytes	Cycles	Flags
RR A	0x03	1	1	None

Description: Shifts the bits of the Accumulator to the right. The right-most bit (bit 0) of the Accumulator is loaded into bit 7.

See Also: RL, RLC, RRC, Instruction Set

RRC

Operation: RRC
Function: Rotate Accumulator Right Through Carry
Syntax: RRC A

Instructions	OpCode	Bytes	Cycles	Flags
RRC A	0x13	1	1	C

Description: Shifts the bits of the Accumulator to the right. The right-most bit (bit 0) of the Accumulator is loaded into the Carry Flag, and the original Carry Flag is loaded into bit 7. This function can be used to quickly divide a byte by 2.

See Also: RL, RLC, RR, Instruction Set

SETB

Operation: SETB
Function: Set Bit
Syntax: SETB bit addr

Instructions	OpCode	Bytes	Cycles	Flags
SETB C	0xD3	1	1	C
SETB bit addr	0xD2	2	1	None

Description: Sets the specified bit.

See Also: CLR, Instruction Set

SJMP

Operation: SJMP
Function: Short Jump
Syntax: SJMP reladdr

Instructions	OpCode	Bytes	Cycles	Flags
SJMP reladdr	0x80	2	2	None

Description: SJMP jumps unconditionally to the address specified reladdr. Reladdr must be within -128 or +127 bytes of the instruction that follows the SJMP instruction.

See Also: LJMP, AJMP, Instruction Set

SUBB

Operation: SUBB
Function: Subtract from Accumulator With Borrow
Syntax: SUBB A,operand

Instructions	OpCode	Bytes	Cycles	Flags

SUBB A,#data	0x94	2	1	C, AC, OV
SUBB A,iram addr	0x95	2	1	C, AC, OV
SUBB A,@R0	0x96	1	1	C, AC, OV
SUBB A,@R1	0x97	1	1	C, AC, OV
SUBB A,R0	0x98	1	1	C, AC, OV
SUBB A,R1	0x99	1	1	C, AC, OV
SUBB A,R2	0x9A	1	1	C, AC, OV
SUBB A,R3	0x9B	1	1	C, AC, OV
SUBB A,R4	0x9C	1	1	C, AC, OV
SUBB A,R5	0x9D	1	1	C, AC, OV
SUBB A,R6	0x9E	1	1	C, AC, OV
SUBB A,R7	0x9F	1	1	C, AC, OV

Description: SUBB subtract the value of operand from the value of the Accumulator, leaving the resulting value in the Accumulator. The value operand is not affected.

The **Carry Bit (C)** is set if a borrow was required for bit 7, otherwise it is cleared. In other words, if the unsigned value being subtracted is greater than the Accumulator the Carry Flag is set.

The **Auxillary Carry (AC)** bit is set if a borrow was required for bit 3, otherwise it is cleared. In other words, the bit is set if the low nibble of the value being subtracted was greater than the low nibble of the Accumulator.

The **Overflow (OV)** bit is set if a borrow was required for bit 6 or for bit 7, but not both. In other words, the subtraction of two signed bytes resulted in a value outside the range of a signed byte (-128 to 127). Otherwise it is cleared.

See Also: ADD, ADDC, DEC, Instruction Set

SWAP

Operation: SWAP
Function: Swap Accumulator Nibbles
Syntax: SWAP A

Instructions	OpCode	Bytes	Cycles	Flags
SWAP A	0xC4	1	1	None

Description: SWAP swaps bits 0-3 of the Accumulator with bits 4-7 of the Accumulator. This instruction is identical to executing "RR A" or "RL A" four times.

See Also: RL, RLC, RR, RRC, Instruction Set

XCH

Operation: XCH
Function: Exchange Bytes
Syntax: XCH A,register

Instructions	OpCode	Bytes	Cycles	Flags
XCH A,@R0	0xC6	1	1	None
XCH A,@R1	0xC7	1	1	None
XCH A,R0	0xC8	1	1	None
XCH A,R1	0xC9	1	1	None
XCH A,R2	0xCA	1	1	None
XCH A,R3	0xCB	1	1	None
XCH A,R4	0xCC	1	1	None
XCH A,R5	0xCD	1	1	None
XCH A,R6	0xCE	1	1	None
XCH A,R7	0xCF	1	1	None
XCH A,iram addr	0xC5	2	1	None

Description: Exchanges the value of the Accumulator with the value contained in register.

See Also: MOV, Instruction Set

XCHD

Operation: XCHD
Function: Exchange Digit
Syntax: XCHD A,[@R0/@R1]

Instructions	OpCode	Bytes	Cycles	Flags
XCHD A,@R0	0xD6	1	1	None
XCHD A,@R1	0xD7	1	1	None

Description: Exchanges bits 0-3 of the Accumulator with bits 0-3 of the Internal RAM address pointed to indirectly by R0 or R1. Bits 4-7 of each register are unaffected.

See Also: DA, Instruction Set

XRL

Operation: XRL
Function: Bitwise Exclusive OR
Syntax: XRL operand1,operand2

Instructions	OpCode	Bytes	Cycles	Flags
XRL iram addr,A	0x62	2	1	None
XRL iram addr,#data	0x63	3	2	None
XRL A,#data	0x64	2	1	None
XRL A,iram addr	0x65	2	1	None
XRL A,@R0	0x66	1	1	None
XRL A,@R1	0x67	1	1	None
XRL A,R0	0x68	1	1	None

XRL A,R1	0x69	1	1	None
XRL A,R2	0x6A	1	1	None
XRL A,R3	0x6B	1	1	None
XRL A,R4	0x6C	1	1	None
XRL A,R5	0x6D	1	1	None
XRL A,R6	0x6E	1	1	None
XRL A,R7	0x6F	1	1	None

Description: XRL does a bitwise "EXCLUSIVE OR" operation between operand1 and operand2, leaving the resulting value in operand1. The value of operand2 is not affected. A logical "EXCLUSIVE OR" compares the bits of each operand and sets the corresponding bit in the resulting byte if the bit was set in either (but not both) of the original operands, otherwise the bit is cleared.

See Also: ANL, ORL, Instruction Set

UNDEFINED

Operation: Undefined Instruction
Function: Undefined
Syntax: ???

Instructions	OpCode	Bytes	Cycles	Flags
???	0xA5	1	1	C

Description: The "Undefined" instruction is, as the name suggests, not a documented instruction. The 8051 supports 255 instructions and OpCode 0xA5 is the single OpCode that is not used by any documented function. Since it is not documented nor defined it is not recommended that it be executed. However, based on my research, executing this undefined instruction takes 1 machine cycle and appears to have no effect on the system except that the Carry Bit always seems to be set.

Programming 8051 Timers

One of the primary uses of timers is to measure time. When a timer is in interval timer mode (as opposed to event counter mode) and correctly configured, it will increment by 1 every machine cycle. A single machine cycle consists of 12 crystal pulses. Thus a running timer will be incremented:11,059,000 / 12 = 921,583 times per second.

Unlike instructions which require 1 machine cycle, others 2, and others 4--the timers are consistent: They will always be incremented once per machine cycle. Thus if a timer has
counted from 0 to 50,000 you may calculate: 50,000 / 921,583 = .0542.0542 seconds have passed. To execute an event once per second you'd have to wait for the timer to count from 0 to 50,000 18.45times.
To calculate how many times the timer will be incremented in .05 seconds, a simple multiplication can be done: 0 .05 * 921,583 = 46,079.15.
This tells us that it will take .05 seconds (1/20th of a second) to count from 0 to
46.0. To work with timers is to control the timers and initialize them.

The TMOD SFR
TMOD (Timer Mode): The TMOD SFR is used to control the mode of operation of both timers. Each bit of the SFR gives the microcontroller specific information concerning how to run a timer. The high four bits (bits 4 through 7) relate to Timer 1whereas the low four bits (bits 0 through 3) perform the exact same functions, but for timer 0. The modes of operation are:

TxM1	TxM0	Timer Mode	Description of Mode
0	0	0	13-bit Timer.
0	1	1	16-bit Timer
1	0	2	8-bit auto-reload
1	1	3	13-bit Time Mode (mode 0)

Timer mode "0" is a 13-bit timer. When the timer is in 13-bit mode, TLx will count from 0 to 31. When TLx is incremented from 31, it will "reset" to 0 and increment THx. Thus, effectively, only 13 bits of the two timer bytes are being used: bits 0-4 of TLx and bits 0-7 of THx. The timer can only contain 8192 values. If you set a 13-bit timer to 0, it will overflow back to zero 8192 machine cycles later.

16-bit Time Mode (mode 1)

Timer mode "1" is a 16-bit timer. TLx is incremented from 0 to 255. When TLx is incremented from 255, it resets to 0 and causes THx to be incremented by 1. Since this is a full 16-bit timer, the timer may contain up to 65536 distinct values. If you set a 16-bit timer to 0, it will overflow back to 0 after 65,536 machine cycles.

8-bit Time Mode (mode 2)

Timer mode "2" is an 8-bit auto-reload mode. When a timer is in mode 2, THx holds the "reload value" and TLx is the timer itself. Thus, TLx starts counting up. When TLx reaches 255 and is subsequently incremented, instead of resetting to 0 (as in the case of modes 0 and 1), it will be reset to the value stored in THx. For example, if TH0 holds the value FDh and TL0 holds the value FEh values of TH0 and TL0 for a few machine cycles:

The value of TH0 never changed. When we use mode 2 you almost always set THx to a known value and TLxis the SFR that is constantly incremented. The benefit of auto-reload mode is the timer always have a value from 200 to 255. If you use mode 0 or 1, you'd have
to check in code to see if the timer had overflowed and, if so, reset the timer to 200. This takes precious instructions of execution time to check the value and/or to reload it. When you use mode 2 the microcontroller takes care of this. Auto-reload mode is very commonly used for establishing a baud rate in Serial Communications.

