NARSIMHA REDDY ENGINEERING COLLEGE
UGC AUTONOMOUS INSTITUTION
Maisammaguda (V), Kompally - 500100, Secunderabad, Telangana State, India
Your roots to success....
UGC - Autonomous Institute
Accredited by NBA & NAAC with 'A' Grade
Approved by AICTE
Permanently affiliated to JNTUH

MICROPROCESSOR AND MICROCONTROLLERS

Mrs. V. Nagalakshmi

Associate Professor
Dept of ECE

CONTENTS

UNIT-I

8086 ARCHITECTURE

NRCM
your roots to success...

UNIT-II

INSTRUCTION SET & ASSEMBLY LANGUAGE PROGRAMMING OF 8086

Instruction Formats

Addressing Modes

Instruction Set

Assembler Directives

Macros

Simple Programs involving Logical Instructions

Simple Programs involving Branch and Call Instructions

Sorting

Evaluating Arithemetic Expressions

String Manipulations

NRCM
your roots to success...

UNIT-III

I/O INTERFACE, INTERFACING WITH ADVANCED DEVICES &COMMUNICATION INTERFACE

8255 PPI

Various Modes of operation of 8255

8255 PPI Interfacing to 8086

Keyboard Interface

Memory Interfacing to 8086

Interrupt Structure of 8086

Vector Interrupt Table, ISR

DOS & BIOS Interrupts

Serial Communication Standards

Serial Data Transfer Schemes

8251 USART Interfacing to 8086

your roots to success...

UNIT-IV

INTRODUCTION TO MICROCONTROLLERS

Overview of 8051

Microcontroller

Architecture

Pin Diagram of 8051,I/O Ports

Memory Organization

Addressing Modes

Instruction set

Simple Programs

NRCM
your roots to success...

UNIT-V

8051 REAL TIME CONTROL

Interrupts

Timer/Counter & Serial Communication

Programming Timer Interrupts

Programming external hardware Interrupts

Programming the serial communication Interrupts

Programming 8051 Timers,Counters

NRCM
your roots to success...

UNIT -1

INTRODUCTION TO 8 BIT/16 BIT MICRO PROCESSORS

1. 8086 flag register

Psw (program status word) is a 16-bit register containing 9 1-bit flags:

- Overflow Flag (OF) - set if the result is too large positive number, or is too small negative number to fit into destination operand.
- Direction Flag (DF) - if set then string manipulation instructions will auto-decrement index registers. If cleared then the index registers will be auto-incremented.
- Interrupt-enable Flag (IF) - setting this bit enables maskable interrupts.
- Single-step Flag (TF) - if set then single-step interrupt will occur after the next instruction.
- Sign Flag (SF) - set if the most significant bit of the result is set.
- Zero Flag (ZF) - set if the result is zero.
- Auxiliary carry Flag (AF) - set if there was a carry from or borrow to bits 0-3 in the AL register.
- Parity Flag (PF) - set if parity (the number of "1" bits) in the low-order byte of the result is even.
- Carry Flag (CF) - set if there was a carry from or borrow to the most significant bit during last result calculation

NRCM
your roots to success...

1.2.4. 8086 pin diagram:

Figure shows the Pin diagram of 8086. The description follows it.

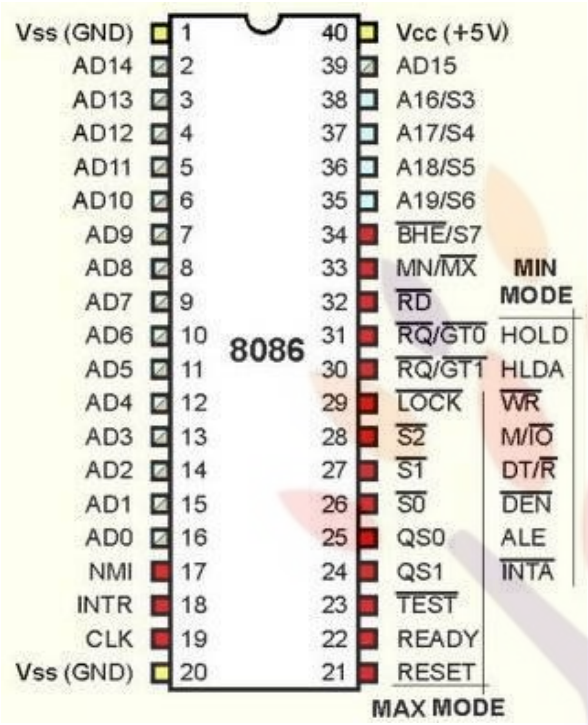


Fig 1.4. 8086 pin diagram

- The Microprocessor 8086 is a 16-bit CPU available in different clock rates and packaged in a 40 pin CERDIP or plastic package.
- The 8086 operates in single processor or multiprocessor configuration to achieve high performance. The pins serve a particular function in minimum mode (single processor mode) and other function in maximum mode configuration (multiprocessor mode).
- The 8086 signals can be categorized in three groups.
    - The first are the signal having common functions in minimum as well as maximum mode.
    - The second are the signals which have special functions for minimum mode
    - The third are the signals having special functions for maximum mode.
- The following signal descriptions are common for both modes.
- **AD15-AD0:** These are the time multiplexed memory I/O address and data lines.
    - Address remains on the lines during T1 state, while the data is available on the data bus during T2, T3, Tw and T4. These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.

- **A19/S6,A18/S5,A17/S4,A16/S3 :** These are the time multiplexed address and status lines.
    - During T1 these are the most significant address lines for memory operations.
    - During I/O operations, these lines are low.
    - During memory or I/O operations, status information is available on those lines for T2,T3,Tw and T4.
    - The status of the interrupt enable flag bit is updated at the beginning of each clock cycle.
    - The S4 and S3 combinely indicate which segment register is presently being used for memory accesses as in below fig.
    - These lines float to tri-state off during the local bus hold acknowledge. The status line S6 is always low.
    - The address bit are separated from the status bit using latches controlled by the ALE signal.

S4	S3	Indication
0	0	Alternate Data
0	1	Stack
1	0	Code or None
1	1	Data
0	0	Whole word
0	1	Upper byte from or to even address
1	0	Lower byte from or to even address

Table 1.1. 8086 status signals

- **BHE/S7:** The bus high enable is used to indicate the transfer of data over the higher order (D15-D8) data bus as shown in table. It goes low for the data transfer over D15-D8 and is used to derive chip selects of odd address memory bank or peripherals. BHE is low during T1 for read, write and interrupt acknowledge cycles, whenever a byte is to

be transferred on higher byte of data bus. The status information is available during T2, T3 and T4. The signal is active low and tristated during hold. It is low during T1 for the first pulse of the interrupt acknowledges cycle.

- **RD – Read:** This signal on low indicates the peripheral that the processor is performing memory or I/O read operation. RD is active low and shows the state for T2, T3, Tw of any read cycle. The signal remains tristated during the hold acknowledge.

- **READY:** This is the acknowledgement from the slow device or memory that they have completed the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the 8086. the signal is active high.

- **INTR-Interrupt Request:** This is a triggered input. This is sampled during the last clock cycles of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle. This can be internally masked by resulting the interrupt enable flag. This signal is active high and internally synchronized.

- **TEST:** This input is examined by a 'WAIT' instruction. If the TEST pin goes low, execution will continue, else the processor remains in an idle state. The input is synchronized internally during each clock cycle on leading edge of clock.

- **CLK- Clock Input:** The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle.
- **ALE- Address Latch Enable**. A HIGH on this line causes the lower order 16bit address bus to belatched that stores the addresses and then, the lower order 16bit of the address bus can be usedas data bus.
- **NMI- non-maskable interrupt**: an edge triggered input which causesan interrupt request to the MP. A subroutine is vectored to via an interrupt vectorlookup table located in system memory. NMI is not maskable internallyby software.
- **RESET:** causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. Itrestarts execution
- **MN/MX -MINIMUM/MAXIMUM**: indicates what mode the processor is to operate in. The two modes arediscussed in the following sections.

- **M/IO:** Differentiate between the Memory and I/O operation. A LOW on this pin indicated I/Ooperation and a HIGH indicated a Memory Operation
- **HOLD:** The 8086 has a pin called HOLD. This pin is used by external devices to gain control ofthe busses.
- **HLDA:**When the HOLD signal is activated by an external device, the 8086 stops executing instructionsand stops using the busses. This would allow external devices to control the information on the8086 MINIMUM AND MAXIMUM MODES of operation MN/MX
  - Minimum mode The 8086 processor works in a single processor environment. All control signals for memory and I/O are generated by the microprocessor.
  - Maximum mode is designed to be used when a coprocessor exists in the system.
  - 8086 works in a multiprocessor environment. Control signals for memory and I/O are generated by an external BUS Controller.

1.2.5. 8086 Minimum Mode System

When the Minimum mode operation is selected, the 8086

Provides all control signals needed to implement the memory and I/O interface.

• The minimum mode signal can be divided into the following basic groups: address/data bus, status, control, interrupt and DMA.

• Address/Data Bus: these lines serve two functions. As an address bus is 20 bits long and consists of signal lines A0 through A19. A19 represents the MSB and A0 LSB. A 20bit address gives the 8086 a 1Mbyte memory address space. More over it has an independent I/O address space which is 64K bytes in length.

The 16 data bus lines D0 through D15 are actually multiplexed with address lines A0 through A15 respectively. By multiplexed we mean that the bus work as an address bus during first machine cycle and as a data bus during next machine cycles. D15 is the MSB and D0 LSB.

• When acting as a data bus, they carry read/write data for memory, input/output data for I/O devices, and interrupt type codes from an interrupt controller.

your roots to success...

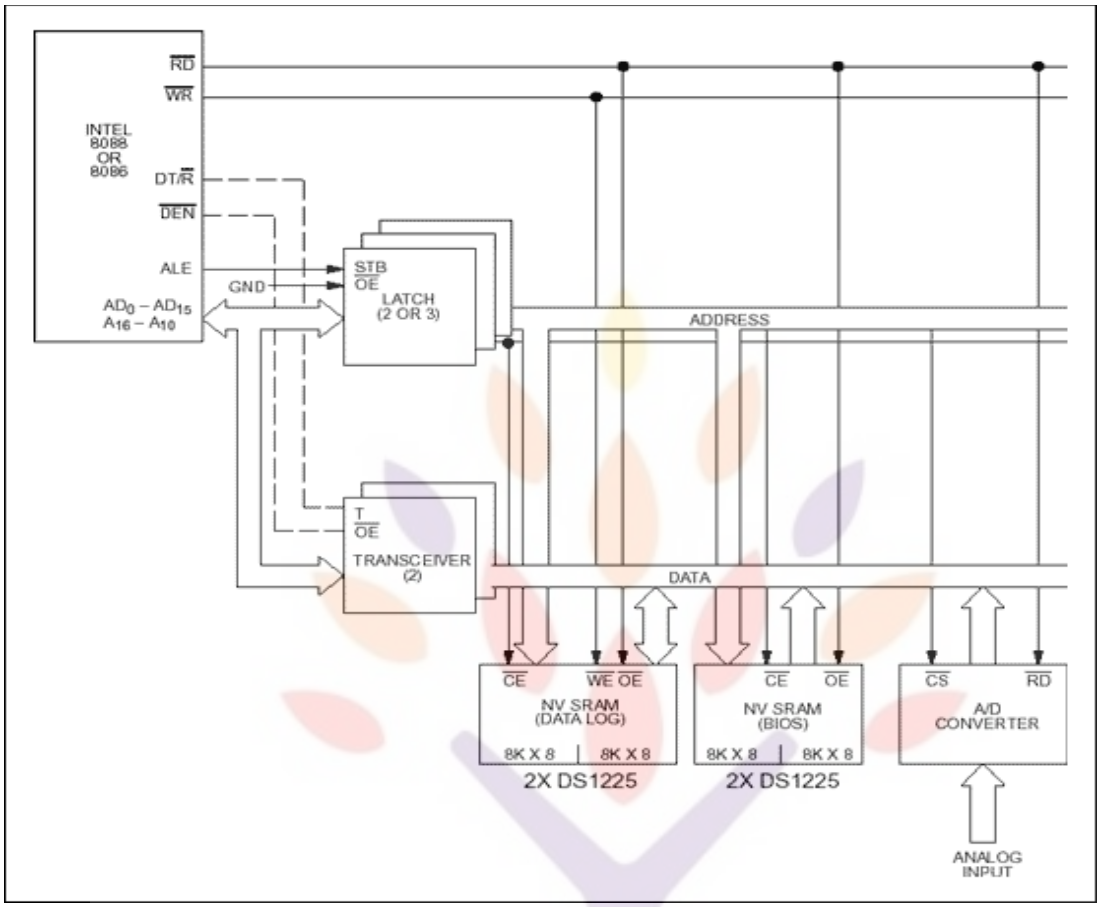


Fig 1.5. 8086 minimum mod system

Status signal: The four most significant address lines A19 through A16 are also multiplexed but in this case with status signals S6 through S3. These status bits are output on the bus at the same time that data are transferred over the other bus lines.

• Bit S4 and S3 together from a 2 bit binary code that identifies which of the 8086 internal segment registers are used to generate the physical address that was output on the address bus during the current bus cycle.

• Code S4 S3 = 00 identifies a register known as extra segment register as the source of the segment address.

S4	S3	Segment Register
0	0	Extra
0	1	Stack
1	0	Code / none
1	1	Data

your roots to success...

Status line S5 reflects the status of another internal characteristic of the 8086. It is the logic level of the internal enable flag. The last status bit S6 is always at the logic 0 level.

• Control Signals: The control signals are provided to support the 8086 memory I/O interfaces. They control functions such as when the bus is to carry a valid address in which direction data are to be transferred over the bus, when valid write data are on the bus and when to put read data on the system bus.

ALE is a pulse to logic 1 that signals external circuitry when a valid address word is on the bus. This address must be latched in external circuitry on the 1-to-0 edge of the pulse at ALE.

• Another control signal that is produced during the bus cycle is BHE bank high enable. Logic 0 on this used as a memory enable signal for the most significant byte half of the data bus D8 through D1. These lines also serve a second function, which is as the S7 status line.

• Using the M/IO and DT/R lines, the 8086 signals which type of bus cycle is in progress and in which direction data are to be transferred over the bus.

The logic level of M/IO tells external circuitry whether a memory or I/O transfer is taking place over the bus. Logic 1 at this output signals a memory operation and logic 0 an I/O operation.

• The direction of data transfer over the bus is signaled by the logic level output at DT/R. When this line is logic 1 during the data transfer part of a bus cycle, the bus is in the transmit mode. Therefore, data are either written into memory or output to an I/O device.

• On the other hand, logic 0 at DT/R signals that the bus is in the receive mode. This corresponds to reading data from memory or input of data from an input port

• The signal read RD and writes WR indicates that a read bus cycle or a write bus cycle is in progress. The 8086 switches WR to logic 0 to signal external device that valid write or output data are on the bus.

• On the other hand, RD indicates that the 8086 is performing a read of data of the bus. During read operations, one other control signal is also supplied. This is DEN (data enable) and it signals external devices when they should put data on the bus.

• There is one other control signal that is involved with the memory and I/O interface. This is the READY signal. READY signal is used to insert wait states into the bus cycle such that it is extended by a number of clock periods. This signal is provided by an external clock generator device and can be supplied by the memory or I/O subsystem to signal the 8086 when they are ready to permit the data transfer to be completed.

• Interrupt signals: The key interrupt interface signals are interrupt request (INTR) and interrupt acknowledge (INTA).

• INTR is an input to the 8086 that can be used by an external device to signal that it needs to be serviced. Logic 1 at INTR represents an active interrupt request. When an interrupt request has been recognized by the 8086, it indicates this fact to external circuit with pulse to logic 0 at the INTA output.

• The TEST input is also related to the external interrupt interface. Execution of a WAIT instruction causes the 8086 to check the logic level at the TEST input.

• If the logic 1 is found, the MPU suspend operation and goes into the idle state. The 8086 no longer executes instructions; instead it repeatedly checks the logic level of the TEST input waiting for its transition back to logic 0.

• As TEST switches to 0, execution resume with the next instruction in the program. This feature can be used to synchronize the operation of the 8086 to an event in external hardware.

• There are two more inputs in the interrupt interface: the non makeable interrupt NMI and the reset interrupt RESET.

• On the 0-to-1 transition of NMI control is passed to a non maskable interrupt service routine. The RESET input is used to provide a hardware reset for the 8086. Switching RESET to logic 0 initializes the internal register of the 8086 and initiates a reset service routine.

DMA Interface signals: The direct memory access DMA interface of the 8086 minimum mode consist of the HOLD and HLDA signals.

• When an external device wants to take control of the system bus, it signals to the 8086 by switching HOLD to the logic 1 level. At the completion of the current bus cycle, the 8086 enters the hold state. In the hold state, signal lines AD0 through AD15, A16 /S3 through A19 /S6 , BHE, M/IO, DT/R, RD, WR, DEN and INTR are all in the high Z state. The 8086 signals external device that it is in this state by switching its HLDA output to logic 1 level.

your roots to success...

\

Timing diagrams of minimum mode:

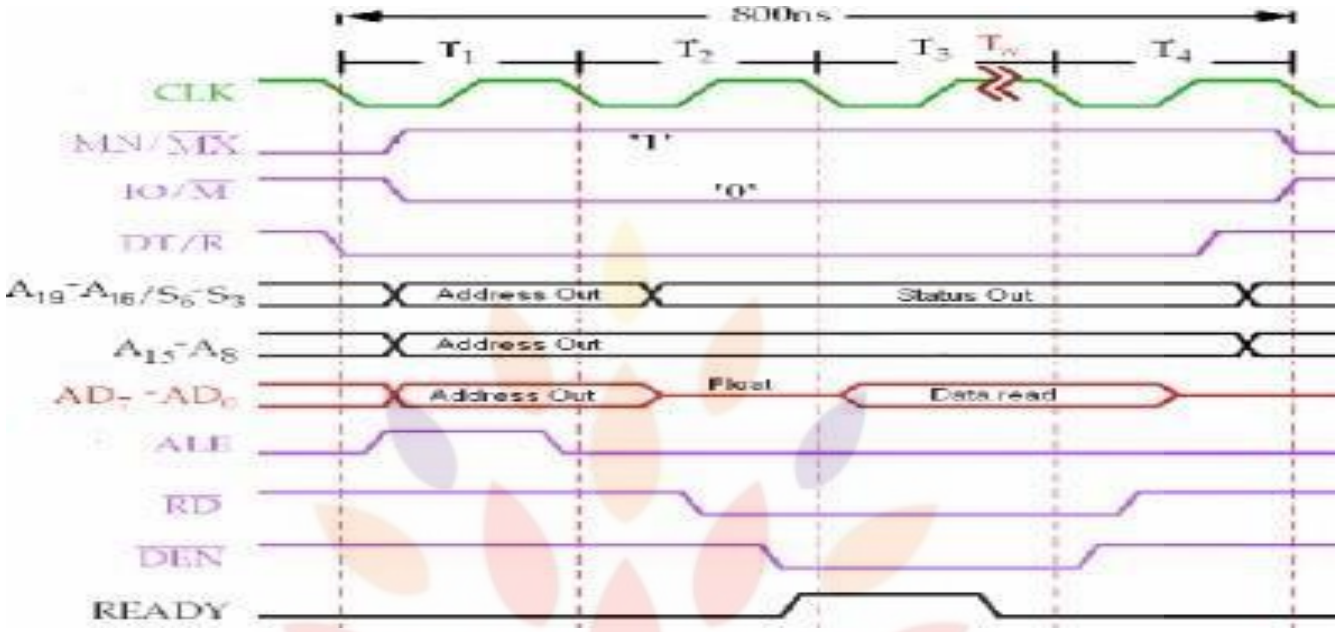


Fig 1.6. Minimum mode timing diagram in read cycle

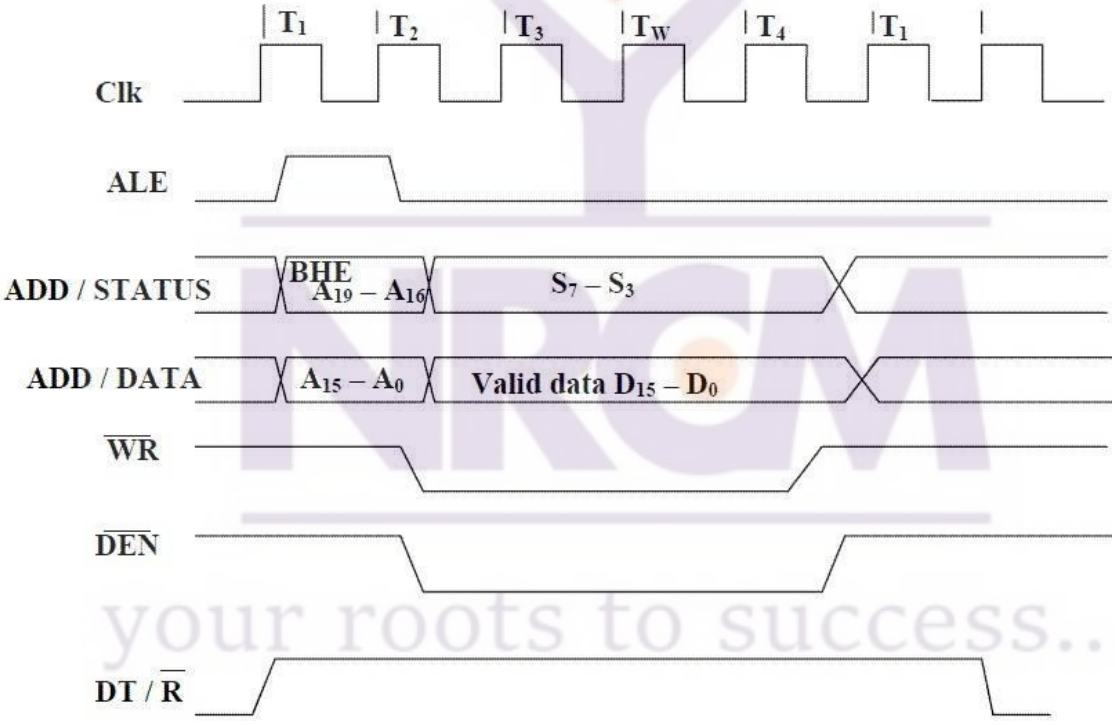


Fig 1.7. Minimum nod timing signals in write cycle

1.2.6. Maximum Mode 8086 System

• In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.

• In this mode, the processor derives the status signal S2, S1, S0. Another chip called bus controller derives the control signal using this status information.

• In the maximum mode, there may be more than one microprocessor in the system configuration.

• The components in the system are same as in the minimum mode system.

The basic function 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 by the processor on the status lines.

• The bus controller chip has input lines S2 , S1 , S0 and CLK. These inputs to 8288 are driven by CPU.

• It derives the outputs ALE, DEN, DT/R, MRDC, 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 cascade 8259A, else it acts as peripheral data enable used in the multiple bus configurations.

• INTA pin 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 address port.

• The MRDC, MWTC are memory read command and memory write command signals respectively and may be used as memory read or write signals.

• All these command signals instructs 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.

NRCM your roots to success

• The maximum mode system timing diagrams are 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 signal used and the available control and advanced command signals.

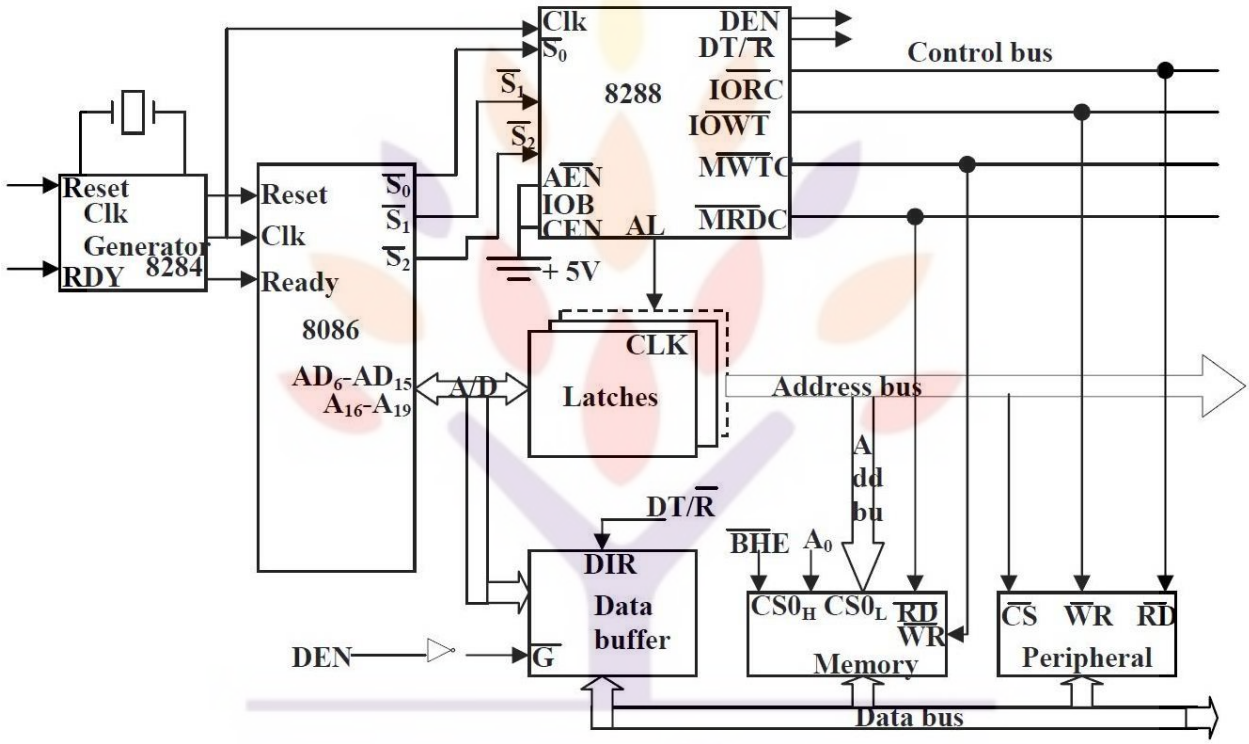


Fig 1.8. 8086 maximum mode system

Here the only difference between in timing diagram between minimum mode and maximum mode is the status signals used and the available control and advanced command signals.

• S0 , S1 , S2 are set at the beginning of bus cycle.8288 bus controller will output a pulse as on the ALE and apply a required signal to its DT / R pin during T1.

In T2 , 8288 will set DEN=1 thus enabling transceivers, and for an input it will activate MRDC or IORC. These signals are activated until T4. For an output, the AMWC or AIOWC is activated from T2

to T4 and MWTC or IOWC is activated from T3 to T4.

• The status bit S0 to S2 remains active until T3 and become passive during T3 and T4

• If reader input is not activated before T3, wait state will be inserted between T3 and T4.

Timings for RQ/ GT Signals: The request/grant response sequence contains a series of three pulses. The request/grant pins are checked at each rising pulse of clock input.

• When a request is detected and if the condition for HOLD request is satisfied, the processor issues a grant pulse over the RQ/GT pin immediately during T4 (current) or T1 (next) state.

• When the requesting master receives this pulse, it accepts the control of the bus, it sends a release pulse to the processor using RQ/GT pin.

Timing diagrams of max mode.

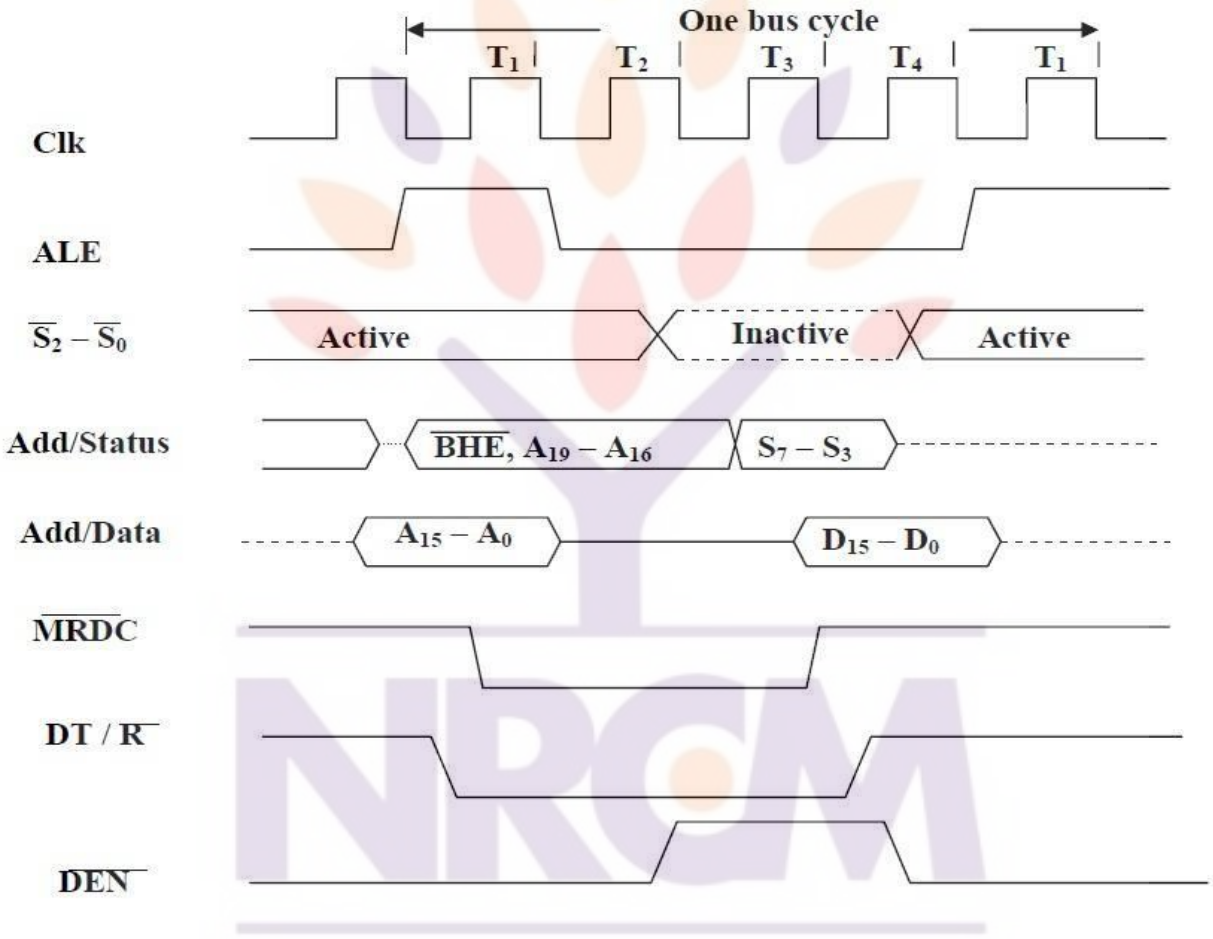


Fig 1.9. Timing diagram of max mode system in read cycle

your roots to success...

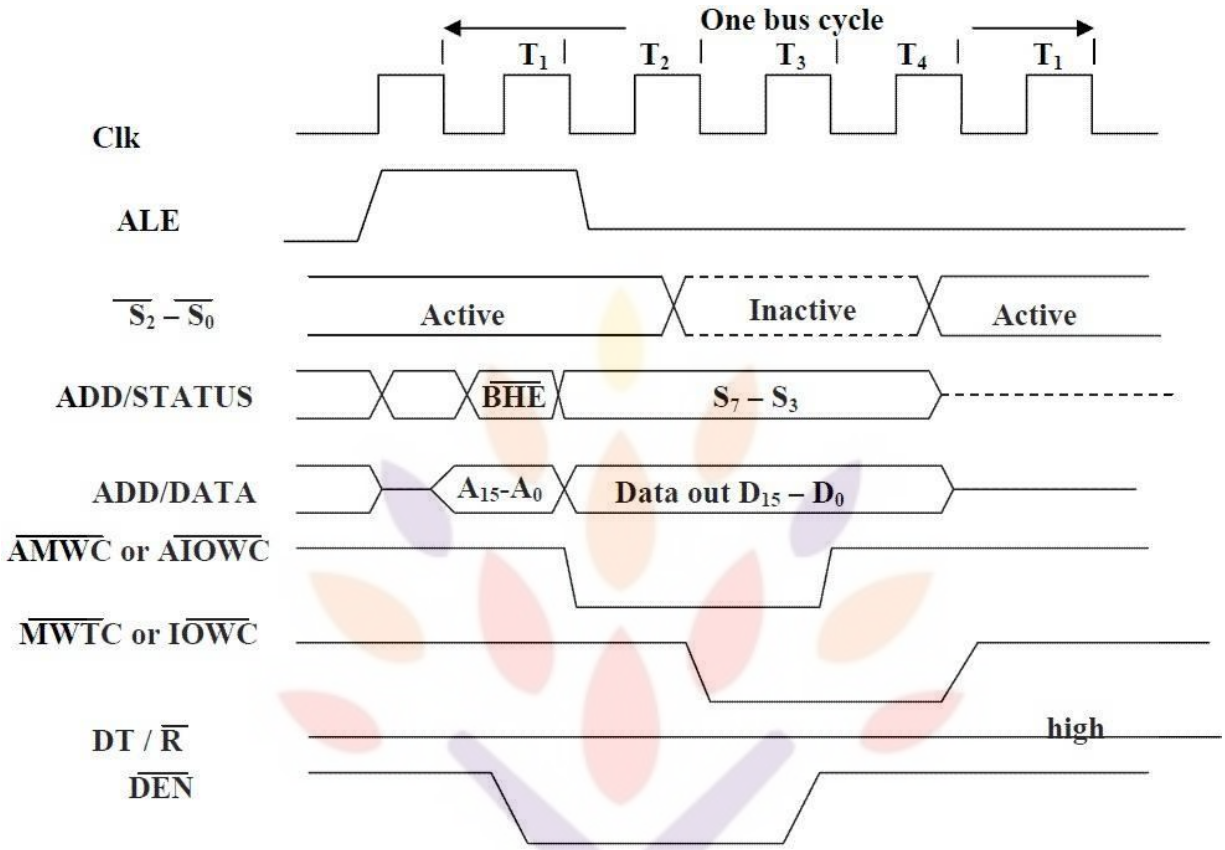


Fig 1.10. Timing diagram of max mode system in write cycle

1.2.7. 8086 Interrupts

The processor has the following interrupts:

• INTR is a maskable hardware interrupt. The interrupt can be enabled/disabled using STI/CLI instructions or using more complicated method of updating the FLAGS register with the help of the POPF instruction.

• When an interrupt occurs, the processor stores FLAGS register into stack, disables further interrupts, fetches from the bus one byte representing interrupt type, and jumps to interrupt processing routine address of which is stored in location 4 * <interrupt type>. Interrupt processing routine should return with the IRET instruction.

NMI is a non-maskable interrupt. Interrupt is processed in the same way as the INTR interrupt. Interrupt type of the NMI is 2, i.e. the address of the NMI processing routine is stored in location 0008h. This interrupt has higher priority then the maskable interrupt.

• Software interrupts can be caused by:

• INT instruction - breakpoint interrupt. This is a type 3 interrupt.

• INT <interrupt number>instruction - any one interrupt from available 256 interrupts.

• INTO instruction - interrupt on overflow.

Single-step interrupt - generated if the TF flag is set. This is a type 1 interrupt. When the CPU processes this interrupt it clears TF flag before calling the interrupt processing routine.

• Processor exceptions: Divide Error (Type 0), Unused Opcode (type 6) and Escape opcode (type 7).

• Software interrupt processing is the same as for the hardware interrupts.

NRCM
your roots to success...

UNIT-2

ASSEMBLY LANGUAGE PROGRAMMING

2.1. Addressing modes of 8086

Various addressing modes of 8086/8088

1) Register Addressing mode
2) Immediate Addressing mode
3) Register indirect addressing mode
4) Direct Addressing mode
5) Indexed Addressing mode
6) Base Relative Addressing mode
7) Base Indexed Addressing mode

Register Addressing Mode

Data transfer using registers is called register addressing mode. Here operand value is present in register. For example

MOV AL,BL;

MOV AX,BX;

Immediate Addressing mode

When data is stored in code segment instead of data segment immediate addressing mode is used. Here operand value is present in the instruction. For example

MOV AX, 12345;

Direct Addressing mode

When direct memory address is supplied as part of the instruction is called direct addressing mode. Operand offset value with respect to data segment is given in instruction. For example

MOV AX, [1234];

NRCM your roots to success...

ADD AX, [1234];

Register indirect addressing mode: Here operand offset is given in a cpu register. Register used are BX, SI(source index), DI(destination index), or BP(base pointer). BP holds offset w.r.t Stack segment, but SI,DI and BX refer to data segment. For example

MOV [BX],AX;

ADD AX, [SI];

Indexed Addressing mode

Here operand offset is given by a sum of a value held in either SI, or DI register and a constant displacement specified as an operand. For example

Lets take arrays as an example. This is very efficient way of accessing arrays.

My_array DB '1', '2', '3','4,'5';

MOV SI, 3;

MOV AL, My_array[3];

So AL holds value 4.

Base Relative addressing mode

Operand offset given by a sum of a value held either in BP, or BX and a constant offset specified as an operand. For example

MOV AX,[BP+1];

JMP [BX+1];

Base Indexed

Here operand offset is given by sum of either BX or BP with either SI or DI. For example

your roots to success...

MOV AX, [BX+SI]

JMP [BP+DI]

2.2. 8086 Assembler directives

ASSUME Directive - The ASSUME directive isused to tell the assembler that the name of the logical segmentshould be used for a specified segment. The 8086 works directly with only 4 physical segments: a Code segment, a data segment, a stack segment, and an extra segment.

Example:

ASUME CS: CODE; This tells the assembler that the logical segment named CODE contains the ; instruction statements for the program and should be treated as a code segment.

ASUME DS: DATA; This tells the assembler that for any instruction which refers to a data in the ;data segment, data will found in the logical segment DATA.

DB - DB directive is used to declare a bytetype variable or to store a byte in memory location.

Example:

1. PRICE DB 49h, 98h, 29h ;Declare an array of 3 bytes, named as PRICE and initialize.

2. NAME DB 'ABCDEF' ;Declare an array of 6 bytes and initialize with ASCII code for letters

3. TEMP DB 100 DUP(?) ;Set 100 bytes of storage in memory and give it the name as TEMP, ;but leave the 100 bytes uninitialized. Program instructions will load values into these locations

DW - The DW directive is used to define a variable of type word or to reserve storage location of type word in memory.

Example:

MULTIPLIER DW437Ah ; this declares a variable of type word and named it as MULTIPLIER. ;This variable is initialized with the value 437Ah when it is loaded into memory to run.

EXP1 DW 1234h, 3456h, 5678h ; this declares an array of 3 words and initialized with specified ;values.

STOR1 DW 100 DUP(0); Reserve an array of 100 words of memory and initialize all words with

;0000.Array is named as STOR1

END - END directive is placed after the last statement of a program to tell the assembler that this is the end of the program module. The assembler will ignore any statement after an END directive. Carriage return is required after the END directive.

ENDP - ENDP directive is used along with the name of the procedure to indicate the end of a procedure to the assembler

Example:

SQUARE_NUM PROCE ; It start the procedureSome steps to find the square root of a number

ENDS - This ENDS directive is used with name of the segment to indicate the end of that logic segment.

Example:

CODE SEGMENT ;Hear it Start the logic segment containing code

; Some instructions statements to perform the logical

;operation

CODE ENDS ;End of segment named as CODE

EQU - This EQU directive is used to give a name to some value or to a symbol. Each time the assembler finds the name in the program, it will replace the name with the value or symbol you given to that name.

Example:

FACTOR EQU 03H ; you has to write this statement at the starting of your program and later in ;the program you can use this as follows

ADD AL, FACTOR; When it codes this instruction the assembler will code it as ADDAL, 03H ;The advantage of using EQU in this manner is, if FACTOR is used many no of times in a ;program and you want to change the value, all you had to do is change the EQU statement at ;beginning, it will changes the rest of all.

EVEN - This EVEN directive instructs the assembler to increment the location of the counter to the next even address if it is not already in the even address. If the word is at even address 8086 can read a memory in 1 bus cycle. If the word starts at an odd address, the 8086 will take 2 bus cycles to get the data. A series of words can be read much more quickly if they are at even address. When EVEN is used the location counter will simply incremented to next address and NOP instruction is inserted in that incremented location

GROUP - The GROUP directive is used to group the logical segments named after the directive into one logical group segment.

INCLUDE - This INCLUDE directive is used to insert a block of source code from the named file into the current source module.

PROC - The PROC directive is used to identify the start of a procedure. The term near or far is used to specify the type of the procedure.

Example:

SMART PROC FAR; This identifies that the start of a procedure named as SMART and ;instructs the assembler that the procedure is far .

SMART ENDP;This PROC is used with ENDP to indicate the break of the procedure.

PTR - This PTR operator is used to assign a specific type of a variable or to a label.

Example:

INC [BX] ; This instruction will not know whether to increment the byte pointed to by BX or a ;word pointed to by BX.

INC BYTE PTR [BX] ;increment the byte pointed to by BX

This PTR operator can also be used to override the declared type of variable. If we want to access the a byte in an

Array WORDS DW 437Ah, 0B97h,

MOV AL, BYTE PTR WORDS

PUBLIC - The PUBLIC directive is used to instruct the assembler that a specified name or label will be accessed from other modules.

Example:

PUBLIC DIVISOR, DIVIDEND; these two variables are public so these are available to all ;modules. If an instruction in a module refers to a variable in another assembly module, we can ;access that module by declaring as EXTRN directive.

TYPE - TYPE operator instructs the assembler to determine the type of a variable and determines the number of bytes specified to that variable.

Example:

Byte type variable – assembler will give a value 1

Word type variable – assembler will give a value 2

Double word type variable – assembler will give a value 4

ADD BX, TYPE WORD_ ARRAY; hear we want to increment BX to point to next word in an array of words.

2.3. Instruction set of 8086

DATA TRANSFER INSTRUCTIONS

GENERAL – PURPOSE BYTE OR WORD TRANSFER INSTRUCTIONS:

MOV

PUSH

POP

XCHG

XLAT

SIMPLE INPUT AND OUTPUT PORT TRANSFER INSTRUCTIONS

IN

OUT

SPECIAL ADDRESS TRANSFER INSTRUCTIONS

LEA

LDS

LES

FLAG TRANSFER INSTRUCTIONS:

LAHF

SAHF

PUSHF

POPF

ADITION INSTRUCTIONS:

ADD

ADC

INC

AAA

DAA

SUBTSUBTRACTION INSTRUCTIONS:

NRCM
your roots to success...

SUB

SBB

DEC

NEG

CMP

AAS

DAS

MULTIPLICATION INSTRUCTIONS:

MUL

IMUL

AAM

DIVISION INSTRUCTIONS:

DIV

IDIV

AAD

CBW

CWD

LOGICAL INSTRUCTIONS:

NOT

AND

OR

XOR

TEST

SHIFT INSTRUCTIONS:

SHL / SAL

NRCM
your roots to success...

SHR

SAR

PROGRAM EXECUTION TRANSFER INSTRUCTIONS

UNCONDITIONAL TRANSFER INSTRUCTIONS:

CALL

RET

JMP

CONDITIONAL TRANSFER INSTRUCTIONS:

JA / JNBE

JAE / JNB

JB / JNAE

JBE / JNA

JC/ JNC

JE / JZ

JG / JNLE

JGE / JNL

JL / JNGE

JLE / JNG

JNE / JNZ

JNO

JNP / JPO

JNS

JO

ITERATION CONTROL INSTRUCTIONS:

LOOP

LOOPE / LOOPZ

your roots to success...

LOOPNE / LOOPNZ

JCXZ

INTERRUPT INSTRUCTIONS:

INT

INTO

IRET

PROCESS CONTROL INSTRUCTIONS

FLAG SET / CLEAR INSTRUCTIONS:

STC

CLC

CMC

STD

CLD

STI

CLI

EXTERNAL HARDWARE SYNCHRONIZATION INSTRUCTIONS:

HLT

WAIT

ESC

LOCK

NOP

AAA Instruction - ASCII Adjust after Addition

AAD Instruction - ASCII adjust before Division

AAM Instruction - ASCII adjust after Multiplication

AAS Instruction - ASCII Adjust for Subtraction

AAA Instruction -AAA converts the result of the addition of two valid unpacked BCD digits to a

valid 2-digit BCD number and takes the AL register as its implicit operand. Two operands of the addition must have its lower 4 bits contain a number in the range from 0-9.The AAA instruction then adjust AL so that it contains a correct BCD digit. If the addition produce carry (AF=1), the AH register is incremented and the carry CF and auxiliary carry AF flags are set to 1. If the addition did not produce a decimal carry, CF and AF are cleared to 0 and AH are not altered. In both cases the higher 4 bits of AL are cleared to 0. AAA will adjust the result of the two ASCII characters that were in the range from 30h ("0") to 39h("9").This is because the lower 4 bits of those character fall in the range of 0-9.The result of addition is not a ASCII character but it is a BCD digit.

AAM Instruction - AAM converts the result of the multiplication of two valid unpacked BCD digits into a valid 2-digit unpacked BCD number and takes AX as an implicit operand. To give a valid result the digits that have been multiplied must be in the range of 0 – 9 and the result should have been placed in the AX register. Because both operands of multiply are required to be 9 or less, the result must be less than 81 and thus is completely contained in AL. AAM unpacks the result by dividing AX by 10, placing the quotient (MSD) in AH and the remainder (LSD) in AL.

AAS Instruction - AAS converts the result of the subtraction of two valid unpacked BCD digits to a single valid BCD number and takes the AL register as an implicit operand. The two operands of the subtraction must have its lower 4 bit contain number in the range from 0 to 9 .The AAS instruction then adjust AL so that it contain a correct BCD digit.

