

Chapter 1

Introduction

A microprocessor is one of the most central parts of a modern personal computer or, in fact, any advanced computer device. It integrates the functions of a central processing unit, the portion of a computer responsible for carrying out programmed instructions, onto a single integrated circuit that couples the important thinking devices of the machine with the electrical infrastructure needed to support them. Microprocessor design is able to incorporate a tremendous amount of processing power in a very small space. Perhaps more than any other component of the modern computer, the microprocessor has a long and storied history, and an almost mythological status thanks to its great importance. Each step of the way to modern micro-processing has spurred more triumphs, innovations, and competition.

4-bit Microprocessors

The first microprocessor was introduced in 1971 by Intel Corp. It was named Intel 4004 as it was a 4 bit processor. It was a processor on a single chip. It could perform simple arithmetic and logic operations such as addition, subtraction, Boolean AND and Boolean OR. It had a control unit capable of performing control functions like fetching an instruction from memory, decoding it, and generating control pulses to execute it. It was able to operate on 4 bits of data at a time. This first microprocessor was quite a success in industry. Soon other microprocessors were also introduced. Intel introduced the enhanced version of 4004, the 4040. Some other 4 bit processors are International's PPS4 and Toshiba T3472.

8-bit Microprocessors

The first 8 bit microprocessor which could perform arithmetic and logic operations on 8 bit words was introduced in 1973 again by Intel. This was Intel 8008 and was later followed by an improved version, Intel 8088. Some other 8 bit processors are Zilog-80 and Motorola M6800.

16-bit Microprocessors

The 8-bit processors were followed by 16 bit processors. They are Intel 8086 and 80286.

32-bit Microprocessors

The 32 bit microprocessors were introduced by several companies but the most popular one is Intel 80386.

Pentium Series

Instead of 80586, Intel came out with a new processor namely Pentium processor. Its performance is closer to RISC performance. Pentium was followed by Pentium Pro CPU. Pentium Pro allows allow multiple CPU s in a single system in order to achieve multiprocessing. The MMX extension was added to Pentium Pro and the result was Pentium II. The low cost version of Pentium II is celeron.

The Pentium III provided high performance floating point operations for certain types of computations by using the SIMD extensions to the instruction set. These new instructions makes the Pentium III faster than high-end RISC CPU s.

Interestingly Pentium IV could not execute code faster than the Pentium III when running at the same clock frequency. So Pentium IV had to speed up by executing at a much higher clock frequency.

The concept and architecture of a microcomputer

A microcomputer is a computer built on the basis of a microprocessor i.e. a processor implemented as an integrated circuit. Since all processors are now produced in the form of integrated circuits, we can say that all computers are microcomputers. The general method for constructing microcomputers consists in connecting to the microprocessor busses additional sub-systems such as memories and peripheral device controllers (input/output units).

The basic block diagram of a simple microcomputer is shown in the figure below. We can see there a microprocessor with three its busses going out: data bus, address bus and control bus. To these busses, the following devices are connected: operational memory composed of RAM (Random Access Memory)and ROM (Read Only Memory) memories, as well as input/output units to which peripheral devices are connected.

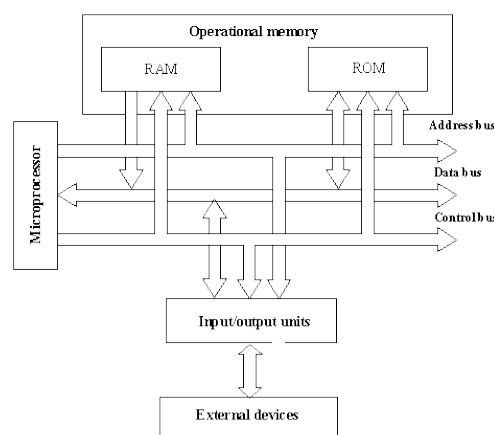


Image 1 Simple microcomputer

A more developed block diagram of a microcomputer is shown in the figure below. Besides RAM and ROM memories, more input/output units are connected to the microprocessor.

These input/output units include:

- ◆ parallel input/output controller in short - parallel I/O, parallel interface
- ◆ serial input/output controller in short - serial I/O, serial interface
- ◆ interrupt controller (handler)
- ◆ timer/counter controller
- ◆ Direct Memory Access (DMA) controller

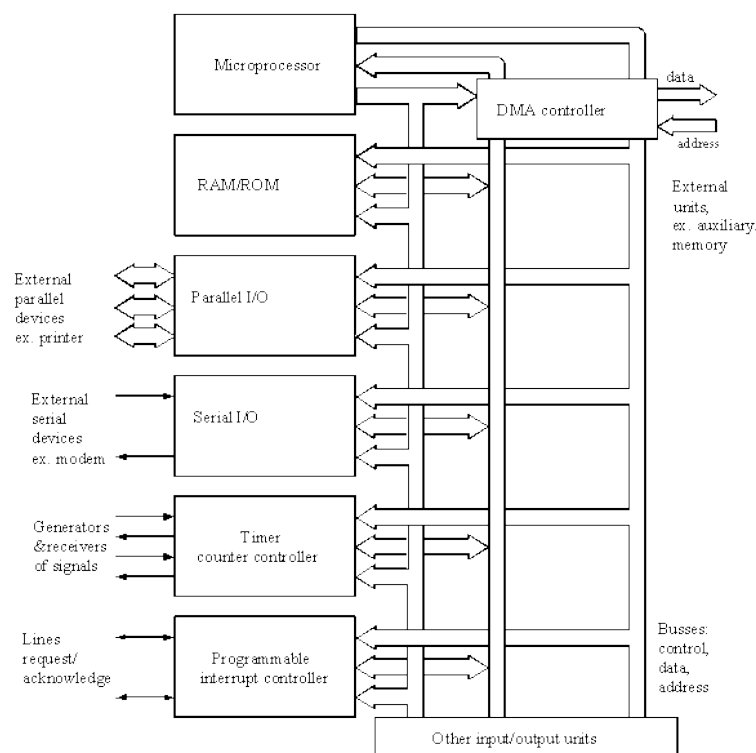


Image 2 General scheme of simple microcomputer

Parallel input/output controller maintains information exchange with peripheral devices, which send data in the parallel form. Examples of such devices are printers, display monitor, hard and floppy disk memories, keyboard. The activity of the controller is supervised by the microprocessor, which intervenes on each transfer of data by execution of respective instructions of data read or write from (to) the controller. The controller itself transfers data from its internal memory (registers) to peripheral devices.

The serial input/output controller maintains information exchange with peripheral devices, which send data in the serial form. Examples of such devices are a mouse and a modem for interconnections through telephone network. The controller implements in hardware conversion of serial data into their parallel form and vice versa (with the use of serial/parallel registers). The functioning of the controller is controlled by the microprocessor, which intervenes on each termination of data conversion by execution of respective data read or write instructions.

Interrupt controller provides servicing interrupts coming in parallel from many external devices. Its task consists in receiving interrupt requests, registering them, performing selection to choose one which is to be serviced by the processor. The controller communicates with the processor to enable sending the identifier of the selected interrupt and to exchange control signals. The interrupt controller is supervised by the processor, which services the interrupt after receiving the interrupt from the controller.

The DMA controller enables parallel data exchange between external devices and the operational memory without involvement of the processor. This controller enables autonomous data transfers to/from operational memory. These transfers do not engage the processor, which can proceed with computations at the same time.

The Microprocessor's Impact on Society

Today, we literally swim in a sea of silicon. There are dozens of chips in every car and dozens in every office. Microprocessors in the form of watches, pagers, cellular phones, and Sharp Wizard's adorn our bodies as jewelry. Beyond embedded systems, the rapid rise of the Internet now appears to be on the verge of ensuring that the personal computer truly becomes ubiquitous. Where in the past PCs have been concentrated in the workplace and scattered in the homes of the wealthier third of American society, the Internet is almost certain to lower the final obstacles and make it possible to create a new class of information appliances that will extend the reach of the Net until it matches that of television or the telephone. Microprocessors are everywhere, but how is modern life different? In many areas the advent of microelectronics obviously does have clear benefits. That can be seen in the effect of computing technology on the disabled; the transformation of certain kinds of boring and dangerous manual labor; advances in medical technology and weather forecasting. Indeed the easy answer is that microprocessors have transformed modern society. They affect the way we work and play, the way we travel and communicate. They offer remarkable processing power at infinitesimal cost.

Chapter 2

Bus architecture

A bus is a collection of wires, chips and slots inside the computer through which data are transmitted from one part of the computer to another from peripheral devices. It is also called a pathway in the computer on which data travels. It is a set of parallel distinct wires, serving different purposes, which allow devices attached to it to communicate with the CPU.

The bus architecture in computer system is as shown below:

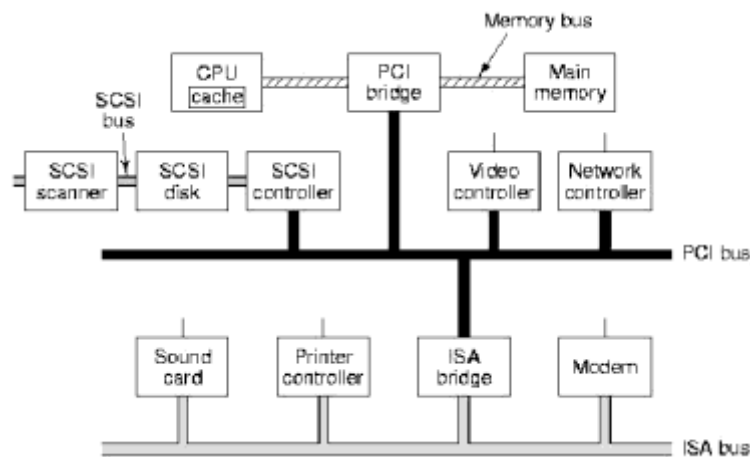


Image 3 Bus architecture

The function of BUS are:

- It carries information from one component to another.
- It carries data, address or control signal.
- One component of the computer can interact with other through a bus.
- There are three main part of bus.