Machine Cycle	TH0 Value	TL0 Value
1	FDh	FEh
2	FDh	FFh
3	FDh	FDh
4	FDh	FEh
5	FDh	FFh
6	FDh	FDh
7	FDh	FEh

Split Timer Mode (mode 3)

Timer mode "3" is a split-timer mode. When Timer 0 is placed in mode 3, it essentially becomes two separate 8-bit timers. Timer 0 is TL0 and Timer 1 is TH0. Both timers count from 0 to 255 and overflow back to 0. All the bits that are related to Timer 1 will now be tied to TH0. While Timer 0 is in split mode, the real Timer 1 (i.e. TH1 and TL1) can be put into modes 0, 1 or 2 normally--however, you may not start or stop the real timer 1 since the bits that do that are now linked to TH0. The real timer 1,e, will be incremented every machine cycle always. The only real use in split timer mode is if you need to have two separate timers and, additionally, a baud rate generator you can use the real Timer 1 as a baud rate generator and use TH0/TL0 as two separate timers.

Reading the Timer

There are two common ways of reading the value of a 16-bit timer; which you use depends on your specific application. You may either read the actual value of the timer as a 16-bit number, or you may simply detect when the timer has overflowed.

Reading the value of a Timer

If timer is in an 8-bit mode either 8-bit Auto Reload mode or in split timer mode, you simply read the 1-byte value of the timer. With a 13-bit or16-bit timer the timer value was
14/255 (High byte 14, low byte 255) but you read 15/255.

Serial Port Programming: 8051 Serial Communication

One of the 8051‘s many powerful features -integrated *UART*, known as a serial port to
easily read and write values to the serial port instead of turning on and off one of the I/O lines in rapid succession to properly "clock out" each individual bit, including start bits, stop bits and parity bits. Setting the Serial Port Mode configures it by specifying 8051 how many data bits.
we want, the baud rate we will be using and how the baud rate will be determined. First, let‘s present the "Serial Control" (SCON) SFR and define what each bit of the SFR

The SCON SFR allows us to configure the Serial Port. The first four bits (bits 4 through 7) are configuration bits: Bits SM0 and SM1 is to set the serial mode to a value between 0 and 3, inclusive as in table above selecting the Serial Mode selects the mode of operation (8-bit/9-bit, UART or Shift Register) and also determines how the baud rate will be calculated. In modes 0 and 2 the baud rate is fixed based on the oscillator‘s frequency.

In modes 1 and 3 the baud rate is variable based on how often Timer 1 overflows. The next bit, SM2, is a flag for " Multiprocessor communication whenever a byte has been received the 8051 will set the "RI" (Receive Interrupt) flag to let the program know that a byte has been received and that it needs to be processed. However, when SM2 is set the "RI" flag will only be triggered if the 9th bit received was a "1". if SM2 is set and a byte is received whose 9th bit is clear, the RI flag will never be set .You will almost always want to clear this bit so that the flag is set upon reception of *any* character. The next bit, **REN**, is "Receiver Enable." is set indicate to data received via the serial port.
The last four bits (bits 0 through 3) are operational bits. They are used when actually sending and receiving data--they are not used to configure the serial port. The **TB8** bit is used in modes 2 and 3.

SM0	SM1	Serial Mode	Explanation Baud Rate
0	0	0	0 8-bit Shift Register Oscillator / 12
0	1	1	8-bit UART Set by Timer 1 (*)
1	0	2	9-bit UART Oscillator / 32 (*)
1	1	3	9-bit UART Set by Timer 1 (*)

In modes 2 and 3, a total of nine data bits are transmitted. The first 8 data bits are the 8 bits of the main value, and the ninth bit is taken from TB8. If TB8 is set and a value is written to the serial port, the data‘s bits will be written to the serial line followed by a "set" ninth bit. If TB8 is clear the ninth bit will be "clear." The **RB8** also operates in modes 2 and 3and functions essentially the same way as TB8, but on the reception side. When a byte is received in modes 2 or 3, a total of nine bits are received. In this case, the first eight bits received are the data of the serial byte received and the value of the nineth bit received will be placed in RB8.**TI** means "Transmit Interrupt."

When a program writes a value to the serial port, a certain amount of time will pass before the individual bits of the byte are "clocked out" the serial port. If the program were to write another byte

to the serial port before the first byte was completely output, the data being sent would be garbled. Thus, the8051 lets the program know that it has "clocked out" the last byte by setting the TI bit.
When the TI bit is set, the program may assume that the serial port is "free" and ready to send the next byte. Finally, the **RI** bit means "Receive Interrupt." It functions similarly to the "TI" bit, but it indicates that a byte has been received. Whenever the 8051 has received a complete byte it will trigger the RI bit to let the program know that it needs to read the value quickly, before another byte is read.

Setting the Serial Port Baud Rate

Once the Serial Port Mode has been configured, the program must configure the serial port‘s baud rate. This only applies to Serial Port modes 1 and 3. The Baud Rate is determined based on the oscillator‘s frequency when in mode 0 and 2. In mode 0, the baud rate is always the oscillator frequency divided by 12. This means if you‘re crystal is 1.059 Mhz, mode 0 baud rate will always be 921,583 baud. In mode 2 the baud rate is always the oscillator frequency divided by 64, so a 11.059Mhz crystal speed will yield a baud rate of172,797.

In modes 1 and 3, the baud rate is determined by how frequently timer 1 overflows. The more frequently timer 1 overflows, the higher the baud rate. There are many ways one can cause timer 1 to overflow at a rate that determines a baud rate, but the most common method is to put timer 1 in 8-bit auto-reload mode (timer mode2) and set a reload value (TH1) that causes Timer 1 to overflow at a frequency appropriate to generate a baud rate.

Writing to the Serial Port

Once the Serial Port has been properly configured as explained above, the serial port is ready to be used to send data and receive data. To write a byte to the serial write the value to the SBUF (99h) SFR. For example, if you wanted to send the letter "A" to the serial port, it could be accomplished as easily as: MOV SBUF, #‘A‘

Upon execution of the above instruction the 8051 will begin transmitting the character via the serial port. Obviously transmission is not instantaneous--it takes a measureable amount of time to transmit. And since the 8051 does not have a serial output buffer we need to be sure that a character is completely transmitted before we try to transmit the next character.

Reading the Serial Port

Reading data received by the serial port is equally easy. To read a byte from the serial port one just needs to read the value stored in the SBUF (99h) SFR after the 8051 has automatically set the RI flag in SCON.

Interrupt Programming:

The following events will cause an interrupt:
Timer 0 Overflow.
Timer 1 Overflow.
Reception/Transmission of Serial Character.
External Event 0.
External Event 1.

To distinguish between various interrupts and executing different code depending on what interrupt was triggered 8051may be jumping to a fixed address when a given interrupt occurs.

Interrupt	Flag	Interrupt Handler Address
External 0	IE0	0003h
Timer 0	TF0	000Bh
External 1	IE1	0013h
Timer 1	TF1	001Bh
Serial	RI/TI	0023h

If Timer 0 overflows (i.e., the TF0 bit is set), the main program will be temporarily suspended and control will jump to 000BH if we have code at address 0003H that handles the situation of Timer 0 overflowing.

Setting Up Interrupts

By default at power up, all interrupts are disabled. Even if, for example, the TF0 bit is set, the 8051 will not execute the interrupt. Your program must specifically tell the
8051 that it wishes to enable interrupts and specifically which interrupts it wishes to enable. Your program may enable and disable interrupts by modifying the IE SFR (A8h)

Each of the 8051'sinterrupts has its own bit in the IE SFR. You enable a given interrupt by setting the corresponding bit. For example, if you wish to enable Timer 1 Interrupt, you would execute either:
MOV IE,#08h || SETB ET1

Both of the above instructions set bit 3 of IE, thus enabling Timer 1 Interrupt. Once Timer 1
Interrupt is enabled, whenever the TF1 bit is set, the 8051 will automatically put "on hold" the main program and execute the Timer 1 Interrupt Handler at address 001Bh. However, before Timer 1 Interrupt (or any other interrupt) is truly enabled, you must also set bit 7 of IE. Bit 7, the Global Interrupt Enable/Disable, enables or disables all interrupts simultaneously. That is to say, if bit 7 is cleared then no interrupts will occur, even if all the other bits of IE are set. Setting bit 7 will enable all the interrupts that have been selected by setting other bits in IE. This is useful in program execution if you have time-critical code that needs to execute.

In this case, you may need the code to execute from start to finish without any interrupt getting in the way. To accomplish this you can simply clear bit 7 of IE (CLR EA) and then set it after your time critical code is done. To enable the Timer 1 Interrupt execute the following two instructions:
SETB ET1
SETB EA
Thereafter, the Timer 1 Interrupt Handler at 01Bh will automatically be called whenever the
TF1 bit is set (upon Timer 1 overflow).

Polling Sequence
The 8051 automatically evaluates whether an interrupt should occur after every instruction. When checking for interrupt conditions, it checks them in the following order:

1) External 0 Interrupt
2) Timer 0 Interrupt
3) External 1 Interrupt
4) Timer 1 Interrupt
5) Serial Interrupt

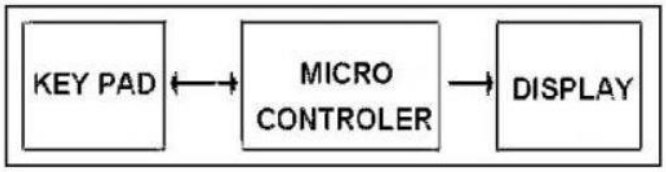
Interrupt Priorities

The 8051 offers two levels of interrupt priority: high and low. By using interrupt priorities you may assign higher priority to certain interrupt conditions. For example, you may have enabled Timer 1 Interrupt which is automatically called every time Timer 1 overflows. Additionally, you may have enabled the Serial Interrupt which is called every time a character is received via the serial port. However, you may consider that receiving a character is much more important than the timer interrupt. In this case, if Timer 1 Interrupt is already executing you may wish that the serial interrupt itself interrupts the Timer 1
Interrupt. When the serial interrupt is complete, control passes back to Timer 1 Interrupt and finally back to the main program. You may accomplish this by assigning a high priority to the Serial Interrupt and a low priority to the Timer 1 Interrupt.