2.4. Assembly language programs

1. Addition of two 16-bit numbers-signed & unsigned

Registers used: AX,DS

Flags affected: AF,CF,OF,PF,SF,ZF

Program:

```
ASSUME CS:CODE,DS:DATA

DATA SEGMENT

OPR1 DW 4269H

OPR2 DW 1000H

RES DW ?

DATA ENDS
```

NRCM your roots to success...

```
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
MOV AX,OPR1
ADD AX,OPR2
MOV RES,AX
MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Result:

INPUT: OPR1 = 4269H

OPR2 = 1000H

OUTPUT: RES = 5269H

2. Multiplication of two 16-bit unsigned numbers

Registers used: AX,DS

Flags affected: OF,CF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
OPR1 DW 2000H
OPR2 DW 4000H
```

your roots to success...

```
RESLW DW ?
RESHW DW ?
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
MOV AX,OPR1
MUL OPR2
MOV RESLW,AX
MOV RESHW,DX
MOV AH,4CH
INT 21H
CODE ENDS
END  START
END
```

Result:

INPUT: OPR1 = 2000H

OPR2 = 4000H

OUTPUT: RESLW = 0000H (AX)

RESHW = 0800H (DX)

NRCM
your roots to success...

3. division of unsigned numbers

Registers used: AX,DS

Flags affected: IF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
OPR1 DW 2C58H
OPR2 DB 56H
RESQ DB ?
RESR DB ?
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
MOV AX,OPR1
DIV OPR2
MOV RESQ,AL
MOV RESR,AH
MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

NRCM
your roots to success...

Result:

INPUT: OPR1 = 2C58H (DIVIDEND)

OPR2 = 56H (DIVISOR)

OUTPUT: RESQ = 84H (AL)

RESR = 00H (AH)

4. Logical and operation

Registers used: AX,DS

Flags affected: PF,SF,ZF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
OPR1 DW 6493H
OPR2 DW 1936H
RES DW ?
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
MOV AX,OPR1
AND AX,OPR2
MOV RES,AX
MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Result:

NRCM your roots to success...

INPUT: OPR1 = 6493H

OPR2 = 1936H

OUTPUT: RES = 0012H

5. Shift arithmetic / logical left operation

Registers used: AX,DS

Flags affected: SF,ZF,PF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
OPR1 DW 1639H
RES DW ?
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
MOV AX,OPR1
SAL AX,01H ←(or)→ SHL AX,01H
MOV RES,AX
MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

NRCM your roots to success...

Result:

INPUT: OPR1 = 1639H

OUTPUT: RES = 2C72H

6. Rotate right with carry

Registers used: AX,DS

Flags affected: CF,OF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
OPR1 DW 1639H
RES DW ?
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
MOV DS,AX
MOV AX,OPR1
RCR AX,01H
MOV RES,AX
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

NRCM
your roots to success...

END

Result:

INPUT: OPR1 = 1639H

OUTPUT: RES = 0B1CH

7. Ascending order

Registers used: AX,DS,ES,SI,DI

Flags affected: AX,DS,SI,CX,DX

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
LIST DW 05H,04H,01H,03H,02H
COUNT EQU 05H
DATA ENDS
CODE SEGMENT
START:MOV AX,DATA
MOV DS,AX
MOV DX,COUNT-1
BACK:MOV CX,DX
MOV SI,OFFSET LIST
AGAIN:MOV AX,[SI]
CMP AX,[SI+2]
JC GO
XCHG AX,[SI+2]
XCHG AX,[SI]
```

NRCM your roots to success...

```
GO:INC SI
INC SI
LOOP AGAIN
DEC DX
JNZ BACK
MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Result:

INPUT: LIST (DS: 0000H) = 05H,04H,01H,03H,02H

OUTPUT: LIST (DS: 0000H) = 01H,02H,03H,04H,05H

8. Packed bcd to unpacked

bcd Registers used:

AX,DS,BL,CL **Flags affected:**

PF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
BCD DB 49H
COUNT DB 04H
UBCD1 DB ?
UBCD2 DB ?
DATA ENDS
```

NRCM your roots to success...

```
CODE SEGMENT
START:MOV AX,DATA
MOV DS,AX
MOV AL,BCD
MOV BL,AL
AND AL,0FH
MOV UBCD1,AL
MOV AL,BL
AND AL,0F0H
MOV CL,COUNT
ROR AL,CL
MOV UBCD2,AL
MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Result:

INPUT: BCD = 49

OUTPUT: UBCD1 = 09

UBCD2 = 04

your roots to success...

9. Reverse string

Registers used: AX,DS,SI,DI,CL

Flags affected: ZF,PF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
STR DB 01H,02H,03H,04H
COUNT EQU 02H
DATA ENDS
CODE SEGMENT
START:MOV AX,DATA
MOV DS,AX
MOV CL,COUNT
MOV SI,OFFSET STR
MOV DI,0003H
BACK:MOV AL,[SI]
XCHG [DI],AL
MOV [SI],AL
INC SI
DEC DI
DEC CL
JNZ BACK
MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Result:

NRCM
your roots to success...

INPUT: STR (DS:0000H) = 01H,02H,03H,04H

OUTPUT: STR (DS:0000H) = 04H,03H,02H,01H

10. Length of the string

Registers used: AX,DS,SI,CL

Flags affected: ZF,PF,SF,AF,CF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
STR DB 01H,03H,08H,09H,05H,07H,02H
LENGTH DB ?
DATA ENDS
CODE SEGMENT
START:MOV AX,DATA
MOV DS,AX
MOV AL,00H
MOV CL,00H
MOV SI,OFFSET STR
BACK:CMP AL,[SI]
JNC GO
INC CL
INC  SI
JNZ BACK
GO:MOV LENGTH,CL
MOV AH,4CH
```

NRCM your roots to success...

```
INT 21H
CODE ENDS
END START
END
```

Result:

INPUT: STR (DS:0000H) = 01H, 03H,08H,09H,05H,07H,02H

OUTPUT: LENGTH = 07H (CL)

11. String comparison

Registers used: AX,DS,SI,DI,CL

Flags affected: ZF,CF

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
STR DB 04H,05H,07H,08H
COUNT EQU 04H
ORG 0010H
STR1 DB 04H,06H,07H,09H
DATA ENDS
CODE SEGMENT
START:MOV AX,DATA
MOV DS,AX
MOV SI,OFFSET STR
MOV DI,OFFSET STR1
```

NRCM your roots to success...

```
MOV CL,COUNT
CLD
REP CMPSB
MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Result:

INPUT: STR (DS:0000H) = 04H,05H,07H,08H

STR1 (DS:0010H) = 04H,06H,07H,09H

OUTPUT: **I):** IF STR = STR1 THEN ZF = 1 &

II): IF STR ≠ STR1 THEN ZF = 0

12. Display the string

Registers used: AX,DS,DX

Flags affected: No flags are affected

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
MSG DB 0DH,0AH,"MICROPROCESSORS ",0DH,0AH,"$"
DATA ENDS
CODE SEGMENT
START:
MOV AX,DATA
```

NRCM
your roots to success...

```
MOV DS,AX
MOV AH,09H
MOV DX,OFFSET MSG
INT 21H
MOV AH,4CH
INT 21H
CODE ENDS
END START
END
```

Result:

MICROPROCESSORS

your roots to success...

UNIT-3

INPUT/ OUTPUT INTERFACE

3.1. 8255 ppi

3.1.1. 8255 architecture

The parallel input-output port chip 8255 is also called as programmable ***peripheral input-output port.*** The Intel's 8255 is designed for use with Intel's 8- bit, 16-bit and higher capability microprocessors. It has 24 input/output lines which may be individually programmed in two groups of twelve lines each, or three groups of eight lines.

• The two groups of I/O pins are named as Group A and Group B. Each of these two groups contains a subgroup of eight I/O lines called as 8-bit port and another subgroup of four lines or a 4-bit port. Thus Group A contains an 8-bit port A along with a 4-bit port C upper.

• The port A lines are identified by symbols PA0-PA7 while the port C lines are identified as PC4-PC7. Similarly, Group B contains an 8-bit port B, containing lines PB0-PB7 and a 4-bit port C with lower bits PC0- PC3.

• Both the port C are assigned the same address. Thus one may have either three 8- bit I/O ports or two 8-bit and two 4-bit ports from 8255. All of these ports can function independently either as input or as output ports. This can be achieved by programming the bits of an internal register of 8255 called as control word register (CWR).

The internal block diagram and the pin configuration of 8255 are shown in fig.3.1. as below

NRCM
your roots to success...

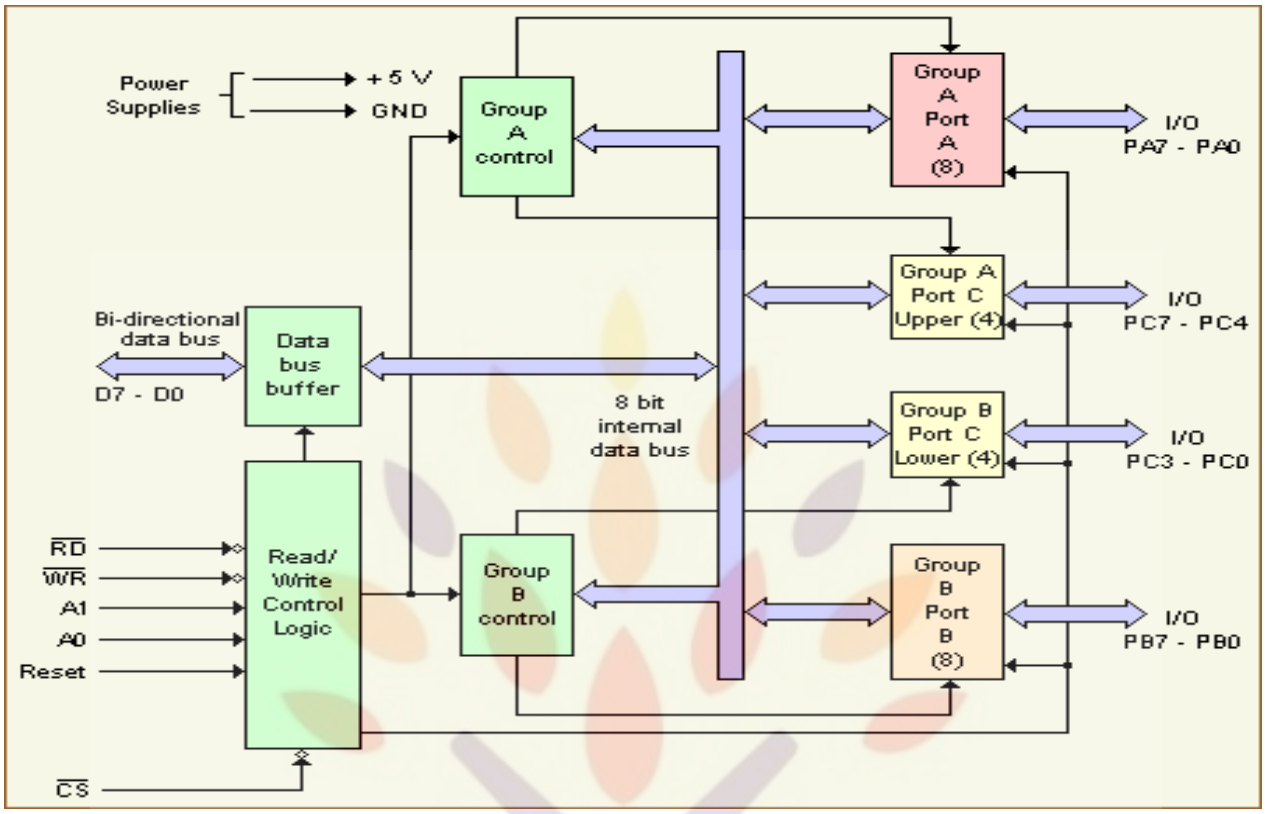


Fig 3.1. 8255 Architecture

The 8-bit data bus buffer is controlled by the read/write control logic. The read/write control logic manages all of the internal and external transfers of both data and control words.

• RD , WR , A_1, A_0 and RESET are the inputs provided by the microprocessor to the READ/ WRITE control logic of 8255. The 8-bit, 3-state bidirectional buffer is used to interface the 8255 internal data bus with the external system data bus.

• This buffer receives or transmits data upon the execution of input or output instructions by the microprocessor. The control words or status information is also
transferred through the buffer

Description

It has a 40 pins of 4 groups.

1. Data bus buffer
2. Read Write control logic
3. Group A and Group B controls
4. Port A, B and C

• ***Data bus buffer***: This is a tri state bidirectional buffer used to interface the 8255 to system data bus. Data is transmitted or received by the buffer on execution of input or output instruction by the CPU.

• Control word and status information are also transferred through this unit.

• ***Read/Write control logic***: This unit accepts control signals (RD , WR) and also inputs from address bus and issues commands to individual group of control blocks (Group A, Group B).
• It has the following pins.
a) CS – Chip select : A low on this PIN enables the communication between CPU and 8255.
b) RD (Read) – A low on this pin enables the CPU to read the data in the ports or the status word through data bus buffer.
c) WR (Write) : A low on this pin, the CPU can write data on to the ports or on to the control register through the data bus buffer.
d) **RESET**: A high on this pin clears the control register and all ports are set to the input mode
e) $\mathbf{A_0}$ and $\mathbf{A_1}$ (Address pins): These pins in conjunction with RD and WR pins control the selection of one of the 3 ports.
• ***Group A and Group B controls*** : These block receive control from the CPU and issues commands to their respective ports.
• Group A - PA and PCU (PC_7 –PC_4)
• Group B - PCL (PC3 – PC0)
• Control word register can only be written into no read operation of the CW register is allowed.
• a) **Port A**: This has an 8 bit latched/buffered O/P and 8 bit input latch. It can be programmed in 3 modes – mode 0, mode 1, mode 2.
b) **Port B**: This has an 8 bit latched / buffered O/P and 8 bit input latch. It can be programmed in mode 0, mode1.
c) **Port C** : This has an 8 bit latched input buffer and 8 bit output latched/buffer. This port can be divided into two 4 bit ports and can be used as control signals for port A and port B.

3.1.2. 8255 Pin Description

The 8255 is a 40 pin integrated circuit (IC), designed to perform a variety of interface functions in a computer environment. The 8255 wasn't originally designed to be connected to the Z80. It was manufactured by Intel for the 8080 microprocessor.

your roots to success...

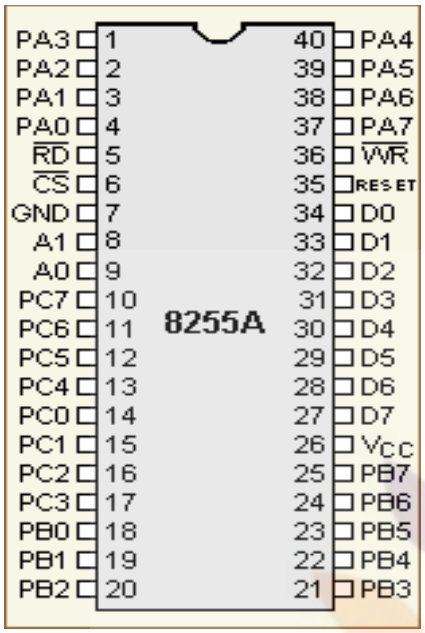


Fig 3.2. 8255 pin diagram

The signal description of 8255 are briefly presented as follows :

• **PA7-PA0**: These are eight port A lines that acts as either latched output orbuffered input lines depending upon the control word loaded into the control wordregister.

• **PC7-PC4**: Upper nibble of port C lines. They may act as either output latches orinput buffers lines.This port also can be used for generation of handshake lines in mode 1 or mode2.

• **PC3-PC0**: These are the lower port C lines, other details are the same as PC7-PC4lines.

• **PB0-PB7** : These are the eight port B lines which are used as latched output linesor buffered input lines in the same way as port A.

• **RD** : This is the input line driven by the microprocessor and should be low toindicate read operation to 8255.

• **WR** : This is an input line driven by the microprocessor. A low on this lineindicates write operation.

• **CS** : This is a chip select line. If this line goes low, it enables the 8255 to respondto RD and WR signals, otherwise RD and WR signal are neglected.

• **A1-A0** : These are the address input lines and are driven by the microprocessor.These lines A1-A0 with RD , WR and CS from the following operations for8255. These address lines are used for addressing any one of the four registers

• In case of 8086 systems, if the 8255 is to be interfaced with lower order data bus,the A0 and A1 pins of 8255 are connected with A1 and A2 respectively.

• **D0-D7** : These are the data bus lines those carry data or control word to/from themicroprocessor.

• **RESET** : A logic high on this line clears the control word register of 8255. Allports are set as input ports by default after reset.

3.1.3. 8255 Control Word Register

Before going to discuss the detailed description about the usage of the 8255 in the MZ-700, you should see the bit definitions of the 8255 control word register (port $E003 of the MZ-700).If bit 7 of the control word is a logical 1 then the 8255 will be configured.

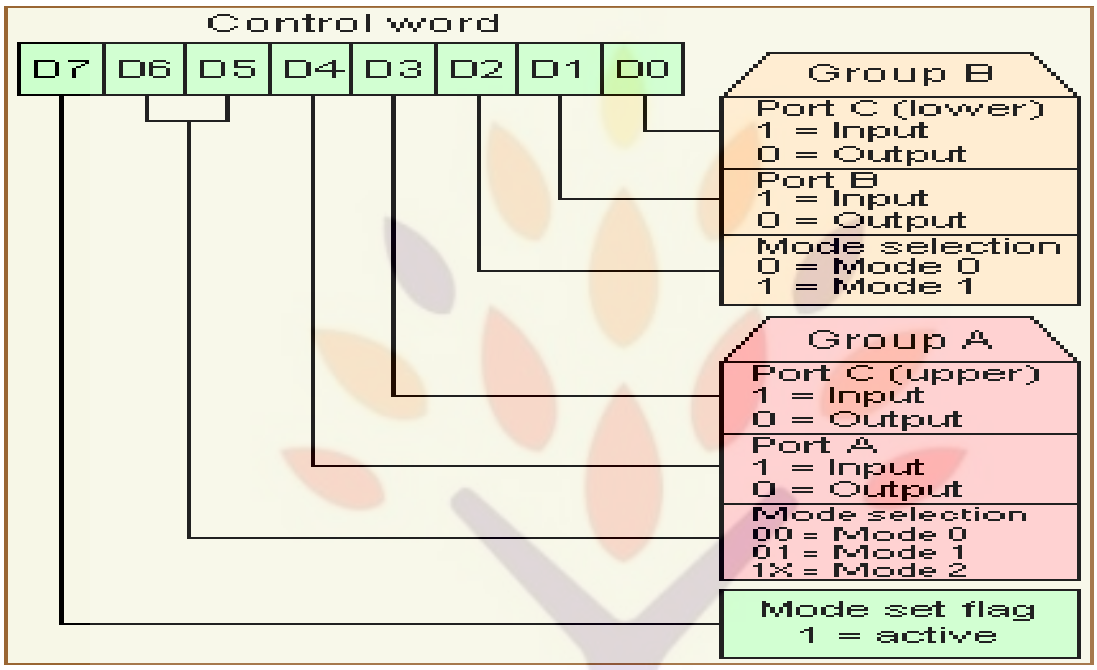


Fig 3.3. 8255 control word register

Mode definition of the 8255 control register to configure the 8255 Bit definitions of the 8255 control register to modify single bits of port C

Examples:
If you want to set/reset bit 0 of port C then set D3 to D1 to 000.
Bit 1 of port C will be set/reset if you code 001 to D3 to D1.
Bit 6 of port C is set/reset if D3 to D1 is 110.

3.1.4. Modes Of Operation Of 8255

• These are two basic modes of operation of 8255. I/O mode and Bit Set-Resetmode (BSR).
• In I/O mode, the 8255 ports work as programmable I/O ports, while in BSR modeonly port C (PC0-PC7) can be used to set or reset its individual port bits.
• Under the I/O mode of operation, further there are three modes of operation of8255, so as to support different types of applications, mode 0, mode 1 and mode2.
• ***BSR Mode***: In this mode any of the 8-bits of port C can be set or reset dependingon D0 of the control word. The bit to be set or reset is selected by bit select flagsD3, D2 and D1 of the CWR.

• **I/O Modes** :

a) ***Mode 0 (Basic I/O mode):*** This mode is also called as basic input/outputmode. This mode provides simple input and output capabilities using each of the threeports. Data can be simply read from and written to the input and output portsrespectively, after appropriate initialization

The salient features of this mode are as listed below:

1. Two 8-bit ports (port A and port B)and two 4-bit ports (port C upper and lower)are available. The two 4-bit ports can be combinedly used as a third 8-bit port.
2. Any port can be used as an input or output port.
3. Output ports are latched. Input ports are not latched.
4. A maximum of four ports are available so that overall 16 I/O configurations arepossible.

• All these modes can be selected by programming a register internal to 8255known as CWR.

• The control word register has two formats. The first format is valid for I/O modesof operation, i.e. modes 0, mode 1 and mode 2 while the second format is validfor bit set/reset (BSR) mode of operation.

b) Mode 1: ***(Strobed input/output mode)*** In this mode the handshaking control the inputand output action of the specified port. Port C lines PC0-PC2; provide strobe orhandshake lines for port B. This group which includes port B and PC0-PC2 is called asgroup B for Strobed data input/output. Port C lines PC3-PC5 provides strobe lines for portA.This group including port A and PC3-PC5 from group A. Thus port C is utilized forgenerating handshake signals. The salient features of mode 1 are listed as follows:

1. Two groups – group A and group B are available for strobed data transfer.
2. Each group contains one 8-bit data I/O port and one 4-bit control/data port.
3. The 8-bit data port can be either used as input and output port. The inputs andoutputs both are latched.
4. Out of 8-bit port C, PC0-PC2 are used to generate control signals for port B andPC3-PC5 are used to generate control signals for port A. the lines PC6, PC7 maybe used as independent data lines.

• **The control signals for both the groups in input and output modes are explained as follows**:

Input control signal definitions (mode 1):

• STB(Strobe input) – If this lines falls to logic low level, the data available at 8-bit input port is loaded into input latches.

• **IBF** (Input buffer full) – If this signal rises to logic 1, it indicates that data hasbeen loaded into latches, i.e. it works as an acknowledgement. IBF is set by a lowon STB and is reset by the rising edge of RD input.

• **INTR** (Interrupt request) – This active high output signal can be used tointerrupt the CPU whenever an input device requests the service. INTR is set by ahigh STB pin and a high at IBF pin. INTE is an internal flag that can be controlledby the bit set/reset mode of either PC4(INTEA) or PC2(INTEB).

• INTR is reset by a falling edge of RD input. Thus an external input device can be request the service of the processor by putting the data on the bus and sending thestrobe signal.

Output control signal definitions (mode 1) :

• **OBF** (Output buffer full) – This status signal, whenever falls to low, indicatesthat CPU has written data to the specified output port. The OBF flip-flop will beset by a rising edge of WR signal and reset by a low going edge at the ACKinput.

• ACK (Acknowledge input) – ACK signal acts as an acknowledgement to begiven by an output device. ACK signal, whenever low, informs the CPU that thedata transferred by the CPU to the output device through the port is received bythe output device.

• **INTR** (Interrupt request) – Thus an output signal that can be used to interruptthe CPU when an output device acknowledges the data received from the CPU.INTR is set when ACK, OBF and INTE are 1. It is reset by a falling edge on WRinput. The INTEA and INTEB flags are controlled by the bit set-reset mode ofPC6 and PC2 respectively.

• **Mode 2** ***(Strobed bidirectional I/O):*** This mode of operation of 8255 is alsocalled as strobed bidirectional I/O. This mode of operation provides 8255 with additional features for communicating with a peripheral device on an 8-bit databus. Handshaking signals are provided to maintain proper data flow andsynchronization between the data transmitter and receiver. The interruptgeneration and other functions are similar to mode 1.

• In this mode, 8255 is a bidirectional 8-bit port with handshake signals. The Rdand WR signals decide whether the 8255 is going to operate as an input port oroutput port.

• The Salient features of Mode 2 of 8255 are listed as follows:

1. The single 8-bit port in group A is available.
2. The 8-bit port is bidirectional and additionally a 5-bit control port is available.
3. Three I/O lines are available at port C.(PC2 – PC0)
4. Inputs and outputs are both latched.
5. The 5-bit control port C (PC3-PC7) is used for generating / accepting handshakesignals for the 8-bit data transfer on port A.

• ***Control signal definitions in mode 2***:

• **INTR** – (Interrupt request) As in mode 1, this control signal is active high and isused to interrupt the microprocessor to ask for transfer of the next data byteto/from it. This signal is used for input (read) as well as output (write)operations.

• ***Control Signals for Output operations***:

• OBF (Output buffer full) – This signal, when falls to low level, indicates that theCPU has written data to port A.

• ACK (Acknowledge) This control input, when falls to logic low level,acknowledges that the previous data byte is received by the destination and nextbyte may be sent by the processor. This signal enables the internal tristate buffersto send the next data byte on port A.

• **INTE1** (A flag associated with OBF) This can be controlled by bit set/resetmode with PC6.

• ***Control signals for input operations :***

your roots to success...

• STB (Strobe input) A low on this line is used to strobe in the data into the inputlatches of 8255.
• **IBF** (Input buffer full) When the data is loaded into input buffer, this signal risesto logic '1'. This can be used as an acknowledge that the data has been receivedby the receiver.
• The waveforms in fig show the operation in Mode 2 for output as well as inputport.

.3.2. Interfacing key board with 8086

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.

1. 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.

2. 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.

• When u press a key, you push the top ink line through the hole to contact the bottom ink line.

• The advantages of membrane keyboards is that they can be made as very thin, sealed units.

• They are often used on cash registers in fast food restaurants. The lifetime of membrane keyboards varies over a wide range.

3. Capacitive key switches: A capacitive keyswitch has two small metal plates on the printed circuit board and another metal plate on the bottom of a piece of foam.

• When u press the key, the movable plate is pushed closer to fixed plate. This changes the capacitance between the fixed plates. Sense amplifier circuitry detects this change in capacitance and produce a logic level signal that indicates a key has been pressed.

• The big advantages of a capacitive switch is that it has no mechanical contacts to become oxidized or dirty.

• A small disadvantage is the specified circuitry needed to detect the change in capacitance.

• Capacitive keyswitches typically have a rated lifetime of about 20 million keystrokes.

4. Hall effect keys witches: This is another type of switch which has no mechanical contact. It takes advantage of the deflection of a moving charge by a magnetic field.

• A reference current is passed through a semiconductor crystal between two opposing faces. When a key is pressed, the crystal is moved through a magnetic field which has its flux lines perpendicular to the direction of current flow in the crystal.

• Moving the crystal through the magnetic field causes a small voltage to be developed between two of the other opposing faces of the crystal.

• This voltage is amplified and used to indicate that a key has been pressed. Hall effect sensors are also used to detect motion in many electrically controlled machines.

• Hall effect keyboards are more expensive because of the more complex switch mechanism, but they are very dependable and have typically rated lifetime of 100 million or more keystrokes.

• In most keyboards, the keyswitches are connecting in a matrix of rows and columns• We will use simple mechanical switches for our examples, but the principle is same for other type of switches.

• Getting meaningful data from a keyboard, it requires the following three major tasks:

1. Detect a keypress.

2. Debounce the keypress.

3. Encode the keypress

• Three tasks can be done with hardware, software, or a combination of two, depending on the application.

1. Software Keyboard Interfacing:

your roots to success...

• Circuit connection and algorithm :

• The rows of the matrix are connected to four output port lines. The column lines of matrix are connected to four input-port lines. To make the program simpler, the row lines are also connected to four input lines.

• When no keys are pressed, the column lines are held high by the pull-up resistor connected to +5V. Pressing a key connects a row to a column. If a low is output on a row and a key in that row is pressed, then the low will appear on the column which contains that key and can be detected on the input port.

• If you know the row and column of the pressed key, you then know which key was pressed, and you can convert this information into any code you want to represent that key.

• An easy way to detect if any key in the matrix is pressed is to output 0's to all rows and then check the column to see if a pressed key has connected a low to a column.

• In the algorithm we first output lows to all the rows and check the columns over and over until the column are all high. This is done before the previous key has been released before looking for the next one. In the standard keyboard terminology, this is called two-key lockout.

• Once the columns are found to be all high, the program enters another loop, which waits until a low appears on one of the columns, indicating that a key has been pressed. This second loop does the detect task for us. A simple 20-ms delay procedure then does the debounce task.

• After the debounce time, another check is made to see if the key is still pressed. If the columns are now all high, then no key is pressed and the initial detection was caused by a noise pulse or a light brushing past a key. If any of the columns are still low, then the assumption is made that it was a valid keypress.

• The final task is to determine the row and column of the pressed key and convert this row and column information to the hex code for the pressed key. To get the row and column information, a low is output to one row and the column are read. If none of the columns is low, the pressed key is not in that row. So the low is rotated to the next row and the column are checked again. The process is repeated until a low on a row produces a low on one of the column.

• The pressed key then is in the row which is low at that time.

your roots to success...

• The connection fig shows the byte read in from the input port will contain a 4-bit code which represents the row of the pressed key and a 4-bit code which represent the column of the pressed key.

• Error trapping: The concept of detecting some error condition such as " no match found" is called error trapping. Error trapping is a very important part of real programs. Even in simple

programs, think what might happen with no error trap if two keys in the same row were pressed at exactly at the same time and a column code with two lows in it was produced.

• This code would not match any of the row-column codes in the table, so after all the values in the table were \checked, assigned register in program would be decremented from 0000H to FFFFH. The compare decrement cycle would continue through 65,536 memory locations until, by change the value in a memory location matched the row-column code. The contents of the lower byte register at hat point would be passed back to the calling routine. The changes are 1 in 256 that would be the correct value for one of the pressed keys. You should keep an error trap in a program whenever there is a chance for it.

Example

• Interface a 4 * 4 keyboard with 8086 using 8255 an write an ALP for detecting a key closure and return the key code in AL. The debounce period for a key is 10ms. Use software debouncing technique. DEBOUNCE is an available 10ms delay routine.

Solution: Port A is used as output port for selecting a row of keys while Port B is used as an input port for sensing a closed key. Thus the keyboard lines are selected one by one through port A and the port B lines are polled continuously till a key closure is sensed. The routine DEBOUNCE is called for key debouncing. The key code is depending upon the selected row and a low sensed column.

• The higher order lines of port A and port B are left unused. The address of port A and port B will respectively 8000H and 8002H while address of CWR will be 8006H. The flow chart of the complete program is as given. The control word for this problem will be 82H. Code segment CS is used for storing the program code.

• **Key Debounce** : Whenever a mechanical push-button is pressed or released once, the mechanical components of the key do not change the position smoothly, rather it generates a transient response .

• These transient variations may be interpreted as the multiple key pressure and responded accordingly by the microprocessor system.

• To avoid this problem, two schemes are suggested: the first one utilizes a bistable multivibrator at the output of the key to debounce .

your roots to success...

• The other scheme suggests that the microprocessor should be made to wait for the transient period (usually 10ms), so that the transient response settles down and reaches a

steady state.

• A logic ‘0’ will be read by the microprocessor when the key is pressed.

• In a number of high precision applications, a designer may have two options- the first is to have more than one 8-bit port, read (write) the port one by one and then from the multibyte data, the second option allows forming 16-bit ports using two 8-bit ports and use 16-bit read or write operations.

Fig 3.4. key board interfacing

3.3. Interfacing 7- Segment display with 8086

• To give directions or data values to users, many microprocessor-controlledinstruments and machines need to display letters of the alphabet and numbers. Insystems where a large amount of data needs to be displayed a CRT is used todisplay the data. In system where only a small amount of data needs to bedisplayed, simple digit-type displays are often used.

• There are several technologies used to make these digit-oriented displays but weare 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, batterypoweredinstruments. They do not emit their own light, they simply change thereflection of available light. Therefore, for an instrument that is to be used in lowlightconditions, you have to include a light source for LCDs or use LEDs whichemit their own light.

• Alphanumeric LED displays are available in three common formats. Fordisplaying only number and hexadecimal letters, simple 7-segment displays suchas that as shown in fig are used.

• To display numbers and the entire alphabet. The 7-segment type is the least expensive, most commonly used and easiest to interfacewith, so we will concentrate first on how to interface with this type.

1. ***Directly Driving LED Displays:*** Figure shows a circuit that you might connect toa parallel port on a microcomputer to drive a single 7-segment , common-anodedisplay. For a common-anode display, a segment is turned on by applying a logiclow to it.

• The 7447 converts a BCD code applied to its inputs to the pattern of lowsrequired to display the number represented by the BCD code. This circuitconnection is referred to as a *static display* because current is being passedthrough the display at all times.

• Each segment requires a current of between 5 and 30mA to light. Let's assumeyou want a current of 20mA. The voltage drop across the LED when it is lit isabout 1.5V.

• The output low voltage for the 7447 is a maximum of 0.4V at 40mA. So assume that it is about 0.2V at 20mA.Subtracting these two voltage drop from the supplyvoltage of 5V leaves 3.3V across the current limiting resistor. Dividing 3.3V by20mA gives a value of 168Ω for the current-limiting resistor. The voltage dropsacross the LED and the output of 7447 are not exactly predictable and exactcurrent through the LED is not critical as long as we don't exceed its maximumrating.

2. Software-Multiplexed LED Display:

• The circuit in fig works for driving just one or two LED digits with a paralleloutput port. However, this scheme has several problems if you want to drive, eightdigits.

• The first problem is power consumption. For worst-case calculations, assume thatall 8 digits are displaying the digit 8, so all 7 segments are all lit. Seven segment times 20mA per segment gives a current of 140mA per digit. Multiplying this by 8digits gives a total current of 1120mA or 1.12A for 8 digits.

• A second problem of the static approach is that each display digit requires aseparate 7447 decoder, each of which uses of another 13mA. The current requiredby the decoders and the LED displays might be several times the current requiredby the reset of the circuitry in the instrument.

• To solve the problem of the static display approach, we use a *multiplex method,*example for an explanation of the multiplexing.

• The question that may occur to you on first seeing this is: Aren't all the digitsgoing to display the same number? The answer is that they would if all the digitswere turned on at the same time. The tricky of multiplexing displays is that onlyone display digit is turned on at a time.

• The PNP transistor is series with the common anode of each digit acts as on/offswitch for that digit. Here's how the multiplexing process works.

• The BCD code for digit 1 is first output from port B to the 7447. the 7447 outputsthe corresponding 7-segment code on the segment bus lines. The transistorconnected to digit 1 is then turned on by outputting a low to the appropriate bit ofport A. All the rest of the bits of port A are made high to make sure no other digitsare turned on. After 1 or 2 ms, digit 1 is turned off by outputting all highs to portA.

• The BCD code for digit 2 is then output to the 7447 on port B, and a word to turnon digit 2 is output on port A.

• After 1 or 2 ms, digit 2 is turned off and the process is repeated for digit 3. Theprocess is continued until all the digits have had a turn. Then digit 1 and thefollowing digits are lit again in turn.

• A procedure which is called on an interrupt basis every 2ms to keep these displaysrefreshed with some values stored in a table. With 8 digits and 2ms per digit, youget back to digit 1 every 16ms or about 60 times a second.

• This refresh rate is fast enough so that the digits will each appear to be lit all time.

Refresh rates of 40 to 200 times a second are acceptable.

• The immediately obvious advantages of multiplexing the displays are that onlyone 7447 is required, and only one digit is lit at a time. We usually increase thecurrent per segment to between 40 and 60 mA for multiplexed displays so thatthey will appear as bright as they would if they were not multiplexed. Even withthis increased segment current, multiplexing gives a large saving in power andparts.

• The software-multiplexed approach we have just described can also be used todrive 18-segment LED devices and dot-matrix LED device. For these devices,however you replace the 7447 in fig with ROM which generates the requiredsegment codes when the ASCII code for a character is applied to the addressinputs of the ROM.

NRCM
your roots to success...

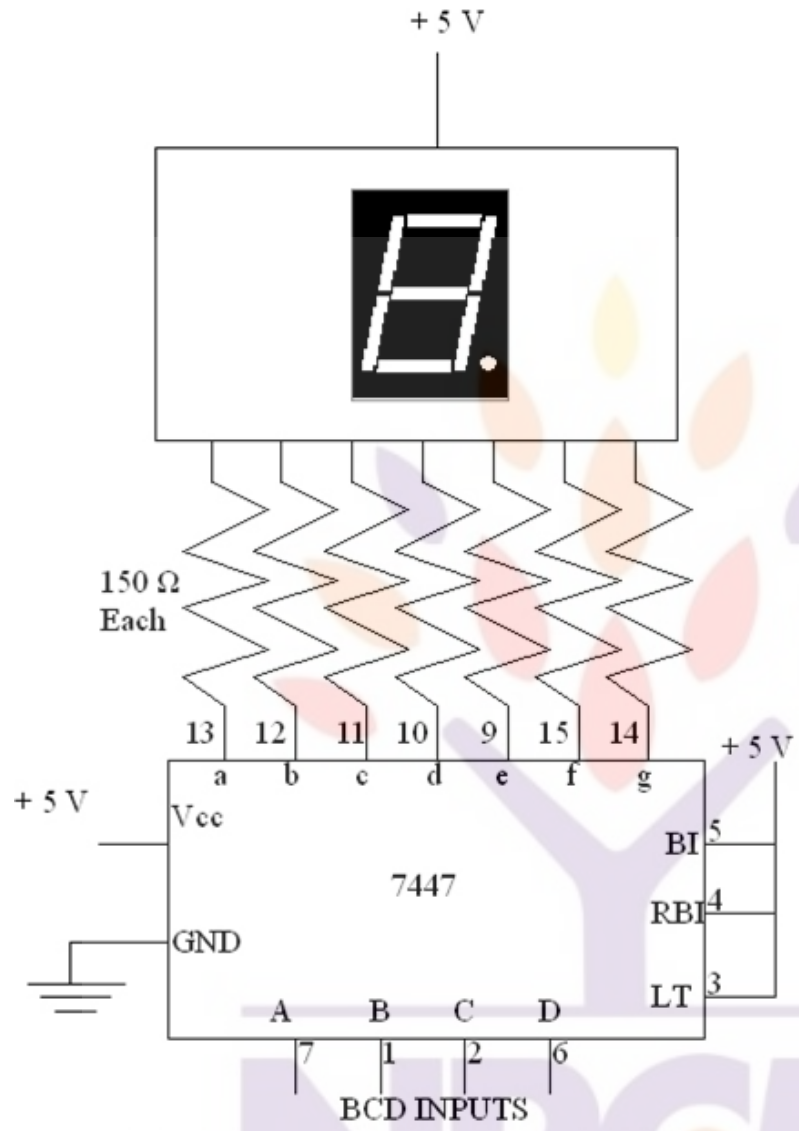


Fig 3.5. 7- Segment display interfacing with 8086

Program:

```
;************************************************************
;************************************************************
; Odd addressed 8255 used ; Port A lines, PA0-PA7 are connected to the FND display
 ; To energize an FND segment the corresponding bit on PA should be active LOW
;************************************************************
CODE SEGMENT
```

```
 ASSUME CS:CODE,DS:CODE,ES:CODE,SS:CODE
 PORT_CON EQU 1FH ; Control Port 8-bit Address
 PORTC EQU 1DH ; Port C 8-bit Address
 PORTB EQU 1BH ; Port B 8-bit Address
 PORTA EQU 19H ; Port A 8-bit Address
ORG 1000H ; Program Effective Address, IP = 1000H ;
MOV AX, CS
 MOV DS, AX ; CS = DS ;
MOV AL, 10000000B ; Configure all ports of 8255 as output
OUT PORT_CON, AL ;
L1: MOV BL, 16 ; Setup number
 MOV SI, OFFSET FONT ; Setup address of font
 L2: MOV AL, [SI] ; Transfer font data
OUT PORTA, AL ; Output data
MOV CX, 0B000H ; Delay
LOOP $
 INC SI ; Font address + 1
 DEC BL ; Next digit
 JNZ L2
JMP L1
; Dgfedcba ; Segment Display
 FONT  DB 11000000B ; 0
          DB 11111001B ; 1
          DB 10100100B ; 2
          DB 10110000B ; 3
```

NRCM
your roots to success...

```
        DB 10011001B ; 4
        DB 10010010B ; 5
        DB 10000010B ; 6
        DB 11011000B ; 7
        DB 10000000B ; 8
        DB 10010000B ; 9
        DB 100010000B ; A
        DB 10000011B ; B
        DB 11000110B ; C
        DB 10100001B ; D
        DB 10000110B ; E
        DB 11000000B ; F
CODE ENDS
END ; End of Assembly Program
```

3.4. Interfacing stepper motor with 8086

The stepping motor is a device which can transfer the incoming pulses to stepping motion of predetermined angular displacement. By using suitable control circuitry, the angular displacement can be made proportional to the number of pulses. Using microcomputer, one can have better control of the angular displacement resolution and angular speed of a stepping motor. Stepping motors are suitable for translating digital inputs into mechanical motion. In general, there are three types of stepping motor:

VR (Variable Reluctance) stepping motors
Hybrid stepping motors
PM (Permanent Magnet) stepping motors

Theory of operation

Stepper motors operate differently from normal DC motors, which simply spin when voltage is applied to their terminals. Stepper motors, on the other hand, effectively have multiple "toothed" electromagnets arranged around a central metal gear, as shown in Fig. 3.6

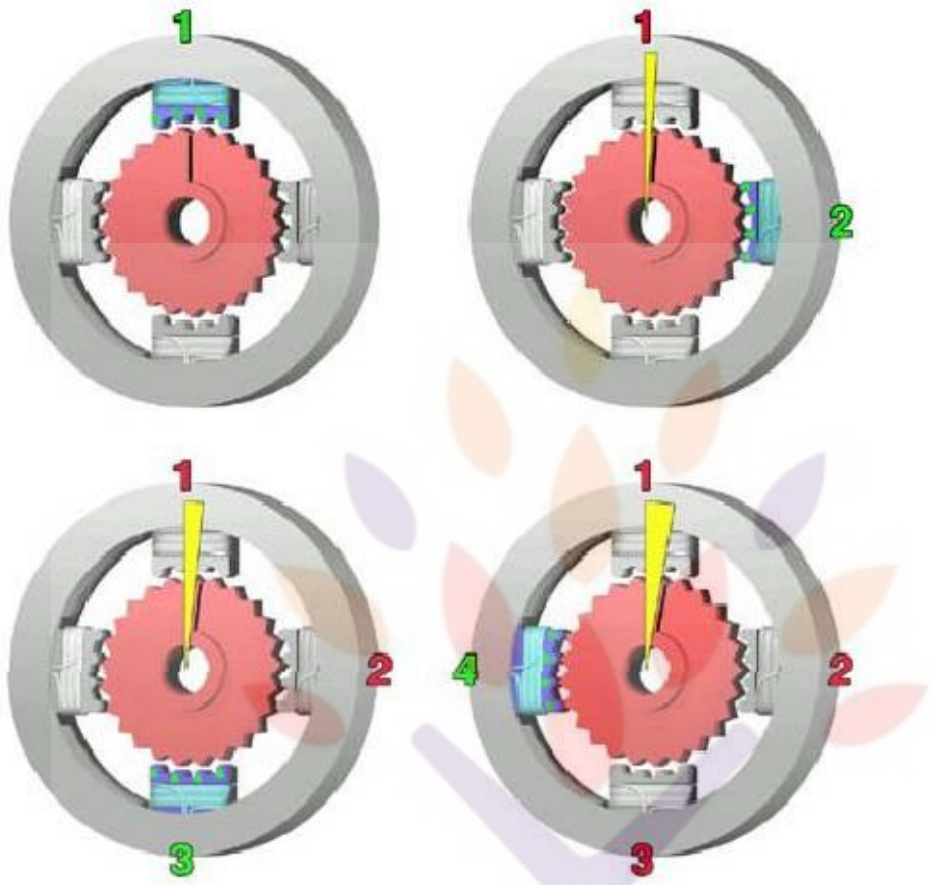


Fig 3.6. Toothed stepper motor and state of motion at different coil excitation.

To make the motor shaft turn, first one electromagnet is given power, which makes the gear's teeth magnetically attracted to the electromagnet's teeth. When the gear's teeth are thus aligned to the first electromagnet, they are slightly offset from the next electromagnet. So, when the next electromagnet is turned on and the first is turned off, the gear rotates slightly to align with the next one, and from there the process is repeated. Each of those slight rotations is called a "step." In that way, the motor can be turned a precise angle. There are two basic arrangements for the electromagnetic coils: bipolar and unipolar. We will experiment on a unipolar 4-phase six wire stepper motor. The step angle for each step depends on the number of teethes on the rotor and pole faces. Stepper motors are mostly Hybrid type. In this experiment we will use a hybrid stepper motor with full step angle of 1.80 and half step angle of 0.90.Commercial stepping motor uses multimotor rotor, the rotor features two bearlike PM cylinders that are turned one-half of tooth spacing. One gear is south pole, the other gear is north pole. If a 50-tooth rotor gear is used, the following movement sequences will proceed.