They are described below:

Control Bus:

It carries the control signal. The control signal is used for controlling and coordinating the various activities of the computer. It is generated from the control unit of the CPU. Different architectures result in a different number of lines within the control bus, as each line is used to perform a specific task.

For instance, different specific lines are used for each of read, write and reset requests. These are not a group of lines like address bus and data bus, but individual lines that provide a pulse to indicate a microprocessor operation. The control unit generates a specific control signal for every operation, such as memory read or input/output operation. This signal is also used to identify a device type, with which the microprocessor intends to communicate.

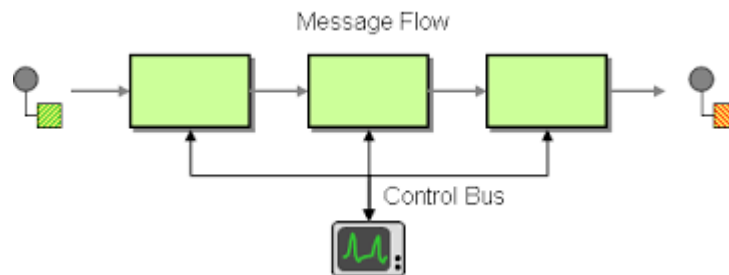


Image 4 Control bus

Address Bus:

Address bus carries memory address within the device. It allows the CPU to reference memory locations within the device. It connects the CPU and other peripherals and carries only memory address. In a computer system, each peripheral or memory location is identified by a numerical value, called an address and the address bus is used to carry this numerical value as well as it also contains a few control lines to carry control commands. The address bus is unidirectional, bits flow in one direction from the processor to peripherals.

The processor uses the address bus to perform the first function of identifying a peripheral or a memory location. The address bus contains the connections between the processor and memory that carry the signals relating to the address which the CPU is processing at that time, such as the locations that the CPU is reading from or writing to. The processor uses the address bus to perform, identifying a peripheral or a memory location.

When the address bus carries 8 bit at a time, the CPU could address only 256 (i.e. 2⁸) bytes of RAM. Most of the early PCs had 20 bit address buses. So, CPU could address 220bytes of data. Now, with 32-bit address buses, CPU can address 4GB of RAM. If there is wider bus path, more information can be processed at a time and hence, it also affects the processing speed of a computer.

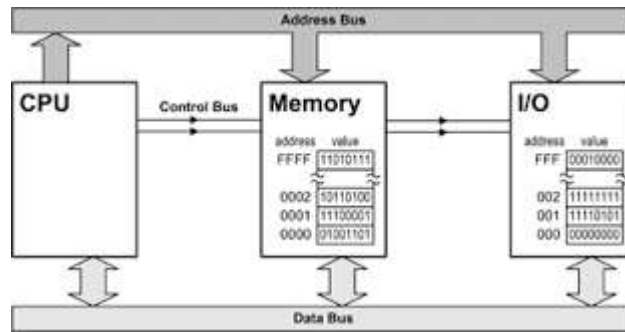


Image 5 Address bus

Data Bus:

Data bus transfer data from one location to another across the computer. On these lines, the meaningful data which is to be retrieved from a device is placed. Data bus is used by CPU to transfer data. It may be 16-bit or 32-bit data bus. It is an electrical path that connects the CPU, memory and other hardware devices on the motherboard. These lines are bidirectional in which the data flows in both directions between the processor and memory and peripheral devices. The number of wires in the bus affects the speed at which data can be travel between hardware components just as the number of lanes on a highway affects the time it takes people to reach their destination. Each wire can transfer 1 bit of data at a time and 8 wire bus can move 8 bit at a time, which is 1-byte data at a time. A 16-bit bus can transfer 2 bytes. 32 bits can transfer 4 bytes and so on. Intel 80286 microprocessor used 16 bit data bus and Intel 80386 used 32-bit data bus. When the data bus width grows larger, more data can be transferred.

The transmission of the data on bus lines takes place between approximately 1M baud for the microcomputer to about 1000 M baud or more for the large more expensive computers (1 baud = 1 bit/sec). Communication between the different units of a processing system is carried out along address and data bus and also along various control lines. All control operations are governed by the master timing source and clock.

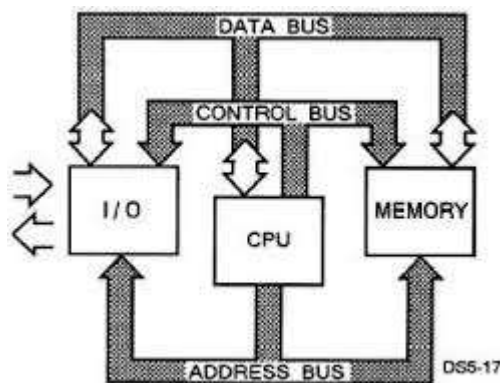


Image 6 data bus

Bus organization of 8085

The microprocessor MPU performs various operations with peripheral devices or a memory location by using three sets of communication lines called buses: the address bus, the data bus and the control bus. And these three combined lines is called as system bus.

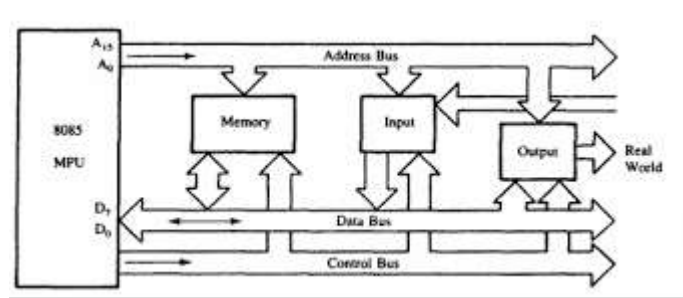


Image 7 bus structure

Address bus:

The address bus is a group of 16 lines generally called as A0 – A15 to carry a 16-bit address of memory location.

In a computer system, each peripheral or memory location is identified by a binary number called an address. This is similar to the postal address of a house.

The address bus is unidirectional, that means bit flow in only one direction from MPU to peripheral.

MPU carries 16-bit address i.e. $2^{16} = 65,536$ or 64K memory locations.

Data Bus:

The data bus is a group of eight bidirectional lines used for data flow in both the directions between MPH and peripheral devices.

The 8 data lines are manipulating 8-bit data ranging from 00 to FF i.e. ($2^8 = 256$) numbers from 0000 0000 -1111 1111

This 8 bit data is called as word length and the register size of a microprocessor and MPH is called 8-bit microprocessor.

Control bus:

Control bus is having various single lines used for sending control signals in the form of pulse to the memory and I/O devices.

The MPU generates specific control signals to perform a particular operations. Some of these control signals are memory read, memory write, I/O read and I/O write.

Pin diagram of 8085

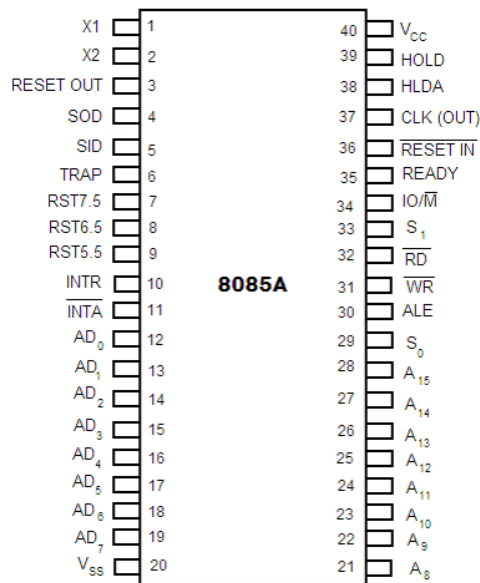


Image 8 Pin diagram of 8085

The 8085 and Its Buses

- The 8085 is an 8-bit general purpose microprocessor that can address 64K Byte of memory.
- It has 40 pins and uses +5V for power. It can run at a maximum frequency of 3 MHz.
 - The pins on the chip can be grouped into 6 groups:
 - Address Bus.
 - Data Bus.
 - Control and Status Signals.
 - Power supply and frequency.
 - Externally Initiated Signals.
 - Serial I/O ports.

8085 Pin description

- Higher Order Address pins- A₁₅ – A₈
 - The address bus has 8 signal lines A₈ – A₁₅ which are unidirectional.
- Lower Order Address/ Data Pins- AD₇-AD₀
 - These are time multiplexed pins and are DE-multiplexed using the pin ALE
 - So, the bits AD₀ – AD₇ are bi-directional and serve as A₀ – A₇ and D₀ – D₇ at the same time.