Interfacing a Microprocessor to Keyboard

When you press a key on your computer, you are activating a switch. There are many different ways of making these switches. An overview of the construction and operation of some of the most common types.

Mechanical key switches: In mechanical-switch keys, two pieces of metal are pushed together when you press the key. The actual switch elements are often made of a phosphor-bronze alloy with gold platting on the contact areas. The key switch usually contains a spring to return the key to the nonpressed position and perhaps a small piece of foam to help damp out bouncing. Some mechanical key switches now consist of a molded silicon dome with a small piece of conductive rubber foam short two trace on the printed-circuit board to produce the key pressed signal. Mechanical switches are relatively inexpensive but they have several disadvantages. First, they suffer from contact bounce. A pressed key may make and break contact several times before it makes solid contact. Second, the contacts may become oxidized or dirty with age so they no longer make A dependable connection. Higher- quality mechanical switches typically have a rated life time of about 1 million keystrokes. The silicone dome type typically last 25 million keystrokes.

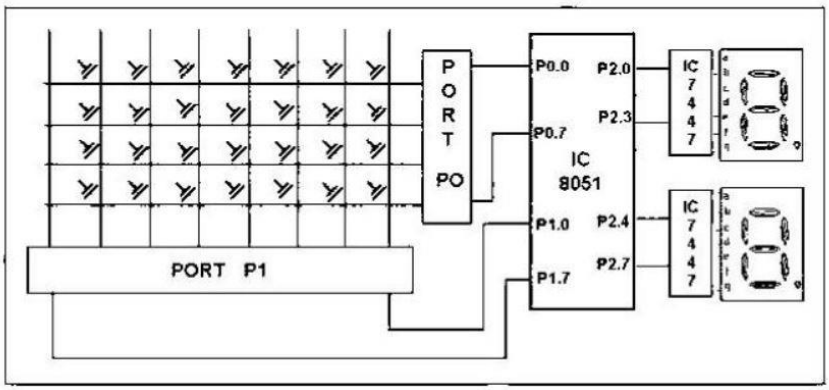


Membrane key switches: These switches are really a special type of mechanical switches. They consist of a three-layer plastic or rubber sandwich. The top layer has a conductive line of silver ink running under each key position. The bottom layer has a conductive line of silver ink running under each column of keys. The key board interfaced is a matrix keyboard. This key board is designed with a particular rows and columns. These rows and columns are connected to the microcontroller through its ports of the micro controller 8051. We normally use 8*8 matrix key board. So only two ports of 8051 can be easily connected to the rows and columns of the key board. Whenever a key is pressed, a row and a column gets shorted through that pressed key and all the other keys are left open. When a

key is pressed only a bit in the port goes high which indicates microcontroller that the key is pressed. By this high on the bit key in the corresponding column is identified.

Interfacing To Alphanumeric Displays

• To give directions or data values to users, many microprocessor-controlled instruments and machines need to display letters of the alphabet and numbers. In systems where a large amount of data needs to be displayed a CRT is used to display the data. In system where only a small amount of data needs to be displayed, simple digit-type displays are often used.
• There are several technologies used to make these digit-oriented displays but we are discussing only the two major types.
• These are *light emitting diodes* (LED) and *liquid-crystal displays* (LCD).
• LCD displays use very low power, so they are often used in portable, battery-powered instruments. They do not emit their own light, they simply change the reflection of available light. Therefore, for an instrument that is to be used in low-light conditions, you have to include a light source for LCDs or use LEDs which emit their own light.



Once we get the row next out job is to find out the column of the pressed key. The column is detected by contents in the input ports with the help of a counter. The content of the input port is rotated with carry until the carry bit is set.
The contents of the counter is then compared and displayed in the display. This display is designed using a seven segment display and a BCD to seven segment decoder IC 7447. The BCD equivalent number of counter is sent through output part of 8051 displays the number of pressed key.

Interfacing Analog to Digital Data Converters

• In most of the cases, the PPI 8255 is used for interfacing the analog to digital converters with microprocessor.
• The analog to digital converters is treaded as an input device by the microprocessor, that sends an initialising signal to the ADC to start the analogy to digital data conversation process. The start of conversation signal is a pulse of a specific duration.
• The process of analog to digital conversion is a slow process, and the microprocessor has to wait for the digital data till the conversion is over. After the conversion is over, the ADC sends end of conversion EOC signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC. These tasks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports.
• The time taken by the ADC from the active edge of SOC pulse till the active edge of EOC

signal is called as the conversion delay of the ADC.
• It may range anywhere from a few microseconds in case of fast ADC to even a few hundred milliseconds in case of slow ADCs.
• The available ADC in the market use different conversion techniques for conversion of analog signal to digitals. Successive approximation techniques and dual slope integration techniques are the most popular techniques used in the integrated ADC chip.
• General algorithm for ADC interfacing contains the following steps:
1. Ensure the stability of analog input, applied to the ADC.
2. Issue start of conversion pulse to ADC
3. Read end of conversion signal to mark the end of conversion processes.
4. Read digital data output of the ADC as equivalent digital output.
5. Analog input voltage must be constant at the input of the ADC right from the start of conversion till the end of the conversion to get correct results. This may be ensured by a sample and hold circuit which samples the analog signal and holds it constant for a specific time duration. The microprocessor may issue a hold signal to the sample and hold circuit.
6. If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct.

Stepper Motor Interface

The complete board consists of transformer, control circuit, keypad and stepper motor as shown in snap.
The circuit has inbuilt 5 V power supply so when it is connected with transformer it will give the supply to circuit and motor both. The 8 Key keypad is connected with circuit through which user can give the command to control stepper motor. The control circuit includes micro controller 89C51, indicating LEDs, and current driver chip ULN2003A. One can program the controller to control the operation of stepper motor. He can give different commands through keypad like, run clockwise, run anticlockwise, increase/decrease RPM, increase/decrease revolutions, stop motor, change the mode, etc. **Unipolar stepper motor:-** unipolar stepper motor has four coils. One end of each coil is tied together and it gives common terminal which is always connected with positive terminal of supply. The other ends of each coil are given for interface. Specific color code may also be given. Like in my motor orange is first coil (L1), brown is second (L2), yellow is third (L3), black is fourth (L4) and red for common terminal.
By means of controlling a stepper motor operation we can
1. Increase or decrease the RPM (speed) of it
2. Increase or decrease number of revolutions of it
3. Change its direction means rotate it clockwise or anticlockwise
To vary the RPM of motor we have to vary the PRF (Pulse Repetition Frequency). Number of applied pulses will vary number of rotations and last to change direction we have to change pulse sequence.
So all these three things just depends on applied pulses. Now there are three different modes to rotate this motor

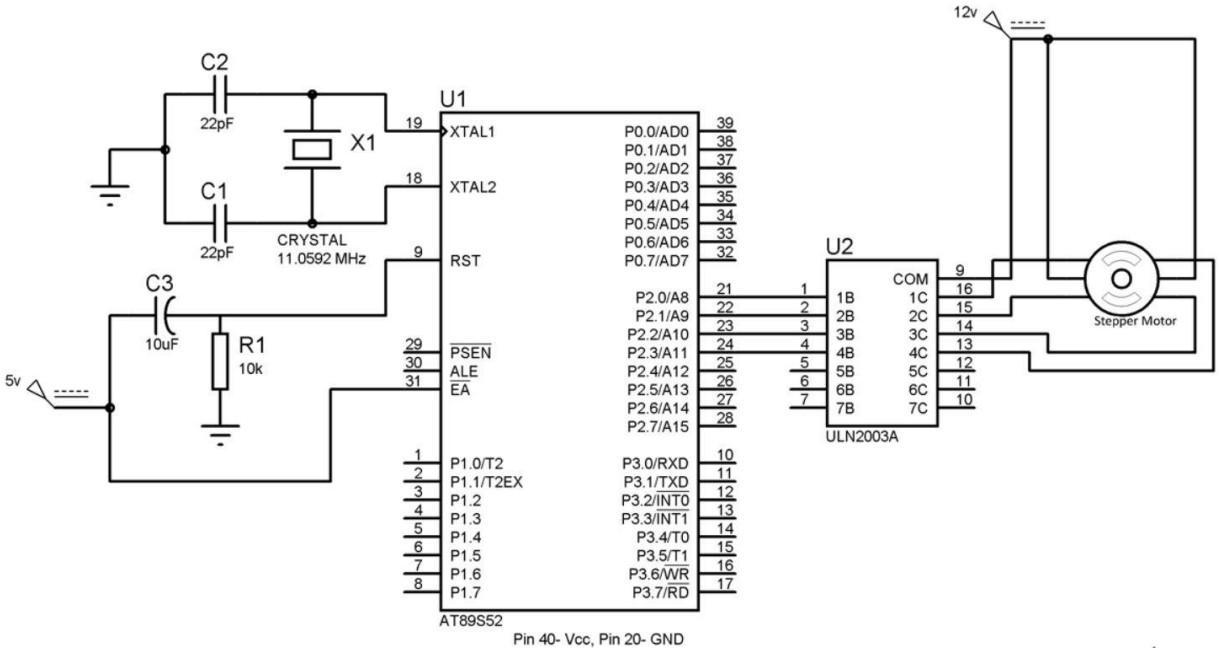
1. Single coil excitation
2. Double coil excitation
3. Half step excitation

The circuit consists of very few components. The major components are 7805, 89C51 and ULN2003A.