Hardware Interface

To run a stepper motor from a microprocessor trainer (microcomputer), we need a parallel port interface. Here we will be using Intel 8255 PPI (programmable peripheral interface) that has three 8-bit ports configurable in few modes. The 8255 has one control port where one can send

the control word for configuring the 8255 ports (For details see 8255 data sheet). In this application, one port of 8255 should be configured as output port for stepper motor control signals to pass through. In MDA8086 trainer, the stepper motor interface is built in the motherboard, as shown in Figs. 8-1, 8-2. Upper 4-bits of Port B of the odd addressed 8255 are connected to the stepper motor circuitry. The signal mappings of the port lines are as follows:

Port Bit	Phase	Terminal Connector P10
PB4	Coil A1	1
PB5	Coil B1	4
PB6	Coil A2	3
PB7	Coil B2	6
-	TAPA	2
-	TAPB	5

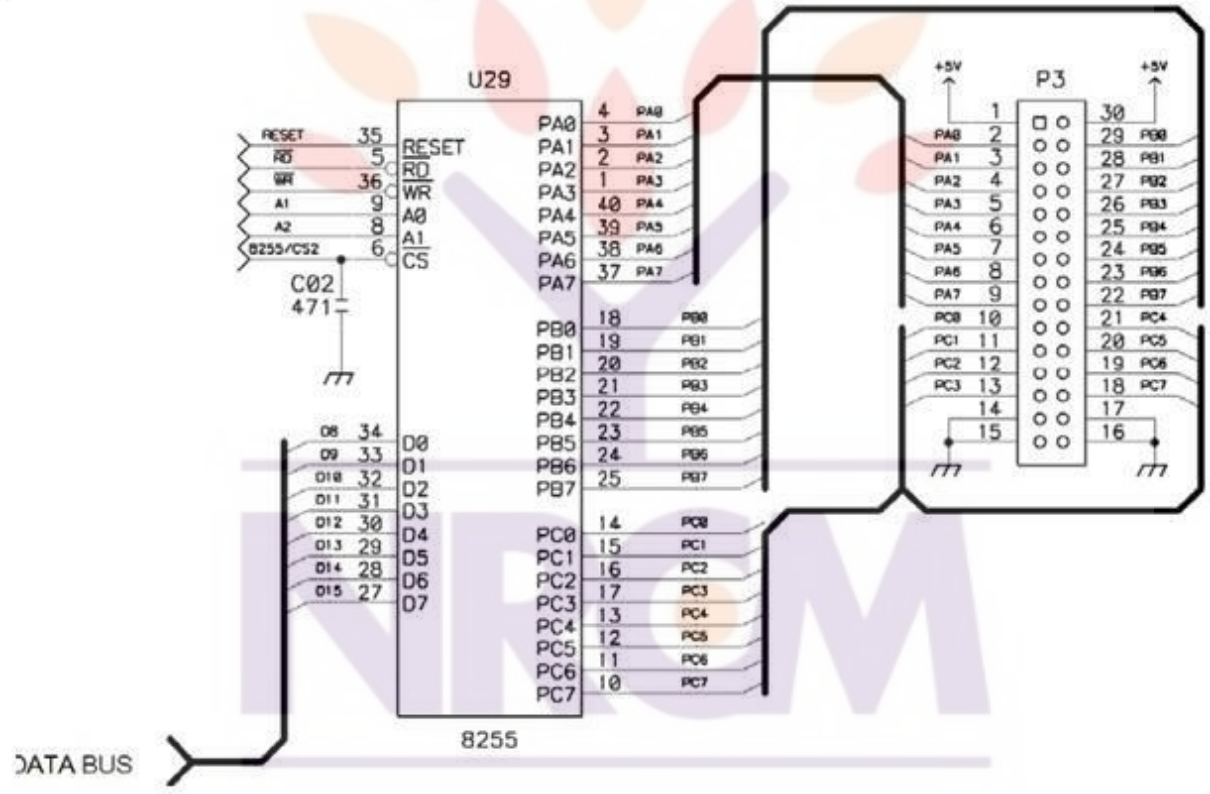


Fig 3.7. Stepper motor interfacing through 8255

Assembly Program

Running a stepper motor in Full-Steps

```
;*************************************************************
;*************************************************************
; PROG1
 ; Odd addressed 8255 used
```

```
 ; Port B (upper 4-lines, PB4-PB7) is connected to the Stepping Motor Interface
 ; PB4 □ Coil 1 (A)
;PB5 □ Coil 1 (B)
 ; PB6 □ Coil 2 (A)
 ; PB7 □ Coil 2 (B)
 ; To energize a coil the corresponding bit on PB should be active LOW
;*************************************************************
CODE SEGMENT ASSUME CS:CODE,DS:CODE,ES:CODE,SS:CODE
 PORT_CON EQU 1FH ; Control Port 8-bit Address
 PORTC EQU 1DH ; Port C 8-bit Address
 PORTB EQU 1BH ; Port B 8-bit Address
 PORTA EQU 19H ; Port A 8-bit Address ;
ORG 1000H ; Program Effective Address, IP = 1000H ;
MOV AX, 0
 MOV DS, AX ; Initialize Data Segment register DS to 0000H ;
MOV AL, 10000000B ; Configure all ports of 8255 as output
OUT PORT_CON, AL ;
MOV AL, 11111111B ; Can write 0FFH as well
OUT PORTA, AL ; All pins of Port A to HIGH
MOV AL, 00000000B
 OUT PORTC, AL ; All pins of Port C to
;MOV AL, 11101110B ; Only one coil to be energized at a time
 L1: OUT PORTB, AL
CALL DELAY ; Call DELAY subroutine
 ROL AL, 1 ; Rotate AL left by 1 bit
 JMP L1
;;Subroutine of DELAY
 DELAY: MOV CX, 0 ; Similar to loading CX by FFFFH
 AGAIN:
 NOP
 NOP ; Dummy instructions to cause time delay
 NOP
 NOP
 LOOP AGAIN
 RET ; Return from subroutine call ;
```

your roots to success...

CODE ENDS ; End of Subroutine DELAY
END ; End of Assembly Program

3.5. Interfacing Analog to Digital Data Converters

• In most of the cases, the PIO 8255 is used for interfacing the analog to digitalconverters with microprocessor.

• We have already studied 8255 interfacing with 8086 as an I/O port, in previoussection. This section we will only emphasize the interfacing techniques of analogto digital converters with 8255.

• The analog to digital converters is treated as an input device by themicroprocessor that sends an initializing signal to the ADC to start the analogy todigital data conversation process. The start of conversation signal is a pulse of aspecific duration.

• The process of analog to digital conversion is a slow process, and themicroprocessor has to wait for the digital data till the conversion is over. After theconversion is over, the ADC sends end of conversion EOC signal to inform themicroprocessor that the conversion is over and the result is ready at the outputbuffer of the ADC. These tasks of issuing an SOC pulse to ADC, reading EOCsignal from the ADC and reading the digital output of the ADC are carried out bythe CPU using 8255 I/O ports.

• The time taken by the ADC from the active edge of SOC pulse till the activeedge 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 afew hundred milliseconds in case of slow ADCs.

• The available ADC in the market use different conversion techniques forconversion of analog signal to digitals. Successive approximation techniques anddual slope integration techniques are the most popular techniques used in theintegrated 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 startof conversion till the end of the conversion to get correct results. This may beensured by a sample and hold circuit which samples the analog signal and holds itconstant for a specific time duration. The microprocessor may issue a hold signalto the sample and hold circuit.
6. If the applied input changes before the complete conversion process is over, thedigital 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, successiveapproximation converters. This technique is one of the fast techniques for analogto digital conversion. The conversion delay is 100μs at a clock frequency of 640KHz, which is quite low as compared to

other converters. These converters do notneed any external zero or full scale adjustments as they are already taken care ofby internal circuits. These converters internally have a 3:8 analog multiplexer sothat at a time eight different analog conversion by using address lines -ADD A, ADD B, ADD C. Using these address inputs, multichannel data acquisitionsystem can be designed using a single ADC. The CPU may drive these lines using outputport lines in case of multichannel applications. In case of single input applications, thesemay be hardwired to select the proper input.

• There are unipolar analog to digital converters, i.e. they are able to convert onlypositive analog input voltage to their digital equivalent. These chips do not containany internal sample and hold circuit.

Analog /P selecte	Address lines		
	C	B	A
I/P_0	0	0	0
I/P_1	0	0	1
I/P_2	0	1	0
I/P_3	0	1	1
I/P_4	1	0	0
I/P_5	1	0	1
I/P_6	1	1	0
I/P_7	1	1	1

Table 3.1. ADC interfacing through 8255

• If one needs a sample and hold circuit for the conversion of fast signal into equivalent digital quantities, it has to be externally connected at each of the analog inputs.

• Vcc Supply pins +5V

• GND GND

• Vref + Reference voltage positive +5 Volts maximum.

• Vref _ Reference voltage negative 0Volts minimum.
• I/P0 –I/P7 Analog inputs
• ADD A,B,C Address lines for selecting analog inputs.
• O7 – O0 Digital 8-bit output with O7 MSB and O0 LSB
• SOC Start of conversion signal pin
• EOC End of conversion signal pin
• OE Output latch enable pin, if high enables output
• CLK Clock input for ADC

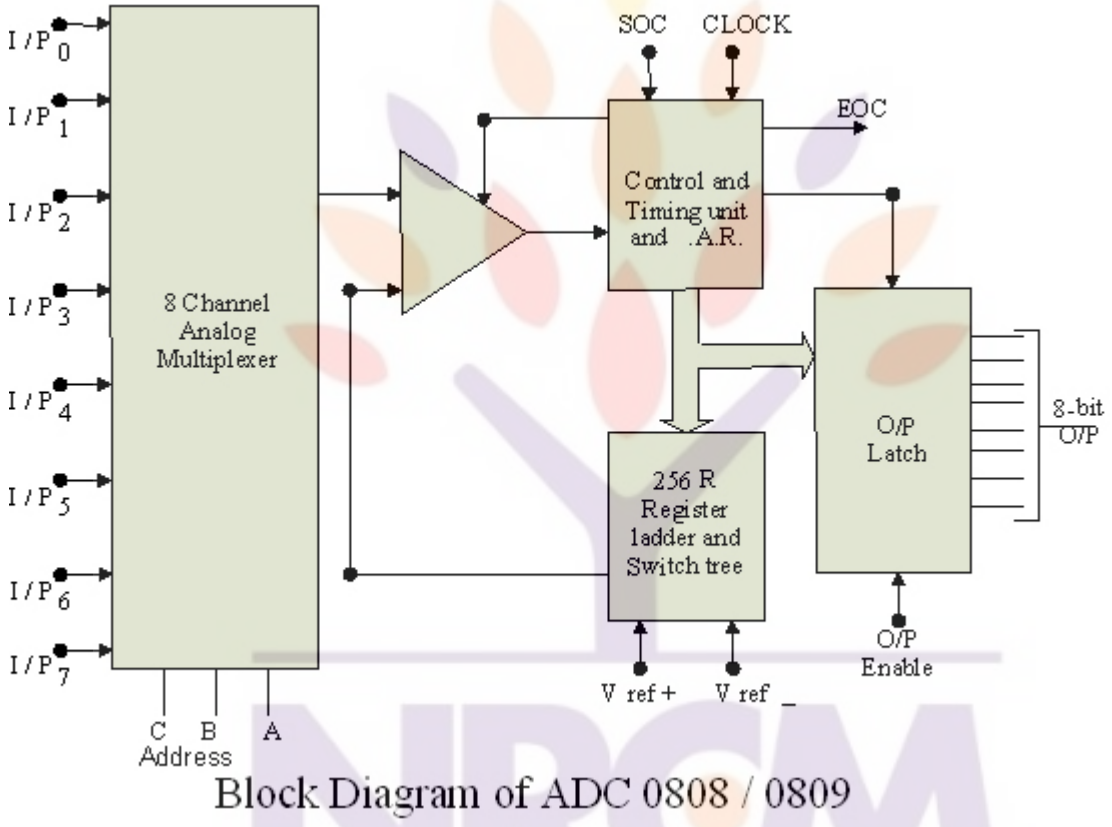


Fig 3.7. Block diagram of 0809 ADC

Example: Interfacing ADC 0808 with 8086 using 8255 ports. Use port A of 8255for transferring digital data output of ADC to the CPU and port C for controlsignals. Assume that an analog input is present at I/P2 of the ADC and a clockinput of suitable frequency is available for ADC.

• **Solution**: The analog input I/P2 is used and therefore address pins A,B,C shouldbe 0,1,0 respectively to select I/P2. The OE and ALE pins are already kept at +5Vto select the ADC and enable the outputs. Port C upper acts as the input port toreceive the EOC signal while port C lower acts as the output port to send SOC tothe ADC.

• Port A acts as a 8-bit input data port to receive the digital data output from theADC. The 8255 control word is written as follows:

D7 D6 D5 D4 D3 D2 D1 D0

1 0 0 1 1 0 0 0
• The required ALP is as follows:

```
MOV AL, 98h ;initialise 8255 as
OUT CWR, AL ;discussed above.
MOV AL, 02h ;Select I/P2 as analog
OUT Port B, AL ;input.
MOV AL, 00h ;Give start of conversion
OUT Port C, AL ; pulse to the ADC
MOV AL, 01h
OUT Port C, AL
MOV AL, 00h
OUT Port C, AL
WAIT: IN AL, Port C ;Check for EOC by
RCR ; reading port C upper and
JNC WAIT ;rotating through carry.
IN AL, Port A ;If EOC, read digital equivalent in AL
HLT ;Stop.
```

3.6. Interfacing Digital To Analog Converters

INTERFACING DIGITAL TO ANALOG CONVERTERS: The digital to analogconverters convert binary number into their equivalent voltages. The DAC findapplications in areas like digitally controlled gains, motors speed controls, programmablegain amplifiers etc.AD 7523 8-bit Multiplying DAC : This is a 16 pin DIP, multiplying digital toanalog converter, containing R-2R ladder for D-A conversion along with single poledouble thrown NMOS switches to connect the digital inputs to the ladder.

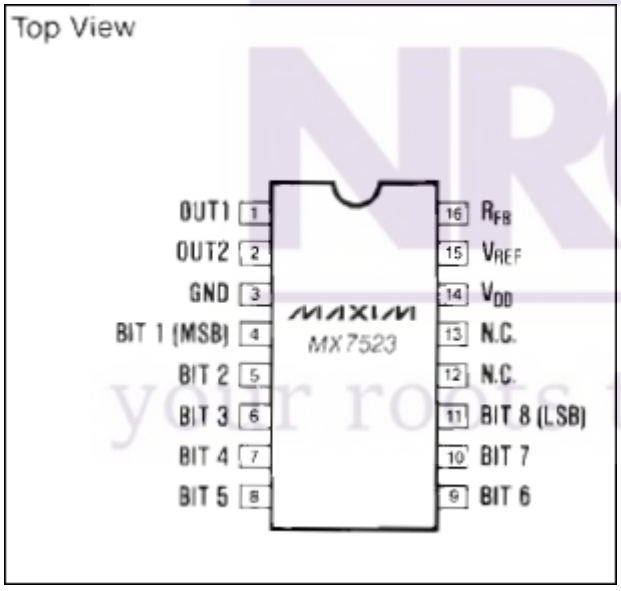


Fig 3.8. Pin diagram of 7253

• The pin diagram of AD7523 is shown in fig the supply range is from +5V to +15V, while Vref may be anywhere between -10V to +10V. The maximum analog output voltage will be any where between -10V to +10V, when all the digital inputs are at logic high state.

• Usually a zener is connected between OUT1 and OUT2 to save the DAC from negative transients. An operational amplifier is used as a current to voltage converter at the output of AD to convert the current output of AD to a proportional output voltage.

• It also offers additional drive capability to the DAC output. An external feedback resistor acts to control the gain. One may not connect any external feedback resistor, if no gain control is required.

• ***EXAMPLE***: Interfacing DAC AD7523 with an 8086 CPU running at 8MHZ and write an assembly language program to generate a sawtooth waveform of period 1ms with Vmax 5V.

Solution: Fig shows the interfacing circuit of AD 74523 with 8086 using 8255. program gives an ALP to generate a sawtooth waveform using circuit.

```
ASSUME CS:CODE
CODE SEGMENT
START :MOV AL,80h ;make all ports output
OUT CW, AL
AGAIN :MOV AL,00h ;start voltage for ramp
BACK : OUT PA, AL
INC AL
CMP AL, 0FFh
JB BACK
JMP AGAIN
CODE ENDS
END START
```

NRCM
your roots to success...

3.7 INTRFACING WITH ADVANCED DEVICS

3.7.1. Memory interfacing

• We have four common types of memory:
• Read only memory (ROM)
• Flash memory (EEPROM)
• Static Random access memory (SARAM)
• Dynamic Random access memory (DRAM).
• Pin connections common to all memory devices are: The address input, dataoutput or input/outputs, selection input and control input used to select a read orwrite operation.
• ***Address connections***: All memory devices have address inputs that select amemory location within the memory device. Address inputs are labeled from A0to An.
• ***Data connections***: All memory devices have a set of data outputs orinput/outputs. Today many of them have bi-directional common I/O pins.
• ***Selection connections***: Each memory device has an input, that selects or enablesthe memory device. This kind of input is most often called a chip select (CS),chip enable (CE) or simply select (S) input

NRCM
your roots to success...

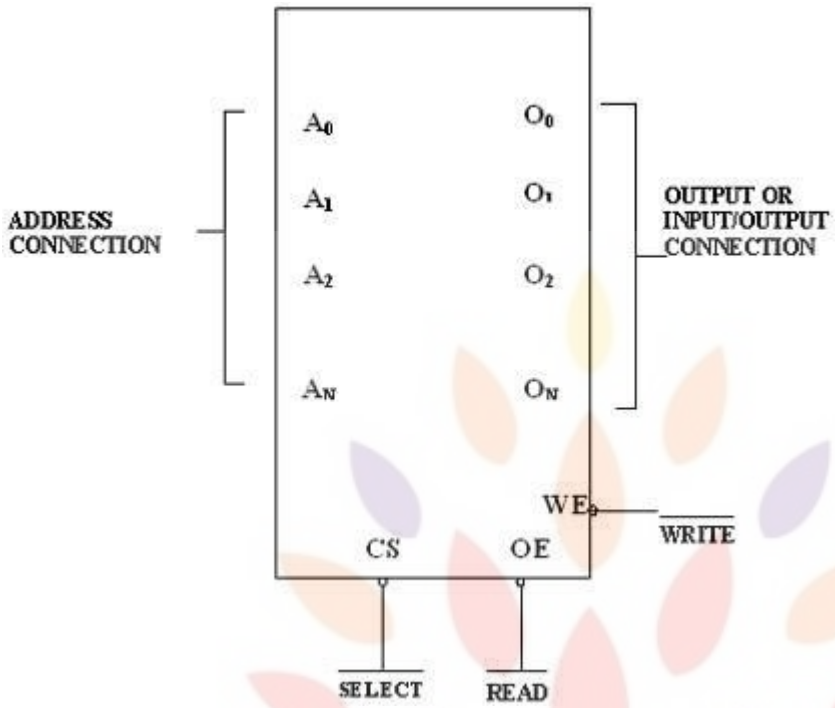


Fig 4.1. Diagram of memory component

• RAM memory generally has at least one CS or S input and ROM at least oneCE .
• If the CE , CS, S input is active the memory device perform the read or write.
• If it is inactive the memory device cannot perform read or write operation.
• If more than one CS connection is present, all must be active to perform read orwrite data.
• ***Control connections***: A ROM usually has only one control input, while a RAMoften has one or two control inputs.
• The control input most often found on the ROM is the output enable (OE) orgate (G), this allows data to flow out of the output data pins of the ROM.
• If OE and the selected input are both active, then the output is enable, if OE isinactive, the output is disabled at its high-impedance state.
• The OE connection enables and disables a set of three-state buffer located withinthe memory device and must be active to read data.
• A RAM memory device has either one or two control inputs. If there is onecontrol input it is often called R/W.
• This pin selects a read operation or a write operation only if the device is selectedby the selection input (CS).
• If the RAM has two control inputs, they are usually labeled WE or W and OEor G .

• (WE) write enable must be active to perform a memory write operation and OEmust be active to perform a memory read operation.
• When these two controls WE and OE are present, they must never be active atthe same time.
• The ROM read only memory permanently stores programs and data and data wasalways present, even when power is disconnected.
• It is also called as nonvolatile memory.
• EPROM (erasable programmable read only memory) is also erasable if exposedto high intensity ultraviolet light for about 20 minutes or less, depending upon thetype of EPROM.
• We have PROM (programmable read only memory)
• RMM (read mostly memory) is also called the flash memory.
• The flash memory is also called as an EEPROM (electrically erasable programmable ROM), EAROM (electrically alterable ROM), or a NOVROM(nonvolatile ROM).
• These memory devices are electrically erasable in the system, but require moretime to erase than a normal RAM.
• EPROM contains the series of 27XXX contains the following part numbers :2704(512 * 8), 2708(1K * 8), 2716(2K * 8), 2732(4K * 8), 2764(8K * 8),27128(16K * 8) etc..
• Each of these parts contains address pins, eight data connections, one or morechip selection inputs (CE) and an output enable pin (OE).
• This device contains **11** address inputs and **8** data outputs.
• If both the pin connection CE and OE are at logic 0, data will appear on the output connection If both the pins are not at logic 0, the data output connectionsremains at their high impedance or off state.
• To read data from the EPROM Vpp pin must be placed at a logic 1.

NRCM
your roots to success...

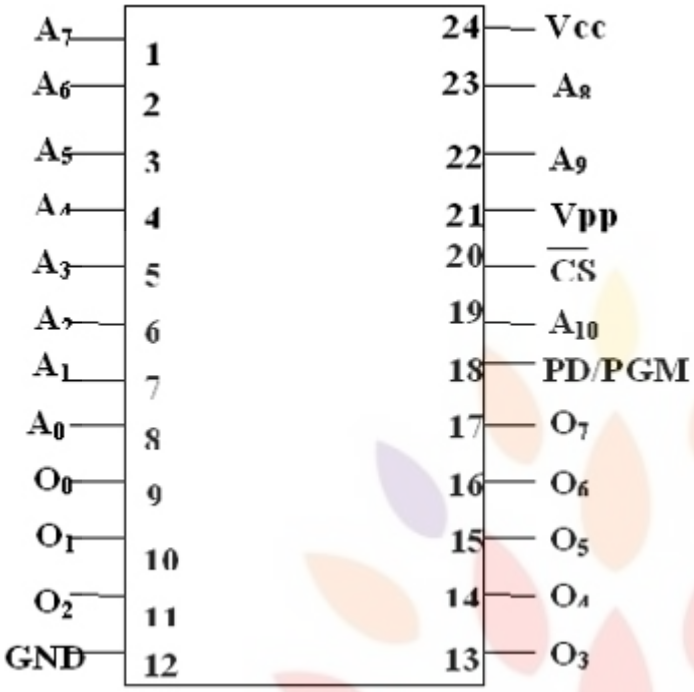


Fig 4.2. Pin diagram of 2716 EPROM

• Static RAM memory device retain data for as long as DC power is applied. Because no special action is required to retain stored data, these devices are called as static memory. They are also called volatile memory because they will not retain data without power.

• The main difference between a ROM and RAM is that a RAM is written under normal operation, while ROM is programmed outside the computer and is only normally read.

• The SRAM stores temporary data and is used when the size of read/write memory is relatively small.

your roots to success...

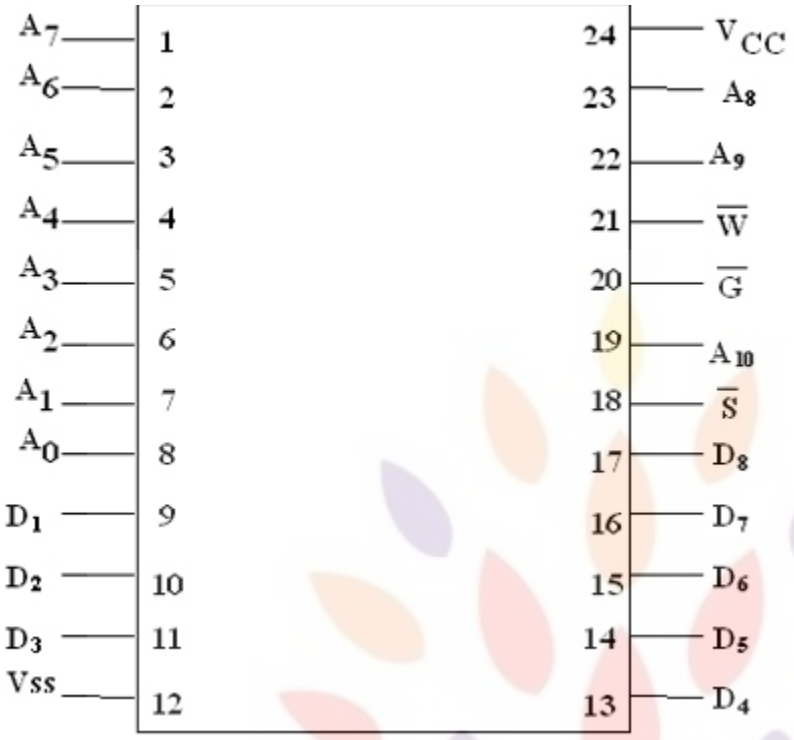


Fig 4.3. Pin diagram of 4016 SRAM

• The control inputs of this RAM are slightly different from those presented earlier. The OE pin is labeled G , the CS pin S and the WE pin W.
• This 4016 SRAM device has 11 address inputs and 8 data input/output connections.

Static RAM Interfacing

• The semiconductor RAM is broadly two types – Static RAM and Dynamic RAM.
• The semiconductor memories are organized as two dimensional arrays of memory locations.
• For example 4K * 8 or 4K byte memory contains 4096 locations, where each locations contains 8-bit data and only one of the 4096 locations can be selected at a time. Once a location is selected all the bits in it are accessible using a group of conductors called Data bus.
• For addressing the 4K bytes of memory, 12 address lines are required.
• In general to address a memory location out of N memory locations, we will require at least n bits of address, i.e. n address lines where n = Log2 N.
• Thus if the microprocessor has n address lines, then it is able to address at the most N locations of memory, where 2n=N. If out of N locations only P memory locations are to be interfaced, then the least significant p address lines out of the available n lines can be directly connected from the microprocessor to the memory chip while the remaining (n-p) higher order address lines may be used for address decoding as inputs to the chip selection logic.
• The memory address depends upon the hardware circuit used for decoding the chip select (CS). The output of the decoding circuit is connected with the CS pin of the memory chip.

• The general procedure of static memory interfacing with 8086 is briefly described as follows:

1. Arrange the available memory chip so as to obtain 16- bit data bus width. The upper 8-bit bank is called as odd address memory bank and the lower 8-bit bank is called as even address memory bank.

2. Connect available memory address lines of memory chip 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 A0 are used for decoding the required chip select signals for the odd and even memory banks. The CS of memory is derived from the o/p of the decoding circuit.

• As a good and efficient interfacing practice, the address map of the system should be continuous as far as possible, i.e. there should not be no windows in the map and no fold back space should be allowed.

• A memory location should have a single address corresponding to it, i.e. absolute decoding should be preferred and minimum hardware should be used for decoding.

Dynamic RAM

• Whenever a large capacity memory is required in a microcomputer system, the memory subsystem is generally designed using dynamic RAM because there are various advantages of dynamic RAM.

• E.g. higher packing density, lower cost and less power consumption. A typical static RAM cell may require six transistors while the dynamic RAM cell requires only a transistors along with a capacitor. Hence it is possible to obtain higher packaging density and hence low cost units are available.

• 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 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 RAM is called as ***Refreshcycle***.

• The refresh activity is similar to reading the data from each and every cell of memory, independent of the requirement of microprocessor. During this refresh period all other operations related to the memory subsystem are suspended. Hence the refresh activity causes loss of time, resulting in reduces system performance.

• However keeping in view the advantages of dynamic RAM, like low power consumption, high packaging density and low cost, most of the advanced computing system are designed using dynamic RAM, at the cost of operating speed.

• A dedicated hardware chip called as dynamic RAM controller is the most important part of the interfacing circuit.

• The ***Refresh cycle*** is different from the memory read cycle in the following aspects.

1. The memory address is not provided by the CPU address bus, rather it is generated by a refresh mechanism counter called as refresh counter.
2. Unlike memory read cycle, more than one memory chip may be enabled at a time so as to reduce the number of total memory refresh cycles.
3. The data enable control of the selected memory chip is deactivated, and data is not allowed to appear on the system data bus during refresh, as more than one memory units are refreshed simultaneously. This is to avoid the data from the different chips to appear on the bus simultaneously.
4. Memory read is either a processor initiated or an external bus master initiated and carried out by the refresh mechanism.

• Dynamic RAM is available in units of several kilobits to megabits of memory. This memory is arranged internally in a two dimensional matrix array so that it will have n rows and m columns. The row address n and column address m are important for the refreshing operation.

• For example, a typical 4K bit dynamic RAM chip has an internally arranged bit array of dimension 64 * 64 , i.e. 64 rows and 64 columns. The row address and column address will require 6 bits each. These 6 bits for each row address and column address will be generated by the refresh counter, during the refresh cycles.

• A complete row of 64 cells is refreshed at a time to minimize the refreshing time. Thus the refresh counter needs to generate only row addresses. The row addresses are multiplexed, over lower order address lines.

• The refresh signals act to control the multiplexer, i.e. when refresh cycle is in process the refresh counter puts the row address over the address bus for refreshing. Otherwise, the address bus of the processor is connected to the address bus of DRAM, during normal processor initiated activities.

• A timer, called refresh timer, derives a pulse for refreshing action after each refresh interval.

• Refresh interval can be qualitatively defined as the time for which a dynamic RAM cell can hold data charge level practically constant, i.e. no data loss takes place.

• Suppose the typical dynamic RAM chip has 64 rows, then each row should be refreshed after each refresh interval or in other words, all the 64 rows are to refreshed in a single refresh interval.

• This refresh interval depends upon the manufacturing technology of the dynamic RAM cell. It may range anywhere from 1ms to 3ms.

• Let us consider 2ms as a typical refresh time interval. Hence, the frequency of the refresh pulses will be calculated as follows:

• Refresh Time (per row) tr = (2 * 10 -3) / 64.

• Refresh Frequency fr = 64 / (2 * 10 -3) = 32 * 103 Hz.

• The following block diagram explains the refreshing logic and 8086 interfacing with dynamic RAM.

• Each chip is of 16K * 1-bit dynamic RAM cell array. The system contains two 16K byte dynamic RAM units. All the address and data lines are assumed to be available from an 8086 microprocessor system.
• The OE pin controls output data buffer of the memory chips. The CE pins are active high chip selects of memory chips. The refresh cycle starts, if the refresh output of the refresh timer goes high, OE and CE also tend to go high.
• The high CE enables the memory chip for refreshing, while high OE prevents the data from appearing on the data bus, as discussed in memory refresh cycle. The 16K * 1-bit dynamic RAM has an internal array of 128*128 cells, requiring 7 bits for row address. The lower order seven lines A0-A6 are multiplexed with the refresh counter output A10-A16.

Fig 4.4. Dynamic ram 2164

• The pin assignment for 2164 dynamic RAM is as in above fig.
• The RAS and CAS are row and column address strobes and are driven by thedynamic RAM controller outputs. A0 –A7 lines are the row or column addresslines, driven by the OUT0 – OUT7 outputs of the controller. The WE pinindicates memory write cycles. The DIN and DOUT pins are data pins for write andread operations respectively.
• In practical circuits, the refreshing logic is integrated inside dynamic RAMcontroller chips like 8203, 8202, 8207 etc.
• Intel's 8203 is a dynamic RAM controller that support 16K or 64K dynamicRAM chip. This selection is done using pin 16K/64K. If it is high, the 8203 isconfigured to control 16K dynamic RAM, else it controls 64K dynamic RAM.The address inputs of 8203 controller accepts address lines A1 to A16 on linesAL0-AL7 and AH0-AH7.
• The A0 lines is used to select the even or odd bank. The RD and WR signalsdecode whether the cycle is a memory read or memory write cycle and areaccepted as inputs to 8203 from the microprocessor.
• The WE signal specifies the memory write cycle and is not output from 8203 thatdrives the WE input of dynamic RAM memory chip. The OUT0 – OUT7 set ofeight pins is an 8-bit output bus that carries multiplexed row and columnaddresses are derived from the address lines A1-A16 accepted by the controller onits inputs AL0-AL7 and AH0-AH7.
• An external crystal may be applied between X0 and X1 pins, otherwise with the

OP2 pin at +12V, a clock signal may be applied at pin CLK.
• The PCS pin accepts the chip select signal derived by an address decoder. The
REFREQ pin is used whenever the memory refresh cycle is to be initiated by anexternal signal.
• The XACK signal indicates that data is available during a read cycle or it has been written if it is a write cycle. It can be used as a strobe for data latches or as aready signal to the processor.
• The SACK output signal marks the beginning of a memory access cycle.
• If a memory request is made during a memory refresh cycle, the SACK signal isdelayed till the starring of memory read or write cycle.
• Following fig shows the 8203 can be used to control a 256K bytes memorysubsystem for a maximum mode 8086 microprocessor system.
• This design assumes that data and address busses are inverted and latched, hencethe inverting buffers and inverting latches are used (8283-inverting buffer and8287- inverting latch).

3.7.2. 8086 Interrupts

- Interrupts provide a mechanism for quickly changing program environment. Transfer of program control is initiated by the occurrence of either an event internal to the microprocessor or an event in its external hardware

- When an interrupt signal occurs in external hardware indicating that an external device, such as a printer, requires service, the MPU must suspend what it is doing in the main part of the program and pass control to a special routine (interrupt-service routine) that performs the function required by the device. In the case of our example of a printer, the routine is usually called the printer driver, which is the piece of software when executed drives the printer output interface.

- The 8088 and 8086 microcomputers are capable of implementing any combination of up to 256 interrupts.

As Fig. 11-2, they are divided into five groups: external hardware interrupts, nonmaskable interrupt, software interrupts, internal interrupts, and reset

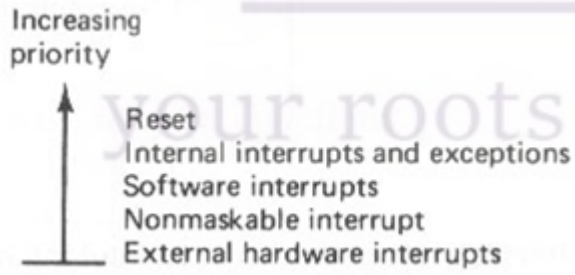


Fig 4.5. Types of interrupts and their priority

- The user defines the function of the external hardware, software, and nonmaskable interrupt. For instant, hardware interrupt are often assigned to devices such as the keyboard, printer, and timers. On the other hand, the function of the internal interrupts and reset are not user defined. They perform dedicated system functions.

- Hardware, software, and internal interrupts are serviced on a priority basis. Priority is achieved in two ways. First, the interrupt-processing sequence implemented in the 8088/8086 tests the occurrence of the various groups based on the hierarchy shown in Fig. 4.5. Thus, we see that internal interrupts are the highest-priority group, and the external hardware interrupts are the lowest-priority group.

- Second, each of the interrupts is given a different priority level by assigning it a type number. Type 0 identifies the highest-priority interrupt, and type 255 identifies the lowest-priority interrupt. Actually, a few of the type numbers are not available for use with software or hardware interrupts. This is because they are reserved for special interrupt functions of the 8088/8086, such as internal interrupts

- For instant, within the internal interrupt group, the interrupt known as divide error is assigned to type number 0. Therefore, it has the highest priority of the internal interrupts. Another internal interrupt, called overflow, is assigned the type number 4, Overflow is lowest-priority internal interrupt

- The importance of priority lies in the fact that, if an interrupt-service routine has been initiated to perform a function assigned to s specific priority level, only devices with higher priority are allowed are allowed to interrupt the active service routine. Lower-priority devices will have to wait until the current routine is completed before their for service can be acknowledged

- For hardware interrupts, this priority scheme is implemented in external hardware. For this reason, the user normally assigns tasks that must not be interrupted frequently to higher-priority levels and those cannot be interrupted to lower-priority levels.

- An example of a high-priority service routine that should not be interrupted is that for power failure. Once initiated, this routine should be quickly run to completion to assure that the microcomputer goes through an orderly power-down.

- A keyboard should also be assigned to a high-priority interrupt. This will assure that the keyboard buffer does not get full and lock out additional entries. On the other hand, device such as the floppy disk or hard disk controller are typically assigned to a lower priority level.

- An interrupt vector table is used to link the interrupt type numbers to the location of their service routine in the program-storage memory.

NRCM your roots to success

- Fig. 4.6 contains 256 address pointers (vectors), which are identified as vector 0 through vector 255. That is, one pointer corresponds to each of the interrupt types 0 through 255. These pointers identify the starting location of their service routines in program memory. The contents of this table may be either held as firmware in EPROMs or loaded into RAM as part of the system initialization routine.

- Fig. 11-3 starts at address 00000_{16} and end at $003FE_{16}$. This represents the first 1Kbytes of memory.

- Each of the 256 pointers requires two words (4 bytes) of memory and is always stored at an even-address boundary. The higher-addressed word of the two-word vector is called the base address. IT identifies the program memory segment in which the service routine resides.

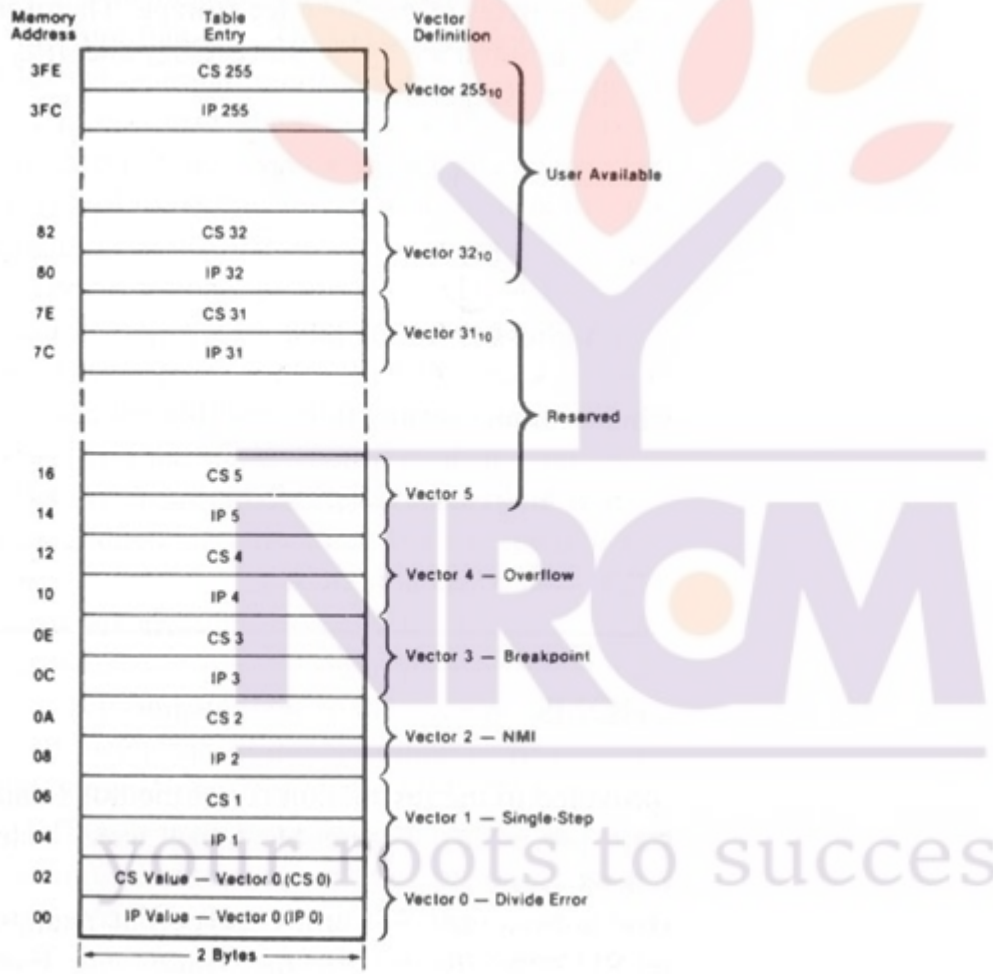


Fig 4.6. Interrupt vector table of 8086

- For example, the offset and base address for type number 255, IP_{255} and CS_{255}, are stored at word addresses $003FC_{16}$ and $003FE_{16}$, respectively. When loaded into the MPU, it points to the instruction at CS_{255}:IP_{255}
- Looking more closely at the table in Fig. 11-3, we find that the first 31 pointers either have dedicated functions or are reserved. For instance, pointers 1, 2, 3, and 4 are used by the 8088's and 8086's internal interrupts: divide error, single step, breakpoint, and overflow. Pointer 2 is used to identify the starting location of the nonmaskable interrupt's service routine. The next 27 pointers, 5 through 31, represent a reserved portion of the pointer table and should not be used.

EXAMPLE 11.1

At what address are CS_{50} and IP_{50} stored in memory?

Solution

Each vector requires four consecutive bytes of memory for storage. Therefore, its address can be found by multiplying the type number by 4. Since CS_{50} and IP_{50} represent the words of the type 50 interrupt pointer, we get

$$\text{Address} = 4 \times 50 = 200$$

Converting to binary form gives

$$\text{Address} = 11001000_2$$

and expressing it as a hexadecimal number results in

$$\text{Address} = C8_{16}$$

Therefore, IP_{50} is stored at $000C8_{16}$ and CS_{50} at $000CA_{16}$.

3.7.3. DOS and BIOS interrupts in 8086

INT 10h / AH = 0 - set video mode.
input:
AL = desired video mode.
these video modes are supported:
00h - text mode. 40x25. 16 colors. 8 pages.
03h - text mode. 80x25. 16 colors. 8 pages.
13h - graphical mode. 40x25. 256 colors. 320x200 pixels. 1 page

NRCM your roots to success...

INT 10h / **AH = 01h** - set text-mode cursor shape.
input:
CH = cursor start line (bits 0-4) and options (bits 5-7).
CL = bottom cursor line (bits 0-4).
when bit 5 of CH is set to **0**, the cursor is visible. when bit 5 is **1**, the cursor is not visible.

INT 10h / **AH = 2** - set cursor position.
input:
DH = row.
DL = column.
BH = page number (0..7).

INT 10h / **AH = 03h** - get cursor position and size.
input:
BH = page number.
return:
DH = row.
DL = column.
CH = cursor start line.
CL = cursor bottom line.

INT 10h / **AH = 05h** - select active video page.
input:
AL = new page number (0..7).
the activated page is displayed.

INT 10h / **AH = 06h** - scroll up window.
INT 10h / **AH = 07h** - scroll down window.
input:
AL = number of lines by which to scroll (00h = clear entire window).
BH = attribute used to write blank lines at bottom of window.
CH, CL = row, column of window's upper left corner.
DH, DL = row, column of window's lower right corner.

your roots to success...

INT 10h / **AH = 08h** - read character and attribute at cursor position.
input:
BH = page number.
return:
AH = attribute.
AL = character.

INT 10h / **AH = 09h -** write character and attribute at cursor position.
input:
AL = character to display.
BH = page number.
BL = attribute.
CX = number of times to write character.

INT 10h / **AH = 0Ah -** write character only at cursor position.
input:
AL = character to display.
BH = page number.
CX = number of times to write character.

INT 10h / **AH = 0Eh -** teletype output.
input:
AL = character to write.
this functions displays a character on the screen, advancing the cursor and scrolling the screen as necessary. the printing is always done to current active page.

INT 10h / **AH = 13h -** write string.
input:
AL = write mode:
bit 0: update cursor after writing;
bit 1: string contains attributes.
BH = page number.
BL = attribute if string contains only characters (bit 1 of AL is zero).
CX = number of characters in string (attributes are not counted).
DL,DH = column, row at which to start writing.
ES:BP points to string to be printed.

INT 10h / **AX = 1003h** - toggle intensity/blinking.
input:
BL = write mode:
0: enable intensive colors.
1: enable blinking (not supported by the emulator and windows command prompt).
BH = 0 (to avoid problems on some adapters).

NRCM your roots to success...

INT 11h - get BIOS equipment list.
return:
AX = BIOS equipment list word, actually this call returns the contents of the word at 0040h:0010h.
Currently this function can be used to determine the number of installed number of floppy disk drives.

Bit fields for BIOS-detected installed hardware:

bit(s)	Description
15-14	Number of parallel devices.
13	Reserved.
12	Game port installed.
11-9	Number of serial devices.
8	Reserved.
7-6	Number of floppy disk drives (minus 1): 00 single floppy disk; 01 two floppy disks; 10 three floppy disks; 11 four floppy disks.
5-4	Initial video mode: 00 EGA,VGA,PGA, or other with on-board video BIOS; 01 40x25 CGA color. 10 80x25 CGA color (emulator default). 11 80x25 mono text.
3	Reserved.
2	PS/2 mouse is installed.
1	Math coprocessor installed.
0	Set when booted from floppy

INT 12h - get memory size.
return:
AX = kilobytes of contiguous memory starting at absolute address 00000h, this call returns the contents of the word at 0040h:0013h.

your roots to success...

INT 15h / **AH = 86h** - BIOS wait function.
input:
CX:DX = interval in microseconds
return:

CF clear if successful (wait interval elapsed),
CF set on error or when wait function is already in progress.

Note:
the resolution of the wait period is 977 microseconds on many systems (1 million microseconds - 1 second).
Windows XP does not support this interrupt (always sets CF=1).

INT 16h / **AH = 00h** - get keystroke from keyboard (no echo).
return:
AH = BIOS scan code.
AL = ASCII character.
(if a keystroke is present, it is removed from the keyboard buffer).

INT 16h / **AH = 01h** - check for keystroke in the keyboard buffer.
return:
ZF = 1 if keystroke is not available.
ZF = 0 if keystroke available.
AH = BIOS scan code.
AL = ASCII character.
(if a keystroke is present, it is not removed from the keyboard buffer).

INT 21h / **AH=1** - read character from standard input, with echo, result is stored in **AL**.
if there is no character in the keyboard buffer, the function waits until any key is pressed.

INT 21h / **AH=2** - write character to standard output.
entry: **DL** = character to write, after execution **AL = DL**.