- During the execution of the instruction, these lines carry the address bits during the early part, then during the late parts of the execution, they carry the 8 data bits.
 - In order to separate the address from the data, we can use a latch to save the value before the function of the bits changes.
- Control Pins – RD, WR
 - These are active low Read & Write pins
- Status Pins – ALE, IO/M (active low), S₁, S₀
 - ALE (Address Latch Enable)-Used to DE-multiplex AD₇-AD₀
 - IO/M – Used to select I/O or Memory operation
 - S₁,S₀ – Denote the status of data on data bus
- Interrupt Pins – TRAP, RST7.5, RST 6.5, RST 5.5, INTR, INTA
 - These are hardware interrupts used to initiate an interrupt service routine stored at predefined locations of the system memory.
- Serial I/O pins – SID (Serial Input Data), SOD (Serial Output Data)
 - These pins are used to interface 8085 with a serial device.
- Clock Pins- X₁, X₂, CLK(OUT)
 - X₁, X₂ – These are clock input pins. A crystal is connected between these pins such that $f_{\text{crystal}} = 2f_{8085}$ where f_{crystal} = crystal frequency & f_{8085} = operating frequency of 8085
 - CLK(OUT) – This is an auxiliary clock output source
- Reset Pins – Reset In (**active low**), Reset Out
 - **Reset In** is used to reset 8085 whereas **Reset Out** can be used to reset other devices in the system
- DMA (Direct Memory Access) pins – HOLD, HLDA
 - These pins are used when data transfer is to be performed directly between an external device and the main memory of the system.
- Power Supply Pins – +V_{CC}, V_{SS}

Architecture of 8085

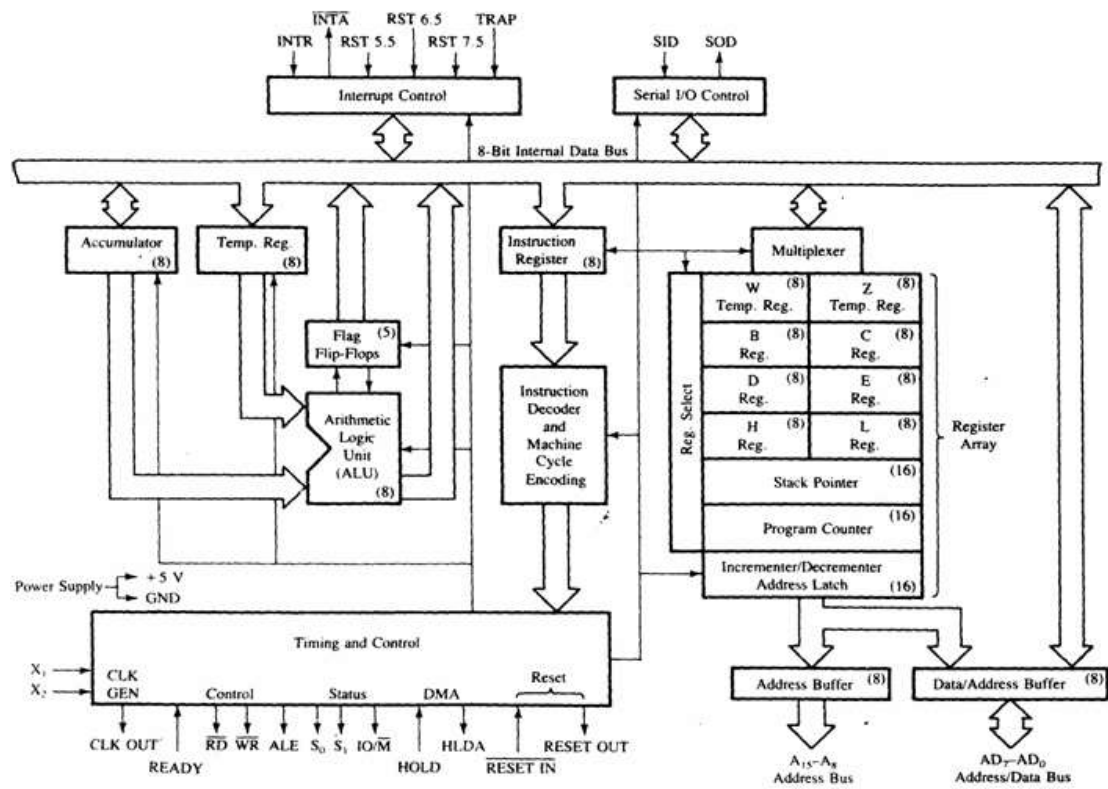


Image 9 function block diagram of 8085

This is the functional Block Diagram of 8085 Microprocessor.

Accumulator:-It is a 8-bit register which is used to perform arithmetical and logical operation. It stores the output of any operation. It also works as registers for i/o accesses.

Temporary Register:-It is a 8-bit register which is used to hold the data on which the accumulator is computing operation. It is also called as operand register because it provides operands to ALU.

Registers:-These are general purposes registers. Microprocessor consists 6 general purpose registers of 8-bit each named as B,C,D,E,H and L. 聽 Generally theses registers are not used for storing the data permanently. It carries the 8-bits data. These are used only during the execution of the instructions.

These registers can also be used to carry the 16 bits data by making the pair of 2 registers. The valid register pairs available are BC,DE HL. We can not use other pairs except BC,DE and HL. These registers are programmed by user.

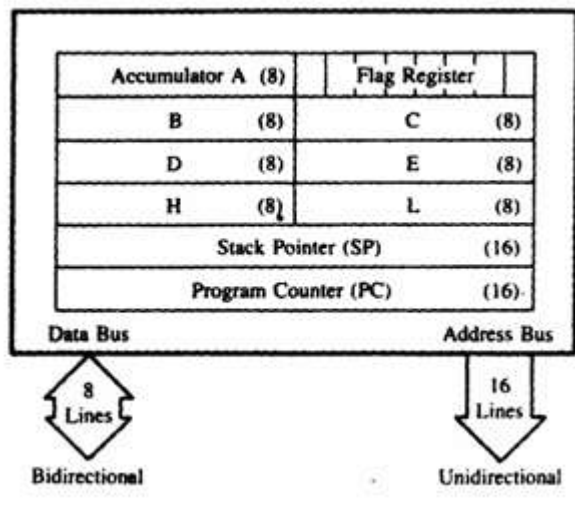


Image 10 Programmatic model of 8085

ALU:-ALU performs the arithmetic operations and logical operation.

Flag Registers:-It consists of 5 flip flop which changes its status according to the result stored in an accumulator. It is also known as status registers. It is connected to the ALU.

There are five flip-flops in the flag register are as follows:

1. Sign(S)
2. zero(z)
3. Auxiliary carry(AC)
4. Parity(P)
5. Carry(C)

The bit position of the flip flop in flag register is:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
S	Z		AC		P		CY

All of the three flip flop set and reset according to the stored result in the accumulator.

1. Sign- If D₇ of the result is 1 then sign flag is set otherwise reset. As we know that a number on the D₇ always decides the sign of the number.

if D₇ is 1: the number is negative.

if D₇ is 0: the number is positive.

2. Zeros(Z)-If the result stored in an accumulator is zero then this flip flop is set otherwise it is reset.

3.Auxiliary carry(AC)-If any carry goes from D3 to D4 in the output then it is set otherwise it is reset.

4.Parity(P)-If the no of 1's is even in the output stored in the accumulator then it is set otherwise it is reset for the odd.

5.Carry(C)-If the result stored in an accumulator generates a carry in its final output then it is set otherwise it is reset.

Instruction registers(IR):-It is a 8-bit register. When an instruction is fetched from memory then it is stored in this register.

Instruction Decoder:- Instruction decoder identifies the instructions. It takes the information from instruction register and decodes the instruction to be performed.

Program Counter:-It is a 16 bit register used as memory pointer. It stores the memory address of the next instruction to be executed. So we can say that this register is used to sequencing the program. Generally the memory have 16 bit addresses so that it has 16 bit memory.

The program counter is set to 0000H.

Stack Pointer:-It is also a 16 bit register used as memory pointer. It points to the memory location called stack. Generally stack is a reserved portion of memory where information can be stores or taken back together.

Timing and Control Unit:-It provides timing and control signal to the microprocessor to perform the various operation.It has three control signal. It controls all external and internal circuits. It operates with reference to clock signal.It synchronizes all the data transfers.

There are three control signal:

1.ALE-Arithmetic Latch Enable, It provides control signal to synchronize the components of microprocessor.

2.RD- This is active low used for reading operation.

3.WR-This is active low used for writing operation.

There are three status signal used in microprocessor S0, S1 and IO/M. It changes its status according the provided input to these pins.

IO/M(Active Low)	S1	S2	Data Bus Status(Output)
0	0	0	Halt
0	0	1	Memory WRITE
0	1	0	Memory READ
1	0	1	IO WRITE
1	1	0	IO READ
0	1	1	Opcode fetch
1	1	1	Interrupt acknowledge

Serial Input Output Control-There are two pins in this unit. This unit is used for serial data communication.

Interrupt Unit-There are 6 interrupt pins in this unit. Generally an external hardware is connected to these pins. These pins provide interrupt signal sent by external hardware to microprocessor and microprocessor sends acknowledgement for receiving the interrupt signal. Generally INTA is used for acknowledgement.

Register Section:-Many registers has been used in microprocessor.

It consists of PIPO(Parallel Input Parallel Output) register.

What is an Opcode?

Opcode is nothing but the machine language instruction which denotes the microprocessor about what operation should be performed on the specific data. Microprocessor converts the instruction into suitable machine language, so that it can understand the operation to be performed and executes it.

Consider MVI A, 18H.

When the above instruction is to be executed, the microprocessor gets the Opcode for MVI A and performs the necessary operation on the data which is 18H in this case. The Opcode for MVI A is 3EH. So the microprocessor first reads this Opcode from the instruction and then performs the operation specified by Opcode over the data given.

Now let us assume we want to store the above instruction in a specific address say 5500H. We know that in 8085 processor only one byte can be stored in each address location. Therefore the Opcode 3EH is stored at the location 5500H and the data 18H is stored at the next location 5501H.