Connections:-

1. The transformer terminals are given to bridge rectifier to generate rectified DC.
2. It is filtered and given to regulator IC 7805 to generate 5 V pure DC. LED indicates supply is ON.
3. All the push button micro switches J1 to J8 are connected with port P1 as shown to form serial keyboard.
4. 12 MHz crystal is connected to oscillator terminals of 89C51 with two biasing capacitors.
5. All the LEDs are connected to port P0 as shown
6. Port P2 drives stepper motor through current driver chip ULN2003A.

The common terminal of motor is connected to Vcc and rest all four terminals are connected to port P2 pins in sequence through ULN chip.



UNIT-V

Pre - requisite:

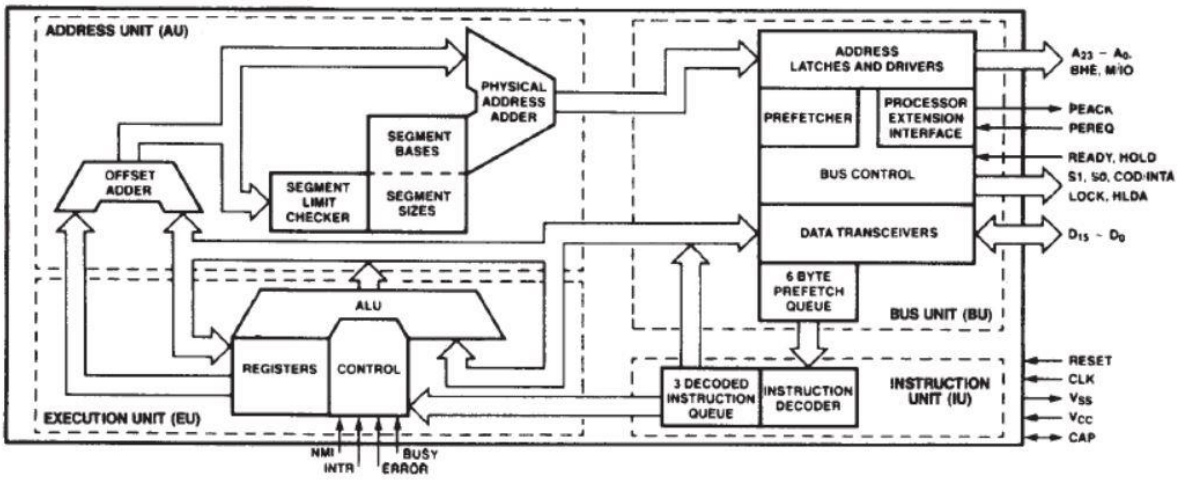
- **Contrast the 8086 and 80186 microprocessors with earlier Intel microprocessors**

Outcomes

- **To get exposed to advance RSIC processors and design ARM microcontroller based systems**

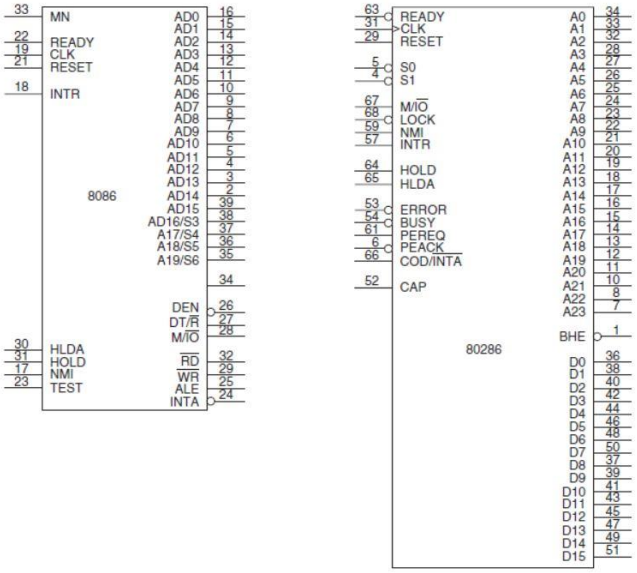
INTRODUCTION TO THE 80286

The 80286 microprocessor is an advanced version of the 8086 microprocessor that was designed for multiuser and multitasking environments. The 80286 addresses 16M bytes of physical memory and 1G bytes of virtual memory by using its memory-management system. This section of the text introduces the 80286 microprocessor, which finds use in earlier AT-style personal computers that once pervaded the computer market and still find some applications. The 80286 is basically an 8086 that is optimized to execute instructions in fewer clocking periods than the 8086. The 80286 is also an enhanced version of the 8086 because it contains a memory manager. At this time, the 80286 no longer has a place in the personal computer system, but it does find applications in control systems as an embedded controller.



As a careful examination of the block diagram reveals, address pins A_{23}–A_0, BUSY, CAP, ERROR PEREQ and are new or additional pins that do not appear on the 8086 microprocessor. The BUSY ERROR PEREQ and PEACK signals are used with the microprocessor extension or coprocessor, of which the 80287 is an example. (Note that the TEST pin is now referred to as the BUSY pin.) The address bus is now 24 bits wide to accommodate the 16M bytes of physical memory. The CAP pin is connected to a 0.047 μF, 20% capacitor that acts as a 12 V filter and connects to ground. The pin-outs of the 8086 and 80286 are illustrated in Figure 16–30 for comparative purposes. Note that the 80286 does not contain a multiplexed address/data bus.

In 80286 operates in both the real and protected modes. In the real mode, the 80286 addresses a 1M-byte memory address space and is virtually identical to the 8086. In the protected mode, the 80286 addresses a 16M-byte memory space.The basic 80286 microprocessor-based system. Notice that the clock is provided by the 82284 clock generator (similar to the 8284A) and the system control signals are provided by the 82288 system bus controller (similar to the 8288). Also, note the absence of the latch circuits used to demultiplex the 8086 address/data bus.



Additional Instructions

The 80286 has even more instructions than its predecessors. These extra instructions control the virtual memory system through the memory manager of the 80286. Table 16–9 lists the additional 80286 instructions with a comment about the purpose of each instruction. These instructions are the only new instructions added to the 80286. Note that the 80286 contains the new instructions added to the 80186/80188 such as INS, OUTS, BOUND, ENTER, LEAVE, PUSHA, POPA, and the immediate multiplication and immediate shift and rotate counts.

CLTS The **clear task-switched flag** (CLTS) instruction clears the TS (task-switched) flag bit to a logic 0. If the TS flag bit is a logic 1 and the 80287 numeric coprocessor is used by the task, an interrupt occurs (vector type 7). This allows the function of the coprocessor to be emulated with software. The CLTS instruction is used in a system and is considered a privileged instruction because it can be executed only in the protected mode at privilege level 0. There is no set TS flag instruction; this is accomplished by writing a logic 1 to bit position 3 (TS) of the machine status word (MSW) by using the LMSW instruction.

CLTS The **clear task-switched flag** (CLTS) instruction clears the TS (task-switched) flag bit to a logic 0. If the TS flag bit is a logic 1 and the 80287 numeric coprocessor is used by the task, an interrupt occurs (vector type 7). This allows the function of the coprocessor to be emulated with software. The CLTS instruction is used in a system and is considered a privileged instruction because it can be executed only in the protected mode at privilege level 0. There is no set TS flag instruction; this is accomplished by writing a logic 1 to bit position 3 (TS) of the machine status word (MSW) by using the LMSW instruction.

LAR The **load access rights** (LAR) instruction reads the segment descriptor and places a copy of the access rights byte into a 16-bit register. An example is the LAR AX,BX instruction that loads AX with the access rights byte from the descriptor selected by the selector value found in BX. This instruction is used to get the access rights so that it can be checked before a program uses the segment of memory described by the descriptor.

LSL The **load segment limit** (LSL) instruction loads a user-specified register with the segment limit. For example, the LSL AX,BX instruction loads AX with the limit of the segment described by the descriptor selected by the selector in BX. This instruction is used to test the limit of a segment.

ARPL The **adjust requested privilege level** (ARPL) instruction is used to test a selector so that the privilege level of the requested selector is not violated. An example is ARPL AX,CX: AX contains the requested privilege level and CX contains the selector value to be used to access a descriptor. If the requested privilege level is of a lower priority than the descriptor under test, the zero flag is set. This may require that a program adjust the requested privilege level or indicate a privilege violation.

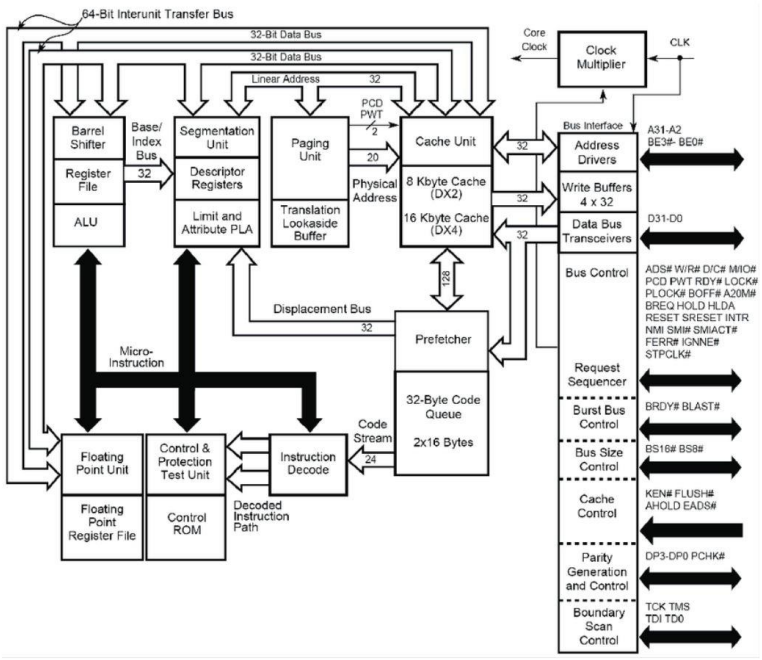
VERR The **verify for read access** (VERR) instruction verifies that a segment can be read. Recall from Chapter 1 that a code segment can be read-protected. If the code segment can be read, the zero flag bit is set. The VERR AX instruction tests the descriptor selected by the AX register.