INT 21h / **AH=5** - output character to printer.
entry: **DL** = character to print, after execution **AL = DL**

INT 21h / **AH=6** - direct console input or output.
parameters for output: **DL** = 0..254 (ascii code)
parameters for input: **DL** = 255
for output returns: AL = DL
for input returns: **ZF** set if no character available and **AL = 00h**, **ZF** clear if character available.
AL = character read; buffer is cleared.

INT 21h / **AH=9** - output of a string at **DS:DX**. String must be terminated by '**$**'.

your roots to success...

INT 21h / **AH=0Ah** - input of a string to **DS:DX**, fist byte is buffer size, second byte is number of chars actually read. this function does **not** add '$' in the end of string. to print using **INT 21h** / **AH=9** you must set dollar character at the end of it and start printing from address **DS:DX + 2**.

INT 21h / **AH=0Ch** - flush keyboard buffer and read standard input.
entry: **AL** = number of input function to execute after flushing buffer (can be 01h,06h,07h,08h, or 0Ah - for other values the buffer is flushed but no input is attempted); other registers as appropriate for the selected input function.

INT 21h / **AH= 0Eh** - select default drive.
Entry: **DL** = new default drive (0=A:, 1=B:, etc)

Return: **AL** = number of potentially valid drive letters

3.7.4. 8259 Interrupt controller

General Description

The Digital Blocks DB8259A Programmable Interrupt Controller core is a full functionequivalent to the Intel 8259A / Intersil 82C59A / NEC uPD8259A devices. TheDB8259A Interrupt Controller manages up to eight vectored priority interrupts for amicroprocessor. Using multiple instantiations of the DB8259A core and programming itto cascade mode enables up to sixty-four vectored priority interrupts. More than sixtyfour vectored interrupts can be accomplished by programming the DB8259A core to PollCommand Mode. Interrupt sources may be either edge or level triggered.

Features

The DB8259A supports eight vectored priority interrupts per core, sixty-four vectoredpriority interrupts with cascading, and more than sixty-four vectored interrupts withprogramming in Poll Command Mode.

Programming for all 8259A modes and operational features:

- MCS-80/85 and 8088/8086 processor modes
- Fully Nested Mode and Special Fully Nested Mode
- Special Mask Mode
- Buffered Mode
- Poll Command Mode
- Cascade Mode with Master or Slave selection
- Automatic End-of-Interrupt Mode

NRCM your roots to success...

- Specific and Non-Specific End-of-Interrupt Commands
- Automatic & Specific Rotation
- Edge and level triggered interrupt input modes
- Reading of Interrupt Request Register (IRR) and In-Service Register (ISR) through data bus
- Writing and reading of Interrupt Mask Register (IMR) through data bus
- Cost-effective CPLD/FPGA replacement solution for 8259A merchant components (Intel/Intersil/NEC)

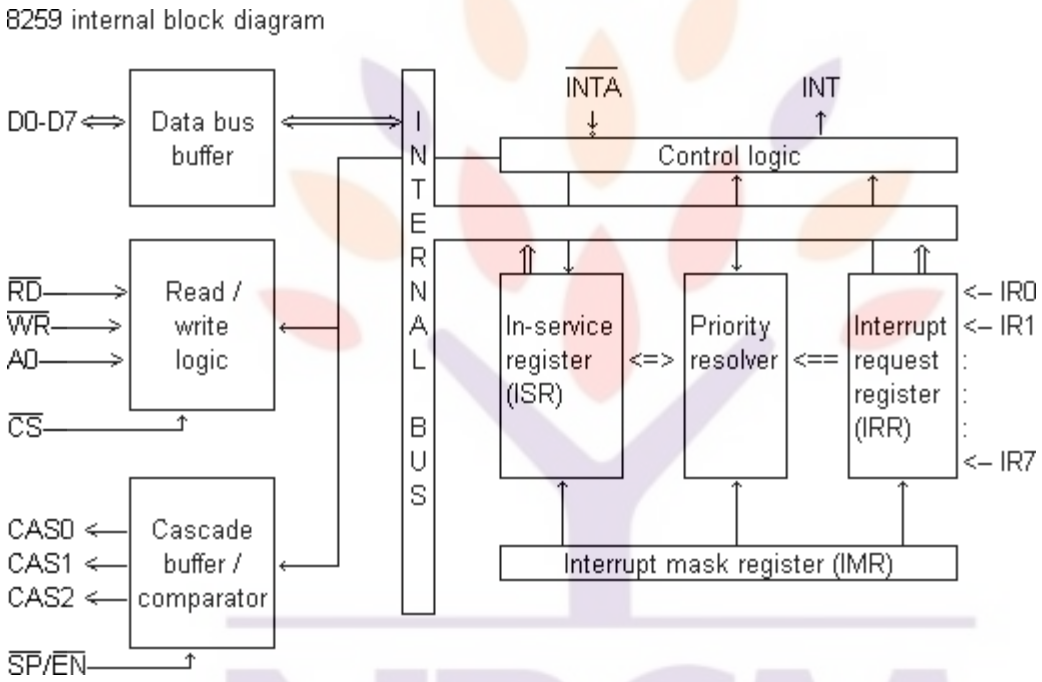


Fig 4.7. 8259 PIC

Functional Description

The DB8259A core is partitioned into modules as shown in Figure 1 and described

below.

Data Bus Buffer

The 3-state, bi-directional 8-bit buffer is used to interface the DB8259A core to themicroprocessor system data bus. Control words and status information are transferredthrough the Data Bus Buffer.

Read / Write Logic

The Read / Write Logic processes the data bus control signals and stores the outputcommands from the microprocessor. The commands are the four Initialization CommandWord (ICW) registers and the three Operation Command Word (OCW) registers. Theseregisters contain the various programming control formats for device operation. TheRead/Write Logic block also enables the status of the DB8259A core to be transferred tothe system data bus.

Cascade Buffer Comparator

The Cascade Buffer Comparator stores and compares the IDs of all DB8259A coresinstantiated in the system. The associated three I/O pins (CAS0-2) are outputs when theDB8259A is used as a Master and are inputs when the DB8259A is used as a Slave. As aMaster, the DB8259A core sends the ID of the interrupting slave device onto the CAS0-2lines. The selected Slave will send its preprogrammed subroutine address onto the systemdata bus during the next one or two consecutive INTAn pulses.

Control Logic

The Control Logic checks for INTAn pulses which cause the DB8259A to release vectorinformation onto the system data bus. The format of this data depends on the systemmode of the DB8259A.

Interrupt Request Register (IRR) and In-Service Register (ISR)

The interrupts at the IR input lines are handled by two registers in tandem, the InterruptRequest Register (IRR) and the In-Service Register (ISR). The IRR is used to store all theinterrupt levels which are requesting service. The ISR is used to store all the interruptlevels that are being serviced by the microprocessor.

Priority Resolver

The Priority Resolver determines the priorities of the bits set in the IRR. The highestpriority is selected and strobed into the corresponding bit of the ISR during an INTAncycle.

Interrupt Mask Register (IMR)

The Interrupt Mask Register (IMR) stores the bits that control the interrupt lines to bemasked. The IMR operates on the IRR. Masking of a higher priority input will not affectthe interrupt request lines of lower priority

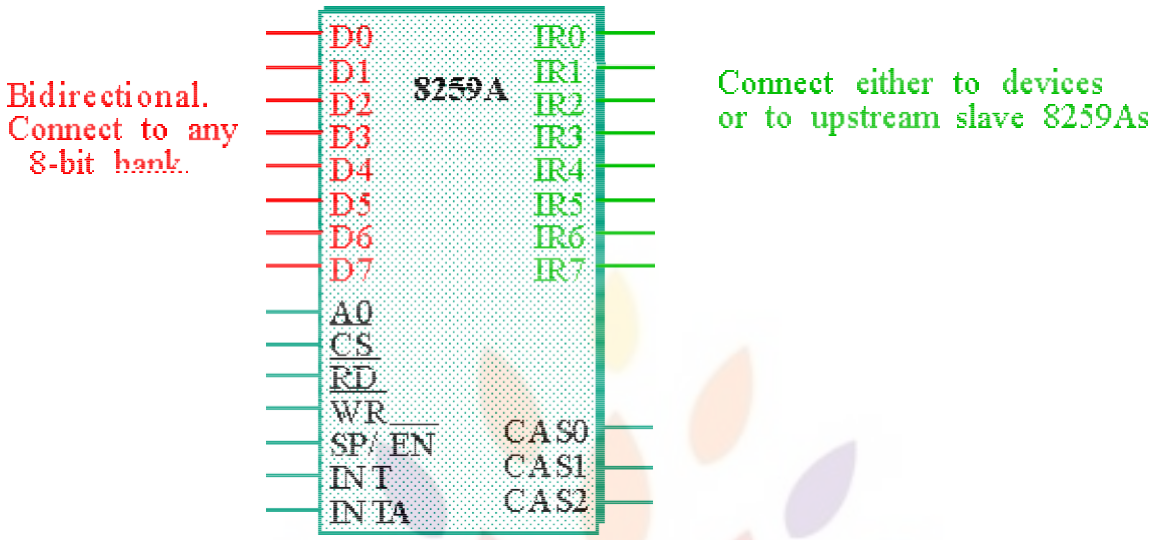


Fig 4.8. 8259 pin diagram

Each of the lines in the above image displays each of the controllers electronic pins. These electronic pins are the connections between the controller and the rest of the system.

This is the chip that we will need to program in order to handle IRQ's within an operating system. Let's look at this closer at each pin.

- **WR Pin:** This pin connects to a write strobe signal (One of 8 on a Pentium)
- **RD Pin:** This connects to the **IOCR (Input Output Control Routine)** signal.
- **INT Pin:** Connects to the INTR pin on the microprocessor.
- **INTA Pin:** Connects to the INTA pin on the microprocessor**.**
- **A0 Pin:** Selects different Command WORDS
- **CS Pin:** Enables the chip for programming and control.
- **SP/EN Pin:** Slave program (SP) / Enable Buffer (EN).
    - Slave Program (1=Master, 0=Slave)
    - Enable Buffer (Controls data bus transievers when in buffered mode)
- **CAS0, CAS1, CAS2 Pins**: Used to output from master to slave PIC controllers in cascaded systems.
- **D0 - D7 Pins:** 8 bit Data connector pins**.**

your roots to success...

There are a couple of important pins here. Pins D0-D7 provide a way for an external device to communicate with the PIC. This is like a small data bus--It provides a way to send data over to the PIC, like...An interrupt number, perhaps?

Remember that we can connect PIC's together. This allows us to provide support for up to 64 IR numbers. In other words--64 hardware interrupts. CAS0, CAS1, and CAS2 pins provide a way to send signals between these PIC's.

Look at the INT and INTA pins. Remember from the **Processor Perspective** section that the processors' own INT and INTA pins connect to these pins on the PIC. Remember that, when about to execute an interrupt, the processor clears the Interrupt (IF) and Trap flags (TF) from the FLAGS register, which disables the INTR pin. **The PIC's INT pin connects to the processors' INTR pin.**

This means that the processor, essentially, disables the PIC's INT pin when executing an interrupt.With this, the pins IR0-IR7 can be streamed to other PIC's. These 8 pins represent the 8 bit interrupt number to be executed. Notice that this, as being an 8 bit value, provides a way to allow up to 256 hardware interrupts. these lines provide a way to send the interrupt number to another PIC controller, so that controller could handle it instead.

The important thing to note is that **We can combine multiple PIC's to support more interrupt routine numbers.** The IR lines connect to another PIC's data lines to transfer data over. As there are only 8 lines (8 bits), we can only connect up to 8 PIC's together, providing support for up to 64 interrupt numbers.

Programming the PIC revolves around the use of sending **Command Bytes** through the 8 bit data line that the PIC's have. This 8 bit command byte follows specific formats that describe what the PIC is to do.

8259A Registers

The 8259A has several internal registers, similar to the processor.

Command Register

This is a write only register that is used to send commands to the microcontroller. There are a lot of different commands that you can send. Some commands are used to read from other registers, while other command are used to initialize and sending data, such as End of Interrupt (EOI). We will cover these commands later.

your roots to success...

Status register

This is a read only register that can be accessed to determine the status of the PIC.

Interrupt Request Register (IRR)

This register specifies which interrupts is pending acknowledgment.

Note: This register is internal, and cannot be accessed directly.

Interrupt Request Register (IRR)		
Bit Number	IRQ Number (Primary controller)	IRQ Number (Slave controller)
0	IRQ0	IRQ8
1	IRQ1	IRQ9
2	IRQ2	IRQ10
3	IRQ3	IRQ11
4	IRQ4	IRQ12
5	IRQ5	IRQ13
6	IRQ6	IRQ14
7	IRQ7	IRQ15

If a bit is set, the interrupt has been signaled by a device, and the PIC has signaled the CPU, but is awaiting acknowledgment from the CPU to go ahead with the interrupt.

In-Sevice Register (ISR)

This register specifies which interrupts have already been acknowledged, but are awaiting for the **End of Interrupt (EOI)** signal. The EOI signal is very important as it determins the end of an interrupt.

Note: We will need to send the EOI signal upon completion of the interrupt to let the 8259A acknowledge the interrupt. Not doing so will result in undefined behavior or malfunction. More on this later.

Note: This register is internal, and cannot be accessed directly.

In Service Register (ISR)		
Bit Number	IRQ Number (Primary controller)	IRQ Number (Slave controller)
0	IRQ0	IRQ8
1	IRQ1	IRQ9
2	IRQ2	IRQ10
3	IRQ3	IRQ11
4	IRQ4	IRQ12
5	IRQ5	IRQ13
6	IRQ6	IRQ14
7	IRQ7	IRQ15

your roots to success...

If a bit is set, the current IRQ has been acknowledged by the CPU to go ahead and begin executing. The PIC uses this register to determine what IRQ is currently being executed.

Interrupt Mask Register (IMR)

This specifies what interrupts are to be ignored, and not acknowledged. this allows us to focus on executing certain, more important interrupts before executing the interrupts specified in this register.This is an 8 bit register, where each bit determines if an interrupt is disabled or not. If the bit is 0, it is enabled. If it is a 1, the interrupt device is disabled.

Interrupt Mask Register (IMR)		
Bit Number	IRQ Number (Primary controller)	IRQ Number (Slave controller)
0	IRQ0	IRQ8
1	IRQ1	IRQ9
2	IRQ2	IRQ10
3	IRQ3	IRQ11
4	IRQ4	IRQ12
5	IRQ5	IRQ13
6	IRQ6	IRQ14
7	IRQ7	IRQ15

This is an important register, as it allows us to enable and disable interrupts from certain devices. Each of these IRQ's represent the device listed in the **x86 Hardware Interrupts** table shown above.

For example, lets say we want to enable COM1 (Serial Port 1). Looking at the x86 Hardware Interrupt Table, we can see that this is mapped to IRQ 4. So, in order to enable COM1 interrupts, all we need to do is set the IRQ4 bit for the primary PIC's Interrupt Mask Register. This register is mapped to the software port number 0x21 (We will cover this later.) So, all we need to do is set the bit by writing to this port location.

```
in      al, 0x21                ; read in the primary PIC Interrupt Mask
Register (IMR)
and     al, 0xEF                ; 0xEF => 11101111b. This sets the IRQ4 bit
(Bit 5) in AL
out     0x21, al                ; write the value back into IMR
```

When a hardware interrupt occurs, **The 8259A Masks out all other interrupts until it recieves an End of Interrupt (EOI) signal.** We will need to send the EOI upon completion of the interrupt. We will look at this later.

8259A Software Port Mappings

Like all hardware controllers, the BIOS POST maps each controller to use a specific region of software ports. Because of this, in order to communicate with the PIC controllers, we need to use software ports.

8259A Software Port Map	
Port Address	Description
0x20	Primary PIC Command and Status Register
0x21	Primary PIC Interrupt Mask Register and Data Register
0xA0	Secondary (Slave) PIC Command and Status Register
0xA1	Secondary (Slave) PIC Interrupt Mask Register and Data Register

Notice the Primary PIC's Interrupt Mask Register is mapped to Port 0x21. We have seen this before, havn't we?

The **Command Register** and **Status Register** are to different registers that share the same port number. The command register is write only, while the status register is read only. This is an important difference, as the PIC determins what register to access depending on weather the write or read lines are set.

We will need to be able to write to these ports to communicate with individual device registers and control the PICs. Lets now take a look at the commands for the PIC.

8259A Commands

Setting up the PIC is fairly complex. It is done through a series of **Command Words**, which are a bit pattern that contains various of states used for initialization and operation. This might seem a little complex, but it is not too hard.Because of this, lets first look at how to initialize the PIC controllers for our use, followed by operating and controlling the PICs.

Initialization Control Words (ICW)

The purpose of initializing the PIC is to remap the PIC's IRQ numbers to our own. this insures the proper IRQ is generated when a hardware interrupt happens.In order to initialize the PIC, we must send a command byte (Known as an **Initialization Control Word (ICW)**) to the primary PIC Command Register. **This is ICW 1**.There can be up to 4 Initialization Control Words. These are not required, but are often needed. Let's take a look at them.

Note: If there are multiple PICs in the system that is to be cascaded with each other, we must send the ICW's to both of the PICs!

ICW 1

This is the primary control word used to initialize the PIC. this is a 7 bit value that must be put in the primary PIC command register. This is the format:

Initialization Control Word (ICW) 1		
Bit Number	Value	Description
0	IC4	If set(1), the PIC expects to recieve IC4 during initialization.
1	SNGL	If set(1), only one PIC in system. If cleared, PIC is cascaded with slave PICs, and ICW3 must be sent to controller.
2	ADI	If set (1), CALL address interval is 4, else 8. This is useually ignored by x86, and is default to 0
3	LTIM	If set (1), Operate in Level Triggered Mode. If Not set (0), Operate in Edge Triggered Mode
4	1	Initialization bit. Set 1 if PIC is to be initialized
5	0	MCS-80/85: Interrupt Vector Address. x86 Architecture: Must be 0
6	0	MCS-80/85: Interrupt Vector Address. x86 Architecture: Must be 0
7	0	MCS-80/85: Interrupt Vector Address. x86 Architecture: Must be 0

As you can see, there is alot going on here. We have seen some of these before. This is not as hard as it seems, as most of these bits are not used on the x86 platform.

To initialize the primary PIC, all we need to do is create the initil ICW and set the appropraite bits. So, lessee...

- **Bit 0** - Set to 1 so we can sent ICW 4
- **Bit 1** - PIC cascading bit. x86 architectures have 2 PICs, so we need the primary PIC cascaded with the slave. Keep it 0
- **Bit 2** - CALL address interval. Ignored by x86 and kept at 8, so keep it 0
- **Bit 3** - Edge triggered/Level triggered mode bit. By default, we are in edge triggered, so leave it 0
- **Bit 4** - Initialization bit. Set to 1
- **Bits 5...7** - Unused on x86, set to 0.

Looking at the above, the final bit pattern becomes **00010001**, or 0x11. So, to initialize the PIC, send 0x11 to the primary PIC controller register, mapped to port 0x20...

```
; Setup to initialize the primary PIC. Send ICW 1
mov     al, 0x11
out     0x20, al
```

```
        ; Remember that we have 2 PICs. Because we are cascading with this
second PIC, send ICW 1 to second PIC command register
        out     0xA0, al        ; slave PIC command register
```

Because we have enabled cascading, we need to send ICW 3 to the controller as well. Also, because we have set bit 0, we must also send ICW 4. More on those later. For now, let's take a look at ICW 2.

ICW 2

This control word is used to map the base address of the IVT of which the PIC are to use. **This is important!**

Initialization Control Word (ICW) 2		
Bit Number	Value	Description
0-2	A8/A9/A10	Address bits A8-A10 for IVT when in MCS-80/85 mode.
3-7	A11(T3)/A12(T4)/A13(T5)/A14(T6)/A15(T7)	Address bits A11-A15 for IVT when in MCS-80/85 mode. **In 80x86 mode, specifies the interrupt vector address.** May be set to 0 in x86 mode.

During initialization, we need to send ICW 2 to the PICs to tell them where the base address of the IRQ's to use. If an ICW1 was sent to the PICs (With the initialization bit set), you must send ICW2 next. **Not doing so can result in undefined results.** Most likley the incorrect interrupt handler will be executed.

Unlike ICW 1, which is placed into the PIC's data registers, ICW 2 is sent to the data Registers, as software ports 0x21 for the primary PIC, and port 0xA1 for the secondary PIC. (Please see the**8259A Software Port Map** table for a complete listing of PIC software ports).

Okay, so assuming we have just sent an ICW 1 to both PICs (Please see the above section), lets send an ICW 2 to both PICs. This will map a base IRQ address to both PICs.This is very simple, but we must be careful at where we map the PICs to. Remember that the first 31 interrupts (0x0-0x1F) are reserved (Please see the above **x86 Interrupt Vector Table (IVT)**table). As such, we have to insure we do not use any of these IRQ numbers.Instead, lets map them to IRQs 32-47, right after these reserved interrupts. the first 8 IRQ's are handled by the primary PIC, so we map the primary PIC to the base address of 0x20 (32 decimal), and the secondary PIC at 0x28 (40 decimal). Remember there are 8 IRQ's for each PIC.

```
        ; send ICW 2 to primary PIC
        mov     al, 0x20                ; Primary PIC handled IRQ 0..7. IRQ 0
is now mapped to interrupt number 0x20
```

```
        out     0x21, al

        ; send ICW 2 to secondary controller
        mov     al, 0x28                ; Secondary PIC handles IRQ's 8..15.
IRQ 8 is now mapped to use interrupt 0x28
        out     0xA1, al
```

ICW 3

This is an important command word. It is used to let the PICs know what IRQ lines to use when communicating with each other.

ICW 3 Command Word for Primary PIC

Initialization Control Word (ICW) 3 - Primary PIC		
Bit Number	Value	Description
0-7	S0-S7	Specifies what Interrupt Request (IRQ) is connected to slave PIC

ICW 3 Command Word for Secondary PIC

Initialization Control Word (ICW) 3 - Secondary PIC		
Bit Number	Value	Description
0-2	ID0	IRQ number the master PIC uses to connect to (**In binary notation**)
3-7	0	Reserved, must be 0

We must send an ICW 3 whenever we enable cascading within ICW 1. this allows us to set which IRQ to use to communicate with each other. Remember that the 8259A Microcontroller relies on the IR0-IR7 pins to connect to other PIC devices. With this, it uses the CAS0-CAS2 pins to communicate with each other.We need to let each PIC know about each other and how they are connected. We do this by sending the ICW 3 to both PICs containing which IRQ line to use for both the master and associated PICs.

Remember: The 80x86 architecture uses IRQ line 2 to connect the master PIC to the slave PIC.

your roots to success...

Knowing this, and remembering that we need to write this to the data registers for both PICs, we need to follow the formats shown above.Note that, in the ICW 3 for the primary PIC, **Each bit represents an interrupt request.** That is...

IRQ Lines for ICW 2 (Primary PIC)	
Bit Number	IRQ Line

0	IR0
1	IR1
2	IR2
3	IR3
4	IR4
5	IR5
6	IR6
7	IR7

Notice that IRQ 2 is Bit 2 within ICW 3. So, in order to set IRQ 2, we need to set bit 2 (Which is at 0100 binary, or 0x4).Here is an example of sending ICW 3 to the primary PIC:

```
; Send ICW 3 to primary PIC
mov     al, 0x4         ; 0x4 = 0100 Second bit (IR Line 2)
out     0x21, al        ; write to data register of primary PIC
```

To send this to the secondary PIC, we must remember that we must send this in **binary notation**. Please refer to the table above. Note that only Bits 0...2 are used to represent the IRQ line. By using binary notation, we can refer to the 8 IRQ lines to choose from:

IRQ Lines for ICW 2 (Secondary PIC)

Binary	IRQ Line
000	IR0
001	IR1
010	IR2
011	IR3
100	IR4
101	IR5
110	IR6
111	IR7

Simple enough. Notice that this just follows a binary<->decimal conversation in the above table.Because we are connected by IRQ line 2, we need to use bit 1 (Shown above).Here is a complete example, that sends a ICW 2 to both primary and secondary PIC controllers:

```
; Send ICW 3 to primary PIC
mov     al, 0x4         ; 0x04 => 0100, second bit (IR line 2)
out     0x21, al        ; write to data register of primary PIC

; Send ICW 3 to secondary PIC
```

```
        mov     al, 0x2         ; 010=> IR line 2
        out     0xA1, al        ; write to data register of secondary PIC
```

Okay, so now both PICs are connected to use IR line 2 to communicate with each other. We have also set a base interrupt number for both PICs to use.This is great, but we are not done yet. Remember that, when building up ICW 1, **if bit 0 is set, the PIC will be expecting us to send it ICW 4.** As such, we need to send ICW 4, the final ICW, to the PICs.

ICW 4

Yey! This is the final initialization control word. This controls how everything is to operate.

Initialization Control Word (ICW) 4		
Bit Number	Value	Description
0	uPM	If set (1), it is in 80x86 mode. Cleared if MCS-80/86 mode
1	AEOI	If set, on the last interrupt acknowledge pulse, controller automatically performs End of Interrupt (EOI) operation
2	M/S	Only use if BUF is set. If set (1), selects buffer master. Cleared if buffer slave.
3	BUF	If set, controller operates in buffered mode
4	SFNM	Special Fully Nested Mode. Used in systems with a large amount of cascaded controllers.
5-7	0	Reserved, must be 0

This is a pretty powerful function. Bits 5..7 are always 0, so lets focus on the other bits and peices (pun entended ;))

The PIC was originally designed to be a generic microcontroller, even before the 80x86 existed. As such, it containes alot of different operation modes designed for different systems. one of these modes is the **Special Fully Nested Mode**.The x86 family does not support this mode, so you can saftley set bit 4 to 0.

Bit 3 is used for buffered mode. For now, set this to 0. We will cover modes of operation later. Bit 2 is only used when bit 3 is set, so set this to 0. With this, Bit 1 is rairly used either.As such, we only need to set bit 0, which enables the PIC for 80x86 mode.Simple enough. So, to send ICW 4, all we need to do is this:

```
        mov     al, 1           ; bit 0 enables 80x86 mode

        ; send ICW 4 to both primary and secondary PICs
        out     0x21, al
        out     0xA1, al
```

Initializing the PIC - Putting it together

Believe it or not, but we have already went over this. In initilizing the PIC, all we need to do is send the correct ICW's to the PIC.Lets put everything from the previous section together to initialize the PIC for better understanding of how everything is put together:

```
;*****************************************************************************
; Map the 8259A PIC to use interrupts 32-47 within our interrupt table
;*****************************************************************************

%define ICW_1 0x11                              ; 00010001 binary. Enables
initialization mode and we are sending ICW 4

%define PIC_1_CTRL 0x20                                 ; Primary PIC control
register
%define PIC_2_CTRL 0xA0                                 ; Secondary PIC control
register

%define PIC_1_DATA 0x21                                 ; Primary PIC data
register
%define PIC_2_DATA 0xA1                                 ; Secondary PIC data
register

%define IRQ_0   0x20                            ; IRQs 0-7 mapped to use
interrupts 0x20-0x27
%define IRQ_8   0x28                            ; IRQs 8-15 mapped to use
interrupts 0x28-0x36

MapPIC:

; Send ICW 1 - Begin initialization -------------------------

        ; Setup to initialize the primary PIC. Send ICW 1

        mov     al, ICW_1
        out     PIC_1_CTRL, al

; Send ICW 2 - Map IRQ base interrupt numbers ---------------

        ; Remember that we have 2 PICs. Because we are cascading with this
second PIC, send ICW 1 to second PIC command register

        out     PIC_2_CTRL, al

        ; send ICW 2 to primary PIC

        mov     al, IRQ_0
        out     PIC_1_DATA, al

        ; send ICW 2 to secondary controller

        mov     al, IRQ_8
        out     PIC_2_DATA, al
```

```
; Send ICW 3 - Set the IR line to connect both PICs ---------

        ; Send ICW 3 to primary PIC

        mov     al, 0x4                 ; 0x04 => 0100, second bit (IR line 2)
        out     PIC_1_DATA, al          ; write to data register of primary PIC

        ; Send ICW 3 to secondary PIC

        mov     al, 0x2                 ; 010=> IR line 2
        out     PIC_2_DATA, al          ; write to data register of secondary
PIC

; Send ICW 4 - Set x86 mode ----------------------------------------------------------------------

        mov     al, 1                   ; bit 0 enables 80x86 mode

        ; send ICW 4 to both primary and secondary PICs

        out     PIC_1_DATA, al
        out     PIC_2_DATA, al

; All done. Null out the data registers

        mov     al, 0
        out     PIC_1_DATA, al
        out     PIC_2_DATA, al
```

Now the PIC is initialized. Whenever an hardware interrupt accors, it will call our interrupts 32 - 47 that we have previously defined somewhere within the Interrupt Vector Table (IVT). This allows us to track hardware interrupts.

Operation Command Words (OCW)

Yippee! Now that the ugly initialization stuff is out of the way, we can finally focus on standard controlling and operation of the PIC. This is done by writing and reading from various registers through**Operation Control Words (OCW)'s**.

OCW 1

OCW 1 represents the value inside of the **Interrupt Mask register (IMR)**. To obtain the current OCW 1, all you need to do is read from the IMR.

Remember that the IMR is mapped to the same port that the status register is at. Because the status register is read only, the PIC can determin what register to access based off if this is a read or write operation.

We have looked at the IMR register above when we covered the PIC registers.

OCW 2

This is the primary control word used to control the PIC. Lets take a look...

Operation Command Word (OCW) 2		
Bit Number	Value	Description
0-2	L0/L1/L2	Interrupt level upon which the controller must react
3-4	0	Reserved, must be 0
5	EOI	End of Interrupt (EOI) request
6	SL	Selection
7	R	Rotation option

OCW2 Commands			
R Bit	SL Bit	EOI Bit	Description
0	0	0	Rotate in Automatic EOI mode (CLEAR)
0	0	1	Non specific EOI command
0	1	0	No operation
0	1	1	Specific EOI command
1	0	0	Rotate in Automatic EOI mode (SET)
1	0	1	Rotate on non specific EOI
1	1	0	Set priority command
1	1	1	Rotate on specific EOI

Sending End of Interrupt (EOI)

As you know, when a hardware interrupt triggers, all other interrupts are masked off inside of the **Interrupt Mask Register** until an EOI signal is sent to the primary controller. This means, we must send an EOI to insure all hardware interrupts are enabled at the end of our **Interrupt Routine (IR)**.

Looking at the above table, we can send a non specific EOI command to signal EOI to the controller. Because the EOI bit is bit 4 within the OCW 2, all we need to do is set bit 4 (010000 = 0x20):

```
; send EOI to primary PIC

mov     al, 0x20        ; set bit 4 of OCW 2
out     0x20, al        ; write to primary PIC command register
```

3.7.5. 8251 DMA controller

- It is a device to transfer the data directly between IO device and memory without through the CPU. So it performs a high-speed data transfer between memory and I/O device.

The features of 8257 is,

- The 8257 has four channels and so it can be used to provide DMA to four I/O devices
- Each channel can be independently programmable to transfer up to 64kb of data by DMA.
- Each channel can be independently perform read transfer, write transfer and verify transfer.

Functional Block Diagram of 8257:

- The functional blocks of 8257 are data bus buffer, read/write logic, control logic, priority resolver and four numbers of DMA channels.

The functional block diagram of 8257 is shown in fig.

NRCM
your roots to success...

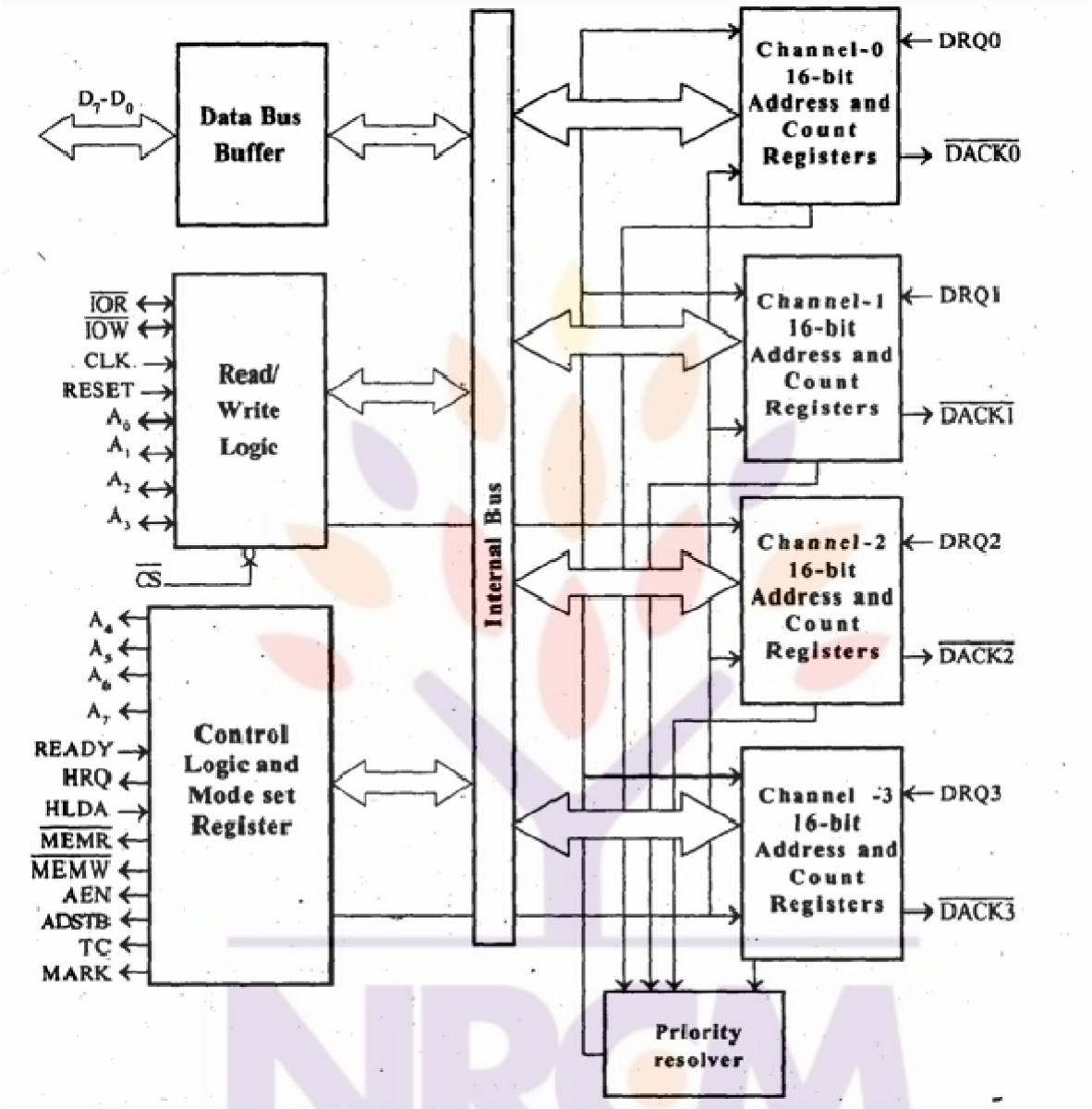


Fig 4.8. 8257 DMA controller

- Each channel of 8257 Block diagram has two programmable 16-bit registers named as address register and count register.
- Address register is used to store the starting address of memory location for DMA data transfer.
- The address in the address register is automatically incremented after every read/write/verify transfer.

- The count register is used to count the number of byte or word transferred by DMA. The format of count register is,

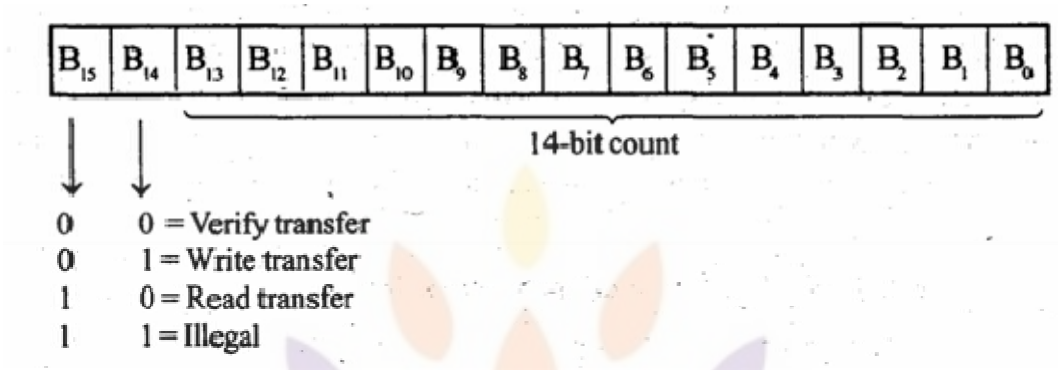


Fig 4.9. 8257 count register

- 14-bits B0-B13 is used to count value and a 2-bits is used for indicate the type of DMA transfer (Read/Write/Veri1 transfer).

- In read transfer the data is transferred from memory to I/O device.

- In write transfer the data is transferred from I/O device to memory.

- Verification operations generate the DMA addresses without generating the DMA memory and I/O control signals.

- The 8257 has two eight bit registers called mode set register and status register. The format of mode set register is,

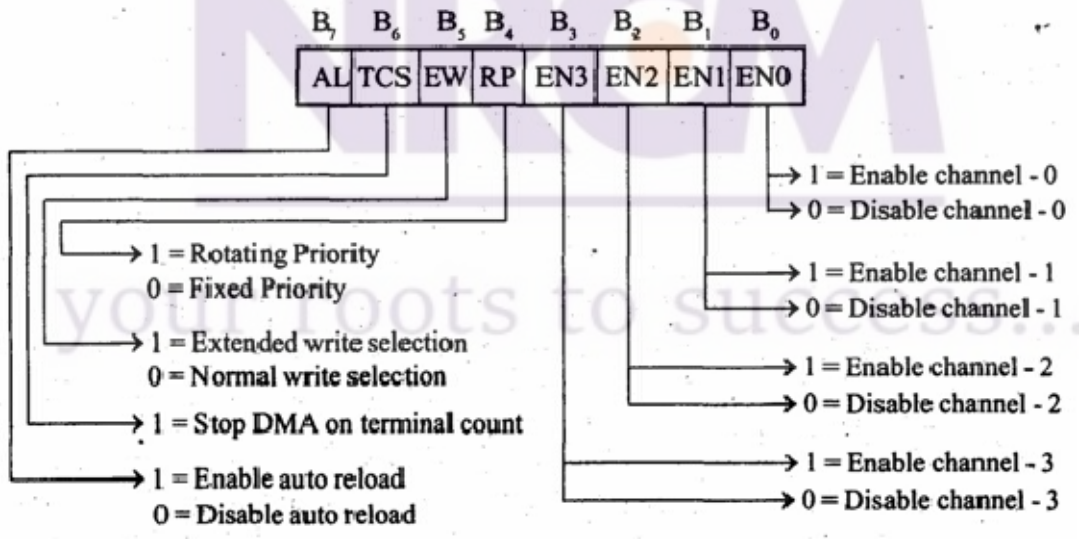


Fig 4.7. 8257 mode set register

- The use of mode set register is,

    1. Enable/disable a channel.
    2. Fixed/rotating priority
    3. Stop DMA on terminal count.
    4.Extended/normal write time.
    5. Auto reloading of channel-2.

- The bits B0, B1, B2, and B3 of mode set register are used to enable/disable channel -0, 1, 2 and 3 respectively. A one in these bit position will enable a particular channel and a zero will disable it.

- If the bit B4 is set to one, then the channels will have rotating priority and if it zero then the channels wilt have fixed priority.

    1. In rotating priority after servicing a channel its priority is made as lowest.
    2. In fixed priority the channel-0 has highest priority and channel-2 has lowest priority.

- If the bit B5 is set to one, then the timing of low write signals (MEMW and IOW) will be extended.

- If the bit B6 is set to one then the DMA operation is stopped at the terminal count.

- The bit B7 is used to select the auto load feature for DMA channel-2.

- When bit B7 is set to one, then the content of channel-3 count and address registers are loaded in channel-2 count and address registers respectively whenever the channel-2 reaches terminal count. When this mode is activated the number of channels available for DMA reduces from four to three.

- The format of status register of 8257 is shown in fig.

your roots to success...

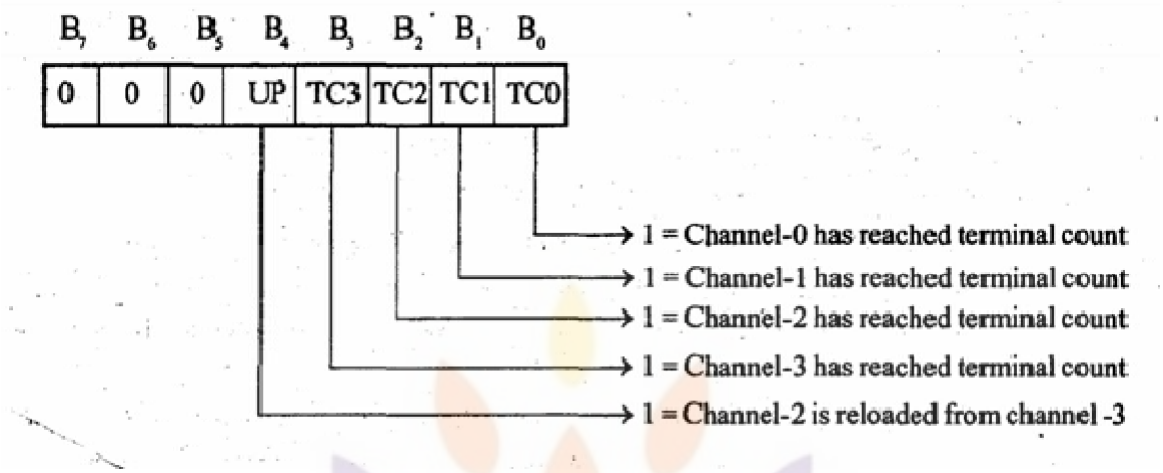


Fig 4.10. 8257 status register

- The bit B0, B1, B2, and B3 of status register indicates the terminal count status of channel-0, 1,2 and 3 respectively. A one in these bit positions indicates that the particular channel has reached terminal count.

- These status bits are cleared after a read operation by microprocessor.

- The bit B4 of status register is called update flag and a one in this bit position indicates that the channel-2 register has been reloaded from channel-3 registers in the auto load mode of operation.

- The internal addresses of the registers of 8257 are listed in table.

NRCM
your roots to success...

Register	Binary Address								Hexa Address
	Decoder input and enable				Input to address pins of 8257				
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Channel-0 DMA address register	0	1	1	0	0	0	0	0	60
Channel-0 count register	0	1	1	0	0	0	0	1	61
Channel-1 DMA address register	0	1	1	0	0	0	1	0	62
Channel-1 count register	0	1	1	0	0	0	1	1	63
Channel-2 DMA address register	0	1	1	0	0	1	0	0	64
Channel-2 count register	0	1	1	0	0	1	0	1	65
Channel-3 DMA address register	0	1	1	0	0	1	1	0	66
Channel-3 count register	0	1	1	0	0	1	1	1	67
Mode set register (Write only)	0	1	1	0	1	0	0	0	68
Status register (Read only)	0	1	1	0	1	0	0	0	68

Table 4.1. Register addresses of 8257

PIN DIAGRAM

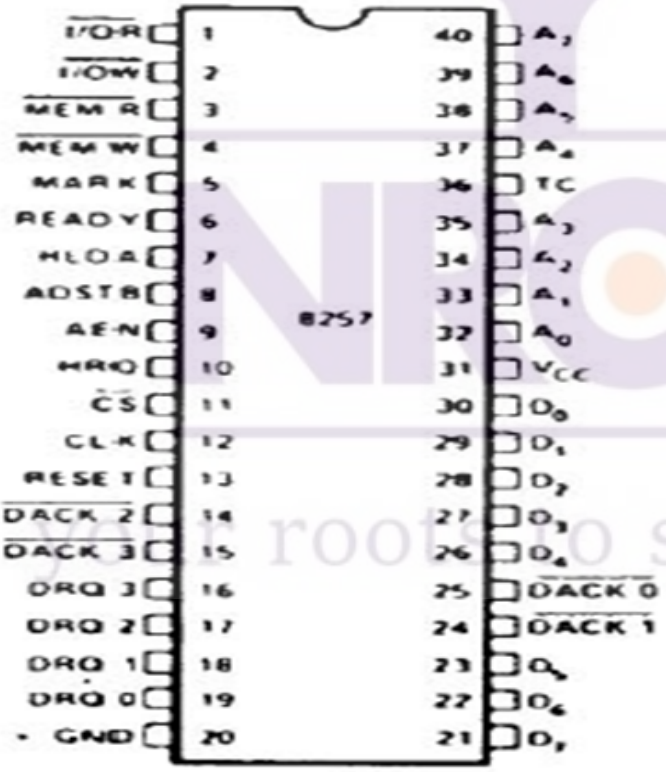


Fig 4.11.8257 pin diagram

- D0-D7: it is a bidirectional, tri state, Buffered, Multiplexed data (D0-D7)and (A8-A15).In the slave mode it is a bidirectional (Data is moving).In the Master mode it is a unidirectional (Address is moving).
- IOR: It is active low, tristate ,buffered ,Bidirectional lines.In the slave mode it functions as a input line. IOR signal is generated by microprocessor to read the contents 8257 registers. In the master mode it functions as a output line. IOR signal is generated by 8257 during write cycle
- IOW:It is active low ,tristate ,buffered ,Bidirectional control lines.In the slave mode it function as a input line. IOR signal is generated by microprocessor to write the contents 8257 registers.In the master mode it function as a output line. IOR signal is generated by 8257 during read cycle
- CLK:It is the input line ,connected with TTL clock generator.This signal is ignored in slave mode.
- RESET:Used to clear mode set registers and status registers
- A0-A3:These are the tristate, buffer, bidirectional address lines.In slave mode ,these lines are used as address inputs lines and internally decoded to access the internal registers.In master mode, these lines are used as address outputs lines,A0-A3 bits of memory address on the lines.
- CS:It is active low, Chip select input line.In the slave mode, it is used to select the chip.
- In the master mode, it is ignored.
- A4-A7:
- These are the tristate, buffer, output address lines.In slave mode ,these lines are used as address outputs lines.In master mode, these lines are used as address outputs lines,A0-A3 bits of memory address on the lines.
- READY:It is an asynchronous input line. In master mode,When ready is high it is received the signal.When ready is low, it adds wait state between S1 and S3In slave mode, this signal is ignored.
- HRQ:It is used to receiving the hold request signal from the output device
- HLDA:It is acknowledgment signal from microprocessor.
- MEMR:It is active low ,tristate ,Buffered control output line.In slave mode, it is tristated.In master mode ,it activated during DMA read cycle.
- MEMW:It is active low ,tristate ,Buffered control input line.In slave mode, it is tristated.In master mode ,it activated during DMA write cycle.
- AEN (Address enable):It is a control output line.In master mode ,it is highIn slave mode ,it is lowUsed it isolate the system address ,data ,and control lines.
- ADSTB: (Address Strobe)It is a control output line.Used to split data and address line.It is working in master mode only.In slave mode it is ignore.
- TC (Terminal Count):It is a status of output line.It is activated in master mode only.It is high ,it selected the peripheral.It is low ,it free and looking for a new peripheral.