Generation of read and write control signals

Control Signals

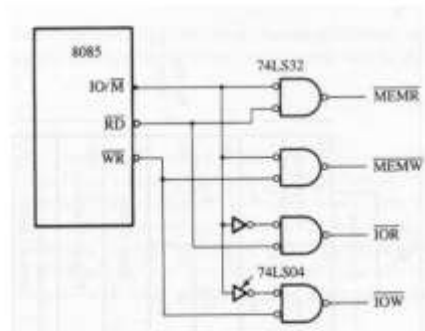


Figure 3: Example of schematic diagram to generate control signals.

Image 11 schematic diagram of control signal generation



READ Operation:

- During T1 state, microprocessor uses IO/M(bar), S0, S1 signals are used to instruct microprocessor to fetch opcode.
- Thus when IO/M(bar)=0, S0=S1= 1, it indicates opcode fetch operation.
- During this operation 8085 transmits 16-bit address and also uses ALE signal for address latching.
- At T2 state microprocessor uses read signal and make data ready from that memory location to read opcode from memory and at the same time program counter increments by 1 and points next instruction to be fetched.
- In this state microprocessor also checks READY input signal, if this pin is at low logic level ie. '0' then microprocessor adds wait state immediately between T2 and T3.
- At T3, microprocessor reads opcode and store it into instruction register to decode it further.

- During T4 microprocessor performs internal operation like decoding opcode and providing necessary actions.

- The opcode is decoded to know whether T5 or T6 states are required, if they are not required then \bar{RD} p performs next operation.

WRITE Operation:

- It is used to fetch one byte from the memory.

- It requires 3 T-States.

- It can be used to fetch operand or data from the memory.

- During T1, A8-A15 contains higher byte of address. At the same time ALE is high. Therefore Lower byte of address A0-A7 is selected from AD0-AD7.

- Since it is memory ready operation, IO/M($\bar{}$) goes low.

- During T2 ALE goes low, RD($\bar{}$) goes low. Address is removed from AD0-AD7 and data D0-D7 appears on AD0-AD7.

- During T3, Data remains on AD0-AD7 till RD($\bar{}$) is at low signal.

Steps to execute a stored program

Two steps are performed that are fetch operation and execution operation.

fetch operation: -

1. microprocessor will transfer 16 bit address from PC to address line to select desired location
2. Microprocessor gives reading location so 8 bit instruction is transferred from selected memory location to data pin
3. instruction code is transferred by microprocessor to instruction register from data bus

Execution operation:-

1. Instruction code is transferred from IR to instruction decoder
2. The instruction code as of 2 or 3 bytes when 1st ckt. Will read 2nd and 3rd byte of instruction from memory and then execute corresponding instruction.

Chapter 3

Instruction cycle

An instruction cycle (also known as the fetch–decode–execute cycle or the fetch-execute cycle) is the basic operational process of a computer. It is the process by which a computer retrieves a program instruction from its memory, determines what actions the instruction dictates, and carries out those actions. This cycle is repeated continuously by a computer's central processing unit (CPU), from boot-up to when the computer is shut down.

In simpler CPU's the instruction cycle is executed sequentially, each instruction being processed before the next one is started. In most modern CPU's the instruction cycles are instead executed concurrently, and often in parallel, through an instruction pipeline: the next instruction starts being processed before the previous instruction has finished, which is possible because the cycle is broken up into separate steps.

1. Initiating the cycle

- The cycle starts immediately when power is applied to the system using an initial PC value that is predefined for the system architecture

- Typically this address points to instructions in a read-only memory (ROM) (not the random access memory or RAM) which begins the process of loading the operating system.

2.Fetch the Instruction

◆ Step 1 of the Instruction Cycle is called the Fetch Cycle. This step is the same for each instruction.

- 1) The CPU sends PC to the MAR and sends a READ command on the control bus
- 2) In response to the read command (with address equal to PC), the memory returns the data stored at the memory location indicated by PC on the data bus.
- 3) The CPU copies the data from the data bus into its MDR (also known as MBR)
- 4) A fraction of a second later, the CPU copies the data from the MDR to the Instruction Register (IR)
- 5) The PC is incremented so that it points to the following instruction in memory. This step prepares the CPU for the next cycle. The Control Unit fetches the instruction's address from the Memory Unit

3.Decode the Instruction

- Step 2 of the instruction Cycle is called the Decode Cycle. The decoding process allows the CPU to determine what instruction is to be performed, so that the CPU can tell how many operands it needs to fetch in order to perform the instruction.
- The opcode fetched from the memory is decoded for the next steps and moved to the appropriate registers. The decoding is done by the CPU's Control Unit.

4.Read the effective address

- Step 3 is deciding which operation it is. If this is a Memory operation - in this step the computer checks if it's a direct or indirect memory operation:
- Direct memory instruction - Nothing is being done.
- Indirect memory instruction - The effective address is being read from the memory. If this is a I/O or Register instruction - the computer checks its kind and executes the instruction.

5.Execute the Instruction

- Step 4 of the Instruction Cycle is the Execute Cycle. Here, the function of the instruction is performed.
- If the instruction involves arithmetic or logic, the Arithmetic Logic Unit is utilized. This is the only stage of the instruction cycle that is useful from the perspective of the end user.
- Everything else is overhead required to make the execute stage happen.

Chapter 4

Machine cycle

A *machine cycle*, also called a *processor cycle* or a *instruction cycle*, is the basic operation performed by a **central processing unit** (CPU). A CPU is the main logic unit of a **computer**.

A machine cycle consists of a sequence of three steps that is performed continuously and at a rate of millions per second while a computer is in operation. They are *fetch*, *decode* and *execute*. There also is a fourth step, *store*, in which input and output from the other three phases is stored in **memory** for later use; however, no actual processing is performed during this step.

In the *fetch* step, the *control unit* requests that main **memory** provide it with the instruction that is stored at the *address* (i.e., location in memory) indicated by the control unit's *program counter*.

The control unit is a part of the CPU that also decodes the instruction in the *instruction register*. A *register* is a very small amount of very fast memory that is built into the CPU in order to speed up its operations by providing quick access to commonly used values; instruction registers are registers that hold the instruction being executed by the CPU. Decoding the instructions in the instruction register involves breaking the *operand field* into its components based on the instructions *opcode*.

Opcode (an abbreviation of *operation code*) is the portion of a *machine language* instruction that specifies what operation is to be performed by the CPU. Machine language, also called *machine code*, refers to instructions coded in patterns of *bits* (i.e., zeros and ones) that are directly readable and executable by a CPU.

A program counter, also called the *instruction pointer* in some computers, is a register that indicates where the computer is in its instruction sequence. It holds either the address of the instruction currently being executed or the address of the next instruction to be executed, depending on the details of the particular computer. The program counter is automatically incremented for each machine cycle so that instructions are normally retrieved sequentially from memory.

The control unit places these instructions into its instruction register and then increments the program counter so that it contains the address of the next instruction stored in memory. It then executes the instruction by activating the appropriate circuitry to perform the requested task. As soon as the instruction has been executed, it restarts the machine cycle, beginning with the fetch step.\

T states

One complete cycle of clock is called as T-state as shown in the above figure. A T-state is measured from the falling edge of one clock pulse to the falling edge of the next clock pulse. Various versions of 8086 have maximum clock frequency from 5MHz to 10MHz. Hence the minimum time for one T-state is between 100 to 200 n sec.

Assembly languages

An assembly (or assembler) language, often abbreviated asm, is a low-level programming language for a computer, or other programmable device, in which there is a very strong (but often not one-to-one) correspondence between the language and the architecture's machine code instructions. Each assembly language is specific to a particular computer architecture. In contrast, most high-level programming languages are generally portable across multiple architectures but require interpreting or compiling. Assembly language may also be called symbolic machine code.

Machine language

Machine code or machine language is a set of instructions executed directly by a computer's central processing unit(CPU). Each instruction performs a very specific task, such as a load, a jump, or an ALU operation on a unit of data in a CPU register or memory. Every program directly executed by a CPU is made up of a series of such instructions. (The phrase 'directly executed' needs some clarification; machine code is by definition the lowest level of programming detail visible to the programmer, but internally many processors use microcode or optimize and transform machine code instructions into sequences of micro-ops in a sophisticated way.)

Mnemonics

Mnemonics allow users to access quickly a wide variety of commands, services, programs and functions without the need to type out extended phrases. One example of a mnemonic code is the term "inc," which on an Intel microprocessor refers to the command "increase by one." Rather than type the entire phrase, the letters "inc" can be entered. Mnemonic code derives from the concept of traditional mnemonics in which abbreviations, rhymes or simple stories are used to help people remember information.

Addressing modes of 8051

1. Immediate addressing mode:

In this type, the operand is specified in the instruction along with the opcode. In simple way, it means data is provided in instruction itself.

Ex: MOV A,#05H -> Where MOV stands for move, # represents immediate data. 05h is the data. It means the immediate data 05h provided in instruction is moved into A register.

2.Register addressing mode:

Here the operand is contained in the specific register of micro-controller. The user must provide the name of register from where the operand/data need to be fetched. The permitted registers are A, R7-R0 of each register bank. Ex: MOV A,R0-> content of R0 register is copied into Accumulator.

3. Direct addressing mode:

In this mode the direct address of memory location is provided in instruction to fetch the operand. Only internal RAM and SFR's address can be used in this type of instruction.

Ex: MOV A, 30H => Content of RAM address 30H is copied into Accumulator.

4. Register Indirect addressing mode:

Here the address of memory location is indirectly provided by a register. The '@' sign indicates that the register holds the address of memory location i.e. fetch the content of memory location whose address is provided in register.

Ex: MOV A,@R0 => Copy the content of memory location whose address is given in R0 register.