A virtual memory machine is a machine that maps a larger memory space (1G bytes for the 80286) into a much smaller physical memory space (l6M bytes for the 80286), which allows a very large system to execute in smaller physical memory systems. This is accomplished by spooling the data and programs between the fixed disk memory system and the physical memory. Addressing a 1G-byte memory system is accomplished by the descriptors in the 80286 microprocessor. Each 80286 descriptor describes a 64K-byte memory segment and the 80286 allows 16K descriptors. This (64K *16K) allows a maximum of 1G bytes of memory to be described for the system.

Advanced coprocessor Architectures- 486

The 80486 microprocessor is a highly integrated device, containing well over 1.2 million transistors. Located within this device circuit are a memory-management unit (MMU), a complete numeric coprocessor that is compatible with the 80387, a high-speed level 1 cache memory that contains 8K bytes of space, and a full 32-bit microprocessor that is upward-compatible with the 80386 microprocessor. The 80486 is currently available as a 25 MHz, 33 MHz, 50 MHz, 66 MHz, or 100 MHz device. Note that the 66 MHz version is double-clocked and the 100 MHz version is triple-clocked. In 1990, Intel demonstrated a 100 MHz version (not double-clocked) of the 80486 for Computer Design magazine, but it has yet to be released. Advanced Micro Devices (AMD) has produced a 40 MHz version that is also available in an 80 MHz (double-clocked) and a 120 MHz (triple-clocked) form. The 80486 is available as an 80486DX or an 80486SX. The only difference between these devices is that the 80486SX does not contain the numeric coprocessor, which reduces its price. The 80487SX numeric coprocessor is available as a separate component for the 80486SX microprocessor.

The architecture of the 80486DX is almost identical to the 80386. Added to the 80386 architecture inside the 80486DX is a math coprocessor and an 8K-byte level 1 cache memory. The 80486SX is almost identical to an 80386 with an 8K-byte cache, but no numeric coprocessor. The only new flag bit is the AC (alignment check), used to indicate that the microprocessor has accessed a word at an odd address or a double word stored at a non-double word boundary. Efficient software and execution require that data be stored at word or double word boundaries.

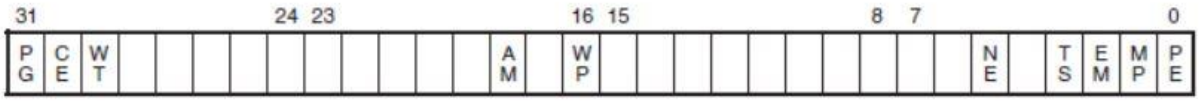


The memory system for the 80486 is identical to the 80386 microprocessor. The 80486 contains 4G bytes of memory, beginning at location 00000000H and ending at location FFFFFFFFH. The major change to the memory system is internal to the 80486 in the form of an 8K-byte cache memory, which speeds the execution of instructions and the acquisition of data. Another addition is the parity checker/generator built into the 80486 microprocessor. commonly used in a program. About the only way that these efficient instructions are slowed is when the microprocessor must fill a line in the cache. Data are also stored in the cache, but it has less of an impact on the execution speed of a program because data are not referenced repeatedly as many portions of a program are. Control register 0 (CR0) is used to control the cache with two new control bits not present in the 80386 microprocessor.

The CD (cache disable) and NW (noncache write-through) bits are new to the 80486 and are used to control the 8K-byte cache. If the CD bit is a logic 1, all cache operations are inhibited. This setting is used only for debugging software and normally remains cleared. The NW bit is used to inhibit cache write-through operations. As with CD, cache write-through is inhibited only for testing. For normal program operation, CD = 0 and NW = 0. Because the cache is new to the 80486 microprocessor and the cache is filled by using burst cycles not present on the 80386, some detail is required to understand bus-filling cycles.

When a bus line is filled, the 80486 must acquire four 32-bit numbers from the memory system to fill a line in the cache. Filling is accomplished with a burst cycle. The burst cycle is a special memory where four 32-bit numbers are fetched from the memory system in five clocking periods. This assumes that the speed of the memory is sufficient and that no wait states are required. If the clock frequency of the 80486 is 33 MHz, we can fill a cache line in 167 ns, which is very efficient considering that a normal, nonburst 32-bit memory read operation requires two clocking periods.

Control register zero (CR0) for the 80486 microprocessor.

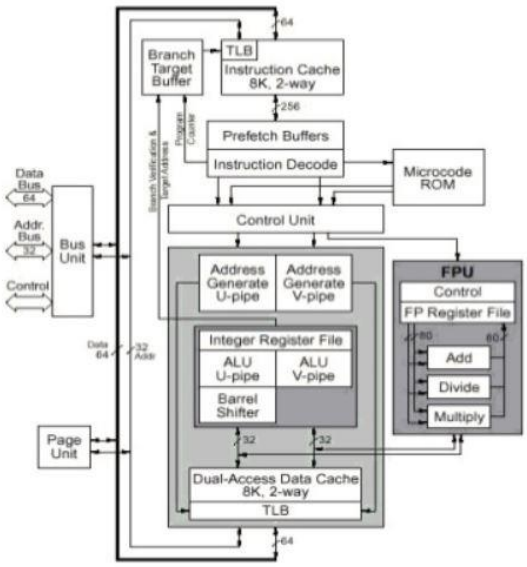


The 80486 33 MHz, 66 MHz, and 100 MHz processors all access bus data at a 33 MHz rate. In other words, the microprocessor may operate at 100 MHz, but the system bus operates at 33 MHz. Notice that the nonburst access timing for the 33 MHz system bus allows 60 ns - 24 ns = 36 ns. It is obvious that wait states are required for operation with standard DRAM memory device. Access time using a 20 MHz version of the 80486 for the second and subsequent double words is 50 ns - 28 ns - 5 ns, or 17 ns, assuming no delays in the system. To use burst mode transfers, we need high-speed memory. Because DRAM memory access times are 40 ns at best, we are forced to use SRAM for burst cycle transfers. The 33 MHz system allows an access time of 30 ns - 19 ns - 5 ns, or 6 ns for the second and subsequent bytes. If an external counter is used in place of address bits A2 and A3, the 19 ns can be eliminated and the access time becomes 30 ns - 5 ns, or 25 ns, which is enough time for even the slowest SRAM connected to the system as a cache.

The PWT controls how the cache functions for a write operation of the external cache memory; it does not control writing to the internal cache. The logic level of this bit is found on the PWT pin of the 80486 microprocessor. Externally, it can be used to dictate the write-through policy of the external cache. The PCD bit controls the on-chip cache. If the PCD = 0, the on-chip cache is enabled for the current page of memory. Note that 80386 page table entries place a logic 0 in the PCD bit position, enabling caching. If PCD = 1, the on-chip cache is disabled. Caching is disabled, regardless of the condition of , CD, and NW.

Pentium

Before the Pentium or any other microprocessor can be used in a system, the function of each pin must be understood. This section of the chapter details the operation of each pin, along with the external memory system and I/O structures of the Pentium microprocessor. As with earlier versions of the Intel family of microprocessors, the early versions of the Pentium require a single +5.0 V power supply for operation. The power supply current averages 3.3 A for the 66 MHz version of the Pentium, and 2.91 A for the 60 MHz version. Because these currents are significant, so are the power dissipations of these microprocessors: 13 W for the 66 MHz version and 11.9 W for the 60 MHz version. The current versions of the Pentium, 90 MHz and above, use a 3.3 V power supply with reduced current consumption. At present, a good heat sink with considerable airflow is required to keep the Pentium cool. The Pentium contains multiple VCC and VSS connections that must all be connected to +5.0 V or +3.3 V and ground for proper operation. Some of the pins are labeled N/C (no connection) and must not be connected. The latest versions of the Pentium have been improved to reduce the power dissipation. For example, the 233 MHz Pentium requires 3.4 A or current, which is only slightly more than the 3.3 A required by the early 66 MHz version.



Each Pentium output pin is capable of providing 4.0 mA of current at a logic 0 level and 2.0 mA at a logic 1 level. This represents an increase in drive current, compared to the 2.0 mA available on earlier 8086, 8088, and 80286 output pins. Each input pin represents a small load requiring only 15 µA of current. In some systems, except the smallest, these current levels require bus buffers.

The memory system for the Pentium microprocessor is 4G bytes in size, just as in the 80386DX and 80486 microprocessors. The difference lies in the width of the memory data bus. The Pentium uses a 64-bit data bus to address memory organized in eight banks that each contain 512M bytes of data. See Figure 18–2 for the organization of the Pentium physical memory system. The Pentium memory system is divided into eight banks where each bank stores byte-wide data with a parity bit. The Pentium, like the 80486, employs internal parity generation and checking logic for the memory system's data bus information. (Note that most Pentium systems do not use parity checks, because ECC is available.) The 64-bit-wide memory is important to double-precision floating-point data. Recall that a double-precision floating-point number is 64 bits wide. Because of the change to a 64-bit-wide data bus, the Pentium is able to retrieve floating- point data with one read cycle, instead of two as in the 80486. This causes the Pentium to function at a higher throughput than an 80486. As with earlier 32-bit Intel microprocessors, the memory system is numbered in bytes, from byte 00000000H to byte FFFFFFFFH. Memory selection is accomplished with the bank enable signals (–). These separate memory banks allow the Pentium to access any single byte, word, doubleword, or quadword with one memory transfer cycle. As with earlier memory selection logic, eight separate write strobes are generated for writing to the memory system.