- MARK:It is a modulo 128 MARK output line.It is activated in master mode only.It goes high, after transferring every 128 bytes of data block.
- DRQ0-DRQ3(DMA Request):These are the asynchronous peripheral request input signal.The request signal is generated by external peripheral device.
- DACK0-DACK3: These are the active low DMA acknowledge output lines.Low level indicate that, peripheral is selected for giving the information (DMA cycle).In master mode it is used for chip select.

Example:

Interface DMA with 8086 so that the channel 0 DMA addresses reg.,TC reg. and MSR has an I/O address 80 H, 81 H and 88 H . Initialize the 8257 with normal priority, TC stop and non-extended write. Auto load is not required. Write an ALP to move 2KB of data from peripheral device to memory address 2000 H: 5000 H, with the above initialization. The transfer has to take place using channel 0.

MSR = 41 H

DMA address register = 5000 H

TC = 47FF H

```
ADDREG EQU 80 H

TC EQU 81 H

MSR EQU 88 H

Code segment

Asssume cs:code

Start:  MOV AX, 2000 H

        MOV     DS,AX

        MOV AX,5000 H

        OUT ADDREG, AL

        OUT ADDREG, AH

        MOV AX,47FF H
```

NRCM your roots to success...

```
        OUT TC, AL

        OUT TC, AH

        MOV AL,41 H

        OUT MSR, AL

        HLT
```

Code ends

End start

your roots to success...

3.8.SERIAL COMMUNICATION INTERFACE

3.8.1 Serial data transfer schemes

Data transfer schemes in Microprocessor: - Data can be transferred between memory, microprocessor and input output devices. the speed and format of all the input output devices does not matches microprocessor. For example some input output devices like ABC' and DAC's are slow as compared to microprocessor. Some devices are serial in nature while microprocessor is parallel in nature. Because of this, number of data transfer schemes have been divided to cope with this problem. The data transfer schemes can be broadly classified into two categories :-
Programmed Data Transfer

Direct Memory Access Data Transfer

Programmed Data Transfer:-In this scheme, data transfer takes place under the control of a program residing in the main memory of the microcomputer system. So microprocessor executes a program to perform all data transfers between the memory and i/o device via register. This data transfer takes place under the control microprocessor. As the transfer of data takes place through a register, generally accumulator and requires execution of several instructions, so programmed data transfer is slow and suitable for small data. It can be classified in four types:-

Synchronous Data Transfer:- In this scheme, timing characteristics of I/O device are precisely known. Speed og I/O devices matches microprocessor always consider the I/O device to ready for data transfer.

Synchronous Data Transfer With Delay: -In this scheme, of I/O device is slow as compare to microprocessor but timing characteristics are precisely known. Microprocessor imitates I/o device to get ready and then waits for some predetermined than executes I/O instruction to complete the data transfer.

Asynchronous Data transfer: -In this scheme data transfer between external device and microprocessor occurs via hand shaking process there is some exchange of signals between I/O and microprocessor before the actual data transfer takes place.

Interrupt Driven Data Transfer:-In this scheme, microprocessor initiates data transfer by requesting the device' to get ready' and then goes executing its main problem instead of wasting its time by continuously checking the statues of input output device. Whenever device is ready to accept or supply data, it informs microprocessor through a special interrupt line.

Direct Memory Access: - In this scheme data is transferred between memory and I/O device without any involvement of microprocessor. Microprocessor is sidelined in this process by tri stating its address bus, data bus and control bus. A direct link is establishment between memory

and I/O device. Data transfer takes place under the control of an external circuit DMA controller. This technique is used to transfer blocks of data between memory and I/O devices

3.8.2. Serial communication standards

5.2.1. RS 232

DCE and DTE Devices

DTE stands for Data Terminal Equipment, and DCE stands for Data CommunicationsEquipment. These terms are used to indicate the pin-out for the connectors on a device and the direction of the signals on the pins. Your computer is a DTE device, while most other devicessuch as modem and other serial devices are usually DCE devices.

RS-232 has been around as a standard for decades as an electrical interface betweenData Terminal Equipment (DTE) and Data Circuit-Terminating Equipment (DCE) such as modems or DSUs. It appears under different incarnations such as RS-232C, RS-232D, V.24,

V.28 or V.10. RS-232 is used for asynchronous data transfer as well as synchronous links such as SDLC, HDLC, Frame Relay and X.25

Synchronous data transfer

In program-to-program communication, synchronous communication requires that each end of an exchange of communication respond in turn without initiating a new communication. A typical activity that might use a synchronous protocol would be a transmission of files from one point to another. As each transmission is received, a response is returned indicating success or the need to resend.

Asynchronous data transfer

The term asynchronous is usually used to describe communications in which data can be transmitted intermittently rather than in a steady stream. For example, a telephone conversation is asynchronous because both parties can talk whenever they like. If the communication were synchronous, each party would be required to wait a specified interval before speaking. The difficulty with asynchronous communications is that the receiver must have a way to distinguish between valid data and noise. In computer communications, this is usually accomplished through a special start bit and stop bit at the beginning and end of each piece of data. For this reason, asynchronous communication is sometimes called start-stop transmission.

RS232

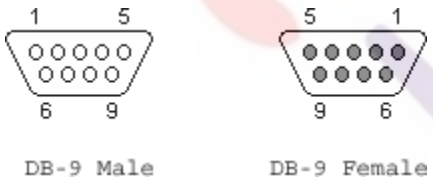
RS-232 (Recommended standard-232) is a standard interface approved by the Electronic Industries Association (EIA) for connecting serial devices. In other words, RS-232 is a longestablished standard that describes the physical interface and protocol for relatively low-speed serial data communication between computers and related devices.

An industry trade group, the Electronic Industries Association (EIA), defined it originally for teletypewriter devices. In 1987, the EIA released a new version of the standard and changed the name to EIA-232-D. Many people, however, still refer to the standard as RS-232C, or just RS-232.

RS-232 is the interface that your computer uses to talk to and exchange data with your modem and other serial devices. The serial ports on most computers use a subset of the RS-232C standard.

RS232 on DB9 (9-pin D-type connector)

There is a standardized pin out for RS-232 on a DB9 connector, as shown below



Pin	SIG.	Signal Name	DTE (PC)
1	DCD	Data Carrier Detect	in
2	RXD	Receive Data	in
3	TXD	Transmit Data	out
4	DTR	Data Terminal Ready	out
5	GND	Signal Ground	-
6	DSR	Data Set Ready	in
7	RTS	Request to Send	out
8	CTS	Clear to Send	in
9	RI	Ring Indicator	in

RS232 on DB25 (25-pin D-type connector)

In DB-25 connector most of the pins are not needed for normal PC communications, and indeed, most new PCs are equipped with male D type connectors having only 9 pins. Using a 25-pin

DB-25 or 9-pin DB-9 connector, its normal cable limitation of 50 feet can be extended to several hundred feet with high-quality cable. RS-232 defines the purpose and signal timing for each of the 25 lines; however, many applications use less than a dozen. There is a standardized

pinout for RS-232 on a DB25 connector, as shown below

1	GND		Shield Ground
2	TXD	—»	Transmit Data
3	RXD	«—	Receive Data
4	RTS	—»	Request to Send
5	CTS	«—	Clear to Send
6	DSR	«—	Data Set Ready
7	GND		System Ground
8	CD	«—	Carrier Detect
9	-	-	RESERVED
10	-	-	RESERVED
11	STF	—»	Select Transmit Channel
12	S.CD	«—	Secondary Carrier Detect
13	S.CTS	«—	Secondary Clear to Send
14	S.TXD	—»	Secondary Transmit Data
15	TCK	«—	Transmission Signal Element Timing
16	S.RXD	«—	Secondary Receive Data
17	RCK	«—	Receiver Signal Element Timing

NRCM
your roots to success...

18	LL	—»	Local Loop Control
19	S.RTS	—»	Secondary Request to Send
20	DTR	—»	Data Terminal Ready
21	RL	—»	Remote Loop Control
22	RI	«—	Ring Indicator
23	DSR	—»	Data Signal Rate Selector
24	XCK	—»	Transmit Signal Element Timing
25	TI	«—	Test Indicator

Signal Description

TxD: - This pin carries data from the computer to the serial device

RXD: - This pin carries data from the serial device to the computer

DTR signals: - DTR is used by the computer to signal that it is ready to communicate with the serial device like modem. In other words, DTR indicates to the Dataset (i.e., the modem or DSU/CSU) that the DTE (computer) is ON.

DSR: - Similarly to DTR, Data set ready (DSR) is an indication from the Dataset that it isON.

DCD: - Data Carrier Detect (DCD) indicates that carrier for the transmit data is ON.

RTS: - This pin is used to request clearance to send data to a modem

CTS: - This pin is used by the serial device to acknowledge the computer's RTS Signal. In most situations, RTS and CTS are constantly on throughout the communication session.

Clock signals (TC, RC, and XTC): - The clock signals are only used for synchronous communications. The modem or DSU extracts the clock from the data stream and provides a steady clock signal to the DTE. Note that the transmit and receive clock signals do not have to be the same, or even at the same baud rate.

NRCM your roots to success...

CD: - CD stands for Carrier Detect. Carrier Detect is used by a modem to signal that it has a made a connection with another modem, or has detected a carrier tone. In other words, this is used by the modem to signal that a carrier signal has been received from a remote modem.

RI: - RI stands for Ring Indicator. A modem toggles(keystroke) the state of this line when an incoming call rings your phone. In other words, this is used by an auto answer modem to signal the receipt of a telephone ring signal The Carrier Detect (CD) and the Ring Indicator (RI) lines are only available in connections to a modem. Because most modems transmit status information to a PC when either a carrier signal is detected (i.e. when a connection is made to another modem) or when the line is ringing, these two lines are rarely used.

Limitations of RS-232

RS-232 has some serious shortcomings as an electrical interface.

Firstly, the interface presupposes a common ground between the DTE and DCE. This is a reasonable assumption where a short cable connects a DTE and DCE in the same room, but with longer lines and connections between devices that may be on different electrical busses, this may not be true. We have seen some spectacular electrical events causes by "uncommon grounds".

Secondly, a signal on a single line is impossible to screen effectively for noise. By screening the entire cable one can reduce the influence of outside noise, but internally generated noise remains a problem. As the baud rate and line length increase, the effect of capacitance between the cables introduces serious crosstalk until a point is reached where the data itself is unreadable. Using low capacitance cable can reduce crosstalk. Also, as it is the higher frequencies that are the problem, control of slew rate in the signal (i.e., making the signal more rounded, rather than square) also decreases the crosstalk. The original specifications for RS-232 had no specification for maximum slew rate.

Voltage levels with respect to ground represent the RS 232 signals. There is a wire for each signal, together with the ground signal (reference for voltage levels). This interface is useful for point-to-point communication at slow speeds. For example, port COM1 in a PC can be used for a mouse, port COM2 for a modem, etc. This is an example of point-to-point communication: one port, one device. Due to the way the signals are connected, a common ground is required. This implies limited cable length - about 30 to 60 meters maximum. (Main problems are interference and resistance of the cable.) Shortly, RS 232 was designed for communication of local devices, and supports one transmitter and one receiver.

5.2.2. IEEE- 488

Hewlett-Packard originally developed the interfacing technique for computer controlled measurement systems in 1960’s. It was called HP-IB(Hewlett-Packard interface bus).HP-IB quickly became very popular, thus IEEE (Institute of Electrical andElectronics Engineers) made

a standard of it and renamed it GP-IB (General Purpose Interface Bus).IEEE-488 was first established in 1978. Later on, during 1980, a new layer was added to IEEE-488 standard and the old standard was renamed IEEE-488.1 and the new one IEEE-488.2. In modern professional measurementdevices equipped with the IEEE-488.2 interface.

IEEE-488.2 provides minimum set requirements for controller and devicecapabilities (talker, listener, controller). Also, data coding and format, message and communication protocol structures between controller and deviceare defined more specificly compared to IEEE-488.1.

Properties of IEEE-488

• 1 Mbyte/sec maximum data transfer rate

• up to 15 devices parallel can be connected to one bus

• total bus length 20 m max. (distance between devices up to 2 m max.)

• messages are sent one byte (8 bits) at time

• message transactions are hardware handshake

GPIB operation

The operation of GPIB is based around the handshaking protocol. Three lines, DAV (DAta Valid), NDAC (Not Data ACepted), and NRFD (Not Ready For Data), control this. All the listeners on the bus use the NRFD line to indicate their state of readiness to accept data. If one listener holds the line low then this prevents any data transfer being initiated. This means that when all the instruments are ready as indicated by the NRFD being line is high and then data can be transferred. Once all the instruments have released the NRFD line and it is in the high state, only then can the next stage be initiated.

Data is placed onto the data lines by the talker and once this has settled, the DAV line is pulled low. This signals to all the listeners that they are able to read the data that is present. During this operation the NDAC line will be held low by all the active listeners, i.e. those which have been instructed to receive the data. Only when they have read the data will each device stop trying to hold this line low. When the last device removes its hold, the level of the line will rise and the talker will know that all the data has been accepted and the next byte of data can be transferred.
By transferring data in this way the data is placed onto the bus at a rate which is suitable for the talker, and it is held until the slowest listener has accepted it. In this way the optimum data transfer rate is always used, and there are no specifications and interface problems associated with the speeds at which data must be transferred.

GPIB Polling

There are two ways in which instruments on GPIB can be polled. One is called parallel polling and the other is serial.

Parallel polling can only operate with up to eight instruments. This is because each of the devices will return a status bit one of the eight data lines. To assert a parallel poll the controller pulls the ATN and EOI lines low. When this occurs each instrument responds by transmitting a one-bit status report.A serial poll is more flexible but takes longer to accomplish. Here the controller sends each of the instruments a serial poll enable command in turn. This is one of the GPIB commands that can be sent when the ATN line is held low. When an instrument receives a serial poll enable it responds by returning eight bits of status information. When the controller has received the status data it sends a serial poll disable command and returns the bus and instruments on it to the normal data mode.

The advantage of a serial poll is that it is far more flexible and enables eight bits of data to be returned. However it is much slower because each instrument has to be polled in turn to find out which one pulled the SRQ line in the first place. In practice the GPIB interface is very easy to use. Ready-made GPIB cables are widely available even if they appear to be a little expensive. However these GPIB cables are fully screened and have the correct lines as twisted pairs. This considerably reduces the susceptibility of the bus to data corruptions. Manufacture of full specification GPIB cables can be difficult in view of the complexity of the cable and having to ensure the integrity of the screening.

A cheap and convenient alternative for GPIB cables is available in the form of insulation displacement connectors. While cables made in this way are much cheaper they are not screened and do not conform to the GPIB / IEEE 488 specifications. In view of this they should only be used where there are a very limited number of instruments, where data rates are likely to be low, for long runs, and where electrical noise is not likely to be a problem. If a cable of this nature is used then it is worth being aware that it could be the cause of random errors when the system is operating.

When setting up a GPIB system linked by the bus few rules need to be observed. The cables can generally be routed as required, linking the instruments as is most convenient. As the connectors can be "piggy backed", this makes linking the instruments very easy. However a little common sense is required, and not too many connectors should be linked to one point.

Before firing up the system, check all the instrument GPIB addresses to ensure that they are correct and match what the software in the controller requires. Also check that none are duplicated. Unfortunately checking the addresses can be a time consuming operation because not all the instruments will have switches that are easily available. To overcome this it is best to have standard addresses for different types of instruments within a factory. This will eliminate the need for any swapping and changing as test stacks are taken down and erected.

Although GPIB normally works very well, occasionally some problems inevitably arise. Sometimes it has been known for the bus to hang, even though all the instruments are operating correctly on their own. Some instruments can be sensitive to their physical position on the bus, particularly if they are at a remote end. In instances like this the topology of the cable routing can be changed to bring the offending instrument closer to the controller

The GPIB / IEEE 488 interface is well established in the electronics industry as a means of providing control of remote test and measurement instruments. Although the GPIB interface has been in widespread use since the early 1970s, the use of the GPIB interface continues in view of its convenience and availability The GPIB cables and GPIB connectors are in widespread use and are available from many stockiest and suppliers.
While GPIB connectors and GPIB cables are widely available and can be used with little knowledge, some background information can be useful.

GPIB connector

The connector used for the IEEE 488 bus is standardized as a 24-way Amphenol 57 series type. This provides an ideal physical interface for the standard. The IEEE 488 or GPIB connector is very similar in format to those that were used for parallel printer ports on PCs. This makes the GPIB connector a sufficiently rugged connector for use in a variety of test equipment and test and measurement environments where unprotected connectors may not survive well.
An additional advantage of the GPIB connector is that it has a screw-lock. In this way, once the connectors have been mated the screw-lock can be used to secure the connectors together. In this way movements of the GPIB cables are unlikely to cause any intermittent connections. With complicated electronics systems such as automatic test systems, intermittent cable connections can cause significant problems which can be difficult to isolate and cure. By using a GPIB connector that has a screw-lock this problem can be overcome.
The basic female connector used on the GPIB equipments follows the standard format for the Amphenol 57 series, the GPIB cable connector has some differences. The basic GPIB cable connector has a male to female capability. In this several GPIB cables connectors can be "piggy-backed" on top of each other. This helps the physical setting up of the bus and prevents complications with special connection boxes or star points, etc.
There are two different types of screw lock used on the GPIB connectors. The metric threads are Black whereas English threads are Silver. Unfortunately the two will not mate together. The most common type of screw lock is the black metric version.

GPIB cables

In practice the GPIB interface is very easy to use. Ready-made GPIB cables are widely available even if they appear to be a little expensive. However these GPIB cables are fully screened and have the correct lines as twisted pairs. This considerably reduces the susceptibility of the bus to data corruptions. Manufacture of full specification GPIB cables can be difficult in view of the complexity of the cable and having to ensure the integrity of the screening.
A cheap and convenient alternative for GPIB cables is available in the form of insulation displacement connectors. While cables made in this way are much cheaper they are not screened and do not conform to the GPIB / IEEE 488 specifications. In view of this they should only be used where there are a very limited number of instruments, where data rates are likely to be low, for long runs, and where electrical noise is not likely to be a problem. If a cable of this nature is

used then it is worth being aware that it could be the cause of random errors when the system is operating.

Although there is a considerable degree of flexibility when setting up a GPIB system, there are some restrictions on the way the GPIB cables are set up. Up to fifteen instruments may be connected together with a maximum bus length not exceeding 20 meters. There must also be no more than 2 meters between any two instruments. Devices on the GPIB can be connected in either a star or linear configuration.

Although a variety of GPIB cable products are available, typically they are available in set lengths. The most common lengths for GPIB cables are: 1 meter and 2 meters. Longer GPIB cables are not normally available in view of the GPIB overall bus restrictions. Some 0.5 meter GPIB cables are available, but these are often too short for many applications in view of the lack of cable flexibility.

Although GPIB cables are normally quite reliable, their construction means that they are not easy to flex. As a result, GPIB cables seem to have a limited life and as a result, care should be taken when re-using old GPIB cables.

GPIB pin-outs

The connections or pinouts for the GPIB connector are given in the table below:

GPIB PIN NO	GPIB LINE NAME	
1	Data Input / Output 1	DIO1
2	Data Input / Output 2	DIO2
3	Data Input / Output 3	DIO3
4	Data Input / Output 4	DIO4
5	End or Identify	EOI
6	Data Valid	DAV
7	Not Ready For Data	NRFD
8	Not Data Accepted	NDAC
9	Interface Clear	IFC
10	Service Request	SRQ
11	Attention	ATN
12	Shield (Connected to Earth)	
13	Data Input / Output 5	DIO5
14	Data Input / Output 6	DIO6
15	Data Input / Output 7	DIO7

NRCM your roots to success...

GPIB PIN NO	GPIB LINE NAME	
16	Data Input / Output 8	DIO8
17	Remote Enable	REN
18	Twisted pair with pin 6	
19	Twisted pair with pin 7	
20	Twisted pair with pin 8	
21	Twisted pair with pin 9	
22	Twisted pair with pin 10	
23	Twisted pair with pin 11	
24	Signal Ground	

GPIB is widely used for test instrumentation and data acquisition. Despite the fact that it has been in use for many years, the GPIB standard offers flexibility and convenience, allowing stand-alone bench test equipment to be sued in an automated fashion. Accordingly many GPIB cables and GPIB connectors are required and these are also in plentiful supply. It is also useful to have a table of the GPIB pinouts for occasions when individual connections may be required

Test and measurement is a key area of electronic engineering.
Test technology is used throughout the life of electronic products from initial development, through verification of the product and manufacture, to maintenance and repair during its service life.
According test techniques, test equipment and test technology are at the very heart of any electronic product from mobile phone, consumer electronics device through to industrial products and systems.
To meet the needs of the industry, there is an enormous variety of test technologies, test equipment and test techniques that can be used.

5.3. 8251 USART

The Intel 8251 Universal Synchronous/Asynchronous Receiver/Transmitter (USART), designed for data communication with Intel's microprocessor families. It is used as a peripheral device and is programmed by the CPU to operate using many serial data transmission techniques. The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream. It accepts serial data streams and converts them into parallel data characters for the CPU. The USART will signal the CPU whenever it can accept a new character for transmission or whenever it has received a character for the CPU. The CPU can read the

status of the USART at any time. The status includes data transmission errors and control signals SYNDET/BD, TxEMPTY, TxRDY, RxRDY

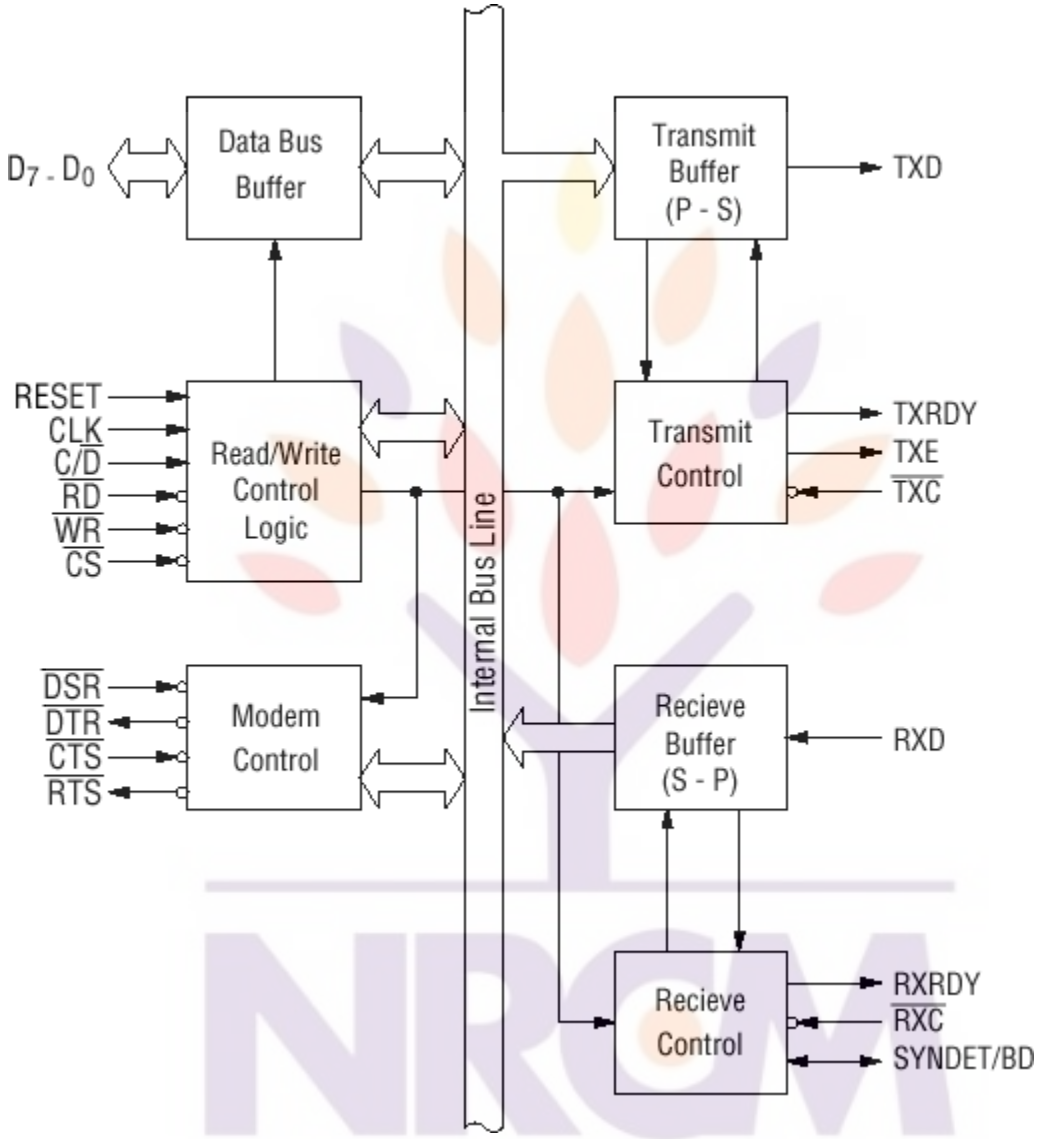


Fig 5.1. 8251 USART Architecture

Ports

RESET (1 BIT, INPUT PORT) : This is the master reset for the 8251 chip.

D_7 to D_0 (8 PINS, 1 BIT each, INOUT PORTS) : These are the bi-directional data bus pins (8 bits) used for transferringdata/control/status words transfer the USART and the CPU. These are usually connected to the CPU's data-bus, although the CPU always remains in control of the bus and initiates all transfers.

CS_BAR (1 BIT, INPUT PORT) : This is the Chip-Select line. A low on this line enables data communication between the CPU and the USART.

RD_BAR (1 BIT, INPUT PORT) : This is the read line. A low on this line causes the USART to place the status word or the (received) data word on the data bus pins("D_7" to "D_0").

WR_BAR (1 BIT, INPUT PORT) : This is the write line. A low on this line causes the USART to accept the data on the data bus pins ("D_7" to "D_0") as either a control word or as a data character (for transmission).

C_D_BAR (1 BIT, INPUT PORT) : This is the "Control/Data" pin. It is used while transferring data to/from the CPU using the data bus pins ("D_7" to "D_0"). During a read operation : If C_D_BAR - 1, the USART places its status on the data bus pins. If C_D_BAR - 0, the USART places the (received) data character on the data bus pins. During a write operation : If C_D_BAR - 1, the USART reads a control word from the data bus pins. If C_D_BAR - 0, the USART reads a data character (for transmission) from the data bus pins.

RxD (1 BIT, INPUT PORT) : This is the receiver data pin. Characters are received serially on this pin and assembled into parallel characters.

TxD (1 BIT, OUTPUT PORT) : This is the transmitter data pin. Parallel characters received by the CPU are transmitted serially by the USART on this line.

RxC_BAR (1 BIT, INPUT PORT) : This is the receiver clock. Data on "RxD" is sampled by the USART on the rising edge of "RxC_BAR".

TxC_BAR (1 BIT, INPUT PORT) : This is the transmitter clock. Data is shifted out serially on "TxD" by the USART, on the falling edge of "TxC_BAR".

CLK (1 BIT, INPUT PORT) : This clock is used for internal device timing. It needs to be faster than "TxC_BAR" and "RxC_BAR".

TxEMPTY (1 BIT, OUTPUT PORT) : A high on this line indicates that the serial buffer in the transmitter is empty. This line goes low only while a data character is being transmitted by the USART. It goes high as soon as the USART completes transmitting a character and a new one has not been loaded in time.

TxRDY (1 BIT, OUTPUT PORT) : This pin signals the CPU that the USART is ready to accept a new data character for transmission. "TxRDY" is reset when the USART receives a data character from the CPU.

RxRDY (1 BIT, OUTPUT PORT) : This pin signals the CPU that the USART has received a character on its serial input "RxD" and is ready to transfer it to the CPU. "RxRDY" is reset when the character is read by the CPU.

SYNDET_BD (1 BIT, INOUT PORT) : In the Synchronous mode, this line can be in two ways (while receiving characters). In the "Internal-Synchronization" mode, this line is used as an output which goes high when the programmed "SYNC-characters" are detected on the "RxD" line. In the "ExternalSynchronization" mode, this line is used as an input and the USART starts assembling data characters at the next clock ("RxC_BAR") edge after a rising edge on this line. In the Asynchronous mode, this line is used as a "Break-Detect" output which goes high if the "RxD" line has stayed low for two consecutive character lengths (including start, stop and parity bits).

RTS_BAR (1 BIT, OUTPUT PORT) : This "Request-To_Send" is a general purpose output signal that can be asserted by a "command word" from the CPU. It may be used to request that the modem prepare itself to transmit.

CTS_BAR (1 BIT, INPUT PORT) : This "Clear-To-Send" is an input signal that can be read by the CPU as part of the "status-word". A low on this line enables the USART to transmit data. A low on "CTS_BAR" is normally generated as a response to an assertion on "RTS_BAR".

DTR_BAR (1 BIT, OUTPUT PORT) : This "Data-Terminal-Ready" is a general purpose output signal that can be asserted by a "command word" from the CPU.

DSR_BAR (1 BIT, INPUT PORT) : This "Data-Set-Ready" is a general purpose input signal that can be read by the CPU as part of the "status-word".

General Operation

Programming the 8251

The complete functional definition of the 8251 is programmed by the system's software. A set of control words must be sent out by the CPU to initialize the 8251 to support the desired communication format. These words must immediately follow a reset (internal/external).

The Mode word

Immediately after a reset, the CPU has to send the 8-bit "mode" word. The 8251 can be used for either synchronous/asynchronous data communication. To understand how the mode instruction works, its best to view the device as two separate components, one synchronous and the other asynchronous.

your roots to success...

Synchronous mode word

Bit 0	Bit 1
0	0

The two least significant bits must be both 0 in Synchronous mode.

Character length : (bits per character)

Bit 3	Bit 2	
0	0	5 bits
0	1	6 bits
1	0	7 bits
1	1	8 bits

Parity :

Bit 5	Bit 4	
0	0	No parity
0	1	Odd parity

| Bit 4 -- Parity Enable

NRCM your roots to success...

| | | | Bit 5 -- Even Parity

| | | |

| 1 | 0 | No parity |

| | | |

| | | |

| 1 | 1 | Even parity |

Synchronization scheme :

| Bit 7 | Bit 6 | |

| | | |

| 0 | 0 | Internal sync detect, Double Sync character |

| | | |

| 0 | 1 | External sync detect (from SYNDET_BD input) |

| | | |

| 1 | 0 | Internal sync detect, Single Sync character |

| | | |

| 1 | 1 | External sync detect (from SYNDET_BD input) |

Bit 6 -- External Synchronization

Bit 7 -- Single Sync character (Internal Synchronization)

your roots to success...

Asynchronous mode word

Baud Rate In asynchronous mode, the baud rate defines the number of clock (RxC_BAR/TxC_BAR) cycles over which each bit is transmitted/received. (e.g. At baud rate 64X, each bit is transmitted over 64 clock cycles).

Bit 1	Bit 0	
0	0	Not relevant (Synchronous mode)
0	1	1X baud rate
1	0	16X baud rate
1	1	64X baud rate

Character length : (bits per character)

Bit 3	Bit 2	
0	0	5 bits
0	1	6 bits

NRCM your roots to success...

1	0	7 bits	
1	1	8 bits	

Parity :

Bit 5	Bit 4		
0	0	No parity	Bit 4 -- Parity Enable
			Bit 5 -- Even Parity
0	1	Odd parity	
1	0	No parity	
1	1	Even parity	

No of Stop Bits :

Bit 7	Bit 6	

NRCM your roots to success...

0	0	Invalid	
0	1	1 stop bit	
1	0	1.5 stop bits	
1	1	2 stop bits	

The Command word and SYNC characters

In the "Internal Synchronization" mode, the control words (from the CPU) that follow the modeword, must be SYNC characters. In Single-Sync mode, only one SYNC character (SYNC1) is loaded. In Double-Sync mode, two consecutive SYNC characters (SYNC1 followed by SYNC2) must be loaded. The SYNC character(s) have the same number of bits as the data characters (as programmed in the mode word).The SYNC characters (if present, i.e. in "Internal Synchronization" mode) are followed by the command word from the CPU. Data words (for transmission) can follow that.Actually, the command word can be written by the CPU at any time in the data block during the operation of the USART. To write a new Mode word, the master reset in the Command instruction can be set to initiate an "Internal Reset".

Command Word

Bit 0 : Transmitter Enable

Bit 1 : DTR (Data Terminal Ready) -- Controls DTR_BAR output(if this is high, DTR_BAR is low)

Bit 2 : Receiver enable

your roots to success...

Bit 3 : Send Break -- Assertion of this forces "TxD" pin low

Bit 4 : Error Reset -- Reset all error flags (parity error, framing error overrun error) in the status word .

Bit 5 : RTS (Request To Send) -- Controls RTS_BAR output (if this is high, RTS_BAR is low)

Bit 6 : Internal Reset -- Resets the USART and makes it ready to accept a new mode word.

Bit 7 : Enter Hunt Mode -- (used only in synchronous receive). If this is high, the USART tries to achieve synchronization by entering the "hunt mode". In "Internal Synchronization" mode, the USART starts looking for the programmed SYNC character(s) at the "RxD" input. In "External Synchronization" mode, the USART starts looking for a rising edge on the "SYNDET_BD" input. Once synchronization is achieved, the USART gets out of "hunt mode" and starts assembling characters at the next rising edge of "RxC_BAR".

The Status Word

The CPU can read the "status word" from the USART at any time.

Bit 0 : TxRDY -- This signifies whether the transmitter is ready to receive a new character for transmission from the CPU. However, in order for the "TxRDY" PIN to be high, three conditions must be satisfied :

(a) TxRDY STATUS BIT must be high

(b) "CTS_BAR" must be low

(c) The transmitter must be enabled (Bit 0 in the Command word must be high).

Bit 1 : RxRDY -- Same as "RxRDY" pin.

Bit 2 : TxEMPTY -- Same as "TxEMPTY" pin.

Bit 3 : Parity Error -- When parity is enabled and a parity error is detected in any received character, this bit is set.

Bit 4 : Overrun Error -- When the CPU does not read a received character before the next one becomes available, this bit is set. However, the previous character is lost.

Bit 5 : Framing Error -- Used only in asynchronous mode. When a valid stop bit (high) is not detected at the end of a received character, this bit is set.

(Note : All three error flags are reset by the "Error Reset" command bit. Also, the setting of these error flags does not inhibit the USART operation.)

Bit 6 : SYNDET_BD -- Same as "SYNDET_BD" pin.

NRCM your roots to success...

Bit 7 : DSR (Data Set Ready) -- Controlled by "DSR_BAR" pin. (If "DSR_BAR" pin is low, his status bit is high.)

your roots to success...

UNIT-4

INTRODUCTION TO MICROCONTROLLERS

4.1. Overview of 8051 microcontroller

Microcontroller manufacturers have been competing for a long time for attracting choosy customers and every couple of days a new chip with a higher operating frequency, more memory and upgraded A/D converters appeared on the market.

However, most of them had the same or at least very similar architecture known in the world of microcontrollers as "8051 compatible". What is all this about?

The whole story has its beginnings in the far 80s when Intel launched the first series of microcontrollers called the MCS 051. Even though these microcontrollers had quite modest features in comparison to the new ones, they conquered the world very soon and became a standard for what nowadays is called the microcontroller.

The main reason for their great success and popularity is a skillfully chosen configuration which satisfies different needs of a large number of users allowing at the same time constant expansions (refers to the new types of microcontrollers). Besides, the software has been developed in great extend in the meantime, and it simply was not profitable to change anything in the microcontroller's basic core. This is the reason for having a great number of various microcontrollers which basically are solely upgraded versions of the 8051 family.

4.1.1 Architecture of 8051

Features:

The main features of 8051 microcontroller are:

i. RAM – 128 Bytes (Data memory)
ii. ROM – 4Kbytes (ROM signify the on – chip program space)
iii. Serial Port – Using UART makes it simpler to interface for serial communication.
iv. Two 16 bit Timer/ Counter
v. Input/output Pins – 4 Ports of 8 bits each on a single chip.
vi. 6 Interrupt Sources
vii. 8 – bit ALU (Arithmetic Logic Unit)
viii. **Harvard Memory Architecture** – It has 16 bit Address bus (each of RAM and ROM) and 8 bit Data Bus.

NRCM your roots to success...

ix. 8051 can execute 1 million one-cycle instructions per second with a clock frequency of 12MHz.

Block Diagram

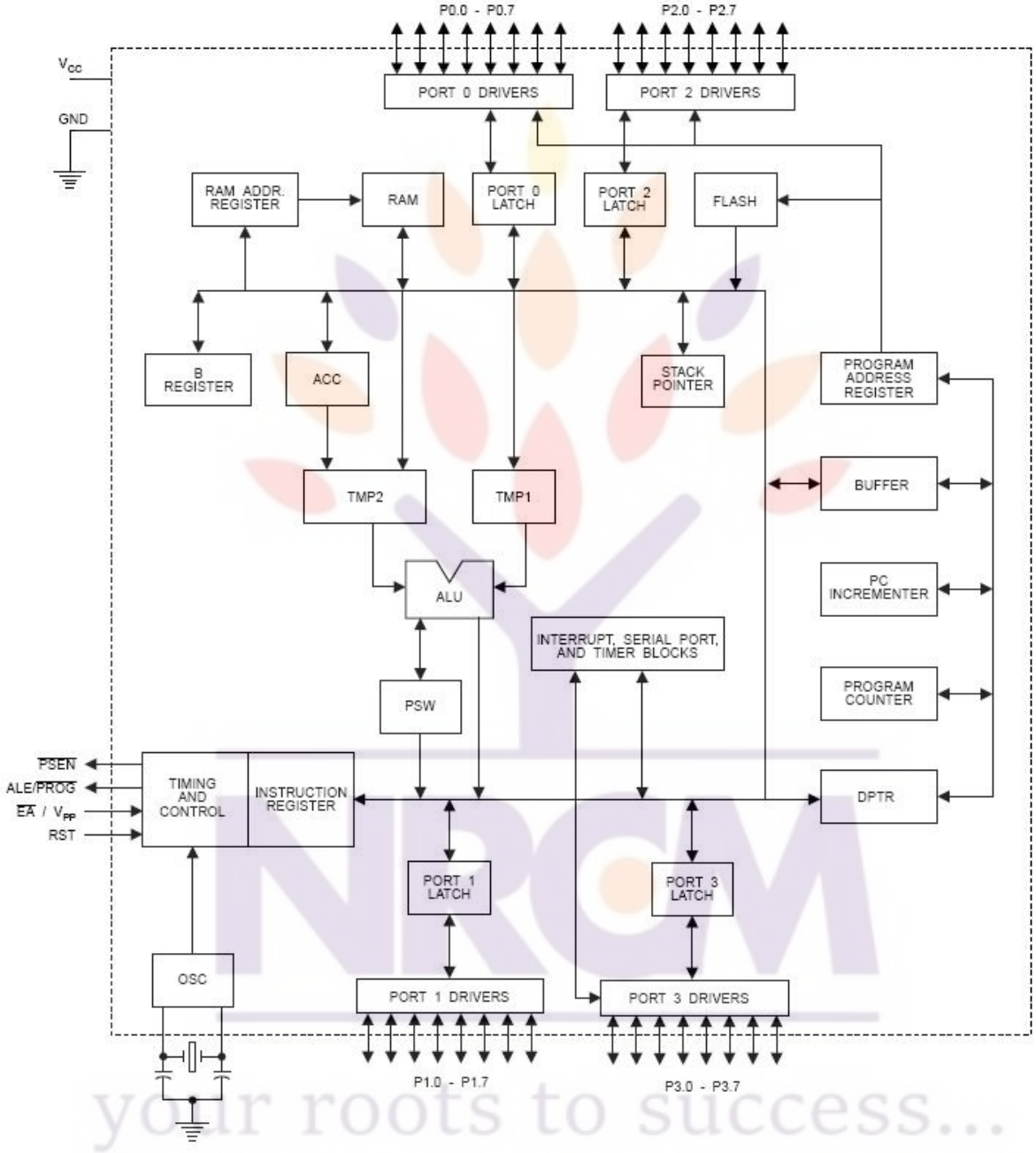


Fig 6.1. 8051 Architecture

The architecture of the 8051 family of microcontrollers is referred to as the MCS-51 architecture, or sometimes simply as MCS-51. The microcontrollers have an 8-bit data bus. They are capable of addressing 64K of program memory and a separate 64K of data memory. The 8051 has 4K of

code memory implemented as on-chip Read Only Memory (ROM). The 8051 has 128 bytes of internal Random Access Memory (RAM). The 8051 has two timer/counters, a serial port, 4 general purpose parallel input/output ports, and interrupt control logic with five sources of interrupts.

Besides internal RAM, the 8051 has various Special Function Registers (SFR), which are the control and data registers for on-chip facilities. The SFRs also include the accumulator, the B register, and the Program Status Word (PSW), which contains the CPU flags. Programming the various internal hardware facilities of the 8051 is achieved by placing the appropriate control words into the corresponding SFRs. The 8031 is similar to the 8051, except it lacks the on-chip ROM.As stated, the 8051 can address 64K of external data memory and 64K of external program memory. These may be separate blocks of memory, so that up to 128K of memory can be attached to the microcontroller. Separate blocks of code and data memory are referred to as the Harvard architecture. The 8051 has two separate read signals, RD# (P3.7) and PSEN#. The first is activated when a byte is to be read from external data memory, the other, from external program memory. Both of these signals are so-called active low signals. That is, they are cleared to logic level 0 when activated. All external code is fetched from external program memory. In addition, bytes from external program memory may be read by special read instructions such as the MOVC instruction. There are separate instructions to read from external data memory, such as the MOVX instruction. That is, the instructions determine which block of memory is addressed, and the corresponding control signal, either RD# or PSEN# is activated during the memory read cycle. A single block of memory may be mapped to act as both data and program memory. This is referred to as the Von Neumann architecture. In order to read from the same block using either the RD#signal or the PSEN# signal, the two signals are combined with a logic AND operation. This way, the output of the AND gate is low when either input is low. The advantage of the Harvard architecture is not simply doubling the memory capacity of the microcontroller. Separating program and data increases the reliability of the microcontroller, since there are no instructions to write to the program memory. A ROM device is ideally suited to serve as program memory. The Harvard architecture is somewhat awkward in evaluation systems, where code needs to be loaded into program memory.

6.1.2. 8051 pin diagram

your roots to success...

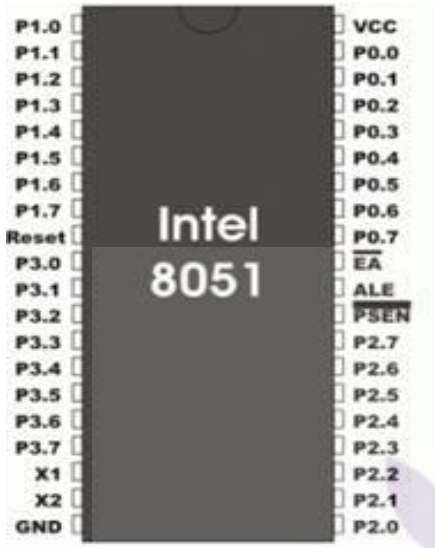


Fig 6.2. 8051 pin diagram

Pins 1-8: Port 1 Each of these pins can be configured as an input or an output.

Pin 9: RS A logic one on this pin disables the microcontroller and clears the contents of most registers. In other words, the positive voltage on this pin resets the microcontroller. By applying logic zero to this pin, the program starts execution from the beginning.

Pins10-17: Port 3 Similar to port 1, each of these pins can serve as general input or output. Besides, all of them have alternative functions:

Pin10: RXD Serial asynchronous communication input or Serial synchronous communication output.

Pin11: TXD Serial asynchronous communication output or Serial synchronous communication clock output.

Pin 12: INT0 Interrupt 0 input.

Pin 13: INT1 Interrupt 1 input.

Pin 14: T0 Counter 0 clock input.

Pin 15: T1 Counter 1 clock input.

Pin 16: WR Write to external (additional) RAM.

Pin 17: RD Read from external RAM.

Pin 18, 19: X2, X1 Internal oscillator input and output. A quartz crystal which specifies operating frequency is usually connected to these pins. Instead of it, miniature ceramics

NROM
your roots to success...

resonators can also be used for frequency stability. Later versions of microcontrollers operate at a frequency of 0 Hz up to over 50 Hz.