5. Indexed Addressing mode:

This addressing mode is basically used for accessing data from look up table. Here the address of memory is indexed i.e. added to form the actual address of memory.

Ex: MOVC A,@A+DPTR => here 'C' means Code. Here the content of A register is added with content of DPTR and the resultant is the address of memory location from where the data is copied to A register.

Its of three types

1. one byte instruction: if operand is not given in instruction then the instruction can be completely represented as one byte instruction.
2. Two byte instruction: if 8 bit no is given as operand in instruction then first byte specify the opcode and second byte specify the operand.
3. Three byte instruction: if 16 bit no is given is operand in the instruction then such instruction are called three byte instruction

Memory Mapping

A memory map is a massive table, in effect a [database](#), that comprises complete information about how the [memory](#) is structured in a computer system. A memory map works something like a gigantic office organizer. In the map, each computer [file](#) has a unique memory [address](#) reserved especially for it, so that no other data can inadvertently overwrite or corrupt it.

In order for a computer to function properly, its [OS](#) (operating system) must always be able to access the right parts of its memory at the right times. When a computer first [boots up](#) (starts), the memory map tells the OS how much memory is available. As the computer runs, the memory map ensures that data is always written to, and read from, the proper places. The memory map also ensures that the computer's [debuggers](#) can resolve memory addresses to actual stored data.

If there were no memory map, or if an existing memory map got corrupted, the OS might (and probably would) write data to, and read data from, the wrong places. As a result, when data was read, it would not always pertain to the appropriate files or [application programs](#). The problem would likely start out small and unnoticeable, worsen with time, and become apparent only after considerable damage had been done to stored data and programs. In the end, some or all of the applications would fail to run, and many critical data files would be ruined.

Memory mapped i/o

Memory-mapped I/O (MMIO) and port-mapped I/O (PMIO) (which is also called isolated I/O[[citation needed](#)]) are two complementary methods of performing

[input/output](#) (I/O) between the [central processing unit](#) (CPU) and [peripheral devices](#) in a [computer](#). An alternative approach is using dedicated I/O processors, commonly known as [channels](#) on [mainframe computers](#), which execute their own [instructions](#).

Memory-mapped I/O uses the same [address space](#) to address both [memory](#) and [I/O devices](#). The memory and [registers](#) of the I/O devices are mapped to (associated with) address values. So when an address is accessed by the CPU, it may refer to a portion of [physical RAM](#), or it can instead refer to memory of the I/O device. Thus, the CPU instructions used to access the memory can also be used for accessing devices. Each I/O device monitors the CPU's address bus and responds to any CPU access of an address assigned to that device, connecting the [data bus](#) to the desired device's [hardware register](#). To accommodate the I/O devices, areas of the addresses used by the CPU must be reserved for I/O and must not be available for normal physical memory. The reservation may be permanent, or temporary (as achieved via [bank switching](#)). An example of the latter is found in the [Commodore 64](#), which uses a form of memory mapping to cause [RAM](#) or I/O hardware to appear in the \$D000-\$DFFF range.

Peripheral mapped I/O

- 8-bit device address
- Data is transfer only between accumulator and I.O port
- The I/O map is independent of the memory map; 256 input device and 256. output device can be connected
- Less hardware is required to decode 8-bit address
- Arithmetic or logical operation cannot be directly performed with I/O data
- Control signals used are like IOR(bar), IOW(bar)
- Instructions used are like IN,OUT

P.S- That bar inside bracket means active low signal

Instruction sets of 8051

Depending on operation they perform, all instructions are divided in several groups:

- Arithmetic Instructions
- Branch Instructions
- Data Transfer Instructions
- Logic Instructions
- Bit-oriented Instructions

The first part of each instruction, called MNEMONIC refers to the operation an instruction performs (copy, addition, logic operation etc.). Mnemonics are abbreviations of the name of operation being executed. For example:

INC R1- Means: Increment register R1 (increment register R1);

LJMP LAB5- Means: Long Jump LAB5 (long jump to the address marked as LAB5);

JNZ LOOP- Means: Jump if Not Zero LOOP (if the number in the accumulator is not 0, jump to the address marked as LOOP);

The other part of instruction, called OPERAND is separated from mnemonic by at least one white-space and defines data being processed by instructions. Some of the instructions have no operand, while some of them have one, two or three. If there is more than one operand in an instruction, they are separated by a comma. For example:

RET- return from a subroutine;

JZ TEMP- if the number in the accumulator is not 0, jump to the address marked as TEMP;

ADD A,R3- add R3 and accumulator;

CJNE A,#20,LOOP- compare accumulator with 20. If they are not equal, jump to the address marked as LOOP;

Arithmetic instructions

Arithmetic instructions perform several basic operations such as addition, subtraction, division, multiplication etc. After execution, the result is stored in the first operand. For example:ADD A,R1- The result of addition (A+R1) will be stored in the accumulator.

Branch Instructions

There are two kinds of branch instructions: Unconditional jump instructions: upon their execution a jump to a new location from where the program continues execution is executed. Conditional jump instructions: a jump to a new program location is executed only if a specified condition is met. Otherwise, the program normally proceeds with the next instruction.

Data Transfer Instructions

Data transfer instructions move the content of one register to another. The register the content of which is moved remains unchanged. If they have the suffix "X" (MOVX), the data is exchanged with external memory.

Logic Instructions

Logic instructions perform logic operations upon corresponding bits of two registers. After execution, the result is stored in the first operand.

Bit-oriented Instructions

Similar to logic instructions, bit-oriented instructions perform logic operations. The difference is that these are performed upon single bits.

Chapter 5

I/O Interface (Interrupt and DMA Mode)

The method that is used to transfer information between internal storage and external I/O devices is known as I/O interface. The CPU is interfaced using special communication links by the peripherals connected to any computer system. These communication links are used to resolve the differences between CPU and peripheral. There exists special hardware components between CPU and peripherals to supervise and synchronize all the input and output transfers that are called interface units.

Mode of Transfer:

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes.

Data transfer to and from the peripherals may be done in any of the three possible ways

1. Programmed I/O.
2. Interrupt- initiated I/O.

3. Direct memory access(DMA).

Now let's discuss each mode one by one.

1. **Programmed I/O:** It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.

Example of Programmed I/O: In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory. In programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it needlessly keeps the CPU busy. This situation can be avoided by using an interrupt facility. This is discussed below.

2. **Interrupt- initiated I/O:** Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.
3. **Direct Memory Access:** The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

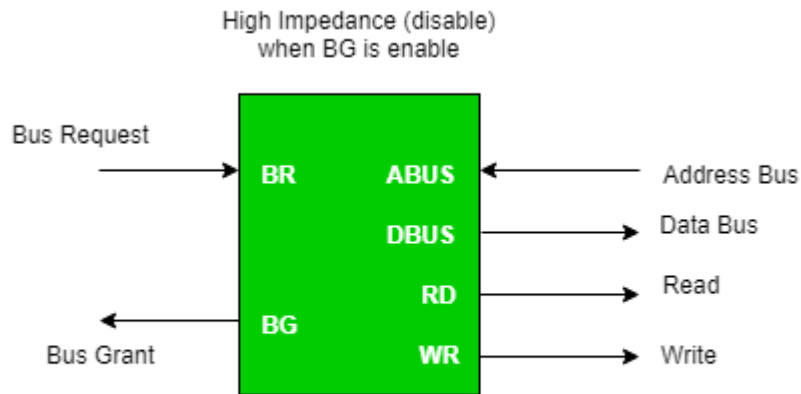


Figure - CPU Bus Signals for DMA Transfer

Bus Request : It is used by the DMA controller to request the CPU to relinquish the control of the buses.

Bus Grant : It is activated by the CPU to Inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data. This transfer can take place in many ways.

Example it can transfer using

Bus Transfer :

In which a block sequence consisting of memory words is transferred in a continuous burst where the DMA controller is the master of the memory buses. This mode is needed for fast devices like magnetic disks.

Cyclic Stealing :

In this DMA controller transfers one word at a time after which it must return the control of the buses to the CPU. The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to “steal” one memory cycle.

Chapter 6

Interrupt

In [system programming](#), an interrupt is a signal to the [processor](#) emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its [state](#), and executing a [function](#) called an [interrupt handler](#) (or an

interrupt service routine, ISR) to deal with the event. This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities.[\[1\]](#) There are two types of interrupts: hardware interrupts and software interrupts.

Hardware interrupts are used by devices to communicate that they require attention from the [operating system](#).[\[2\]](#) Internally, hardware interrupts are implemented using electronic alerting signals that are sent to the processor from an external device, which is either a part of the computer itself, such as a [disk controller](#), or an external [peripheral](#). For example, pressing a key on the [keyboard](#) or moving the [mouse](#) triggers hardware interrupts that cause the processor to read the keystroke or mouse position. Unlike the software type (described below), hardware interrupts are [asynchronous](#) and can occur in the middle of instruction execution, requiring additional care in programming. The act of initiating a hardware interrupt is referred to as an [interrupt request](#) (IRQ).

A software interrupt is caused either by an exceptional condition in the processor itself, or a special [instruction](#) in the [instruction set](#) which causes an interrupt when it is executed. The former is often called a [trap](#) or [exception](#) and is used for errors or events occurring during program execution that are exceptional enough that they cannot be handled within the program itself. For example, a divide-by-zero exception will be thrown if the processor's [arithmetic logic unit](#) is commanded to divide a number by zero as this instruction is an error and impossible. The operating system will catch this exception, and can choose to abort the instruction. Software interrupt instructions can function similarly to [subroutine calls](#) and are used for a variety of purposes, such as to request services from [device drivers](#), like interrupts sent to and from a [disk controller](#) to request reading or writing of data to and from the disk.