The Pentium processor has two primary operating modes -

Protected Mode - In this mode all instructions and architectural features are available, providing the highest performance and capability. This is the recommended mode that all new applications and operating systems should target.

Real-Address Mode - This mode provides the programming environment of the Intel 8086 processor, with a few extensions. Reset initialization places the processor in real mode where, with a single instruction, it can switch to protected mode.

The Pentium's basic integer pipeline is five stages long, with the stages broken down as follows:

Pre-fetch/Fetch: Instructions are fetched from the instruction cache and aligned in pre-fetch buffers for decoding.
Decode1: Instructions are decoded into the Pentium's internal instruction format. Branch prediction also takes place at this stage.
Decode2: Same as above, and microcode ROM kicks in here, if necessary. Also, address computations take place at this stage.
Execute: The integer hardware executes the instruction.
Write-back: The results of the computation are written back to the register file.

COMPARISON OF RISC AND CISC

RISC is stand for Reduced Instruction Set Computer. Nowadays mostly Mobile Phones Based on RISC architecture Like MIPS and ARM etc. RISC has simple and small Instruction. RISC chips Comes around the mid 80's because the reaction of CISC chips. The philosophy behind that almost no one use complex instructions and mostly people uses compilers which never use complex instructions. So for Apple uses RISC chips [14-20]. So therefore simple and faster instructions are better than large complex and slower (CISC) instructions. However, RISC required more instruction to complete a task than CISC. An advantage of RISC is that because it more simple instructions. RISC chips require less transistors which makes easier to design and cheaper to produce. So now it easier to write powerful optimal compilers since fewer instructions exists.

Properties of CISC:

1. Some simple and very complex instructions
2. In CISC instructions take more than 1 clock per Cycle to execute
3. Variable size instructions
4. No pipelining
5. Few registers
6. Not a load and store machine
7. For Compilation not so good in term of speed
8. Emphasis of Hardware
9. Transistors are used for storing complex Instructions

Properties of RISC:

1. Small and simple instructions
2. In RISC Instructions are execute in one clock cycle per Instructions
3. All instructions have the same length
4. Load and Store architecture implemented due to the desired single-cycle operation
5. Have Pipelining
6. More register than CISC
7. Optimal compilation speed as compared to CISC

8. Emphasis on software
9. Compare to CISC a RISC Spends more transistors on memory registers

Advantages of RISC:

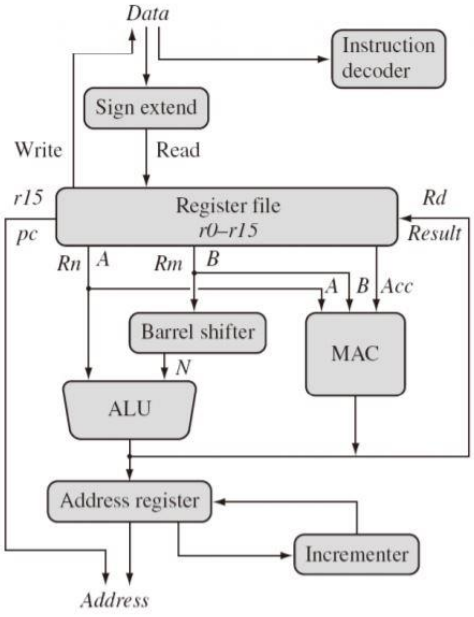
Implementation with simple instructions provides many advantages over implementing as compared to CISC Processors. Simple instruction set allow for pipeline superscalar designing RISC processor often achieved two to four times performance of CISC processors using [21-27] comparable semiconductor technology and similar clock rates. Simple hardware. Because instructions set of a (RISC) processor is so simple, it uses up much less chips spaces and extra functions i.e. memory management unit or floating point arithmetic units, can also be placed on the similar chip. Smaller chips allows a semiconductor manufacturers to placed more parts on single silicon wafer which can lower per chips cost dramatically and have short design cycles. Since RISC processors are simpler than corresponding CISC processors they can be design more quickly and take advantage of other technological [28-33] developments sooner than corresponds CISC design leading to great leaps in performance between generations.

Advantages of CISC:

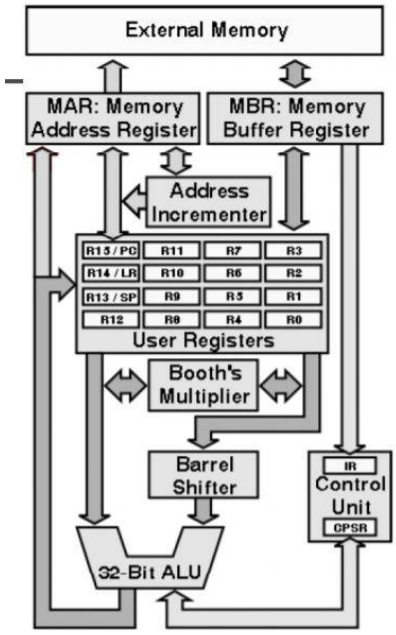
At the time of their initial development CISC machines use technologies to optimize the performance of a computer. Microprogramming is easy as assembly Programming language to implement and less expensive than hardwiring a control unit. The ease of micro coding newly instructions allows designers to make (CISC) machines upwardly compatible new computer run the same programs as early computers because the new computers would contained a superset of instructions of earlier computers. As each instruction became more capable less instruction used to implement the given task. This made efficient uses of the relative slow main

ARM architecture

The microcontroller market is vast, with more than 20 billion devices per year estimated to be shipped in 2010. A bewildering array of vendors, devices, and architectures is competing in this market. The requirement for higher performance microcontrollers has been driven globally by the industry's changing needs; for example, microcontrollers are required to handle more work without increasing a product's frequency or power. In addition, microcontrollers are becoming increasingly connected, whether by Universal Serial Bus (USB), Ethernet, or wireless radio, and hence, the processing needed to support these communication channels and advanced peripherals are growing. Similarly, general application complexity is on the increase, driven by more sophisticated user interfaces, multimedia requirements, system speed, and convergence of functionalities.



Microcontrollers based on the Cortex-M3 processor already compete head-on with devices based on a wide variety of other architectures. Designers are increasingly looking at reducing the system cost, as opposed to the traditional device cost. As such, organizations are implementing device aggregation, whereby a single, more powerful device can potentially replace three or four traditional 8-bit devices. Other cost savings can be achieved by improving the amount of code reuse across all systems. Because Cortex-M3 processor-based microcontrollers can be easily programmed using the C language and are based on a well-established architecture, application code can be ported and reused easily, reducing development time and testing costs.



It is worthwhile highlighting that the Cortex-M3 processor is not the first ARM processor to be used to create generic microcontrollers. The venerable ARM7 processor has been very successful in this market,

with partners such as NXP (Philips), Texas Instruments, Atmel, OKI, and many other vendors delivering robust 32-bit Microcontroller Units (MCUs). The ARM7 is the most widely used 32-bit embedded processor in history, with over 1 billion processors produced each year in a huge variety of electronic products, from mobile phones to cars.

The Cortex-M3 processor builds on the success of the ARM7 processor to deliver devices that are significantly easier to program and debug and yet deliver a higher processing capability. Additionally, the Cortex-M3 processor introduces a number of features and technologies that meet the specific requirements of the microcontroller applications, such as non maskable interrupts for critical tasks, highly deterministic nested vector interrupts, atomic bit manipulation, and an optional Memory Protection Unit (MPU). These factors make the Cortex-M3 processor attractive to existing ARM processor users as well as many new users considering use of 32-bit MCUs in their products.

Cortex-M3 Processor Applications

With its high performance and high code density and small silicon footprint, the Cortex-M3 processor is ideal for a wide variety of applications:

• Low-cost microcontrollers: The Cortex-M3 processor is ideally suited for low-cost microcontrollers, which are commonly used in consumer products, from toys to electrical appliances. It is a highly competitive market due to the many well-known 8-bit and 16-bit microcontroller products on the market. Its lower power, high performance, and ease-of-use advantages enable embedded developers to migrate to 32-bit systems and develop products with the ARM architecture.

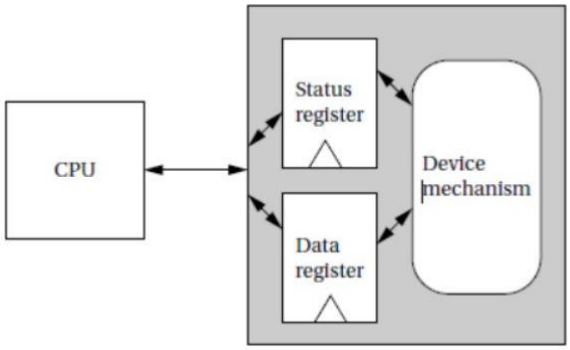
• Automotive: Another ideal application for the Cortex-M3 processor is in the automotive industry. The Cortex-M3 processor has very high-performance efficiency and low interrupt latency, allowing it to be used in real-time systems. The Cortex-M3 processor supports up to 240 external vectored interrupts, with a built-in interrupt controller with nested interrupt supports and an optional MPU, making it ideal for highly integrated and cost-sensitive automotive applications.

•Data communications: The processor's low power and high efficiency, coupled with instructions in Thumb-2 for bit-field manipulation, make the Cortex-M3 ideal for many communications applications, such as Bluetooth and ZigBee.

• Industrial control: In industrial control applications, simplicity, fast response, and reliability are key factors. Again, the Cortex-M3 processor's interrupt feature, low interrupt latency, and enhanced fault-handling features make it a strong candidate in this area.

• Consumer products: In many consumer products, a high-performance microprocessor (or several of them) is used. The Cortex-M3 processor, being a small processor, is highly efficient and low in power and supports an MPU enabling complex software to execute while providing robust memory protection.