Pin 20: GND Ground.

Pin 21-28: Port 2 If there is no intention to use external memory then these port pins are configured as general inputs/outputs. In case external memory is used, the higher address byte, i.e. addresses A8-A15 will appear on this port. Even though memory with capacity of 64Kb is not used, which means that not all eight port bits are used for its addressing, the rest of them are not available as inputs/outputs.

Pin 29: PSEN If external ROM is used for storing program then a logic zero (0) appears on it every time the microcontroller reads a byte from memory.

Pin 30: ALE Prior to reading from external memory, the microcontroller puts the lower address byte (A0-A7) on P0 and activates the ALE output. After receiving signal from the ALE pin, the external register (usually 74HCT373 or 74HCT375 add-on chip) memorizes the state of P0 and uses it as a memory chip address. Immediately after that, the ALU pin is returned its previous logic state and P0 is now used as a Data Bus. As seen, port data multiplexing is performed by means of only one additional (and cheap) integrated circuit. In other words, this port is used for both data and address transmission.

Pin 31: EA By applying logic zero to this pin, P2 and P3 are used for data and address transmission with no regard to whether there is internal memory or not. It means that even there is a program written to the microcontroller, it will not be executed. Instead, the program written to external ROM will be executed. By applying logic one to the EA pin, the microcontroller will use both memories, first internal then external (if exists).

Pin 32-39: Port 0 Similar to P2, if external memory is not used, these pins can be used as general inputs/outputs. Otherwise, P0 is configured as address output (A0-A7) when the ALE pin is driven high (1) or as data output (Data Bus) when the ALE pin is driven low (0).

Pin 40: VCC +5V power supply.

6.2. 8051 in/ out ports

All 8051 microcontrollers have 4 I/O ports each comprising 8 bits which can be configured as inputs or outputs. Accordingly, in total of 32 input/output pins enabling the microcontroller to be connected to peripheral devices are available for use.

Pin configuration, i.e. whether it is to be configured as an input (1) or an output (0), depends on its logic state. In order to configure a microcontroller pin as an input, it is necessary to apply a logic zero (0) to appropriate I/O port bit. In this case, voltage level on appropriate pin will be 0.

Similarly, in order to configure a microcontroller pin as an input, it is necessary to apply a logic one (1) to appropriate port. In this case, voltage level on appropriate pin will be 5V (as is the case with any TTL input

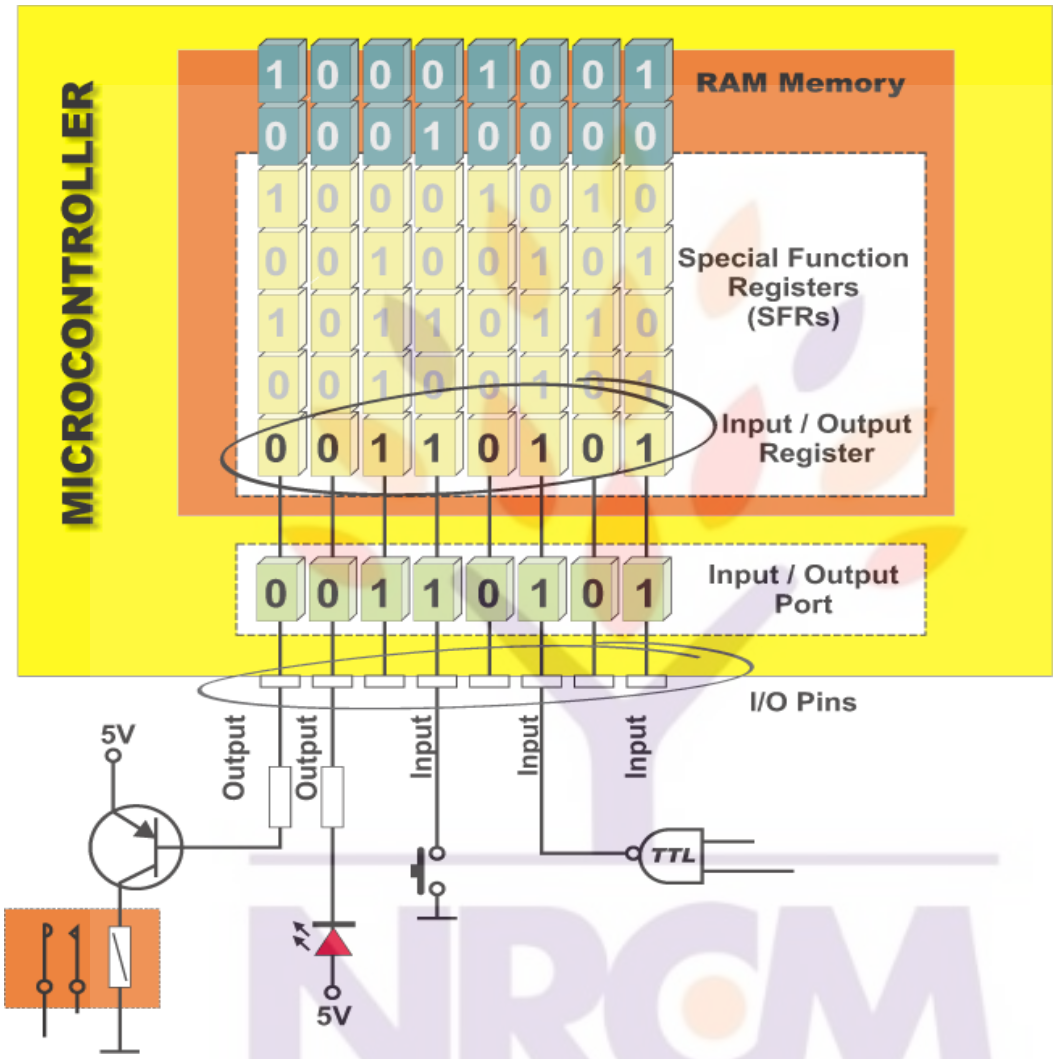
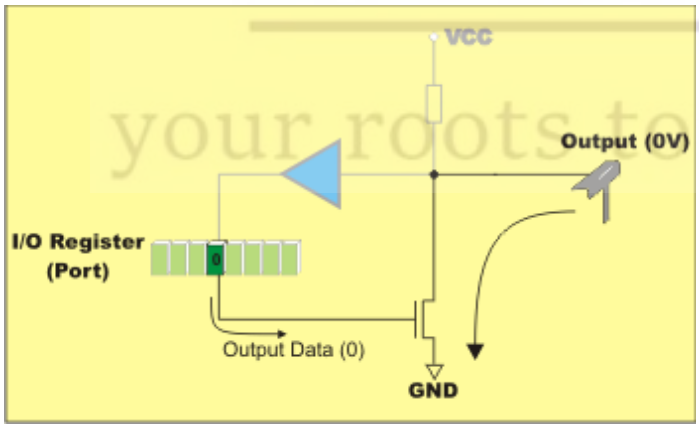
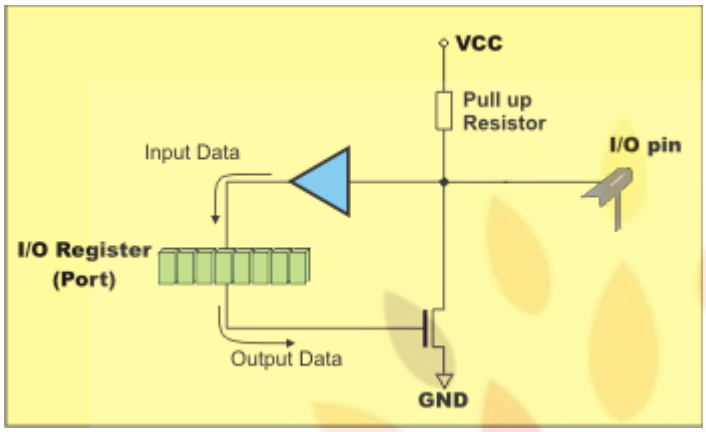


Fig 6.3. input/ output pin structure



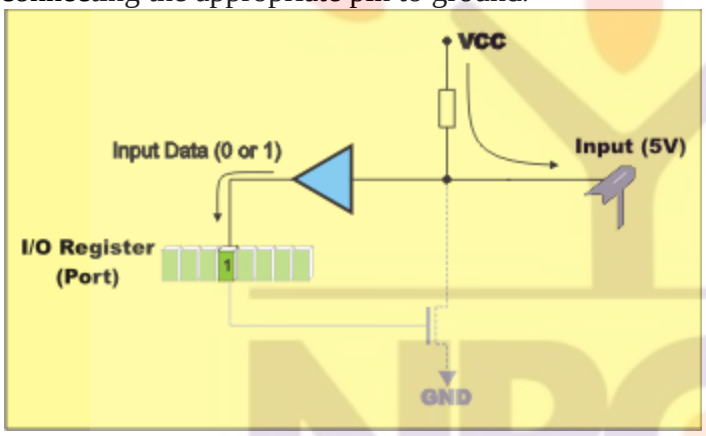
Input /Output (I/O)pin

Figure above illustrates a simplified schematic of all circuits within the microcontroller connected to one of its pins. It refers to all the pins except those of the P0 port which do not have pull-up resistors built-in.



Output pin

A logic zero (0) is applied to a bit of the P register. The output FE transistor is turned on, thus connecting the appropriate pin to ground.



Input pin

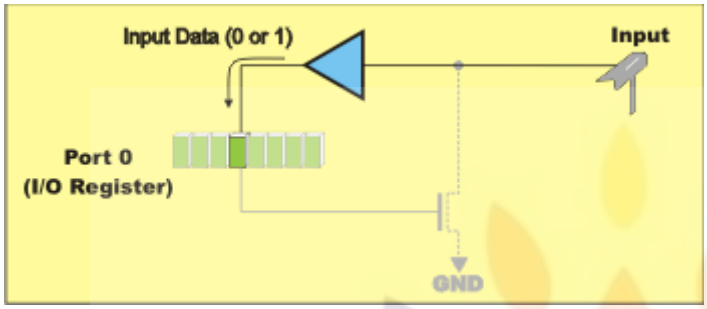
Logic one (1) is applied to a bit of the P register. The output FE transistor is turned off and the appropriate pin remains connected to the power supply voltage over a pull-up resistor of high resistance.

your roots to success...

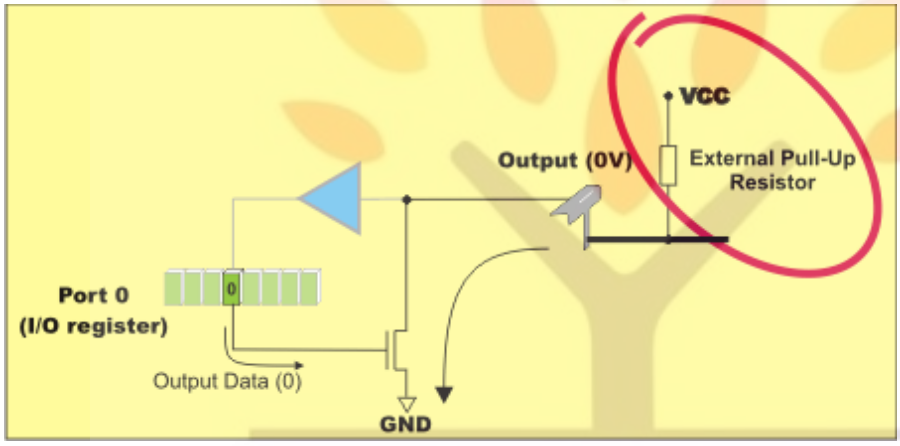
Port 0

The P0 port is characterized by two functions. If external memory is used then the lower address byte (addresses A0-A7) is applied on it. Otherwise, all bits of this port are configured as inputs/outputs.

The other function is expressed when it is configured as an output. Unlike other ports consisting of pins with built-in pull-up resistor connected by its end to 5 V power supply, pins of this port have this resistor left out. This apparently small difference has its consequences:



If any pin of this port is configured as an input then it acts as if it "floats". Such an input has unlimited input resistance and in determined potential.



When the pin is configured as an output, it acts as an "open drain". By applying logic 0 to a port bit, the appropriate pin will be connected to ground (0V). By applying logic 1, the external output will keep on "floating". In order to apply logic 1 (5V) on this output pin, it is necessary to built in an external pull-up resistor.

Port 1

P1 is a true I/O port, because it doesn't have any alternative functions as is the case with P0, but can be configured as general I/O only. It has a pull-up resistor built-in and is completely compatible with TTL circuits.

Port 2

P2 acts similarly to P0 when external memory is used. Pins of this port occupy addresses intended for external memory chip. This time it is about the higher address byte with addresses A8-A15. When no memory is added, this port can be used as a general input/output port showing features similar to P1.

Port 3

All port pins can be used as general I/O, but they also have an alternative function. In order to use these alternative functions, a logic one (1) must be applied to appropriate bit of the P3 register. In tems of hardware, this port is similar to P0, with the difference that its pins have a pull-up resistor built-in.

Special functions of port 3 are given by

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{\text{INT0}}$ (external interrupt)
P3.3	$\overline{\text{INT1}}$ (external interrupt)
P3.4	T0 (Timer/Counter 0 external input)
P3.5	T1 (Timer/Counter 1 external input)
P3.6	$\overline{\text{WR}}$ (external data memory write strobe)
P3.7	$\overline{\text{RD}}$ (external data memory read strobe)

Table 6.1. port 3 special functions

6.3. Memory organization

The 8051 has two types of memory and these are Program Memory and Data Memory. Program Memory (ROM) is used to permanently save the program being executed, while Data Memory (RAM) is used for temporarily storing data and intermediate results created and used during the operation of the microcontroller. Depending on the model in use (we are still talking about the 8051 microcontroller family in general) at most a few Kb of ROM and 128 or 256 bytes of RAM is used. All 8051 microcontrollers have a 16-bit addressing bus and are capable of addressing 64 kb memory. It is neither a mistake nor a big ambition of engineers who were working on basic core development. It is a matter of smart memory organization which makes these microcontrollers a real "programmers' goody".

Program Memory

The first models of the 8051 microcontroller family did not have internal program memory. It was added as an external separate chip. These models are recognizable by their label beginning

with 803 (for example 8031 or 8032). All later models have a few Kbyte ROM embedded. Even though such an amount of memory is sufficient for writing most of the programs, there are situations when it is necessary to use additional memory as well. A typical example are so called lookup tables. They are used in cases when equations describing some processes are too complicated or when there is no time for solving them. In such cases all necessary estimates and approximates are executed in advance and the final results are put in the tables

How does the microcontroller handle external memory depend on the EA pin logic state?

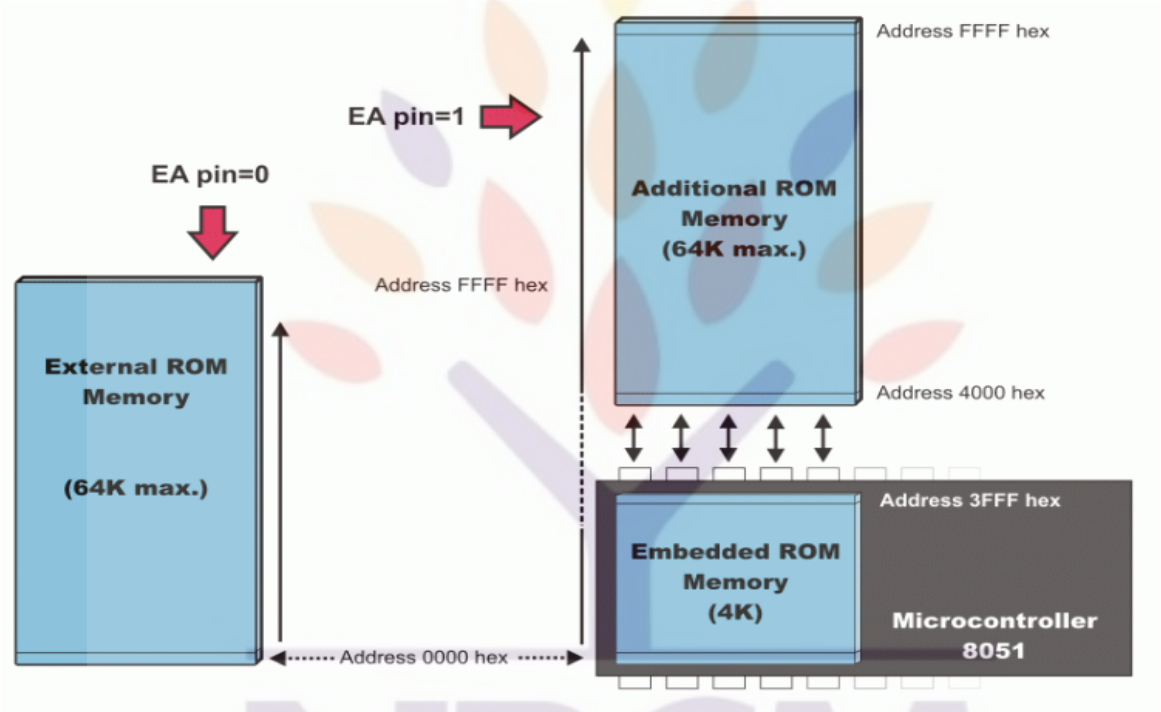


Fig 6.3. 8051 program memory

EA=0 In this case, the microcontroller completely ignores internal program memory and executes only the program stored in external memory.
EA=1 In this case, the microcontroller executes first the program from built-in ROM, then the program stored in external memory.
In both cases, P0 and P2 are not available for use since being used for data and address transmission. Besides, the ALE and PSEN pins are also used.

Data Memory

As already mentioned, Data Memory is used for temporarily storing data and intermediate results created and used during the operation of the microcontroller. Besides, RAM memory built in the 8051 family includes many registers such as hardware counters and timers, input/output ports, serial data buffers etc. The previous models had 256 RAM locations, while for the later models

this number was incremented by additional 128 registers. However, the first 256 memory locations (addresses 0-FFh) are the heart of memory common to all the models belonging to the 8051 family. Locations available to the user occupy memory space with addresses 0-7Fh, i.e. first 128 registers. This part of RAM is divided in several blocks.

The first block consists of 4 banks each including 8 registers denoted by R0-R7. Prior to accessing any of these registers, it is necessary to select the bank containing it. The next memory block (address 20h-2Fh) is bit- addressable, which means that each bit has its own address (0-7Fh). Since there are 16 such registers, this block contains in total of 128 bits with separate addresses (address of bit 0 of the 20h byte is 0, while address of bit 7 of the 2Fh byte is 7Fh). The third group of registers occupies addresses 2Fh-7Fh, i.e. 80 locations, and does not have any special functions or features.

Additional RAM

In order to satisfy the programmers' constant hunger for Data Memory, the manufacturers decided to embed an additional memory block of 128 locations into the latest versions of the 8051 microcontrollers. However, it's not as simple as it seems to be… The problem is that electronics performing addressing has 1 byte (8 bits) on disposal and is capable of reaching only the first 256 locations, therefore. In order to keep already existing 8-bit architecture and compatibility with other existing models a small trick was done.

What does it mean? It means that additional memory block shares the same addresses with locations intended for the SFRs (80h- FFh). In order to differentiate between these two physically separated memory spaces, different ways of addressing are used. The SFRs memory locations are accessed by direct addressing, while additional RAM memory locations are accessed by indirect addressing.

NRCM
your roots to success...

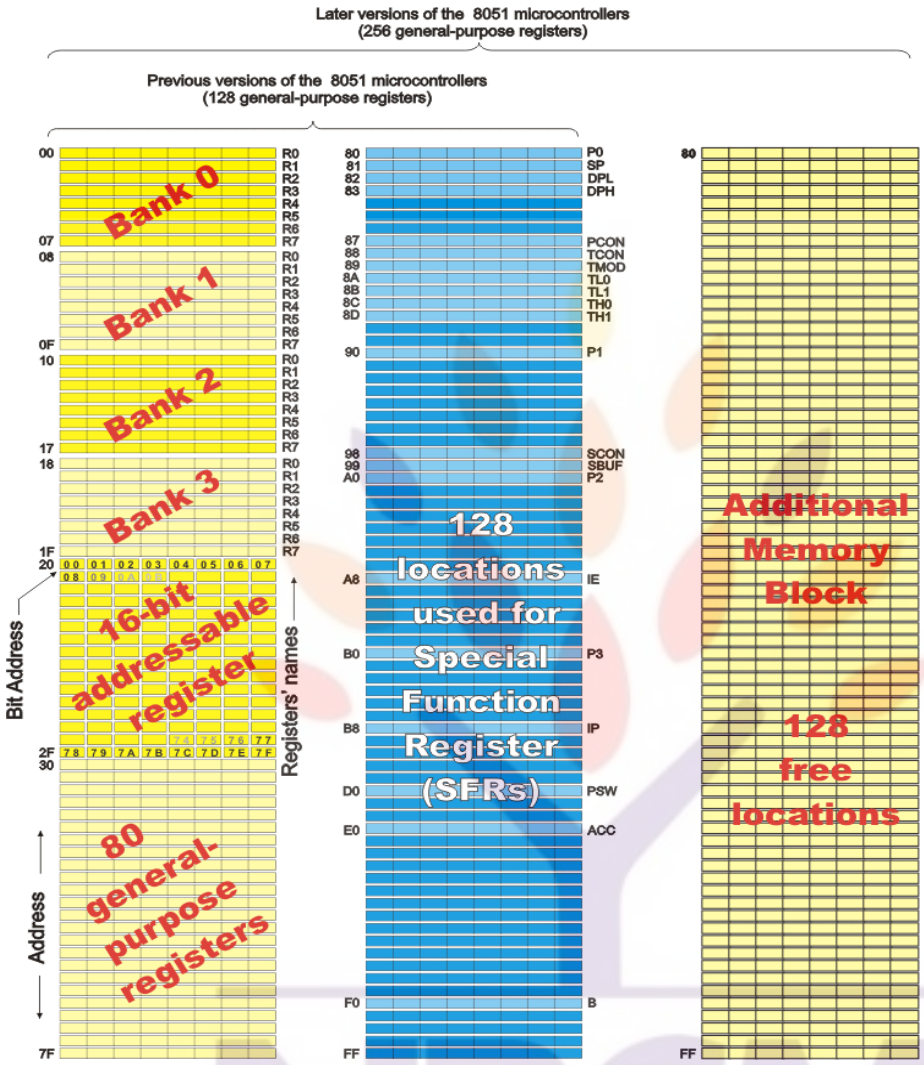


Fig 6.4. Data memory organization

Register Banks: 00h to 1Fh

The 8051 uses 8 general-purpose registers R0 through R7 (R0, R1, R2, R3, R4, R5,R6, and R7). These registers are used in instructions such as:

ADD A, R2 ; adds the value contained in R2 to the accumulator

Note since R2 happens to be memory location 02h in the Internal RAM the following instruction has the same effect as the above instruction.

ADD A, 02h; Now, things get more complicated when we see that there are four banks of the general-purpose registers defined within the Internal RAM. For the moment we will consider register bank 0 only. Register banks 1 to 3 can be ignored when writing introductory level assembly language programs.

Bit Addressable RAM: 20h to 2Fh

The 8051 supports a special feature which allows access to *bit variables*. This is where individual memory bits in Internal RAM can be set or cleared. In all there are 128 bits numbered 00h to 7Fh. Being bit variables any one variable can have a value 0 or 1. A bit variable can be set with a command such as SETB and cleared with a command such as CLR. Example instructions are:

SETB 25h ; sets the bit 25h (becomes 1)

CLR 25h ; clears bit 25h (becomes 0)

Note, bit 25h is actually bit b5 of Internal RAM location 24h.

The Bit Addressable area of the RAM is just 16 bytes of Internal RAM located between 20h and 2Fh. So if a program writes a byte to location 20h, for example, it writes 8 *bit variables*, bits 00h to 07h at once. Note bit addressing can also be performed on some of the SFR registers,

General Purpose RAM: 30h to 7Fh

These 80 bytes of Internal RAM memory are available for general-purpose data storage. Access to this area of memory is fast compared to access to the main memory and special instructions with single byte operands are used. However, these 80 bytes are used by the system stack and in practice little space is left for general storage. The general purpose RAM can be accessed using direct or indirect addressing modes.

Examples of direct addressing:

MOV A, 6Ah ; reads contents of address 6Ah to accumulator

Examples for indirect addressing (**use registers R0 or R1**):

MOV R1, #6Ah ; move immediate 6Ah to R1

MOV A, @R1 ; move indirect: R1 contains address of Internal RAM which contains data that is moved to A. These two instructions have the same effect as the direct instruction above.

SFR Registers

The SFR registers are located within the Internal Memory in the address range 80h to FFh, as shown in figure 6.7. Not all locations within this range are defined. Each SFR has a very specific function. Each SFR has an address (within the range 80h to FFh) and a name which reflects the purpose of the SFR. Although 128 byes of the SFR address space is defined only 21 SFR registers are defined in the standard 8051. Undefined SFR addresses should not be accessed as this might lead to some unpredictable results. Note some of the SFR registers are *bit addressable*. SFRs are accessed just like normal Internal RAM locations.

your roots to success...

6.4. 8051 addressing modes

There are a number of addressing modes available to the 8051 instruction set, as follows:

Immediate Addressing	Register Addressing	Direct Addressing
Indirect Addressing	Relative Addressing	Absolute addressing

Long Addressing Indexed Addressing

Immediate Addressing

Immediate addressing simply means that the operand (which immediately follows the instruction op. code) is the data value to be used.

For example the instruction:

MOV A, #99d-- Moves the value 99 into the accumulator (note this is 99 decimal since we used 99d).

The # symbol tells the assembler that the immediate addressing mode is to be used.

One of the eight general-registers, R0 to R7, can be specified as the instruction operand. The assembly language documentation refers to a register generically as *Rn*. An example instruction using register addressing is :

ADD A, R5 ; Adds register R5 to A (accumulator)-Here the contents of R5 is added to the accumulator. One advantage of register addressing is that the instructions tend to be short, single byte instructions.

Direct Addressing

Direct addressing means that the data value is obtained directly from the memory location specified in the operand.

For example consider the instruction:

MOV A, 47h

The instruction reads the data from Internal RAM address 47h and stores this in the accumulator. Direct addressing can be used to access Internal RAM , including the SFR registers.

Indirect Addressing

Indirect addressing provides a powerful addressing capability, which needs to be appreciated. An example instruction, which uses indirect addressing, is as follows:

MOV A, @R0

Note the @ symbol indicated that the indirect addressing mode is used. R0 contains a value, for example 54h, which is to be used as the address of the internal RAM location, which contains the operand data. Indirect addressing refers to Internal RAM only and cannot be used to refer to SFR registers. Note, only R0 or R1 can be used as register data pointers for indirect addressing when using MOV instructions.

The 8052 (as opposed to the 8051) has an additional 128 bytes of internal RAM. These 128 bytes of RAM can be accessed only using indirect addressing.

Relative Addressing

This is a special addressing mode used with certain jump instructions. The relative address, often referred to as an offset, is an 8-bit signed number, which is automatically added to the PC to

NRCM your roots to success...

make the address of the next instruction. The 8-bit signed offset value gives an address range of + 127 to –128 locations.

Consider the following example:

SJMP LABEL_X

An advantage of relative addressing is that the program code is easy to relocate in memory in that the addressing is relative to the position in memory.

Absolute addressing

Absolute addressing within the 8051 is used only by the AJMP (Absolute Jump) and ACALL (Absolute Call) instructions, which will be discussed later.

Long Addressing

The long addressing mode within the 8051 is used with the instructions LJMP and LCALL. The address specifies a full 16 bit destination address so that a jump or a call can be made to a location within a 64KByte code memory space (216 = 64K).

An example instruction is:

LJMP 5000h ; full 16 bit address is specified in operand

Indexed Addressing

With indexed addressing a separate register, either the program counter, PC, or the data pointer DTPR, is used as a base address and the accumulator is used as an offset address. The effective address is formed by adding the value from the base address to the value from the offset address. Indexed addressing in the 8051 is used with the JMP or MOVC instructions. Look up tables are easy to implement with the help of index addressing.

Consider the example instruction:

MOVC A, @A+DPTR

MOVC is a move instruction, which moves data from the external code memoryspace. The address operand in this example is formed by adding the content of theDPTR register to the accumulator value. Here the DPTR value is referred to as the*base address* and the accumulator value us referred to as the *index address*.

6.5. 8051 instruction set

The assembly level instructions include: data transfer instructions, arithmeticinstructions, logical instructions, program control instructions, and some specialinstructions such as the rotate instructions.

Data Transfer

Many computer operations are concerned with moving data from one location toanother. The 8051 uses five different types of instruction to move data:

MOV	MOVX	MOVC
PUSH	POP	XCH

MOV

In the 8051 the MOV instruction is concerned with moving data internally, i.e.between Internal RAM, SFR registers, general registers etc. MOVX and MOVC areused in accessing external memory data. The MOV instruction has the followingformat:

MOV *destination <- source*

The instruction copies *(copy* is a more accurate word than *move)* data from a definedsource location to a destination location. Example MOV instructions are:

MOV R2, #80h ; Move immediate data value 80h to register R2
MOV R4, A ; Copy data from accumulator to register R4
MOV DPTR, #0F22Ch ; Move immediate value F22Ch to the DPTR register
MOV R2, 80h ; Copy data from 80h (Port 0 SFR) to R2
MOV 52h, #52h ; Copy immediate data value 52h to RAM location 52h
MOV 52h, 53h ; Copy data from RAM location 53h to RAM 52h
MOV A, @R0 ; Copy contents of location addressed in R0 to A(indirect addressing).

MOVX

The 8051 the external memory can be addressed using *indirect* addressing only. TheDPTR register is used to hold the address of the external data (since DPTR is a 16-bitregister it can address 64KByte locations: 216 = 64K). The 8 bit registers R0 or R1 canalso be used for indirect addressing of external memory but the address range islimited to the lower 256 bytes of memory (28 = 256 bytes).

The MOVX instruction is used to access the external memory (X indicates externalmemory access). All external moves must work through the A register (accumulator).

Examples of MOVX instructions are:

MOVX @DPTR, A ; Copy data from A to the address specified in DPTR
MOVX A, @DPTR ; Copy data from address specified in DPTR to A

MOVC

MOVX instructions operate on RAM, which is (normally) a volatile memory.Program tables often need to be stored in ROM since ROM is non volatile memory.The MOVC instruction is used to read data from the external code memory (ROM).Like the MOVX instruction the DPTR register is used as the indirect address register.The indirect addressing is enhanced to realize an indexed addressing mode whereregister A can be used to provide an offset in the address specification. Like theMOVX instruction all moves must be done through register A. The followingsequence of instructions provides an example:

MOV DPTR, # 2000h ; Copy the data value 2000h to the DPTR register
MOV A, #80h ; Copy the data value 80h to register A
MOVC A, @A+DPTR ; Copy the contents of the address 2080h (2000h + 80h) to register A

Note, for the MOVC the program counter, PC, can also be used to form the address.

PUSH and POP

PUSH and POP instructions are used with the stack only. The SFR register SPcontains the current stack address. Direct addressing is used as shown in the following examples:

PUSH 4Ch ; Contents of RAM location 4Ch is saved to the stack. SP isincremented.

PUSH 00h ; The content of R0 (which is at 00h in RAM) is saved to the stack andSP is incremented.

POP 80h ; The data from current SP address is copied to 80h and SP isdecremented.

XCH

The above move instructions copy data from a source location to a destinationlocation, leaving the source data unaffected. A special XCH (exchange) instructionwill actually swap the data between source and destination, effectively changing thesource data. Immediate addressing may not be used with XCH. XCH instructionsmust use register A. XCHD is a special case of the exchange instruction where justthe lower nibbles are exchanged. Examples using the XCH instruction are:

XCH A, R3 ; Exchange bytes between A and R3

XCH A, @R0 ; Exchange bytes between A and RAM location whose address is in R0

XCH A, A0h ; Exchange bytes between A and RAM location A0h (SFR port 2)

Arithmetic

Some key flags within the PSW, i.e. C, AC, OV, P, are utilized in many of thearithmetic instructions. The arithmetic instructions can be grouped as follows:

Addition

Subtraction

Increment/decrement

Multiply/divide

Decimal adjust

Addition

Register A (the accumulator) is used to hold the result of any addition operation.Some simple addition examples are:

ADD A, #25h ; Adds the number 25h to A, putting sum in A

ADD A, R3 ; Adds the register R3 value to A, putting sum in A

The flags in the PSW register are affected by the various addition operations, asfollows:

The C (carry) flag is set to 1 if the addition resulted in a carry out of the accumulator's MSB bit, otherwise it is cleared.

The AC (auxiliary) flag is set to 1 if there is a carry out of bit position 3 of theaccumulator, otherwise it is cleared.

For signed numbers the OV flag is set to 1 if there is an arithmetic overflow(described elsewhere in these notes)

Simple addition is done within the 8051 based on 8 bit numbers, but it is oftenrequired to add 16 bit numbers, or 24 bit numbers etc. This leads to the use ofmultiple byte (multi-precision) arithmetic. The least significant bytes are first added,and if a carry results, this carry is carried over in the addition of the next significantbyte etc. This addition process is done at 8-bit precision steps to achieve multiprecisionarithmetic. The ADDC instruction is used to include the carry bit in theaddition process. Example instructions using ADDC are:

ADDC A, #55h ; Add contents of A, the number 55h, the carry bit and put thesum in A

ADDC A, R4 ; Add the contents of A, the register R4, the carry bit and putthe sum in A.

Subtraction

Computer subtraction can be achieved using 2's complement arithmetic. Mostcomputers also provide instructions to directly subtract signed or unsigned numbers.The accumulator, register A, will contain the result (difference) of the subtractionoperation. The C (carry) flag is treated as a borrow flag, which is always subtractedfrom the minuend during a subtraction operation. Some examples of subtractioninstructions are:

SUBB A, #55d ; Subtract the number 55 (decimal) and the C flag from A; andput the result in A.

SUBB A, R6 ; Subtract R6 the C flag from A; and put the result in A.

SUBB A, 58h ; Subtract the number in RAM location 58h and the C flagFrom A; and put the result in A.

Increment/Decrement

The increment (INC) instruction has the effect of simply adding a binary 1 to anumber while a decrement (DEC) instruction has the effect of subtracting a binary 1from a number. The increment and decrement instructions can use the addressingmodes: direct, indirect and register. The flags C, AC, and OV are **not** affected by theincrement or decrement instructions. If a value of FFh is increment it overflows to00h. If a value of 00h is decrement it underflows to FFh. The DPTR can overflowfrom FFFFh to 0000h. The DPTR register cannot be decremented using a DECinstruction (unfortunately!). Some example INC and DEC instructions are as follows:

INC R7 ; Increment register R7

INC A ; Increment A

INC @R1 ; Increment the number which is the content of the address in R1

DEC A ; Decrement register A

DEC 43h ; Decrement the number in RAM address 43h

INC DPTR ; Increment the DPTR register

Multiply / Divide

The 8051 supports 8-bit multiplication and division. This is low precision (8 bit)arithmetic but is useful for many simple control applications. The arithmetic isrelatively fast since multiplication

and division are implemented as singleinstructions. If better precision, or indeed, if floating point arithmetic is required thenspecial software routines need to be written. For the MUL or DIV instructions the Aand B registers must be used and only unsigned numbers are supported.

Multiplication

The MUL instruction is used as follows (note absence of a comma between the A and B operands):

MUL AB ; Multiply A by B.

The resulting product resides in registers A and B, the low-order byte is in A and thehigh order byte is in B.

Division

The DIV instruction is used as follows:

DIV AB ; A is divided by B.

The remainder is put in register B and the integer part of the quotient is put in registerA.

Decimal Adjust (Special)

The 8051 performs all arithmetic in binary numbers (i.e. it does not support BCDarithmetic). If two BCD numbers are added then the result can be adjusted by usingthe DA, decimal adjust, instruction:

DA A ; Decimal adjust A following the addition of two BCD numbers.

Logical

Boolean Operations

Most control applications implement control logic using Boolean operators to act onthe data. Most microcomputers provide a set of Boolean instructions that act on bytelevel data. However, the 8051 (somewhat uniquely) additionally provides Booleaninstruction which can operate on bit level data.The following Boolean operations can operate on byte level or bit level data:

ANL Logical AND

ORL Logical OR

CPL Complement (logical NOT)

XRL Logical XOR (exclusive OR)

Logical operations at the BYTE level

The destination address of the operation can be the accumulator (register A), ageneral register, or a direct address. Status flags are not affected by these logicaloperations (unless PSW is directly manipulated). Example instructions are:

ANL A, #55h ; AND each bit in A with corresponding bit in number 55h, leavingthe result in A.

ANL 42h, R4 ; AND each bit in RAM location 42h with corresponding bit in R4,leaving the result in RAM location 42h.

ORL A,@R1 ; OR each bit in A with corresponding bit in the number whose addressis contained in R1 leaving the result in A.
XRL R4, 80h ; XOR each bit in R4 with corresponding bit in RAM location 80h(port 0), leaving result in A.
CPL R0 ; Complement each bit in R0

Logical operations at the BIT level

The C (carry) flag is the destination of most bit level logical operations. The carry flagcan easily be tested using a branch (jump) instruction to quickly establish programflow control decisions following a bit level logical operation.The following SFR registers only are addressable in bit level operations:

PSW IE IP TCON SCON

Examples of bit level logical operations are as follows:
SETB 2Fh ; Bit 7 of Internal RAM location 25h is set
CLR C ; Clear the carry flag (flag =0)
CPL 20h ; Complement bit 0 of Internal RAM location 24h
MOV C, 87h ; Move to carry flag the bit 7of Port 0 (SFR at 80h)
ANL C,90h ; AND C with the bit 0 of Port 1 (SFR at 90)
ORL C, 91h ; OR C with the bit 1 of Port 1 (SFR at 90)

Rotate Instructions

The ability to rotate the A register (accumulator) data is useful to allow examinationof individual bits. The options for such rotation are as follows:
RL A ; Rotate A one bit to the left. Bit 7 rotates to the bit 0 position

RLC A ; The Carry flag is used as a ninth bit in the rotation loop

RR A ; Rotates A to the right (clockwise)

RRC A ; Rotates to the right and includes the carry bit as the 9th bit.

Swap = special

The Swap instruction swaps the accumulator's high order nibble with the low-ordernibble using the instruction:
SWAP A

your roots to success...

Program Control Instructions

The 8051 supports three kinds of jump instructions:
LJMP SJMP AJMP

LJMP

LJMP (long jump) causes the program to branch to a destination address defined bythe 16-bit operand in the jump instruction. Because a 16-bit address is used theinstruction can cause a jump to any location within the 64KByte program space (216 =64K). Some example instructions are:
LJMP LABEL_X ; Jump to the specified label
LJMP 0F200h ; Jump to address 0F200h
LJMP @A+DPTR ; Jump to address which is the sum of DPTR and Reg. A

SJMP

SJMP (short jump) uses a single byte address. This address is a signed 8-bit numberand allows the program to branch to a distance –128 bytes back from the current PC address or +127 bytes forward from the current PC address. The address mode usedwith this form of jumping (or branching) is referred to as *relative addressing*,introduced earlier, as the jump is calculated relative to the current PC address.

AJMP

This is a special 8051 jump instruction, which allows a jump with a 2KByte addressboundary (a 2K page)There is also a generic JMP instruction supported by many 8051 assemblers. Theassembler will decide which type of jump instruction to use, LJMP, SJMP or AJMP,so as to choose the most efficient instruction.

Subroutines and program flow control

A subroutine is called using the LCALL or the ACALL instruction.

LCALL

This instruction is used to call a subroutine at a specified address. The address is 16bits long so the call can be made to any location within the 64KByte memory space.When a LCALL instruction is executed the current PC content is automaticallypushed onto the stack of the PC. When the program returns from the subroutine thePC contents is returned from the stack so that the program can resume operation fromthe point where the LCALL was made.The return from subroutine is achieved using the RET instruction, which simply popsthe PC back from the stack.

ACALL

The ACALL instruction is logically similar to the LCALL but has a limited addressrange similar to the AJMP instruction.CALL is a generic call instruction supported by many 8051 assemblers. Theassembler will decide which type of call instruction, LCALL or ACALL, to use so asto choose the most efficient instruction.

Program control using conditional jumps

NRCM your roots to success...

Most 8051 jump instructions use an 8-bit destination address, based on relativeaddressing, i.e. addressing within the range –128 to +127 bytes.When using a conditional jump instruction the programmer can simply specify aprogram label or a full 16-bit address for the conditional jump instruction'sdestination. The assembler will position the code and work out the correct 8-bitrelative address for the instruction. Some example conditional jump instructions are:

JZ LABEL_1 ; Jump to LABEL_1 if accumulator is equal to zero

JNZ LABEL_X ; Jump to LABEL_X if accumulator is not equal to zero

JNC LABEL_Y ; Jump to LABEL_Y if the carry flag is not set

DJNZ R2, LABEL ; Decrement R2 and jump to LABEL if the resulting value ofR2 is not zero.

CJNE R1, #55h , LABEL_2; Compare the magnitude of R1 and the number 55h and jump to LABEL_2 if themagnitudes are not equal.

your roots to success...

UNIT – 5

8051 REAL TIME CONTROL

7.1. Special function registers:

Special Function Registers (SFRs) are a sort of control table used for running and monitoring the operation of the microcontroller. Each of these registers as well as each bit they include, has its name, address in the scope of RAM and precisely defined purpose such as timer control, interrupt control, serial communication control etc. Even though there are 128 memory locations intended to be occupied by them, the basic core, shared by all types of 8051 microcontrollers, has only 21 such registers. Rest of locations is intentionally left unoccupied in order to enable the manufacturers to further develop microcontrollers keeping them compatible with the previous versions. It also enables programs written a long time ago for microcontrollers which are out of production now to be used today.

F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8									CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	P0	SP	DPL	DPH				PCON	87

Bit-addressable Registers

Fig 7.1. 8051 SFR'S

A Register (Accumulator)

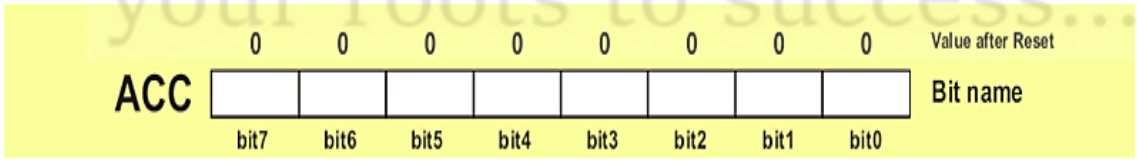


Fig 7.2. 8051 Accumulator

A register is a general-purpose register used for storing intermediate results obtained during operation. Prior to executing an instruction upon any number or operand it is necessary to store it

in the accumulator first. All results obtained from arithmetical operations performed by the ALU are stored in the accumulator. Data to be moved from one register to another must go through the accumulator. In other words, the A register is the most commonly used register and it is impossible to imagine a microcontroller without it. More than half instructions used by the 8051 microcontroller use somehow the accumulator.

B Register

Multiplication and division can be performed only upon numbers stored in the A and B registers. All other instructions in the program can use this register as a spare accumulator (A).

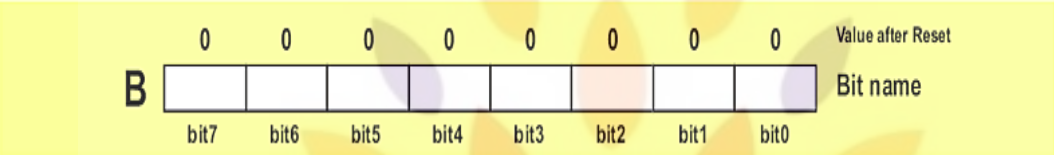


Fig 7.3. 8051 b register

R Registers (R0-R7)

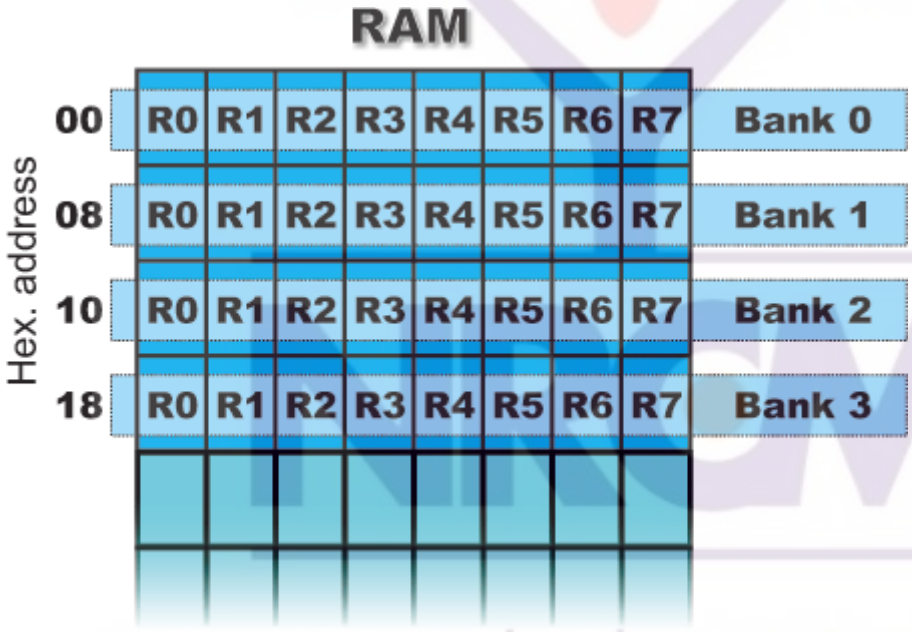


Fig 7.4. 8051 Register banks

This is a common name for 8 general-purpose registers (R0, R1, R2 ...R7). Even though they are not true SFRs, they deserve to be discussed here because of their purpose. They occupy 4 banks within RAM. Similar to the accumulator, they are used for temporary storing variables and

intermediate results during operation. Which one of these banks is to be active depends on two bits of the PSW Register. Active bank is a bank the registers of which are currently used.