Each interrupt has its own interrupt handler. The number of hardware interrupts is limited by the number of interrupt request (IRQ) lines to the processor, but there may be hundreds of different software interrupts. Interrupts are a commonly used technique for [computer multitasking](#), especially in [real-time computing](#). Such a system is said to be interrupt-driven.[\[3\]](#)

Interrupts are similar to [signals](#), the difference being that signals are used for [IPC](#), mediated by the kernel (possibly via system calls) and handled by processes, while interrupts are mediated by the processor and handled by the [kernel](#). The kernel may pass an interrupt as a signal to the process that caused it (typical examples are [SIGSEGV](#), [SIGBUS](#), [SIGILL](#) and [SIGFPE](#)).

Interrupt

Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an **ISR** (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

The following image shows the types of interrupts we have in a 8086 microprocessor

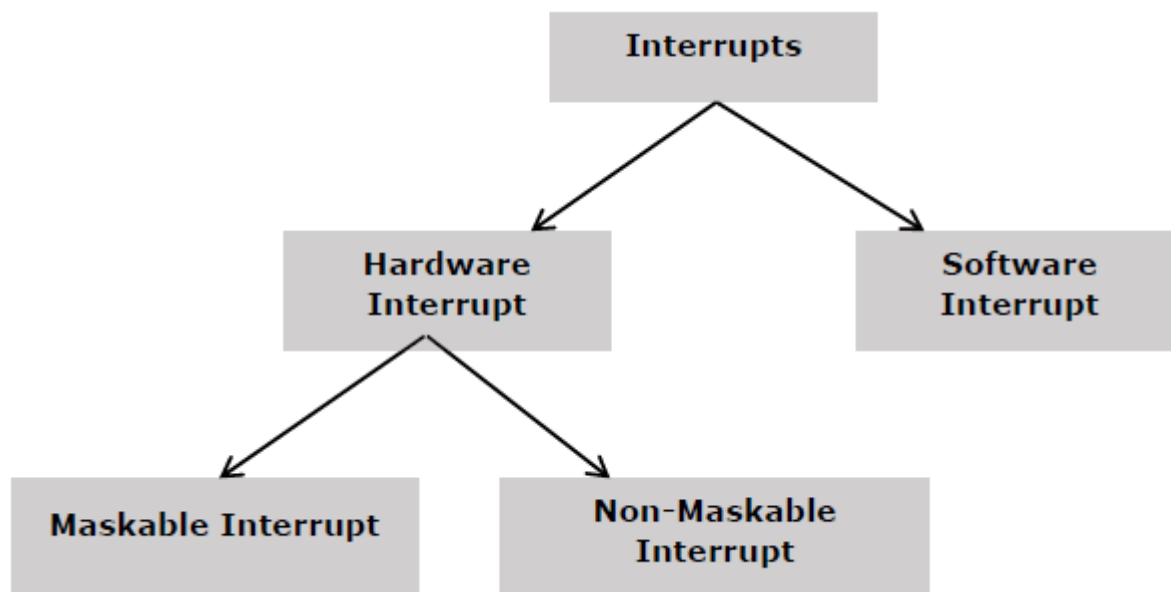


Image 12 Interrupt

Hardware Interrupts

Hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor.

The 8086 has two hardware interrupt pins, i.e. NMI and INTR. NMI is a non-maskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.

NMI

It is a single non-maskable interrupt pin (NMI) having higher priority than the maskable interrupt request pin (INTR) and it is of type 2 interrupt.

When this interrupt is activated, these actions take place –

- Completes the current instruction that is in progress.
- Pushes the Flag register values on to the stack.
- Pushes the CS (code segment) value and IP (instruction pointer) value of the return address on to the stack.
- IP is loaded from the contents of the word location 00008H.
- CS is loaded from the contents of the next word location 0000AH.
- Interrupt flag and trap flag are reset to 0.

INTR

The INTR is a mask-able interrupt because the microprocessor will be interrupted only if interrupts are enabled using set interrupt flag instruction. It should not be enabled using clear interrupt Flag instruction.

The INTR interrupt is activated by an I/O port. If the interrupt is enabled and NMI is disabled, then the microprocessor first completes the current execution and sends '0' on INTA pin twice. The first '0' means INTA informs the external device to get ready and during the second '0' the microprocessor receives the 8 bit, say X, from the programmable interrupt controller.

These actions are taken by the microprocessor –

- First completes the current instruction.
- Activates INTA output and receives the interrupt type, say X.
- Flag register value, CS value of the return address and IP value of the return address are pushed on to the stack.
- IP value is loaded from the contents of word location $X \times 4$
- CS is loaded from the contents of the next word location.
- Interrupt flag and trap flag is reset to 0

Software Interrupts

Some instructions are inserted at the desired position into the program to create interrupts. These interrupt instructions can be used to test the working of various interrupt handlers. It includes –

INT- Interrupt instruction with type number

It is 2-byte instruction. First byte provides the op-code and the second byte provides the interrupt type number. There are 256 interrupt types under this group.

Its execution includes the following steps –

- Flag register value is pushed on to the stack.
- CS value of the return address and IP value of the return address are pushed on to the stack.
- IP is loaded from the contents of the word location 'type number' $\times 4$.
- CS is loaded from the contents of the next word location.
- Interrupt Flag and Trap Flag are reset to 0

The starting address for type0 interrupt is 000000H, for type1 interrupt is 00004H similarly for type2 is 00008H andso on. The first five pointers are dedicated interrupt pointers. i.e. -

- **TYPE 0** interrupt represents division by zero situation.
- **TYPE 1** interrupt represents single-step execution during the debugging of a program.
- **TYPE 2** interrupt represents non-mask-able NMI interrupt.
- **TYPE 3** interrupt represents break-point interrupt.
- **TYPE 4** interrupt represents overflow interrupt.

The interrupts from Type 5 to Type 31 are reserved for other advanced microprocessors, and interrupts from 32 to Type 255 are available for hardware and software interrupts.

INT 3-Break Point Interrupt Instruction

It is a 1-byte instruction having op-code is CCH. These instructions are inserted into the program so that when the processor reaches there, then it stops the normal execution of program and follows the break-point procedure.

Its execution includes the following steps –

- Flag register value is pushed on to the stack.
- CS value of the return address and IP value of the return address are pushed on to the stack.
- IP is loaded from the contents of the word location $3 \times 4 = 0000CH$
- CS is loaded from the contents of the next word location.
- Interrupt Flag and Trap Flag are reset to 0.

INTO - Interrupt on overflow instruction

It is a 1-byte instruction and their mnemonic **INTO**. The op-code for this instruction is CEH. As the name suggests it is a conditional interrupt instruction, i.e. it is active only when the overflow flag is set to 1 and branches to the interrupt handler whose interrupt type number is 4. If the overflow flag is reset then, the execution continues to the next instruction.

Its execution includes the following steps –

- Flag register values are pushed on to the stack.

- CS value of the return address and IP value of the return address are pushed on to the stack.
- IP is loaded from the contents of word location $4 \times 4 = 00010H$
- CS is loaded from the contents of the next word location.
- Interrupt flag and Trap flag are reset to 0

Non- mask-able interrupt

A non-mask-able interrupt (NMI) is a type of hardware interrupt (or signal to the processor) that prioritizes a certain thread or process. Unlike other types of interrupts, the non-mask-able interrupt cannot be ignored through the use of interrupt masking techniques.

mask-able interrupt

An Interrupt that can be disabled or ignored by the instructions of CPU are called as mask-able Interrupt.

Eg: RST6.5, RST7.5, RST5.5 OF 8085 are mask-able Interrupts.

Chapter 7

Programmed I/O

Programmed input/output (PIO) is a method of transferring data between the CPU and a peripheral, such as a network adapter or an ATA storage device.

For programmed I/O, the software that is running on the CPU uses instructions to perform data transfers to or from an I/O device. This is in contrast to **Direct Memory Access (DMA)** transfers. The term *Programmed I/O* can refer to either MMIO or PMIO. Port-mapped I/O (PMIO) refers to a special address space outside of normal memory that is accessed with instructions such as IN and OUT. **Memory-mapped I/O[1]** (MMIO) refers to I/O devices being allocated addresses inside the normal Von Neumann address space that is primarily used for program and data. Such I/O is done using instructions such as LOAD and STORE. PMIO was very useful for early microprocessors with small address spaces, since the valuable resource was not consumed by the I/O devices.

The best known example of a PC device that uses programmed I/O is the ATA interface; however, this interface can also be operated in any of several DMA modes. Many older devices in a PC also use PIO, including legacy serial ports, legacy parallel ports when not in ECP mode, the PS/2 keyboard and mouse ports, legacy MIDI and joystick ports, the interval timer, and older network interfaces.

Synchronous data transmission

Synchronous transmission is transmission of signals in a fixed interval based on a predefined clocking signal and is meant for constant and reliable transmission of time-sensitive data such as VoIP and audio/video streaming.

This method of transmission is used when large amounts of data need to be transferred quickly since data is transferred in large blocks instead of individual characters. The data blocks are spaced and grouped in regular intervals and preceded by synchronous characters that a remote device decode and use to synchronize the connection between the end points.

After synchronization is complete, the transmission can begin.

Asynchronous data transfer

The transmission of asynchronous data is not prompted by a clock signal when sending the data to the receiver, unlike in synchronous methods, where sending data is measured against a time reference. Compared to synchronous transmission, asynchronous communication has a few advantages:

- It is more flexible and devices can exchange information at their own pace. Individual data characters can complete themselves so that even if one packet is corrupted, its predecessors and successors will not be affected.
- It does not require complex processes by the receiving device. This means that an inconsistency in the transmission of data does not result in a big crisis, since the device can keep up with the data stream. This also makes asynchronous transfers suitable for applications where character data is generated in an irregular manner.