CPU: Programming input and output:



The basic techniques for I/O programming can be understood relatively independent of the instruction set. In this section, we cover the basics of I/O programming and place them in the contexts of both the ARM and C55x.We begin by discussing the basic characteristics of I/O devices so that we can understand the requirements they place on programs that communicate with them.

1. Input and Output Devices:

Input and output devices usually have some analog or non electronic component for instance, a disk drive has a rotating disk and analog read/write electronics. But the digital logic in the device that is most closely connected to the CPU very strongly resembles the logic you would expect in any computer system.

The interface between the CPU and the device's internals (e.g.,the rotating disk and read/write electronics in a disk drive) is a set of registers. The CPU talks to the device by reading and writing the registers.

Data registers hold values that are treated as data by the device, such as the data read or written by a disk. Status registers provide information about the device's operation, such as whether the current transaction has completed. Some registers may be read-only, such as a status register that indicates when the device is done, while others may be readable or writable.

2. Input and Output Primitives:

Microprocessors can provide programming support for input and output in two ways: I/O instructions and memory-mapped I/O Some architectures, such as the Intel x86, provide special instructions (in and out in the case of the Intel x86) for input and output. These instructions provide a separate address space for I/O devices .But the most common way to implement I/O is by memory mapping even CPUs that provide I/O instructions can also implement memory-mapped I/O.As the name implies, memory-mapped I/O provides addresses for the registers in each I/O device. Programs use the CPU's normal read and write instructions to communicate with the devices.

3. Busy-Wait I/O:

The most basic way to use devices in a program is busy-wait I/O. Devices are typically slower than the CPU and may require many cycles to complete an operation. If the CPU is performing multiple operations on a single device, such as writing several characters to an output device, then it must wait for one operation

to complete before starting the next one. (If we try to start writing the second character before the device has finished with the first one, for example, the device will probably never print the first character.) Asking an I/O device whether it is finished by reading its status register is often called polling.

Supervisor Mode:

As will become clearer in later chapters, complex systems are often implemented as several programs that communicate with each other. These programs may run under the command of an operating system. It may be desirable to provide hardware checks to ensure that the programs do not interfere with each other—for example, by erroneously writing into a segment of memory used by another program. Software debugging is important but can leave some problems in a running system; hardware checks ensure an additional level of safety.

In such cases it is often useful to have a supervisor mode provided by the CPU. Normal programs run in user mode. The supervisor mode has privileges that user modes do not. Control of the memory management unit (MMU) is typically reserved for supervisor mode to avoid the obvious problems that could occur when program bugs cause inadvertent changes in the memory management registers.

Not all CPUs have supervisor modes. Many DSPs, including the C55x, do not provide supervisor modes. The ARM, however, does have such a mode. The ARM instruction that puts the CPU in supervisor mode is called SWI:

```
SWI CODE_1
```

It can, of course, be executed conditionally, as with any ARM instruction. SWI causes the CPU to go into supervisor mode and sets the PC to 0x08.The argument to SWI is a 24-bit immediate value that is passed on to the supervisor mode code; it allows the program to request various services from the supervisor mode.

In supervisor mode, the bottom 5 bits of the CPSR are all set to 1 to indicate that the CPU is in supervisor mode. The old value of the CPSR just before the SWI is stored in a register called the saved program status register (SPSR). There are in fact several SPSRs for different modes; the supervisor mode SPSR is referred to as SPSR_svc. To return from supervisor mode, the supervisor restores the PC from register r14 and restores the CPSR from the SPSR_svc.

Exceptions:

An exception is an internally detected error. A simple example is division by zero. One way to handle this problem would be to check every divisor before division to be sure it is not zero, but this would both substantially increase the size of numerical programs and cost a great deal of CPU time evaluating the divisor's value.

The CPU can more efficiently check the divisor's value during execution. Since the time at which a zero divisor will be found is not known in advance, this event is similar to an interrupt except that it is generated inside the CPU. The exception mechanism provides a way for the program to react to such unexpected events.

Just as interrupts can be seen as an extension of the subroutine mechanism, exceptions are generally implemented as a variation of an interrupt. Since both deal with changes in the flow of control of a program, it makes sense to use similar mechanisms. However, exceptions are generated internally.

Exceptions in general require both prioritization and vectoring. Exceptions must be prioritized because a single operation may generate more than one exception for example, an illegal operand and an illegal memory access.The priority of exceptions is usually fixed by the CPU architecture. Vectoring provides a

way for the user to specify the handler for the exception condition. The vector number for an exception is usually predefined by the architecture; it is used to index into a table of exception handlers.

Traps:

A trap, also known as a software interrupt, is an instruction that explicitly generates an exception condition. The most common use of a trap is to enter supervisor mode.

The entry into supervisor mode must be controlled to maintain security—if the interface between user and supervisor mode is improperly designed, a user program may be able to sneak code into the supervisor mode that could be executed to perform harmful operations.

The ARM provides the SWI interrupt for software interrupts. This instruction causes the CPU to enter supervisor mode. An opcode is embedded in the instruction that can be read by the handler.

CO-PROCESSORS:

CPU architects often want to provide flexibility in what features are implemented in the CPU. One way to provide such flexibility at the instruction set level is to allow co-processors, which are attached to the CPU and implement some of the instructions. For example, floating-point arithmetic was introduced into the Intel architecture by providing separate chips that implemented the floating-point instructions.

To support co-processors, certain opcodes must be reserved in the instruction set for co-processor operations. Because it executes instructions, a co-processor must be tightly coupled to the CPU. When the CPU receives a co-processor instruction, the CPU must activate the co-processor and pass it the relevant instruction. Co-processor instructions can load and store co-processor registers or can perform internal operations. The CPU can suspend execution to wait for the co-processor instruction to finish; it can also take a more superscalar approach and continue executing instructions while waiting for the co-processor to finish.

A CPU may, of course, receive co-processor instructions even when there is no coprocessor attached. Most architectures use illegal instruction traps to handle these situations. The trap handler can detect the co-processor instruction and, for example, execute it in software on the main CPU. Emulating co-processor instructions in software is slower but provides compatibility.

The ARM architecture provides support for up to 16 co-processors. Co-processors are able to perform load and store operations on their own registers. They can also move data between the co-processor registers and main ARM registers.

An example ARM co-processor is the floating-point unit. The unit occupies two co-processor units in the ARM architecture, numbered 1 and 2, but it appears as a single unit to the programmer. It provides eight 80-bit floating-point data registers, floating-point status registers, and an optional floating-point status register.

MEMORY SYSTEM MECHANISMS:

Modern microprocessors do more than just read and write a monolithic memory. Architectural features improve both the speed and capacity of memory systems. Microprocessor clock rates are increasing at a faster rate than memory speeds, such that memories are falling further and further behind microprocessors every day. As a result, computer architects resort to caches to increase the average performance of the memory system.

Although memory capacity is increasing steadily, program sizes are increasing as well, and designers may not be willing to pay for all the memory demanded by an application. Modern microprocessor units (MMUs) perform address translations that provide a larger virtual memory space in a small physical memory. In this section, we review both caches and MMUs.

1. Caches:

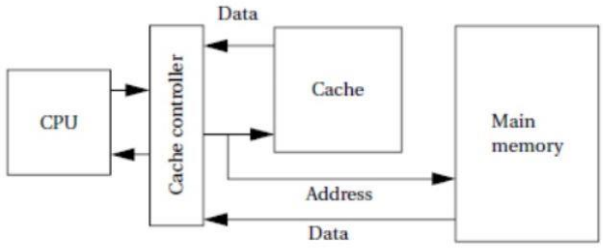
Caches are widely used to speed up memory system performance. Many microprocessor architectures include caches as part of their definition.

The cache speeds up average memory access time when properly used. It increases the variability of memory access times accesses in the cache will be fast, while access to locations not cached will be slow. This variability in performance makes it especially important to understand how caches work so that we can better understand how to predict cache performance and factor variabilities into system design.

A cache is a small, fast memory that holds copies of some of the contents of main memory. Because the cache is fast, it provides higher-speed access for the CPU; but since it is small, not all requests can be satisfied by the cache, forcing the system to wait for the slower main memory. Caching makes sense when the CPU is using only a relatively small set of memory locations at any one time; the set of active locations is often called the working set.

The cache controller sends a memory request to the cache and main memory. If the requested location is in the cache, the cache controller forwards the location's contents to the CPU and aborts the main memory request; this condition is known as a cache hit.

If the location is not in the cache, the controller waits for the value from main memory and forwards it to the CPU; this situation is known as a cache miss.



We can classify cache misses into several types depending on the situation that generated them:

Even before we consider ways to implement caches, we can write some basic formulas for memory system performance. Let h be the hit rate, the probability that a given memory location is in the cache. It follows that 1_h is the miss rate, or the probability that the location is not in the cache. Then we can compute the average memory access time as

where tcache is the access time of the cache and tmain is the main memory access time. The memory access times are basic parameters available from the memory manufacturer.

The hit rate depends on the program being executed and the cache organization, and is typically measured using simulators. The best-case memory access time (ignoring cache controller overhead) is tcache, while the worst-case access time is tmain. Given that tmain is typically 50–60 ns for DRAM, while tcache is at most a few nanoseconds, the spread between worst-case and best-case memory delays is substantial.

where tcache is the access time of the cache and tmain is the main memory access time. The memory access times are basic parameters available from the memory manufacturer. The hit rate depends on the program being executed and the cache organization, and is typically measured using simulators.
The best-case memory access time (ignoring cache controller overhead) is tcache, while the worst-case access time is tmain. Given that tmain is typically 50–60 ns for DRAM,while tcache is at most a few nanoseconds, the spread between worst-case and best-case memory delays is substantial.