The following example best illustrates the purpose of these registers. Suppose it is necessary to perform some arithmetical operations upon numbers previously stored in the R registers: (R1+R2) - (R3+R4). Obviously, a register for temporary storing results of addition is needed. This is how it looks in the program:

```
MOV A,R3; Means: move number from R3 into accumulator
ADD A,R4; Means: add number from R4 to accumulator (result remains in
accumulator)
MOV R5,A; Means: temporarily move the result from accumulator into R5
MOV A,R1; Means: move number from R1 to accumulator
ADD A,R2; Means: add number from R2 to accumulator
SUBB A,R5; Means: subtract number from R5 (there are R3+R4)
```

Program Status Word (PSW) Register

Fig 7.5. 8051 PSW

PSW register is one of the most important SFRs. It contains several status bits that reflect the current state of the CPU. Besides, this register contains Carry bit, Auxiliary Carry, two register bank select bits, Overflow flag, parity bit and user-definable status flag.

P - Parity bit. If a number stored in the accumulator is even then this bit will be automatically set (1), otherwise it will be cleared (0). It is mainly used during data transmit and receive via serial communication.

- Bit 1. This bit is intended to be used in the future versions of microcontrollers.

OV Overflow occurs when the result of an arithmetical operation is larger than 255 and cannot be stored in one register. Overflow condition causes the OV bit to be set (1). Otherwise, it will be cleared (0).

RS0, RS1 - Register bank select bits. These two bits are used to select one of four register banks of RAM. By setting and clearing these bits, registers R0-R7 are stored in one of four banks of RAM.

RS1	RS2	SPACE IN RAM
0	0	Bank0 00h-07h
0	1	Bank1 08h-0Fh

1	0	Bank2 10h-17h
1	1	Bank3 18h-1Fh

Table 7.1. Register bank selection

F0 - Flag 0. This is a general-purpose bit available for use.
AC - Auxiliary Carry Flag is used for BCD operations only.
CY - Carry Flag is the (ninth) auxiliary bit used for all arithmetical operations and shift instructions.

Data Pointer Register (DPTR)

DPTR register is not a true one because it doesn't physically exist. It consists of two separate registers: DPH (Data Pointer High) and (Data Pointer Low). For this reason it may be treated as a 16-bit register or as two independent 8-bit registers. Their 16 bits are primarly used for external memory addressing. Besides, the DPTR Register is usually used for storing data and intermediate results.

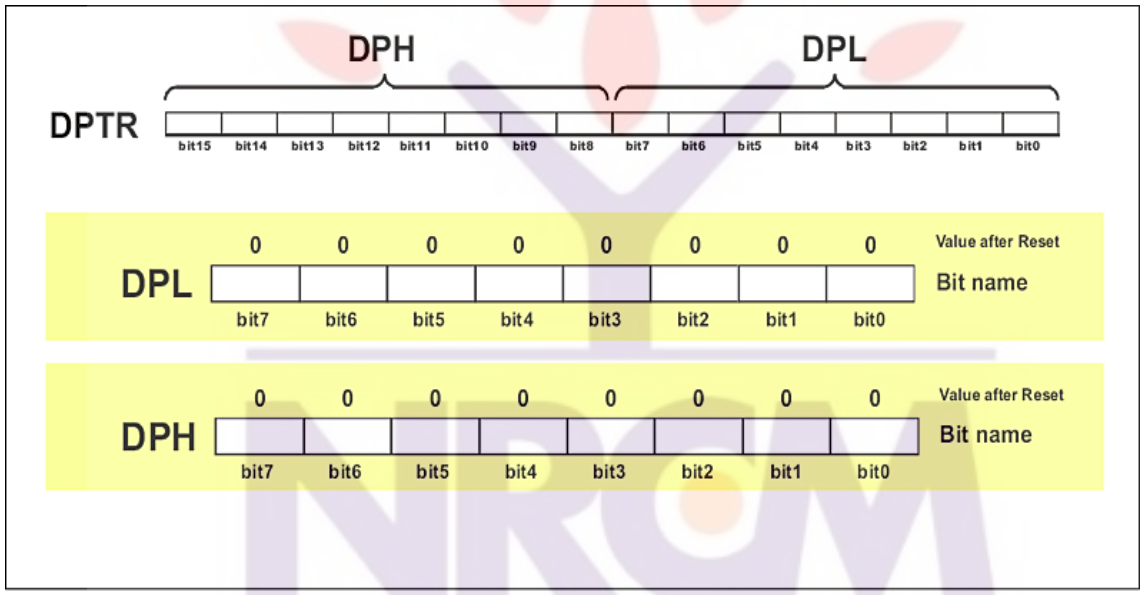


Fig 7.6. 8051 Data Pointer

Stack Pointer (SP) Register

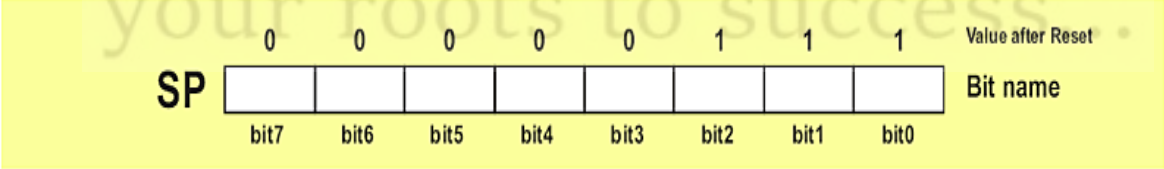


Fig 7.7. 8051 Stack Pointer

A value stored in the Stack Pointer points to the first free stack address and permits stack availability. Stack pushes increment the value in the Stack Pointer by 1. Likewise, stack pops decrement its value by 1. Upon any reset and power-on, the value 7 is stored in the Stack Pointer, which means that the space of RAM reserved for the stack starts at this location. If another value is written to this register, the entire Stack is moved to the new memory location.

P0, P1, P2, P3 – Input/Output Registers

If neither external memory nor serial communication system are used then 4 ports with in total of 32 input/output pins are available for connection to peripheral environment. Each bit within these ports affects the state and performance of appropriate pin of the microcontroller. Thus, bit logic state is reflected on appropriate pin as a voltage (0 or 5 V) and vice versa, voltage on a pin reflects the state of appropriate port bit.

As mentioned, port bit state affects performance of port pins, i.e. whether they will be configured as inputs or outputs. If a bit is cleared (0), the appropriate pin will be configured as an output, while if it is set (1), the appropriate pin will be configured as an input. Upon reset and power-on, all port bits are set (1), which means that all appropriate pins will be configured as inputs.

7.2. Counters and Timers

As you already know, the microcontroller oscillator uses quartz crystal for its operation. As the frequency of this oscillator is precisely defined and very stable, pulses it generates are always of the same width, which makes them ideal for time measurement. Such crystals are also used in quartz watches. In order to measure time between two events it is sufficient to count up pulses coming from this oscillator. That is exactly what the timer does. If the timer is properly programmed, the value stored in its register will be incremented (or decremented) with each coming pulse, i.e. once per each machine cycle. A single machine-cycle instruction lasts for 12 quartz oscillator periods, which means that by embedding quartz with oscillator frequency of 12MHz, a number stored in the timer register will be changed million times per second, i.e. each microsecond.

The 8051 microcontroller has 2 timers/counters called T0 and T1. As their names suggest, their main purpose is to measure time and count external events. Besides, they can be used for generating clock pulses to be used in serial communication, so called Baud Rate.

Timer T0

As seen in figure below, the timer T0 consists of two registers – TH0 and TL0 representing a low and a high byte of one 16-digit binary number.

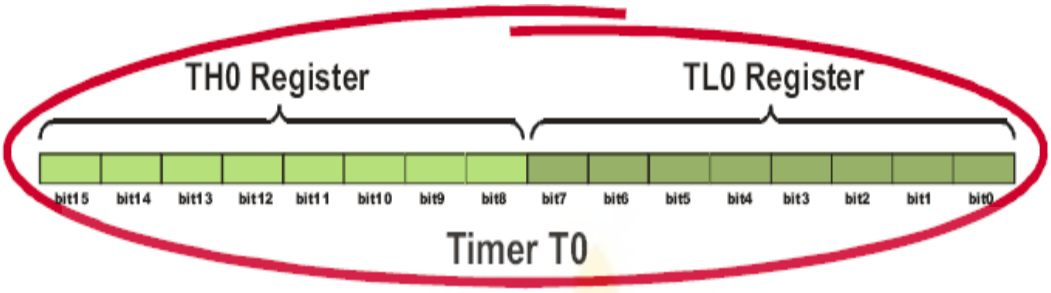
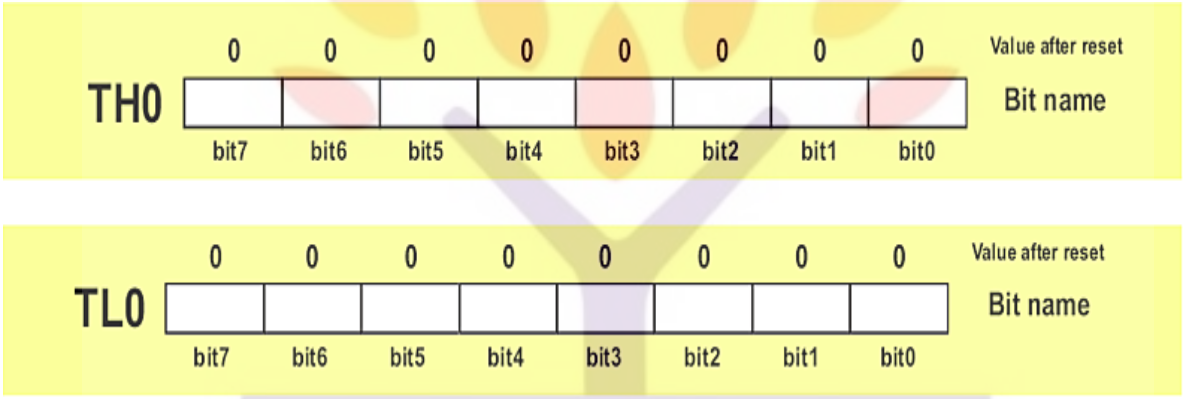


Fig 7.8. Timer-0 register

Accordingly, if the content of the timer T0 is equal to 0 (T0=0) then both registers it consists of will contain 0. If the timer contains for example number 1000 (decimal), then the TH0 register (high byte) will contain the number 3, while the TL0 register (low byte) will contain decimal number 232.

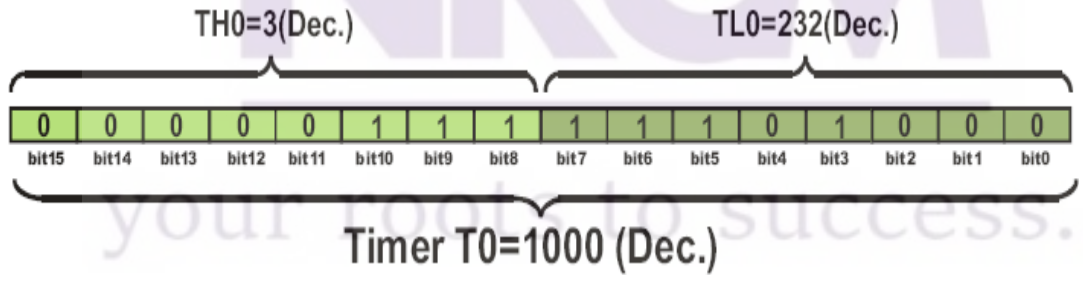


Formula used to calculate values in these two registers is very simple:

TH0 × 256 + TL0 = T

Matching the previous example it would be as follows:

3 × 256 + 232 = 1000



Since the timer T0 is virtually 16-bit register, the largest value it can store is 65 535. In case of exceeding this value, the timer will be automatically cleared and counting starts from 0. This condition is called an overflow. Two registers TMOD and TCON are closely connected to this timer and control its operation.

TMOD Register (Timer Mode)

The TMOD register selects the operational mode of the timers T0 and T1. As seen in figure below, the low 4 bits (bit0 - bit3) refer to the timer 0, while the high 4 bits (bit4 - bit7) refer to the timer 1. There are 4 operational modes and each of them is described herein.

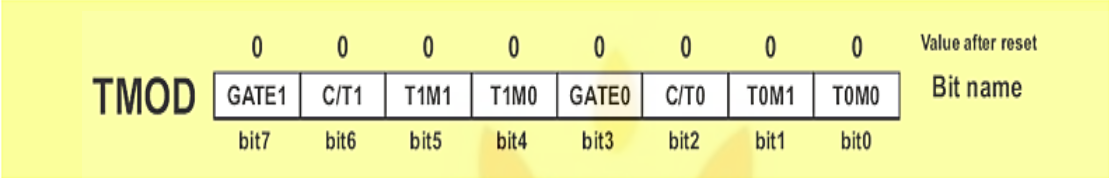


Fig 7.9. 8051 TMOD Register

Bits of this register have the following function:

- **GATE1** enables and disables Timer 1 by means of a signal brought to the INT1 pin (P3.3):
  - **1** - Timer 1 operates only if the INT1 bit is set.
  - **0** - Timer 1 operates regardless of the logic state of the INT1 bit.
- **C/T1** selects pulses to be counted up by the timer/counter 1:
  - **1** - Timer counts pulses brought to the T1 pin (P3.5).
  - **0** - Timer counts pulses from internal oscillator.
- **T1M1,T1M0** These two bits select the operational mode of the Timer 1.

T1M1	T1M0	MODE	DESCRIPTION
0	0	0	13-bit timer
0	1	1	16-bit timer
1	0	2	8-bit auto-reload
1	1	3	Split mode

Table 7.2. Timer Mode Selection

- **GATE0** enables and disables Timer 1 using a signal brought to the INT0 pin (P3.2):
-
  - **1** - Timer 0 operates only if the INT0 bit is set.
  - **0** - Timer 0 operates regardless of the logic state of the INT0 bit.
- **C/T0** selects pulses to be counted up by the timer/counter 0:
  - **1** - Timer counts pulses brought to the T0 pin (P3.4).
  - **0** - Timer counts pulses from internal oscillator.
- **T0M1,T0M0** These two bits select the oprtaional mode of the Timer 0.

Timer 0 in mode 0 (13-bit timer)

This is one of the rarities being kept only for the purpose of compatibility with the previuos versions of microcontrollers. This mode configures timer 0 as a 13-bit timer which consists of all

8 bits of TH0 and the lower 5 bits of TL0. As a result, the Timer 0 uses only 13 of 16 bits. How does it operate? Each coming pulse causes the lower register bits to change their states. After receiving 32 pulses, this register is loaded and automatically cleared, while the higher byte (TH0) is incremented by 1. This process is repeated until registers count up 8192 pulses. After that, both registers are cleared and counting starts from 0.

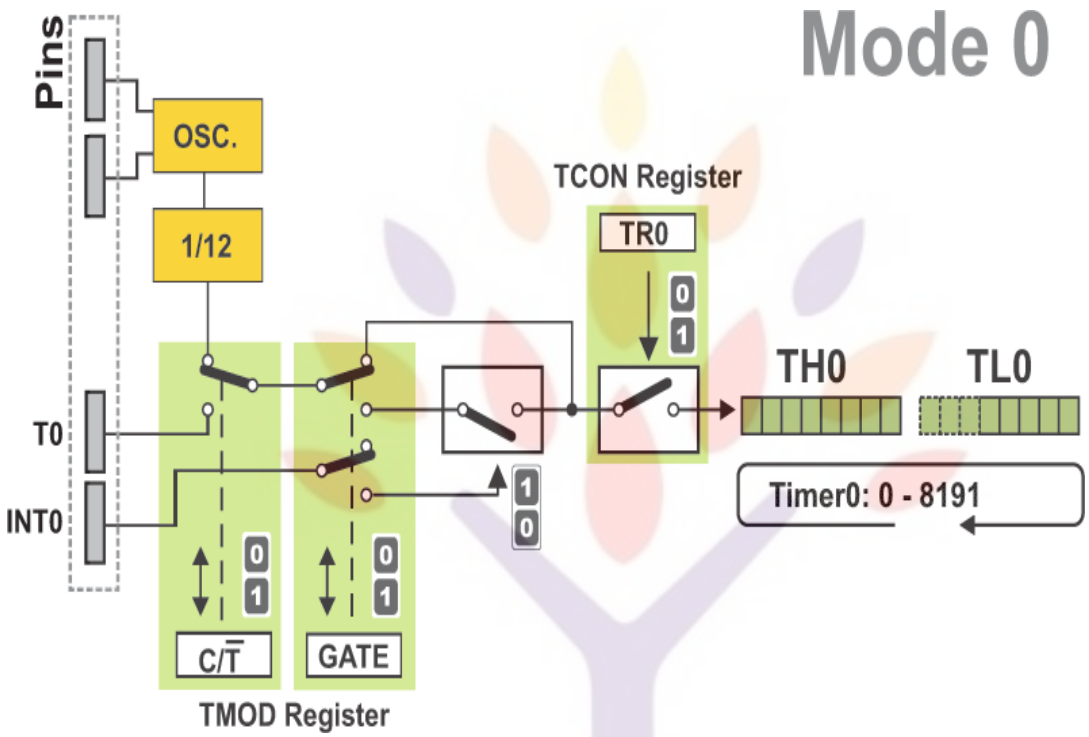


fig 7.10. Timer-0 in Mode -0

Timer 0 in mode 1 (16-bit timer)

Mode 1 configures timer 0 as a 16-bit timer comprising all the bits of both registers TH0 and TL0. That's why this is one of the most commonly used modes. Timer operates in the same way as in mode 0, with difference that the registers count up to 65 536 as allowable by the 16 bits.

NRCM
your roots to success...

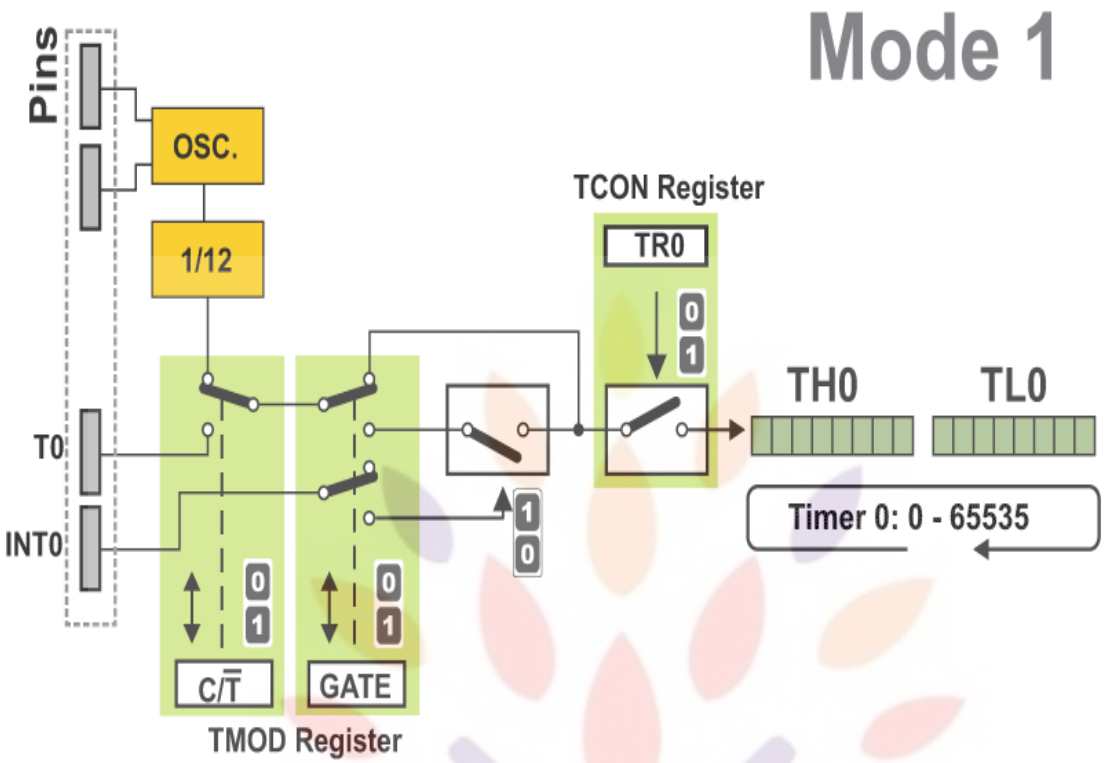


Fig 7.11. Timer -0 in Mode -1

Timer 0 in mode 2 (Auto-Reload Timer)

Mode 2 configures timer 0 as an 8-bit timer. Actually, timer 0 uses only one 8-bit register for counting and never counts from 0, but from an arbitrary value (0-255) stored in another (TH0) register.The following example shows the advantages of this mode. Suppose it is necessary to constantly count up 55 pulses generated by the clock.

If mode 1 or mode 0 is used, It is necessary to write the number 200 to the timer registers and constantly check whether an overflow has occured, i.e. whether they reached the value 255. When it happens, it is necessary to rewrite the number 200 and repeat the whole procedure. The same procedure is automatically performed by the microcontroller if set in mode 2. In fact, only the TL0 register operates as a timer, while another (TH0) register stores the value from which the counting starts. When the TL0 register is loaded, instead of being cleared, the contents of TH0 will be reloaded to it. Referring to the previous example, in order to register each 55th pulse, the best solution is to write the number 200 to the TH0 register and configure the timer to operate in mode 2.

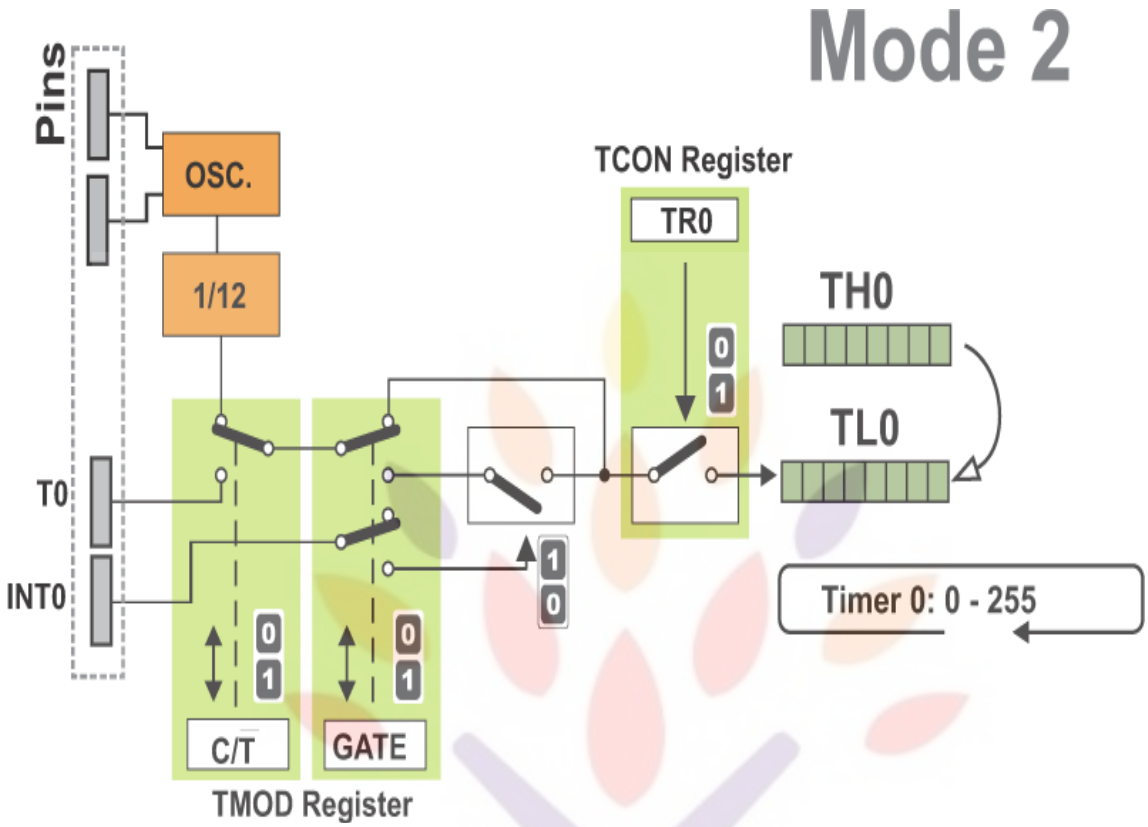


Fig 7.12. Timer-0 in Auto Reload Mode

Timer 0 in Mode 3 (Split Timer)

Mode 3 configures timer 0 so that registers TL0 and TH0 operate as separate 8-bit timers. In other words, the 16-bit timer consisting of two registers TH0 and TL0 is split into two independent 8-bit timers. This mode is provided for applications requiring an additional 8-bit timer or counter. The TL0 timer turns into timer 0, while the TH0 timer turns into timer 1. In addition, all the control bits of 16-bit Timer 1 (consisting of the TH1 and TL1 register), now control the 8-bit Timer 1. Even though the 16-bit Timer 1 can still be configured to operate in any of modes (mode 1, 2 or 3), it is no longer possible to disable it as there is no control bit to do it. Thus, its operation is restricted when timer 0 is in mode 3.

NROM
your roots to success...

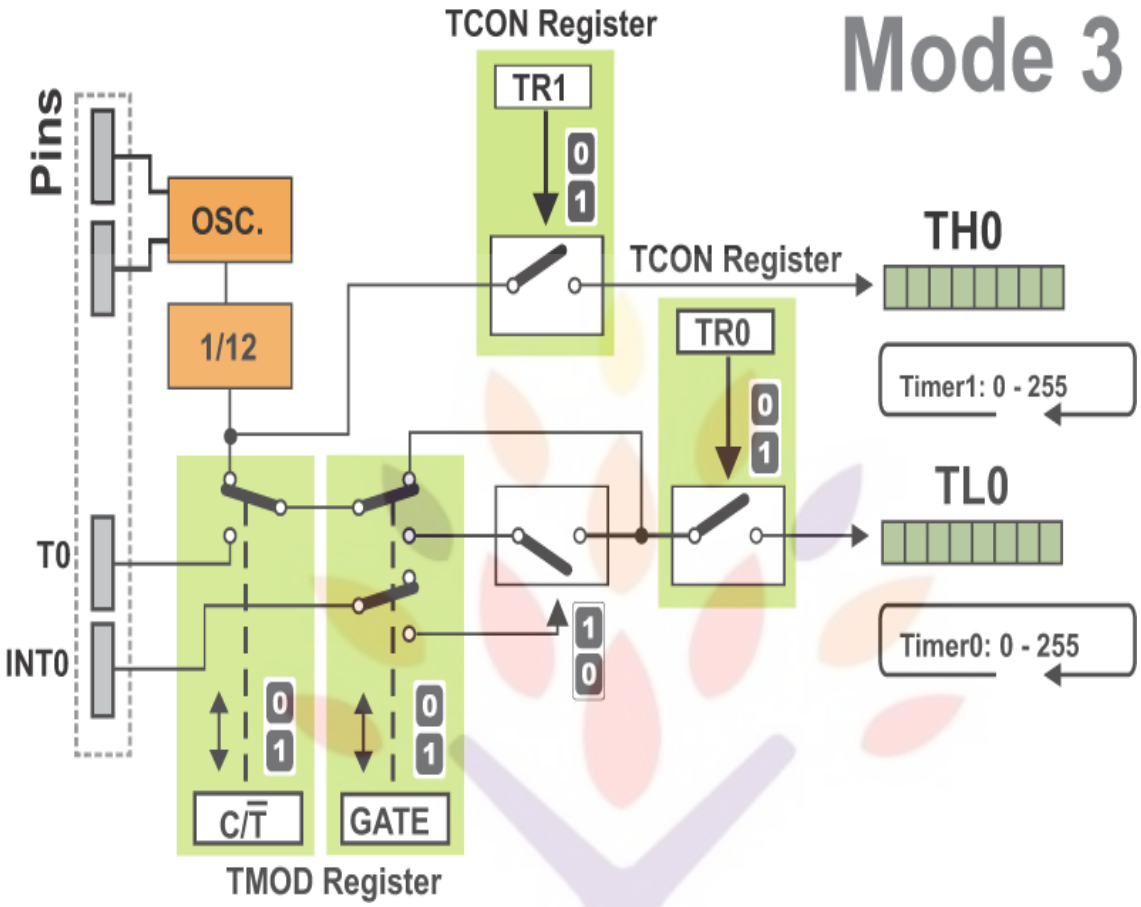


Fig 7.13. Timer -0 in Split Timer Mode

The only application of this mode is when two timers are used and the 16-bit Timer 1 the operation of which is out of control is used as a baud rate generator.

Timer Control (TCON) Register

TCON register is also one of the registers whose bits are directly in control of timer operation. Only 4 bits of this register are used for this purpose, while rest of them is used for interrupt control to be discussed later.

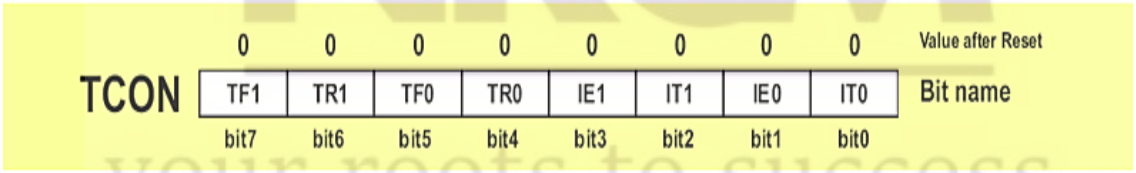


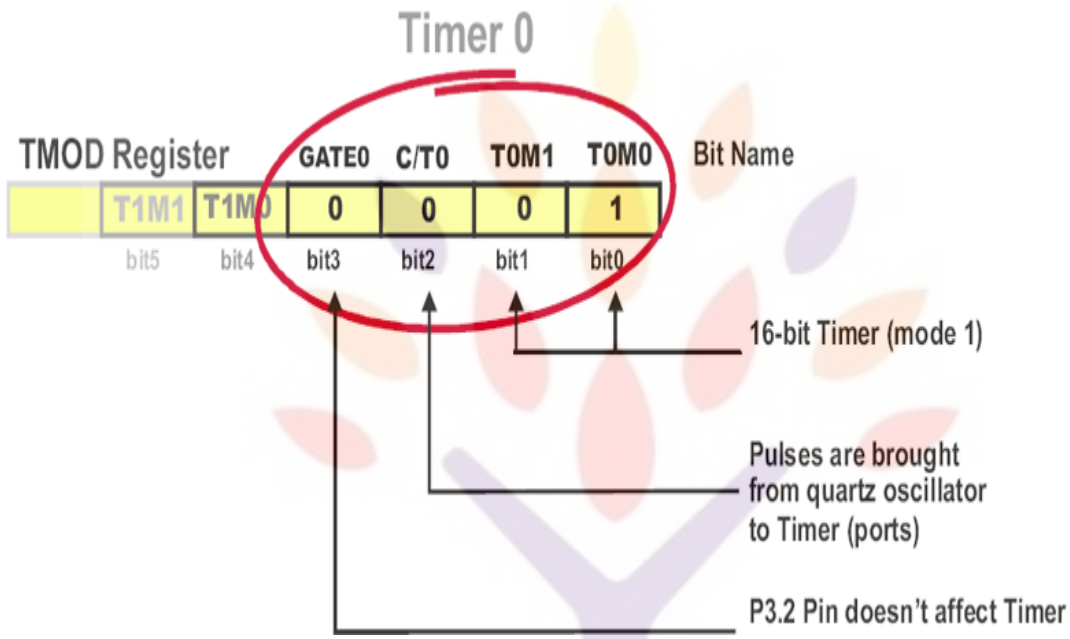
Fig 7.14. 8051 TCON Register

- **TF1** bit is automatically set on the Timer 1 overflow.
- **TR1** bit enables the Timer 1.
  - **1** - Timer 1 is enabled.
  - **0** - Timer 1 is disabled.
- **TF0** bit is automatically set on the Timer 0 overflow.
- **TR0** bit enables the timer 0.

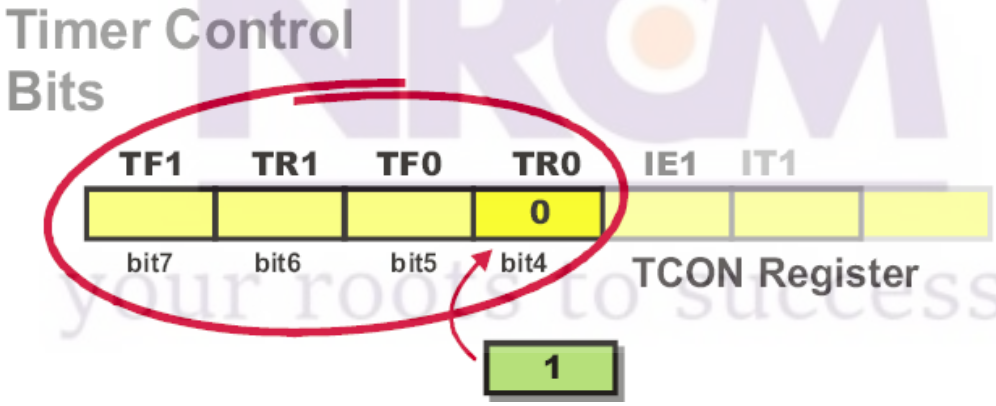
- **1** - Timer 0 is enabled.
- **0** - Timer 0 is disabled.

How to use the Timer 0 ?

In order to use timer 0, it is first necessary to select it and configure the mode of its operation. Bits of the TMOD register are in control of it:



Referring to figure above, the timer 0 operates in mode 1 and counts pulses generated by internal clock the frequency of which is equal to 1/12 the quartz frequency. Turn on the timer:



The TR0 bit is set and the timer starts operation. If the quartz crystal with frequency of 12MHz is embedded then its contents will be incremented every microsecond. After 65.536 microseconds,

the both registers the timer consists of will be loaded. The microcontroller automatically clears them and the timer keeps on repeating procedure from the beginning until the TR0 bit value is logic zero (0).

How to 'read' a timer?

Depending on application, it is necessary either to read a number stored in the timer registers or to register the moment they have been cleared.

- It is extremely simple to read a timer by using only one register configured in mode 2 or 3. It is sufficient to read its state at any moment. That's all!

- It is somehow complicated to read a timer configured to operate in mode 2. Suppose the lower byte is read first (TL0), then the higher byte (TH0). The result is:

TH0 = 15 TL0 = 255

Everything seems to be ok, but the current state of the register at the moment of reading was:

TH0 = 14 TL0 = 255

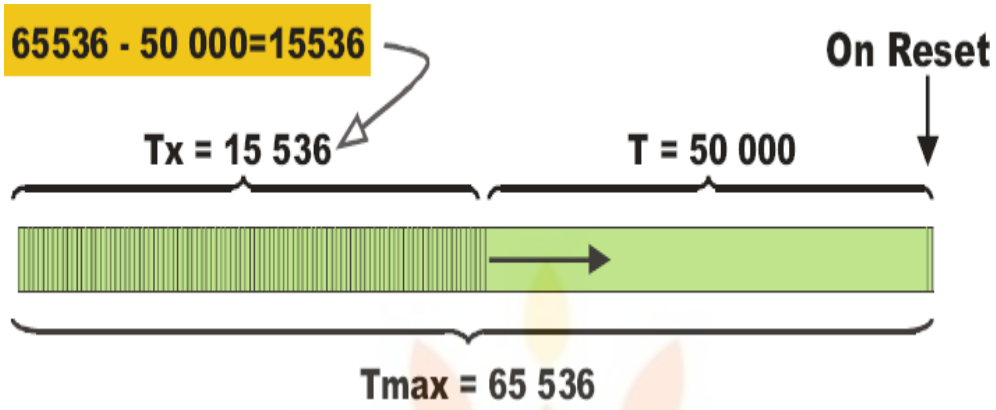
In case of negligence, such an error in counting (255 pulses) may occur for not so obvious but quite logical reason. The lower byte is correctly read (255), but at the moment the program counter was about to read the higher byte TH0, an overflow occurred and the contents of both registers have been changed (TH0: 14→15, TL0: 255→0). This problem has a simple solution. The higher byte should be read first, then the lower byte and once again the higher byte. If the number stored in the higher byte is different then this sequence should be repeated. It's about a short loop consisting of only 3 instructions in the program.

There is another solution as well. It is sufficient to simply turn the timer off while reading is going on (the TR0 bit of the TCON register should be cleared), and turn it on again after reading is finished.

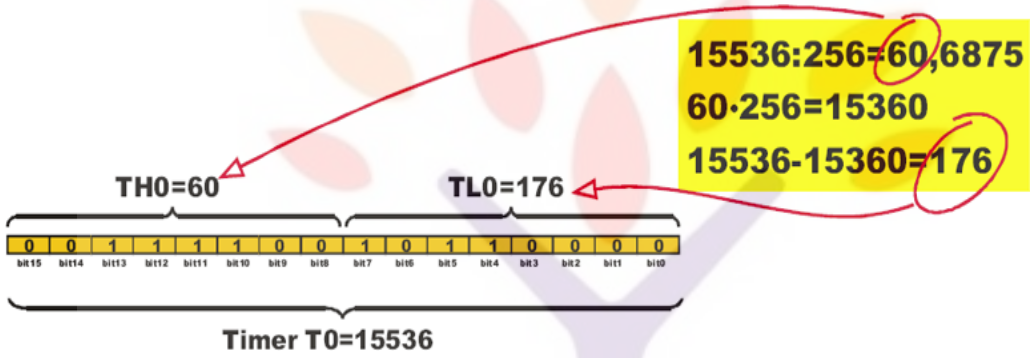
Timer 0 Overflow Detection

Usually, there is no need to constantly read timer registers. It is sufficient to register the moment they are cleared, i.e. when counting starts from 0. This condition is called an overflow. When it occurrs, the TF0 bit of the TCON register will be automatically set. The state of this bit can be constantly checked from within the program or by enabling an interrupt which will stop the main program execution when this bit is set. Suppose it is necessary to provide a program delay of 0.05 seconds (50 000 machine cycles), i.e. time when the program seems to be stopped:

First a number to be written to the timer registers should be calculated:



Then it should be written to the timer registers TH0 and TL0:



When enabled, the timer will resume counting from this number. The state of the TF0 bit, i.e. whether it is set, is checked from within the program. It happens at the moment of overflow, i.e. after exactly 50.000 machine cycles or 0.05 seconds.

How to measure pulse duration?

NRCM
your roots to success...

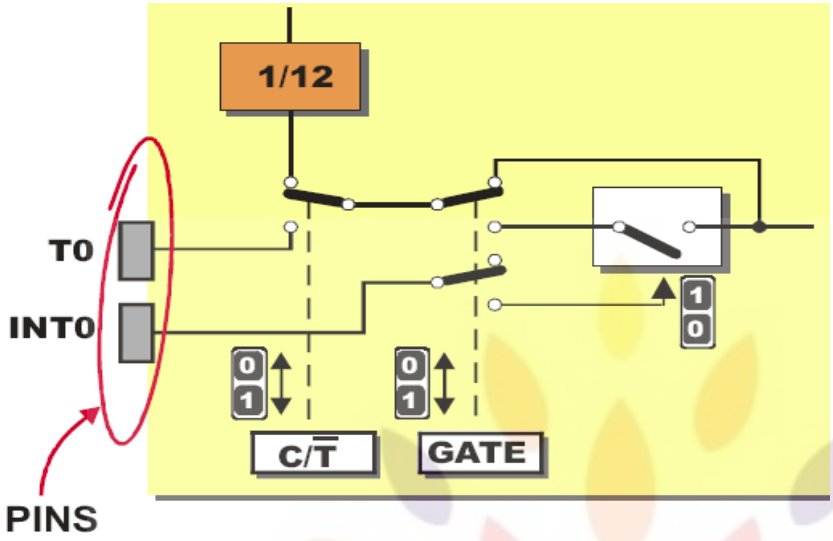


Fig 7.14. Internal operation of the timer

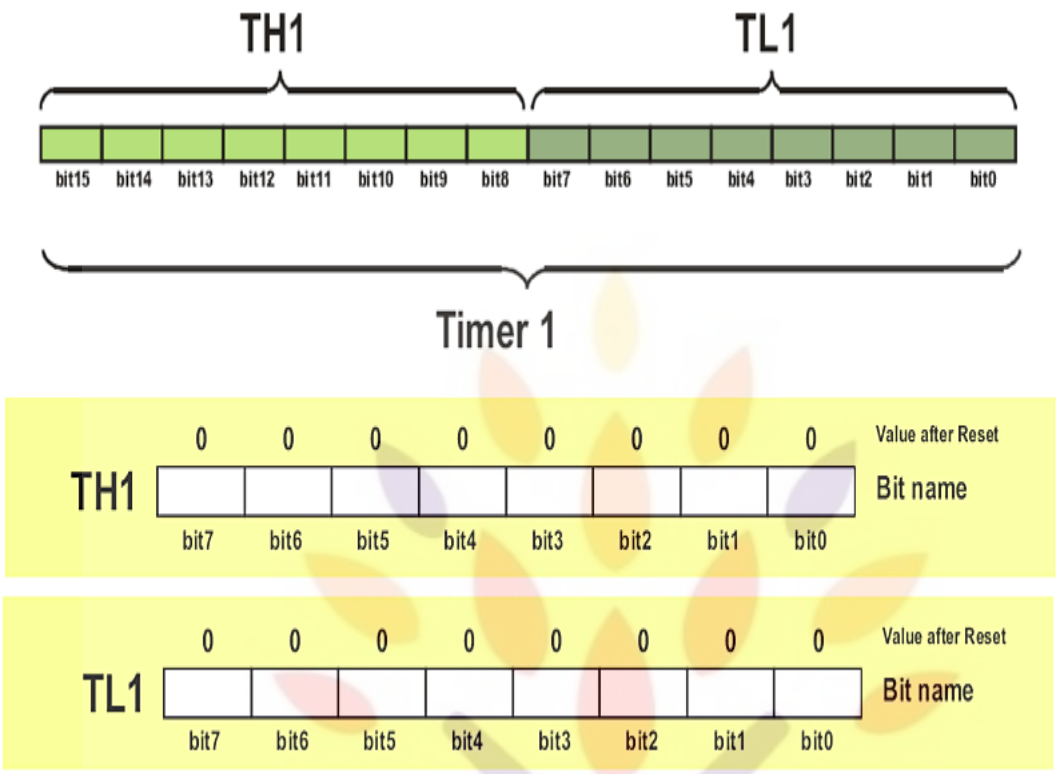
Suppose it is necessary to measure the duration of an operation, for example how long a device has been turned on? Look again at the figure illustrating the timer and pay attention to the function of the GATE0 bit of the TMOD register. If it is cleared then the state of the P3.2 pin doesn't affect timer operation. If GATE0 = 1 the timer will operate until the pin P3.2 is cleared. Accordingly, if this pin is supplied with 5V through some external switch at the moment the device is being turned on, the timer will measure duration of its operation, which actually was the objective.

How to count up pulses?

Similarly to the previous example, the answer to this question again lies in the TCON register. This time it's about the C/T0 bit. If the bit is cleared the timer counts pulses generated by the internal oscillator, i.e. measures the time passed. If the bit is set, the timer input is provided with pulses from the P3.4 pin (T0). Since these pulses are not always of the same width, the timer cannot be used for time measurement and is turned into a counter, therefore. The highest frequency that could be measured by such a counter is 1/24 frequency of used quartz-crystal.

Timer 1

Timer 1 is identical to timer 0, except for mode 3 which is a hold-count mode. It means that they have the same function, their operation is controlled by the same registers TMOD and TCON and both of them can operate in one out of 4 different modes.



7.3. Serial Communication

One of the microcontroller features making it so powerful is an integrated UART, better known as a serial port. It is a full-duplex port, thus being able to transmit and receive data simultaneously and at different baud rates. Without it, serial data send and receive would be an enormously complicated part of the program in which the pin state is constantly changed and checked at regular intervals. When using UART, all the programmer has to do is to simply select serial port mode and baud rate. When it's done, serial data transmit is nothing but writing to the SBUF register, while data receive represents reading the same register. The microcontroller takes care of not making any error during data transmission.

Fig 7.15. 8051 SBUF register

Serial port must be configured prior to being used. In other words, it is necessary to determine how many bits is contained in one serial "word", baud rate and synchronization clock source. The whole process is in control of the bits of the SCON register (Serial Control).

Serial Port Control (SCON) Register

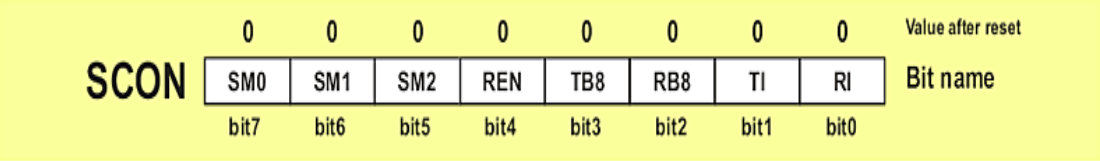


Fig 7.16. Serial Control Register

- **SM0** - Serial port mode bit 0 is used for serial port mode selection.
- **SM1** - Serial port mode bit 1.
- **SM2** - Serial port mode 2 bit, also known as multiprocessor communication enable bit. When set, it enables multiprocessor communication in mode 2 and 3, and eventually mode 1. It should be cleared in mode 0.
- **REN** - Reception Enable bit enables serial reception when set. When cleared, serial reception is disabled.
- **TB8** - Transmitter bit 8. Since all registers are 8-bit wide, this bit solves the problem of transmiting the 9th bit in modes 2 and 3. It is set to transmit a logic 1 in the 9th bit.
- **RB8** - Receiver bit 8 or the 9th bit received in modes 2 and 3. Cleared by hardware if 9th bit received is a logic 0. Set by hardware if 9th bit received is a logic 1.
- **TI** - Transmit Interrupt flag is automatically set at the moment the last bit of one byte is sent. It's a signal to the processor that the line is available for a new byte transmite. It must be cleared from within the software.
- **RI** - Receive Interrupt flag is automatically set upon one byte receive. It signals that byte is received and should be read quickly prior to being replaced by a new data. This bit is also cleared from within the software.