There are also some disadvantages of using asynchronous data for transmission:

- The success of these transmissions depends on the start bits and their recognition. This can be easily susceptible to line interference, causing these bits to be corrupted or distorted.
- A large portion of the transmitted data is used for control and identification bits for headers and thus carries no useful information related to the transmitted data. This invariably means that more data packets need to be sent.

Interrupt driven I/O

Interrupt driven I/O is an alternative scheme dealing with I/O. Interrupt I/O is a way of controlling input/output activity whereby a peripheral or terminal that needs to make or receive a data transfer sends a signal. This will cause a program interrupt to be set. At a time appropriate to the priority level of the I/O interrupt. Relative to the total interrupt system, the processors enter an interrupt service routine. The function of the routine will depend upon the system of interrupt levels and priorities that is implemented in the processor. The interrupt technique requires more complex hardware and software, but makes far more efficient use of the computer's time and capacities

For **input**, the device interrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports or memory mapping.

For **output**, the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they are done with the buffer.

Here the CPU works on its given tasks continuously. When an input is available, such as when someone types a key on the keyboard, then the CPU is interrupted from its work to take care of the input data. The CPU can work continuously on a task without checking the input devices, allowing the devices themselves to interrupt it as necessary.

Direct Memory Access

Direct memory access (DMA) is a feature of computer systems that allows certain hardware subsystems to access main system [memory \(Random-access memory\)](#), independent of the [central processing unit \(CPU\)](#).

Without DMA, when the CPU is using [programmed input/output](#), it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU first initiates the transfer, then it does other operations while the transfer is in progress, and it finally receives an [interrupt](#) from the DMA controller when the operation is done. This feature is useful at any time that the CPU cannot keep up with the rate of data transfer, or when the CPU needs to perform work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, including [disk drive controllers](#), [graphics cards](#), [network cards](#) and [sound cards](#). DMA is also used for intra-chip data transfer in [multi-core processors](#). Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without DMA channels. Similarly, a processing element inside a multi-core processor can transfer data to and from its local memory without occupying its processor time, allowing computation and data transfer to proceed in parallel.

DMA can also be used for "memory to memory" copying or moving of data within memory. DMA can offload expensive memory operations, such as large copies or [scatter-gather](#) operations, from the CPU to a dedicated DMA engine. An implementation example is the [I/O Acceleration Technology](#).

Serial I/O

Serial input/output (SIO) A method of communicating data between devices, typically a computer and its peripherals, the individual data bits being sent sequentially. Serial communication may be *asynchronous*, where the data characters include start and stop bits to delimit the data, or *synchronous*, where such additional bits are omitted and the delimiting of the data depends purely on timing. Asynchronous serial communication is more flexible whereas synchronous serial communication makes better use of the available bandwidth. Asynchronous methods are generally used with dial-up modems or for general connection of simple serial peripherals. Synchronous methods are usually to be found where [leased lines](#) or proprietary interfaces are used.

Until recently SIO was a relatively slow mode, but advances in silicon technology

means that very fast SIO (hundreds of megabits per second) now make SIO a very cost-effective alternative to parallel interfaces. See [USB](#), [Firewire](#), [serial IDE](#).

Chapter 8

8255 Pin Diagram

Let us first take a look at the pin diagram of Intel 8255A –

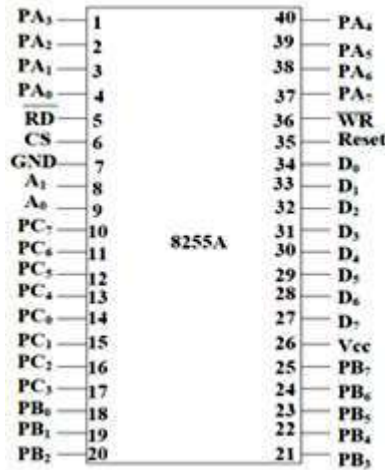


Image 13 pin diagram 8255 A

Now let us discuss the functional description of the pins in 8255A.

Data Bus Buffer

It is a tri-state 8-bit buffer, which is used to interface the microprocessor to the system data bus. Data is transmitted or received by the buffer as per the instructions by the CPU. Control words and status information is also transferred using this bus.

Read/Write Control Logic

This block is responsible for controlling the internal/external transfer of data/control/status word. It accepts the input from the CPU address and control buses, and in turn issues command to both the control groups.

CS

It stands for Chip Select. A LOW on this input selects the chip and enables the communication between the 8255A and the CPU. It is connected to the decoded address, and A₀ & A₁ are connected to the microprocessor address lines.

WR

It stands for write. This control signal enables the write operation. When this signal goes low, the microprocessor writes into a selected I/O port or control register.

RESET

This is an active high signal. It clears the control register and sets all ports in the input mode.

RD

It stands for Read. This control signal enables the Read operation. When the signal is low, the microprocessor reads the data from the selected I/O port of the 8257.

8257

Here is a list of some of the prominent features of 8257 –

- It has four channels which can be used over four I/O devices.
- Each channel has 16-bit address and 14-bit counter.
- Each channel can transfer data up to 64kb.
- Each channel can be programmed independently.
- Each channel can perform read transfer, write transfer and verify transfer operations.
- It generates MARK signal to the peripheral device that 128 bytes have been transferred.
- It requires a single phase clock.
- Its frequency ranges from 250Hz to 3MHz.
- It operates in 2 modes, i.e., **Master mode** and **Slave mode**.

8257 Architecture

The following image shows the architecture of 8257 –

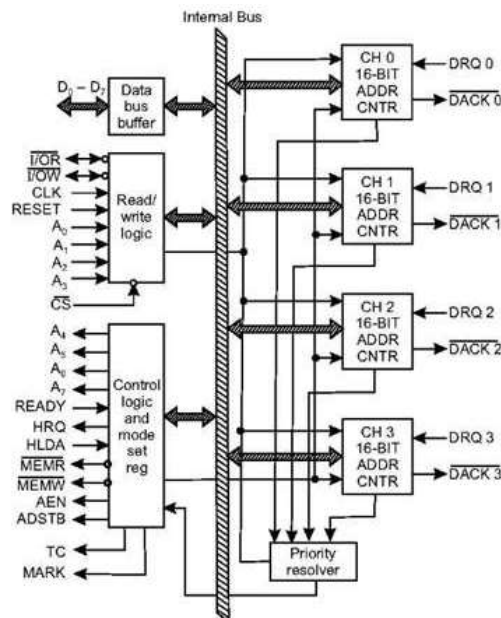


Image 14 Pin diagram of 8257

8257 Pin Description

The following image shows the pin diagram of a 8257 DMA controller —

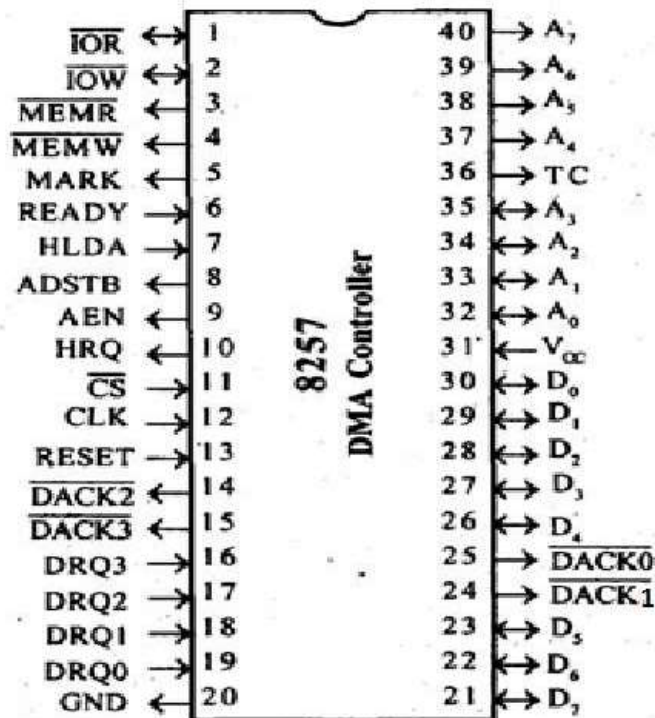


Image 15 8257 pin description

DRQ₀ – DRQ₃

These are the four individual channel DMA request inputs, which are used by the peripheral devices for using DMA services. When the fixed priority mode is selected, then DRQ₀ has the highest priority and DRQ₃ has the lowest priority among them.

DACK₀ – DACK₃

These are the active-low DMA acknowledge lines, which updates the requesting peripheral about the status of their request by the CPU. These lines can also act as strobe lines for the requesting devices.

D₀ – D₇

These are bidirectional, data lines which are used to interface the system bus with the internal data bus of DMA controller. In the Slave mode, it carries command words to 8257 and status word from 8257. In the master mode, these lines are used to send higher byte of the generated address to the latch. This address is further latched using ADSTB signal.

IOR

It is an active-low bidirectional tri-state input line, which is used by the CPU to read internal registers of 8257 in the Slave mode. In the master mode, it is used to read data from the peripheral devices during a memory write cycle.

IOW

It is an active low bi-direction tri-state line, which is used 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 used to load the data to the peripheral devices during DMA memory read cycle.

CLK

It is a clock frequency signal which is required for the internal operation of 8257.

RESET

This signal is used to RESET the DMA controller by disabling all the DMA channels.