Modern CPUs may use multiple levels of cache as shown in Figure 1.20. The first-level cache (commonly known as L1 cache) is closest to the CPU, the second-level cache (L2 cache) feeds the first-level cache, and so on.The second-level cache is much larger but is also slower. If h1 is the first-level hit rate and h2 is the rate at which access hit the second-level cache but not the first-level cache, then the average access time for a two-level cache system,As the program's working set changes, we expect locations to be removed from the cache to make way for new locations. When set-associative caches are used, we have to think about what happens when we throw out a value from the cache to make room for a new value.

We do not have this problem in direct-mapped caches because every location maps onto a unique block, but in a set-associative cache we must decide which set will have its block thrown out to make way for the new block.

One possible replacement policy is least recently used (LRU), that is, throw out the block that has been used farthest in the past. We can add relatively small amounts of hardware to the cache to keep track of the time since the last access for each block. Another policy is random replacement, which requires even less hardware to implement.

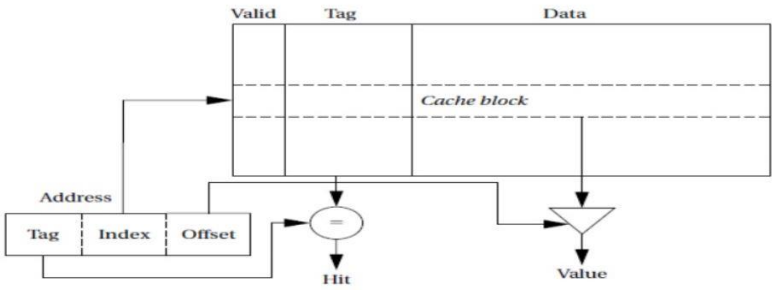
The simplest way to implement a cache is a direct-mapped cache, as shown in Figure 1.20. The cache consists of cache blocks, each of which includes a tag to show which memory location is represented by this block, a data field holding the contents of that memory, and a valid tag to show whether the contents of this cache block are valid. An address is divided into three sections.

The index is used to select which cache block to check. The tag is compared against the tag value in the block selected by the index. If the address tag matches the tag value in the block, that block includes the desired memory location.

If the length of the data field is longer than the minimum addressable unit, then the lowest bits of the address are used as an offset to select the required value from the data field. Given the structure of the cache, there is only one block that must be checked to see whether a location is in the cache—the index uniquely determines that block. If the access is a hit, the data value is read from the cache.

Writes are slightly more complicated than reads because we have to update main memory as well as the cache. There are several methods by which we can do this. The simplest scheme is known as write-through—every write changes both the cache and the corresponding main memory location (usually through a write buffer).

This scheme ensures that the cache and main memory are consistent, but may generate some additional main memory traffic. We can reduce the number of times we write to main memory by using a write-back policy: If we write only when we remove a location from the cache, we eliminate the writes when a location is written several times before it is removed from the cache.



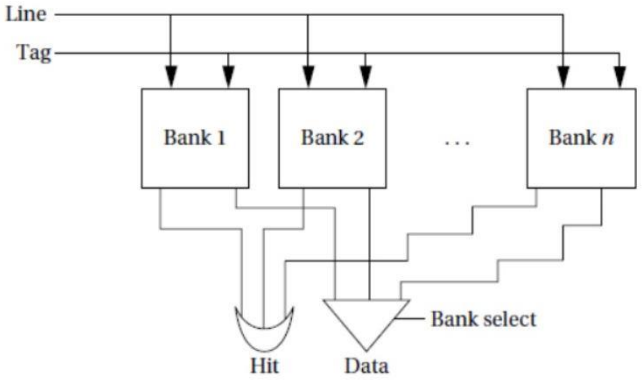
The direct-mapped cache is both fast and relatively low cost, but it does have limits in its caching power due to its simple scheme for mapping the cache onto main memory. Consider a direct-mapped cache with four blocks, in which locations 0, 1, 2, and 3 all map to different blocks. But locations 4, 8, 12…all map to the same block as location 0; locations 1, 5, 9, 13…all map to a single block; and so on. If two popular locations in a program happen to map onto the same block, we will not gain the full benefits of the cache. As seen in Section 5.6, this can create program performance problems.

The limitations of the direct-mapped cache can be reduced by going to the set-associative cache structure shown in Figure 1.21.A set-associative cache is characterized by the number of banks or ways it uses, giving an n-way set-associative cache.

A set is formed by all the blocks (one for each bank) that share the same index. Each set is implemented with a direct-mapped cache. A cache request is broadcast to all banks simultaneously. If any of the sets has the location, the cache reports a hit.

Although memory locations map onto blocks using the same function, there are n separate blocks for each set of locations. Therefore, we can simultaneously cache several locations that happen to map onto the same cache block. The set associative cache structure incurs a little extra overhead and is slightly slower than a direct-mapped cache, but the higher hit rates that it can provide often compensate.

The set-associative cache generally provides higher hit rates than the direct mapped cache because conflicts between a small number of locations can be resolved within the cache. The set-associative cache is somewhat slower, so the CPU designer has to be careful that it doesn't slow down the CPU's cycle time too much. A more important problem with set-associative caches for embedded program.



Design is predictability. Because the time penalty for a cache miss is so severe, we often want to make sure that critical segments of our programs have good behavior in the cache. It is relatively easy to determine when two memory locations will conflict in a direct-mapped cache.

Conflicts in a set-associative cache are more subtle, and so the behavior of a set-associative cache is more difficult to analyze for both humans and programs.

CPU PERFORMANCE:

Now that we have an understanding of the various types of instructions that CPUs can execute, we can move on to a topic particularly important in embedded computing: How fast can the CPU execute instructions? In this section, we consider three factors that can substantially influence program performance: pipelining and caching.

1. Pipelining

Modern CPUs are designed as pipelined machines in which several instructions are executed in parallel. Pipelining greatly increases the efficiency of the CPU. But like any pipeline, a CPU pipeline works best when its contents flow smoothly.Some sequences of instructions can disrupt the flow of information in the pipeline and, temporarily at least, slow down the operation of the CPU.

The ARM7 has a three-stage pipeline:

Fetch the instruction is fetched from memory.

Decode the instruction's opcode and operands are decoded to determine what function to perform.

Execute the decoded instruction is executed.

Each of these operations requires one clock cycle for typical instructions. Thus, a normal instruction requires three clock cycles to completely execute, known as the latency of instruction execution. But since the pipeline has three stages, an instruction is completed in every clock cycle. In other words, the pipeline has a throughput of one instruction per cycle.illustrates the position of instructions in the pipeline during execution using the notation introduced by Hennessy and Patterson [Hen06]. A vertical slice through the timeline shows all instructions in the pipeline at that time. By following an instruction horizontally, we can see the progress of its execution.

The C55x includes a seven-stage pipeline [Tex00B]:

Fetch.

Decode.

Address computes data and branch addresses.

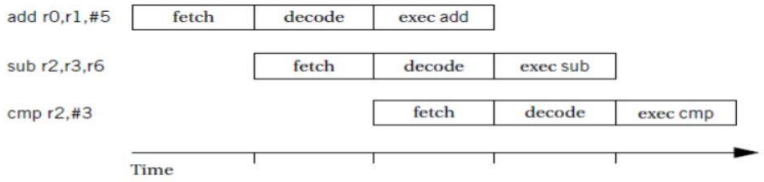
Access 1 reads data.

Access 2 finishes data read.

Read stage puts operands onto internal busses.

Execute performs operations.

RISC machines are designed to keep the pipeline busy. CISC machines may display a wide variation in instruction timing. Pipelined RISC machines typically have more regular timing characteristics most instructions that do not have pipeline hazards display the same latency.



Caching

However, the desired location is not always in the cache since it is considerably smaller than main memory. As a result, caches cause the time required to access memory to vary considerably. The extra time required to access a memory location not in the cache is often called the cache miss penalty.The amount of variation depends on several factors in the system architecture, but a cache miss is often several clock cycles slower than a cache hit. The time required to access a memory location depends on whether the requested location is in the cache. However, as we have seen, a location may not be in the cache for several reasons.

At a compulsory miss, the location has not been referenced before.

At a conflict miss, two particular memory locations are fighting for the same cache line.

At a capacity miss, the program's working set is simply too large for the cache.

The contents of the cache can change considerably over the course of execution of a program. When we have several programs running concurrently on the CPU,

CPU POWER CONSUMPTION:

Power consumption is, in some situations, as important as execution time. In this section we study the characteristics of CPUs that influence power consumption and mechanisms provided by CPUs to control how much power they consume. First, it is important to distinguish between energy and power. Power is, of course, energy consumption per unit time. Heat generation depends on power consumption. Battery life, on the other hand, most directly depends on energy consumption. Generally, we will use the term power as shorthand for energy and power consumption, distinguishing between them only when necessary.

The high-level power consumption characteristics of CPUs and other system components are derived from the circuits used to build those components. Today, virtually all digital systems are built with complementary metal oxide semiconductor (CMOS) circuitry. The detailed circuit characteristics are best left to a study of VLSI design [Wol08], but the basic sources of CMOS power consumption are easily identified and briefly described below.

Voltage drops: The dynamic power consumption of a CMOS circuit is proportional to the square of the power supply voltage (V2). Therefore, by reducing the power supply voltage to the lowest level that provides the required performance, we can significantly reduce power consumption. We also may be able to add parallel hardware and even further reduce the power supply voltage while maintaining required performance.

Toggling: A CMOS circuit uses most of its power when it is changing its output value. This provides two ways to reduce power consumption. By reducing the speed at which the circuit operates, we can reduce its power consumption (although not the total energy required for the operation, since the result is available later).We can actually reduce energy consumption by eliminating unnecessary changes to the inputs of a CMOS circuit—eliminating unnecessary glitches at the circuit outputs eliminates unnecessary power consumption.