A₀ - A₃

These are the four least significant address lines. In the slave mode, they act as an input, which selects 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

It is an active-low chip select line. In the Slave mode, it enables the read/write operations to/from 8257. In the master mode, it disables the read/write operations to/from 8257.

A₄ - A₇

These are the higher nibble of the lower byte address generated by DMA in the master mode.

READY

It is an active-high asynchronous input signal, which makes DMA ready by inserting wait states.

HRQ

This signal is used to receive the hold request signal from the output device. In the slave mode, it is connected with a DRQ input line 8257. In Master mode, it is connected with HOLD input of the CPU.

HLDA

It is the hold acknowledgement signal which indicates the DMA controller that the bus has been granted to the requesting peripheral by the CPU when it is set to 1.

MEMR

It is the low memory read signal, which is used to read the data from the addressed memory locations during DMA read cycles.

MEMW

It is the active-low three state signal which is used to write the data to the addressed memory location during DMA write operation.

ADST

This signal is used to convert the higher byte of the memory address generated by the DMA controller into the latches.

AEN

This signal is used to disable the address bus/data bus.

TC

It stands for 'Terminal Count', which indicates the present DMA cycle to the present peripheral devices.

MARK

The mark will be activated after each 128 cycles or integral multiples of it from the beginning. It indicates the current DMA cycle is the 128th cycle since the previous MARK output to the selected peripheral device.

V_{cc}

It is the power signal which is required for the operation of the circuit.

8279

8279 programmable keyboard/display controller is designed by Intel that interfaces a keyboard with the CPU. The keyboard first scans the keyboard and identifies if any key has been pressed. It then sends their relative response of the pressed key to the CPU and vice-a-versa.

How Many Ways the Keyboard is Interfaced with the CPU?

The Keyboard can be 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 will continue with its main task.

In the **Polled mode**, the CPU periodically reads an internal flag of 8279 to check whether any key is pressed or not with key pressure.

How Does 8279 Keyboard Work?

The keyboard consists of maximum 64 keys, which are interfaced with the CPU by using the key-codes. These key-codes are de-bounced and stored in an 8-byte FIFORAM, which can be accessed by the CPU. If more than 8 characters are entered

in the FIFO, then it means more than eight keys are pressed at a time. This is when the overrun status is set.

If a FIFO contains a valid key entry, then the CPU is interrupted in an interrupt mode else the CPU checks the status in polling to read the entry. Once the CPU reads a key entry, then FIFO is updated, and the key entry is pushed out of the FIFO to generate space for new entries.

Architecture and Description

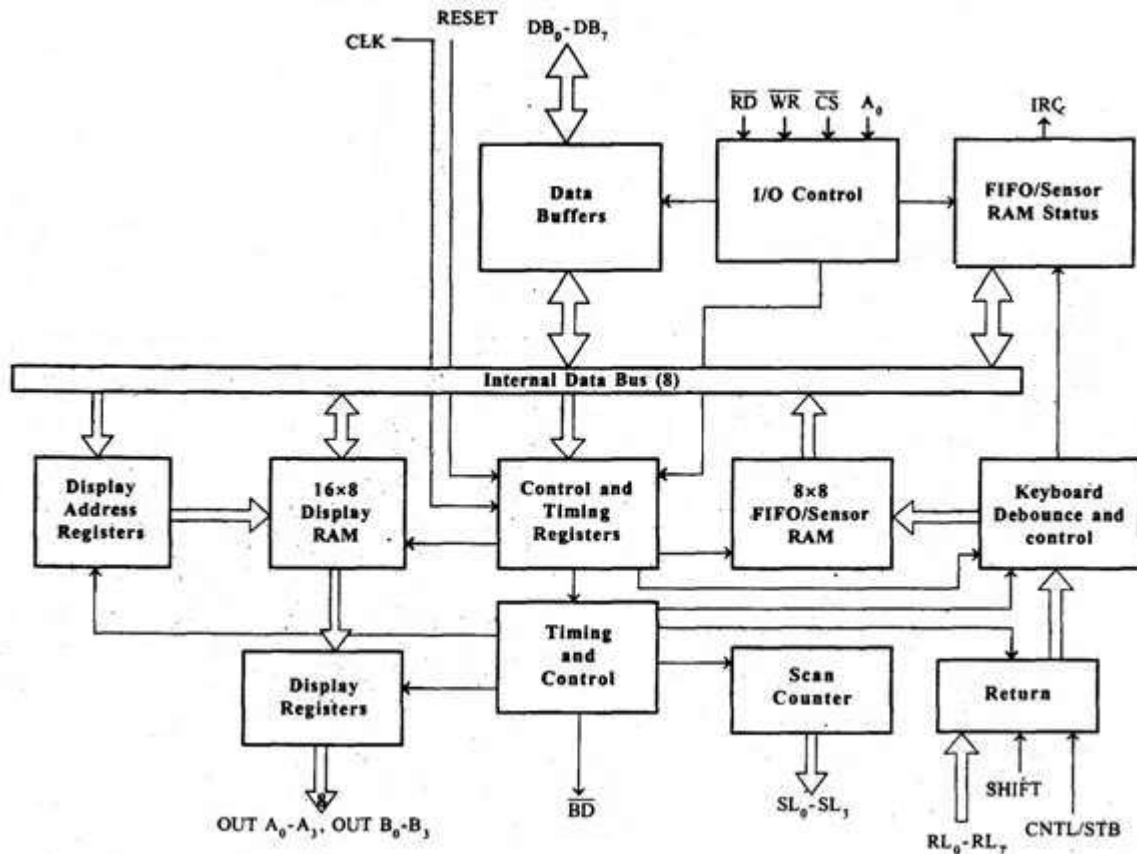


Image 16 block diagram of 8279

I/O Control and Data Buffer

This unit controls the flow of data through the microprocessor. It is enabled only when D is low. Its data buffer interfaces the external bus of the system with the internal bus of the microprocessor. The pins A0, RD, and WR are used for command, status or data read/write operations.

Control and Timing Register and Timing Control

This unit contains registers to store the keyboard, display modes, and other operations as programmed by the CPU. The timing and control unit handles the timings for the operation of the circuit.

Scan Counter

It has two modes i.e. **Encoded mode** and Decoded mode. In the encoded mode, the counter provides the binary count that is to be externally decoded to provide the scan lines for the keyboard and display.

In the **decoded scan mode**, the counter internally decodes the least significant 2 bits and provides a decoded 1 out of 4 scan on SL₀-SL₃.

Return Buffers, Keyboard Debounce, and Control

This unit first scans the key closure row-wise, if found then the keyboard debounce unit debounces the key entry. In case, the same key is detected, then the code of that key is directly transferred to the sensor RAM along with SHIFT & CONTROL key status.

FIFO/Sensor RAM and Status Logic

This unit acts as 8-byte first-in-first-out (FIFO) RAM where the key code of every pressed key is entered into the RAM as per their sequence. The status logic generates an interrupt request after each FIFO read operation till the FIFO gets empty.

In the scanned sensor matrix mode, this unit acts as sensor RAM where its each row is loaded with the status of their corresponding row of sensors into the matrix. When the sensor changes its state, the IRQ line changes to high and interrupts the CPU.

Display Address Registers and Display RAM

This unit consists of display address registers which holds the addresses of the word currently read/written by the CPU to/from the display RAM.

8279 – Pin Description

The following figure shows the pin diagram of 8279 –



Data Bus Lines, DB₀ - DB₇

These are 8 bidirectional data bus lines used to transfer the data to/from the CPU.

CLK

The clock input is used to generate internal timings required by the microprocessor.

RESET

As the name suggests this pin is used to reset the microprocessor.

CS Chip Select

When this pin is set to low, it allows read/write operations, else this pin should be set to high.

A₀

This pin indicates the transfer of command/status information. When it is low, it indicates the transfer of data.

RD, WR

This Read/Write pin enables the data buffer to send/receive 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.

V_{SS}, V_{CC}

These are the ground and power supply lines of the microprocessor.

SL₀ – SL₃

These are the scan lines used to scan the keyboard matrix and display the digits. These lines can be programmed as encoded or decoded, using the mode control register.

RL₀ – RL₇

These are the Return Lines which are connected to one terminal of keys, while the other terminal of the keys is connected to the decoded scan lines. These lines are set to 0 when any key is pressed.

SHIFT

The Shift input line status is stored along with every 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

It stands for blank display. It is used to blank the display during digit switching.

OUTA₀ – OUTA₃ and OUTB₀ – OUTB₃

These are the output ports for two 16x4 or one 16x8 internal display refresh registers. The data from these lines is synchronized with the scan lines to scan the display and the keyboard.

Operational Modes of 8279

There are two modes of operation on 8279 – Input Mode and Output Mode.

Input Mode

This mode deals with the input given by the keyboard and this mode is further classified into 3 modes.

- **Scanned Keyboard Mode** – In this mode, the key matrix can be interfaced using either encoded or decoded scans. In the encoded scan, an 8×8 keyboard or in the decoded scan, a 4×8 keyboard can be interfaced. The code of key pressed with SHIFT and CONTROL status is stored into the FIFO RAM.
- **Scanned Sensor Matrix** – In this mode, a sensor array can be interfaced with the processor using either encoder or decoder scans. In the encoder scan, 8×8 sensor matrix or with decoder scan 4×8 sensor matrix can be interfaced.
- **Strobed Input** – In this mode, when the control line is set to 0, the data on the return lines is stored in the FIFO byte by byte.

Output Mode

This mode deals with display-related operations. This mode is further classified into two output modes.

- **Display Scan** – This mode allows 8/16 character multiplexed displays to be organized as dual 4-bit/single 8-bit display units.
- **Display Entry** – This mode allows the data to be entered for display either from the right side/left side.