SM0	SM1	MODE	DESCRIPTION	BAUD RATE
0	0	0	8-bit Shift Register	1/12 the quartz frequency
0	1	1	8-bit UART	Determined by the timer 1
1	0	2	9-bit UART	1/32 the quartz frequency (1/64 the quartz frequency)
1	1	3	9-bit UART	Determined by the timer 1

Table 7.3. Serial communication mode selection

NRCM your roots to success...

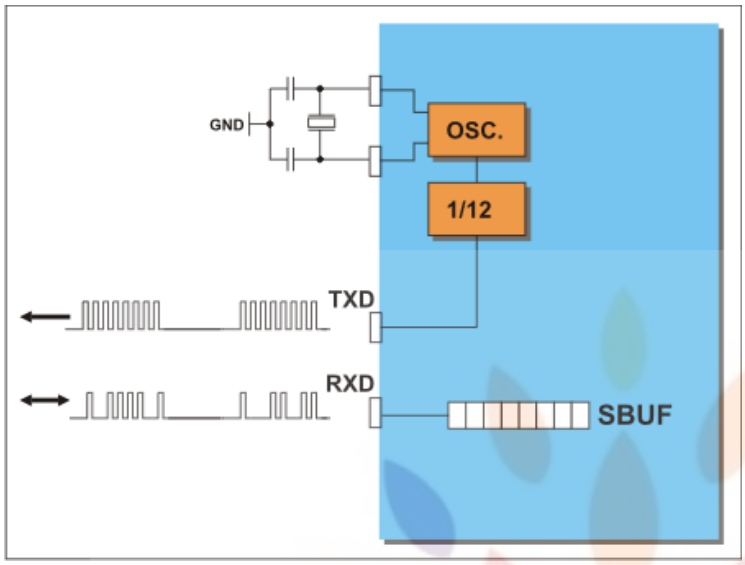
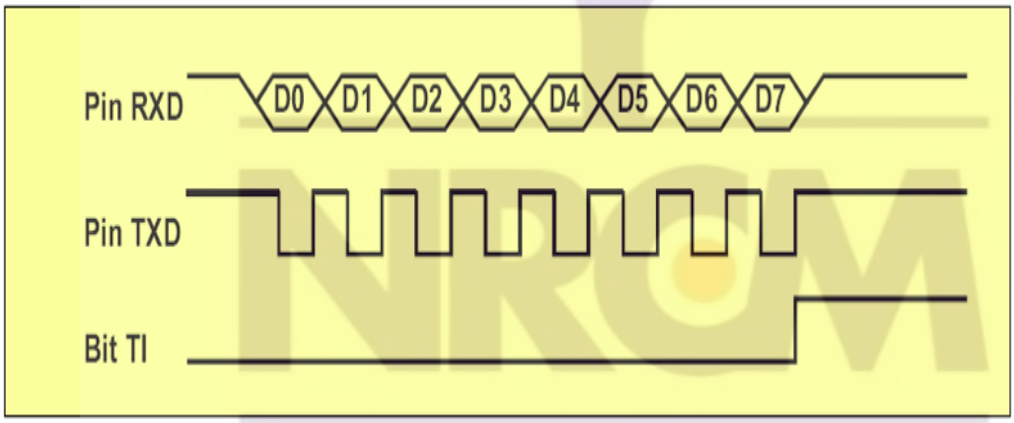


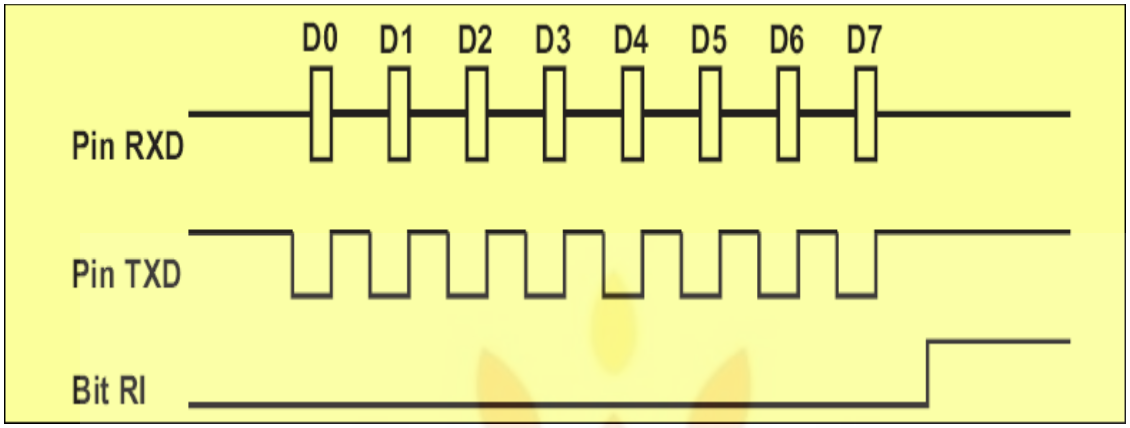
Fig 7.16. Serial communication overview

In mode 0, serial data are transmitted and received through the RXD pin, while the TXD pin output clocks. The bout rate is fixed at 1/12 the oscillator frequency. On transmit, the least significant bit (LSB bit) is sent/received first.

TRANSMIT - Data transmit is initiated by writing data to the SBUF register. In fact, this process starts after any instruction being performed upon this register. When all 8 bits have been sent, the TI bit of the SCON register is automatically set.



RECEIVE - Data receive through the RXD pin starts upon the two following conditions are met: bit REN=1 and RI=0 (both of them are stored in the SCON register). When all 8 bits have been received, the RI bit of the SCON register is automatically set indicating that one byte receives is complete.



Since there are no START and STOP bits or any other bit except data sent from the SBUF register in the pulse sequence, this mode is mainly used when the distance between devices is short, noise is minimized and operating speed is of importance. A typical example is I/O port expansion by adding a cheap IC (shift registers 74HC595, 74HC597 and similar).

Mode 1

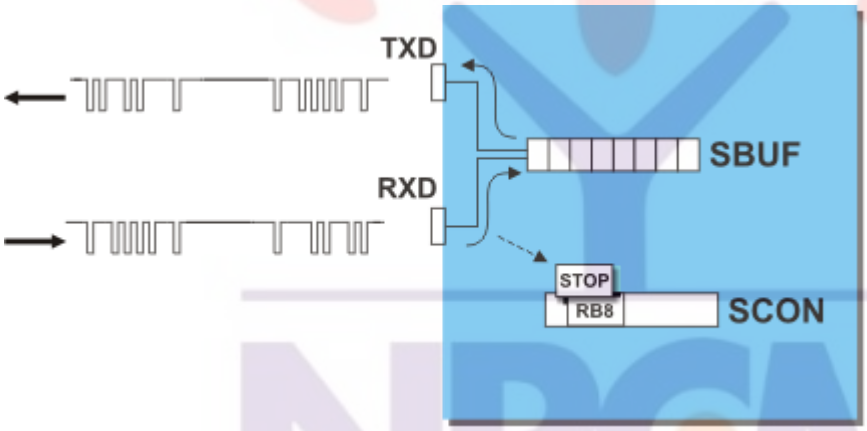
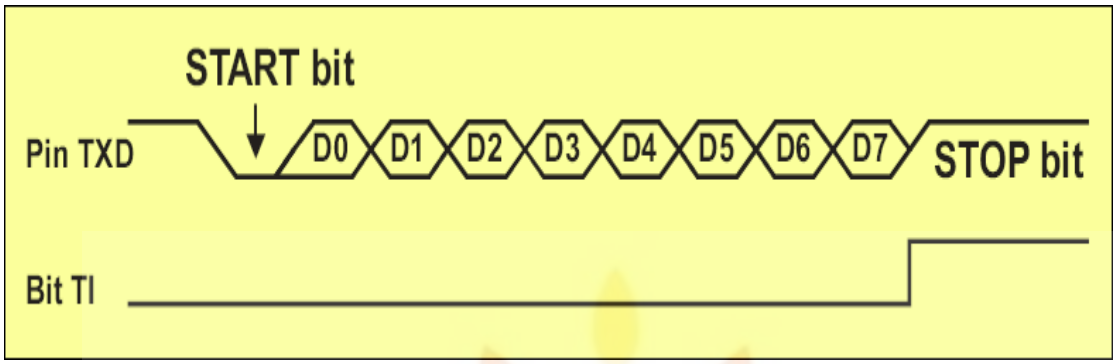


Fig 7.17. Serial communication in mode1

In mode 1, 10 bits are transmitted through the TXD pin or received through the RXD pin in the following manner: a START bit (always 0), 8 data bits (LSB first) and a STOP bit (always 1). The START bit is only used to initiate data receive, while the STOP bit is automatically written to the RB8 bit of the SCON register.

TRANSMIT - Data transmit is initiated by writing data to the SBUF register. End of data transmission is indicated by setting the TI bit of the SCON register.



RECEIVE - The START bit (logic zero (0)) on the RXD pin initiates data receive. The following two conditions must be met: bit REN=1 and bit RI=0. Both of them are stored in the SCON register. The RI bit is automatically set upon data reception is complete.

The Baud rate in this mode is determined by the timer 1 overflow.

Mode 2

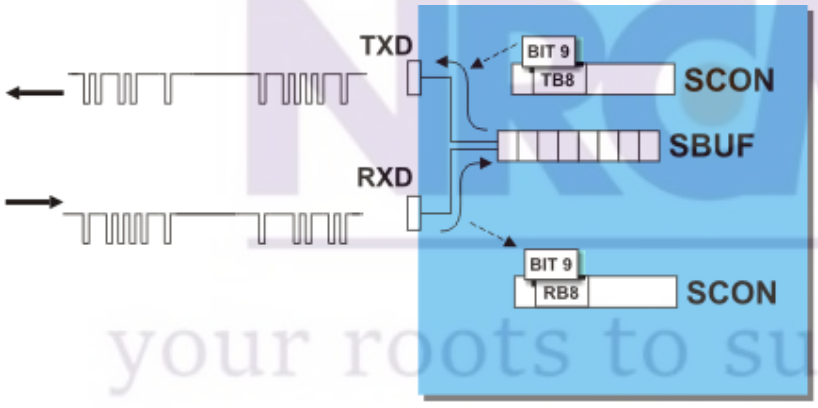
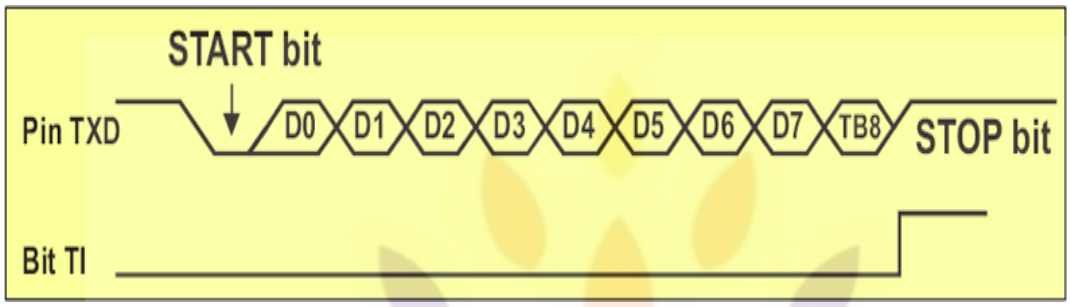


Fig 7.18. Serial communication in mode 2

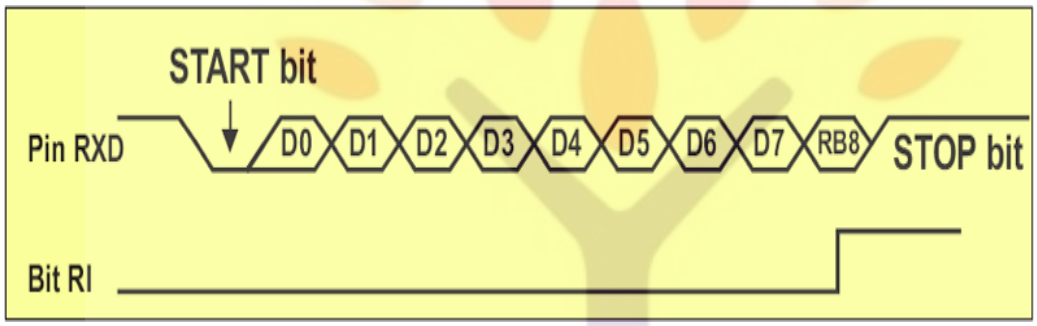
In mode 2, 11 bits are transmitted through the TXD pin or received through the RXD pin: a START bit (always 0), 8 data bits (LSB first), a programmable 9th data bit and a STOP bit (always 1). On transmit, the 9th data bit is actually the TB8 bit of the SCON register. This bit

usually has a function of parity bit. On receive, the 9th data bit goes into the RB8 bit of the same register (SCON).The baud rate is either 1/32 or 1/64 the oscillator frequency.

TRANSMIT - Data transmit is initiated by writing data to the SBUF register. End of data transmission is indicated by setting the TI bit of the SCON register.



RECEIVE - The START bit (logic zero (0)) on the RXD pin initiates data receive. The following two conditions must be met: bit REN=1 and bit RI=0. Both of them are stored in the SCON register. The RI bit is automatically set upon data reception is complete.



Mode 3

Mode 3 is the same as Mode 2 in all respects except the baud rate. The baud rate in Mode 3 is variable.

Baud Rate

Baud Rate is a number of sent/received bits per second. In case the UART is used, baud rate depends on: selected mode, oscillator frequency and in some cases on the state of the SMD bit of the SCON register. All the nec

	BAUD RATE	BITSMOD
Mode 0	Fosc. / 12	
Mode 1	1 Fosc. 16 12 (256-TH1)	BitSMOD
Mode 2	Fosc. / 32 Fosc. / 64	1 0
Mode 3	1 Fosc. 16 12 (256-TH1)	

your roots to success...

essary formulas are specified in the table:

Timer 1 as a clock generator

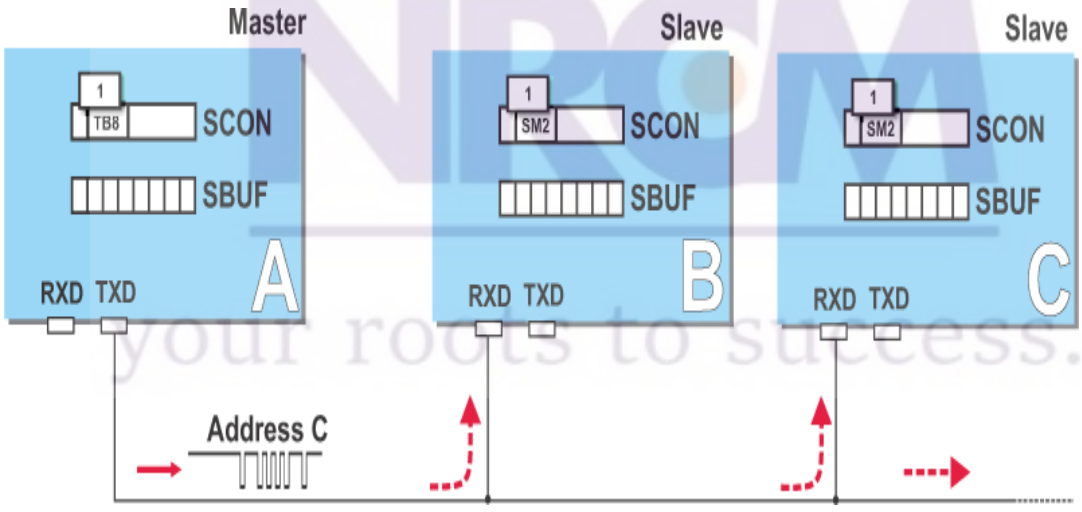
Timer 1 is usually used as a clock generator as it enables various baud rates to be easily set. The whole procedure is simple and is as follows:

- First, enable Timer 1 overflow interrupt.
- Configure Timer T1 to operate in auto-reload mode.
- Depending on needs, select one of the standard values from the table and write it to the TH1 register. That's all.

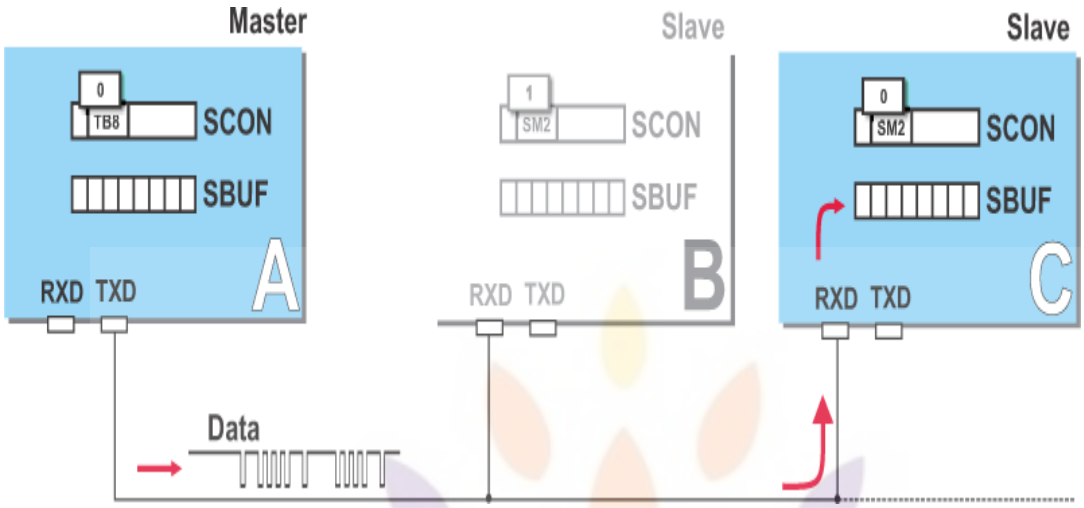
Multiprocessor Communication

As you may know, additional 9th data bit is a part of message in mode 2 and 3. It can be used for checking data via parity bit. Another useful application of this bit is in communication between two or more microcontrollers, i.e. multiprocessor communication. This feature is enabled by setting the SM2 bit of the SCON register. As a result, after receiving the STOP bit, indicating end of the message, the serial port interrupt will be generated only if the bit RB8 = 1 (the 9th bit).

Suppose there are several microcontrollers sharing the same interface. Each of them has its own address. An address byte differs from a data byte because it has the 9th bit set (1), while this bit is cleared (0) in a data byte. When the microcontroller A (master) wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte will generate an interrupt in all slaves so that they can examine the received byte and check whether it matches their address.



Of course, only one of them will match the address and immediately clear the SM2 bit of the SCON register and prepare to receive the data byte to come. Other slaves not being addressed leave their SM2 bit set ignoring the coming data bytes.



7.4. 8051 Interrupts

There are five interrupt sources for the 8051, which means that they can recognize 5 different events that can interrupt regular program execution. Each interrupt can be enabled or disabled by setting bits of the IE register. Likewise, the whole interrupt system can be disabled by clearing the EA bit of the same register. Refer to figure below.

Now, it is necessary to explain a few details referring to external interrupts- INT0 and INT1. If the IT0 and IT1 bits of the TCON register are set, an interrupt will be generated on high to low transition, i.e. on the falling pulse edge (only in that moment). If these bits are cleared, an interrupt will be continuously executed as far as the pins are held low.

NRCM
your roots to success...

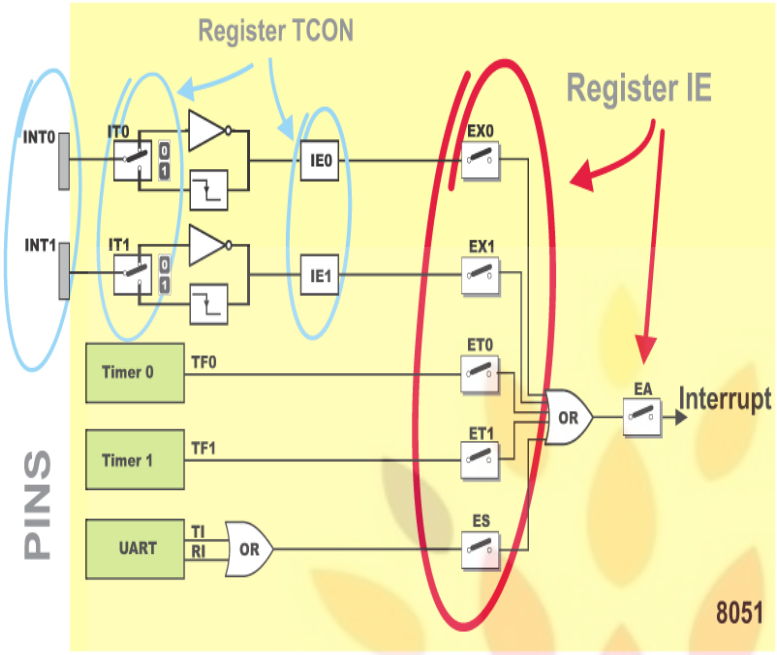


Fig 7.19. Interrupt sources of 8051

IE Register
(InterruptEnable)

Fig 7.20. IE register of 8051

- **EA** - global interrupt enable/disable:
  - 0 - disables all interrupt requests.
  - 1 - enables all individual interrupt requests.
- **ES** - enables or disables serial interrupt:
  - 0 - UART system cannot generate an interrupt.
  - 1 - UART system enables an interrupt.
- **ET1** - bit enables or disables Timer 1 interrupt:
  - 0 - Timer 1 cannot generate an interrupt.
  - 1 - Timer 1 enables an interrupt.
- **EX1** - bit enables or disables external 1 interrupt:
  - 0 - change of the pin INT0 logic state cannot generate an interrupt.
  - 1 - enables an external interrupt on the pin INT0 state change.
- **ET0** - bit enables or disables timer 0 interrupt:
  - 0 - Timer 0 cannot generate an interrupt.
  - 1 - enables timer 0 interrupt.

- **EX0** - bit enables or disables external 0 interrupt:
  - 0 - change of the INT1 pin logic state cannot generate an interrupt.
  - 1 - enables an external interrupt on the pin INT1 state change.

Interrupt Priorities

It is not possible to forseen when an interrupt request will arrive. If several interrupts are enabled, it may happen that while one of them is in progress, another one is requested. In order that the microcontroller knows whether to continue operation or meet a new interrupt request, there is a priority list instructing it what to do.

The priority list offers 3 levels of interrupt priority:

1. Reset! The absolute masterInterrupt priority 1 can be disabled by Reset only.
2. Interrupt priority 0 can be disabled by both Reset and interrupt priority 1.

The IP Register (Interrupt Priority Register) specifies which one of existing interrupt sources have higher and which one has lower priority. Interrupt priority is usually specified at the beginning of the program. According to that, there are several possibilities:

- If an interrupt of higher priority arrives while an interrupt is in progress, it will be immediately stopped and the higher priority interrupt will be executed first.
- If two interrupt requests, at different priority levels, arrive at the same time then the higher priority interrupt is serviced first.
- If the both interrupt requests, at the same priority level, occur one after another, the one which came later has to wait until routine being in progress ends.
- If two interrupt requests of equal priority arrive at the same time then the interrupt to be serviced is selected according to the following priority list:
  1. External interrupt INT0
  2. Timer 0 interrupt
  3. External Interrupt INT1
  4. Timer 1 interrupt
  5. Serial Communication Interrupt

IP Register (Interrupt Priority)

The IP register bits specify the priority level of each interrupt (high or low priority).

your roots to success...

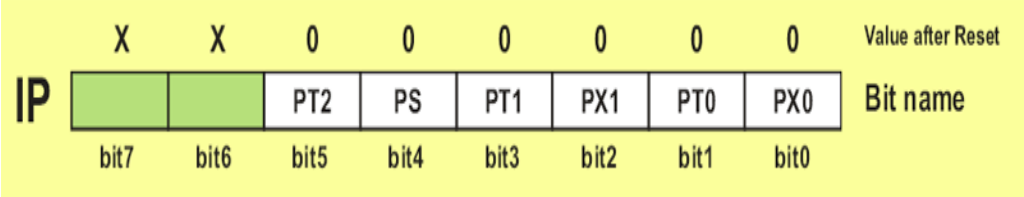


Fig 7.21. IP register of 8051

- **PS** - Serial Port Interrupt priority bit
  - Priority 0
  - Priority 1
- **PT1** - Timer 1 interrupt priority
  - Priority 0
  - Priority 1
- **PX1** - External Interrupt INT1 priority
  - Priority 0
  - Priority 1
- **PT0** - Timer 0 Interrupt Priority
  - Priority 0
  - Priority 1
- **PX0** - External Interrupt INT0 Priority
  - Priority 0
  - Priority 1

Handling Interrupt

When an interrupt request arrives the following occurs:

- Instruction in progress is ended.
- The address of the next instruction to execute is pushed on the stack.
- Depending on which interrupt is requested, one of 5 vectors (addresses) is written to the program counter in accordance to the table below:

INTERRUPT SOURCE	VECTOR (ADDRESS)
IE0	3 h
TF0	B h
TF1	1B h
RI, TI	23 h

NRCM your roots to success...

All addresses are in hexadecimal format

Table 7.4. vector Addresses of 8051 Interrupts

- These addresses store appropriate subroutines processing interrupts. Instead of them, there are usually jump instructions specifying locations on which these subroutines reside.
- When an interrupt routine is executed, the address of the next instruction to execute is popped from the stack to the program counter and interrupted program resumes operation from where it left off.

Reset

Reset occurs when the RS pin is supplied with a positive pulse in duration of at least 2 machine cycles (24 clock cycles of crystal oscillator). After that, the microcontroller generates an internal reset signal which clears all SFRs, except SBUF registers, Stack Pointer and ports (the state of the first two ports is not defined, while FF value is written to the ports configuring all their pins as inputs). Depending on surrounding and purpose of device, the RS pin is usually connected to a power-on reset push button or circuit or to both of them. Figure below illustrates one of the simplest circuit providing safe power-on reset.

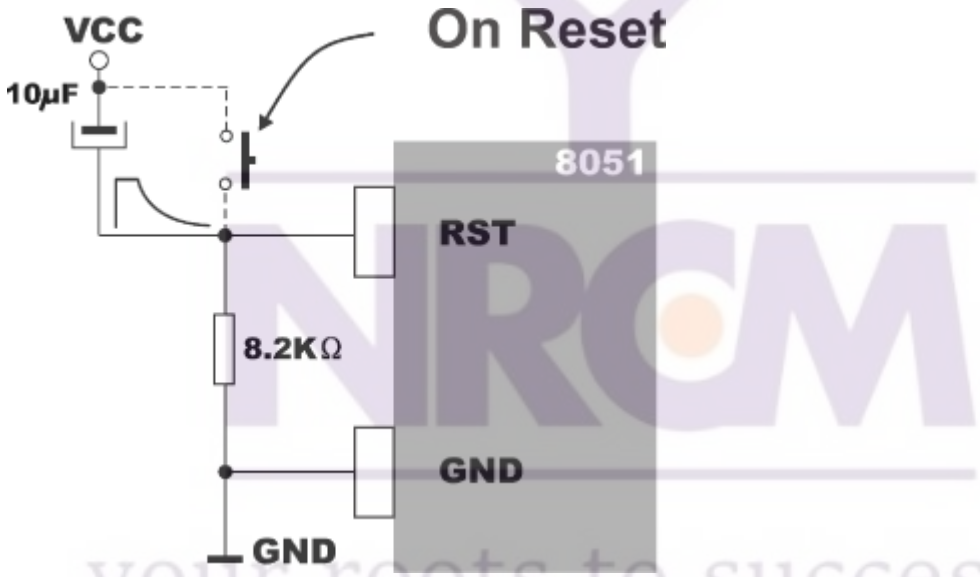


Fig 7.22. 8051 Reset

Basically, everything is very simple: after turning the power on, electrical capacitor is being charged for several milliseconds throgh a resistor connected to the ground. The pin is driven high

during this process. When the capacitor is charged, power supply voltage is already stable and the pin remains connected to the ground, thus providing normal operation of the microcontroller. Pressing the reset button causes the capacitor to be temporarily discharged and the microcontroller is reset. When released, the whole process is repeated…

Through the program- step by step…

Microcontrollers normally operate at very high speed. The use of 12 Mhz quartz crystal enables 1.00.00 instructions to be executed per second. Basically, there is no need for higher operating rate. In case it is needed, it is easy to built in a crystal for high frequency. The problem arises when it is necessary to slow down the operation of the microcontroller. For example during testing in real environment when it is necessary to execute several instructions step by step in order to check I/O pins' logic state.

Interrupt system of the 8051 microcontroller practically stops operation of the microcontroller and enables instructions to be executed one after another by pressing the button. Two interrupt features enable that:

- Interrupt request is ignored if an interrupt of the same priority level is in progress.
- Upon interrupt routine execution, a new interrupt is not executed until at least one instruction from the main program is executed.

In order to use this in practice, the following steps should be done:

- External interrupt sensitive to the signal level should be enabled (for example INT0).
- Three following instructions should be inserted into the program (at the 03hex. address):

```
JNB    P3.2$   ← Means: wait here until the pin P3.2 (INT0) is set to "1".
JB     P3.2$   ← Means: wait here until the pin P3.2 (INT0) is set to "0".
RETI           ← Means: go back to the main program
```

As soon as the P3.2 pin is cleared (for example, by pressing the button), the microcontroller will stop program execution and jump to the 03hex address will be executed. This address stores a short interrupt routine consisting of 3 instructions.

The first instruction is executed until the push button is realised (logic one (1) on the P3.2 pin). The second instruction is executed until the push button is pressed again. Immediately after that, the RETI instruction is executed and the processor resumes operation of the main program. Upon execution of any program instruction, the interrupt INT0 is generated and the whole procedure is repeated (push button is still pressed). In other words, one button press - one instruction.

2.9 8051 Microcontroller Power Consumption Control

Generally speaking, the microcontroller is inactive for the most part and just waits for some external signal in order to takes its role in a show. This can cause some problems in case batteries are used for power supply. In extreme cases, the only solution is to set the whole electronics in sleep mode in order to minimize consumption. A typical example is a TV remote controller: it can be out of use for months but when used again it takes less than a second to send a command to TV receiver. The AT89S53 uses approximately 25mA for regular operation, which doesn't make it a pover-saving microcontroller. Anyway, it doesn't have to be always like that, it can easily switch the operating mode in order to reduce its total consumption to approximately 40uA. Actually, there are two power-saving modes of operation:*Idle* and *Power Down.*

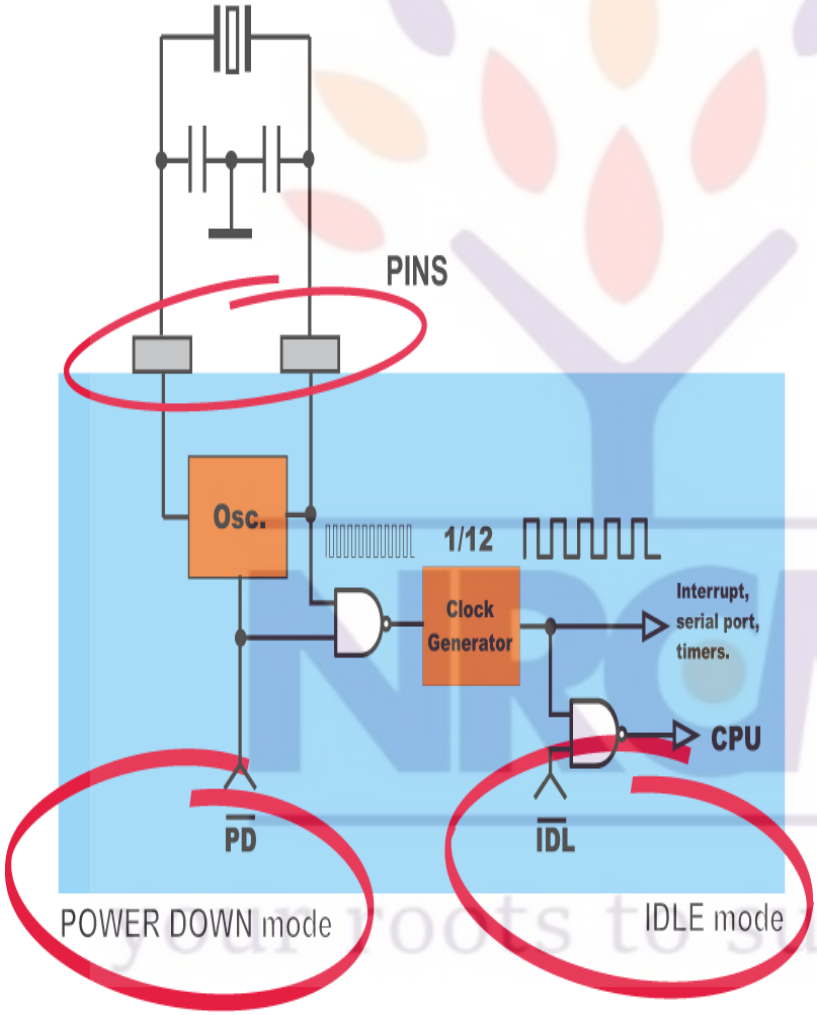


Fig 7.23. power down and idle modes of 8051

Idle mode

Upon the IDL bit of the PCON register is set, the microcontroller turns off the greatest power consumer- CPU unit while peripheral units such as serial port, timers and interrupt system continue operating normally consuming 6.5mA. In Idle mode, the state of all registers and I/O ports remains unchanged.

In order to exit the Idle mode and make the microcontroller operate normally, it is necessary to enable and execute any interrupt or reset. It will cause the IDL bit to be automatically cleared and the program resumes operation from instruction having set the IDL bit. It is recommended that first three instructions to execute now are NOP instructions. They don't perform any operation but provide some time for the microcontroller to stabilize and prevents undesired changes on the I/O ports.

Power Down mode

By setting the PD bit of the PCON register from within the program, the microcontroller is set to Power down mode, thus turning off its internal oscillator and reduces power consumption enormously. The microcontroller can operate using only 2V power supply in power- down mode, while a total power consumption is less than 40uA. The only way to get the microcontroller back to normal mode is by reset.

While the microcontroller is in Power Down mode, the state of all SFR registers and I/O ports remains unchanged. By setting it back into the normal mode, the contents of the SFR register is lost, but the content of internal RAM is saved. Reset signal must be long enough, approximately 10mS, to enable stable operation of the quartz oscillator.

PCON register

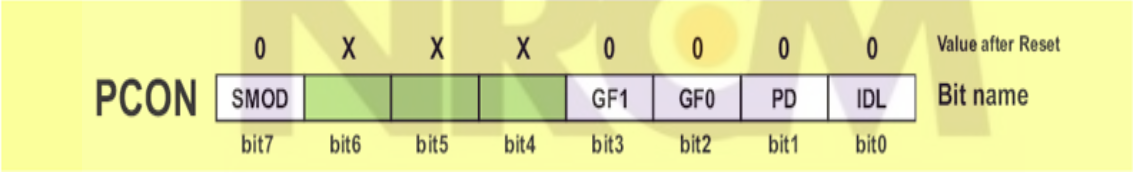


Fig 7.24. PCON register of 8051

your roots to success...

The purpose of the Register PCON bits is:

- SMOD Baud rate is twice as much higher by setting this bit.
- GF1 General-purpose bit (available for use).
- GF1 General-purpose bit (available for use).
- GF0 General-purpose bit (available for use).
- PD By setting this bit the microcontroller enters the *Power Down* mode.
- IDL By setting this bit the microcontroller enters the *Idle* mode.

CONTENTS BEYOND SYLLABUS

THE AVR RISC MICROCONTROLLER ARCHITECTURE

8.1. Introduction

Features

• Utilizes the AVR® RISC Architecture
• AVR – High-performance and Low-power RISC Architecture
– 118 Powerful Instructions – Most Single Clock Cycle Execution
– 32 x 8 General Purpose Working Registers
– Up to 10 MIPS Throughput at 10 MHz
• Data and Non-volatile Program Memory
– 2K Bytes of In-System Programmable FlashEndurance 1,000 Write/Erase Cycles
– 128 Bytes of SRAM
– 128 Bytes of In-System Programmable EEPROM
Endurance: 100,000 Write/Erase Cycles
– Programming Lock for Flash Program and EEPROM Data Security
• Peripheral Features
– One 8-bit Timer/Counter with Separate Prescalar
– One 16-bit Timer/Counter with Separate Prescalar,Compare, Capture Modes and 8-, 9-, or 10-bit PWM
– On-chip Analog Comparator
– Programmable Watchdog Timer with On-chip Oscillator
– SPI Serial Interface for In-System Programming
– FullDuplexUART
• • Special Microcontroller Features
– Low-power Idle and Power-down Modes
– External and Internal Interrupt Sources
• • Specifications
– Low-power, High-speed CMOS Process Technology
– Fully Static Operation
• Power Consumption at 4 MHz, 3V, 25°C
– Active: 2.8 mA
– Idle Mode: 0.8 mA
– Power-down Mode: <1 μA
• Operating Voltages
– 2.7 - 6.0V (AT90S2313-4)
– 4.0 - 6.0V (AT90S2313-10)
• Speed Grades

NRCM your roots to success...

– 0 - 4 MHz (AT90S2313-4)
– 0 - 10 MHz (AT90S2313-10)

8.2. Architecture of AT90S2313

The AT90S2313 is a low-power CMOS 8-bit microcontroller based on the AVR RISC architecture. By executing powerful instructions in a single clock cycle, the AT90S2313 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.The AVR core combines a rich instruction set with 32 general purpose working registers.All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowingtwo independent registers to be accessed in one single instruction executed in one clockcycle. The resulting architecture is more code efficient while achieving throughputs up toten times faster than conventional CISC microcontrollers.

NRCM
your roots to success...

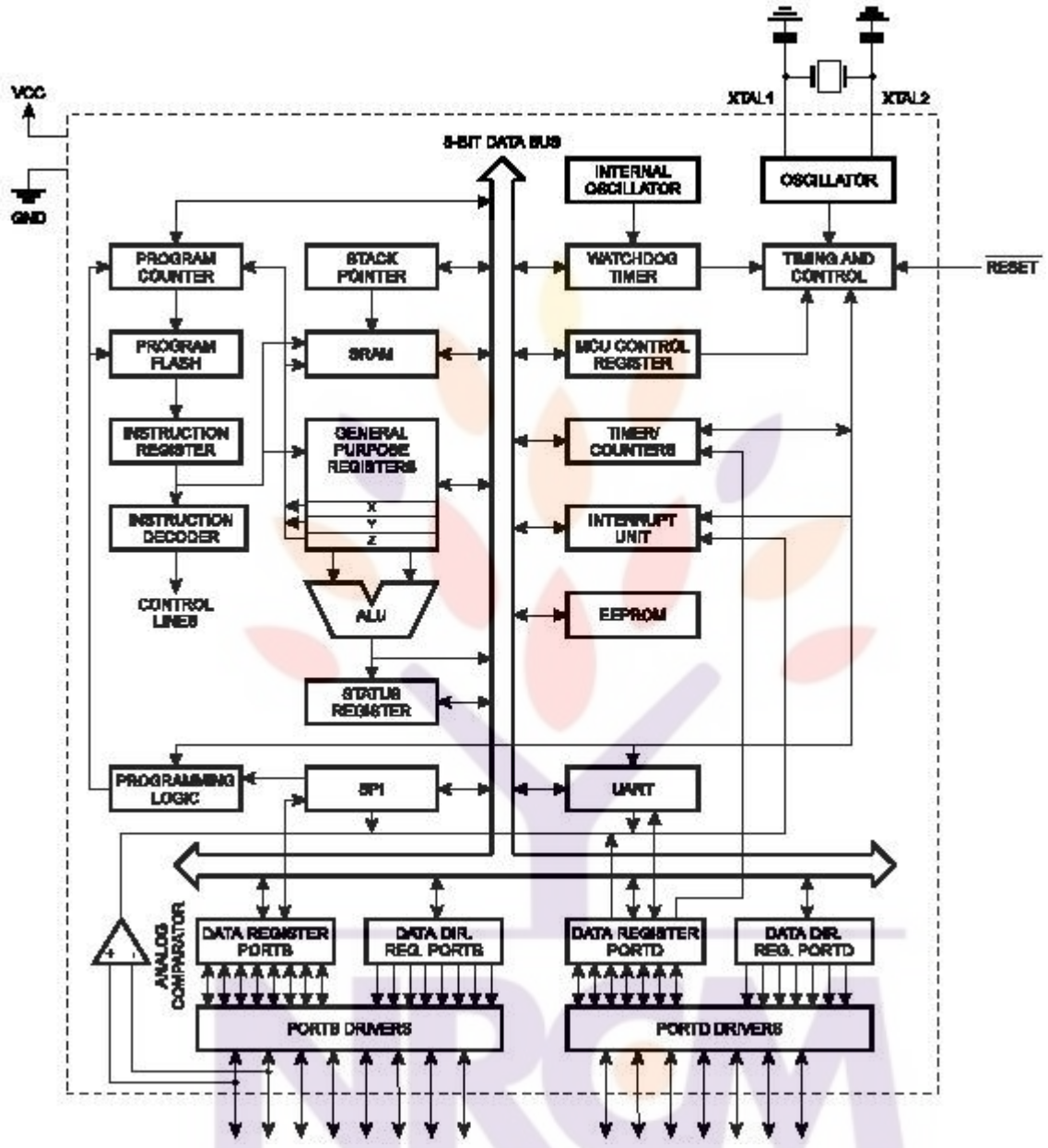


Fig 8.1. Architecture of AT90S2313

The AT90S2313 provides the following features: 2K bytes of In-System ProgrammableFlash, 128 bytes EEPROM, 128 bytes SRAM, 15 general purpose I/O lines, 32 generalpurpose working registers, flexible Timer/Counters with compare modes, internal andexternal interrupts, a programmable serial UART, programmable Watchdog Timer withinternal Oscillator, an SPI serial port for Flash memory downloading and two software selectable power-saving modes. The Idle mode stops the CPU while allowing theSRAM, Timer/Counters, SPI port and interrupt system to continue functioning. ThePower-down mode saves the register contents but freezes the Oscillator, disabling allother chip functions until the next external interrupt or Hardware Reset.
The device is manufactured using Atmel's high-density non-volatile memory technology.

The On-chip In-System Programmable Flash allows the Program memory to be reprogrammedin-system through an SPI serial interface or by a conventional non-volatilememory programmer. By combining an enhanced RISC 8-bit CPU with In-System ProgrammableFlash on a monolithic chip, the Atmel AT90S2313 is a powerfulmicrocontroller that provides a highly flexible and cost-effective solution to many embeddedcontrol applications. The AT90S2313 AVR is supported with a full suite of program and system developmenttools including: C compilers, macro assemblers, program debugger/simulators, In-CircuitEmulators and evaluation kits.

8.3. Pin diagram of AT90S2313

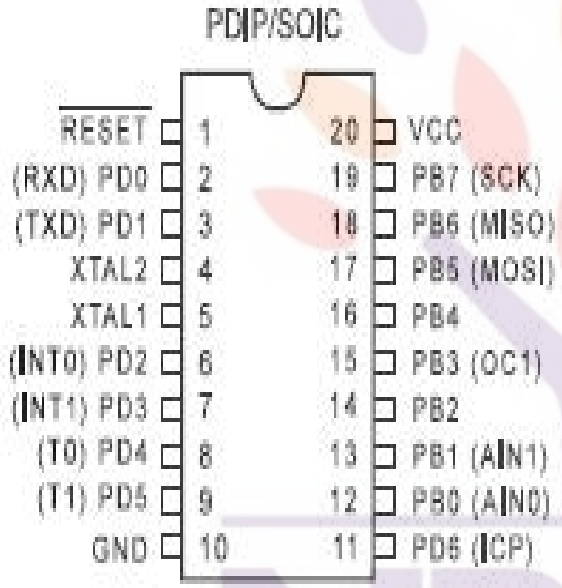


Fig 8.2. Pin diagram of AT90S2313

VCC Supply voltage pin.

GND Ground pin.

Port B (PB7..PB0) Port B is an 8-bit bi-directional I/O port. Port pins can provide internal pull-up resistors(selected for each bit). PB0 and PB1 also serve as the positive input (AIN0) and thenegative input (AIN1), respectively, of the On-chip Analog Comparator. The Port B outputbuffers can sink 20 mA and can drive LED displays directly. When pins PB0 to PB7are used as inputs and are externally pulled low, they will source current if the internalpull-up resistors are activated. The Port B pins are tri-stated when a reset conditionbecomes active, even if the clock is not active.

Port D (PD6..PD0) Port D has seven bi-directional I/O ports with internal pull-up resistors, PD6..PD0. ThePort D output buffers can sink 20 mA. As inputs, Port D pins that are externally pulledlow will source current if the pull-up resistors are activated. The Port D pins are tri-statedwhen a reset condition becomes active, even if the clock is not active.

RESET Reset input. A low level on this pin for more than 50 ns will generate a Reset, even if theclock is not running. Shorter pulses are not guaranteed to generate a Reset.

XTAL1 Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

XTAL2 Output from the inverting Oscillator amplifier.

Crystal Oscillator XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier that canbe configured for use as an On-chip Oscillator, as shown in Figure 2. Either a quartzcrystal or a ceramic resonator may be used. To drive the device from an external clocksource, XTAL2 should be left unconnected while XTAL1 is driven

8.4. AT90S2313 Architectural Overview

NRCM
your roots to success...

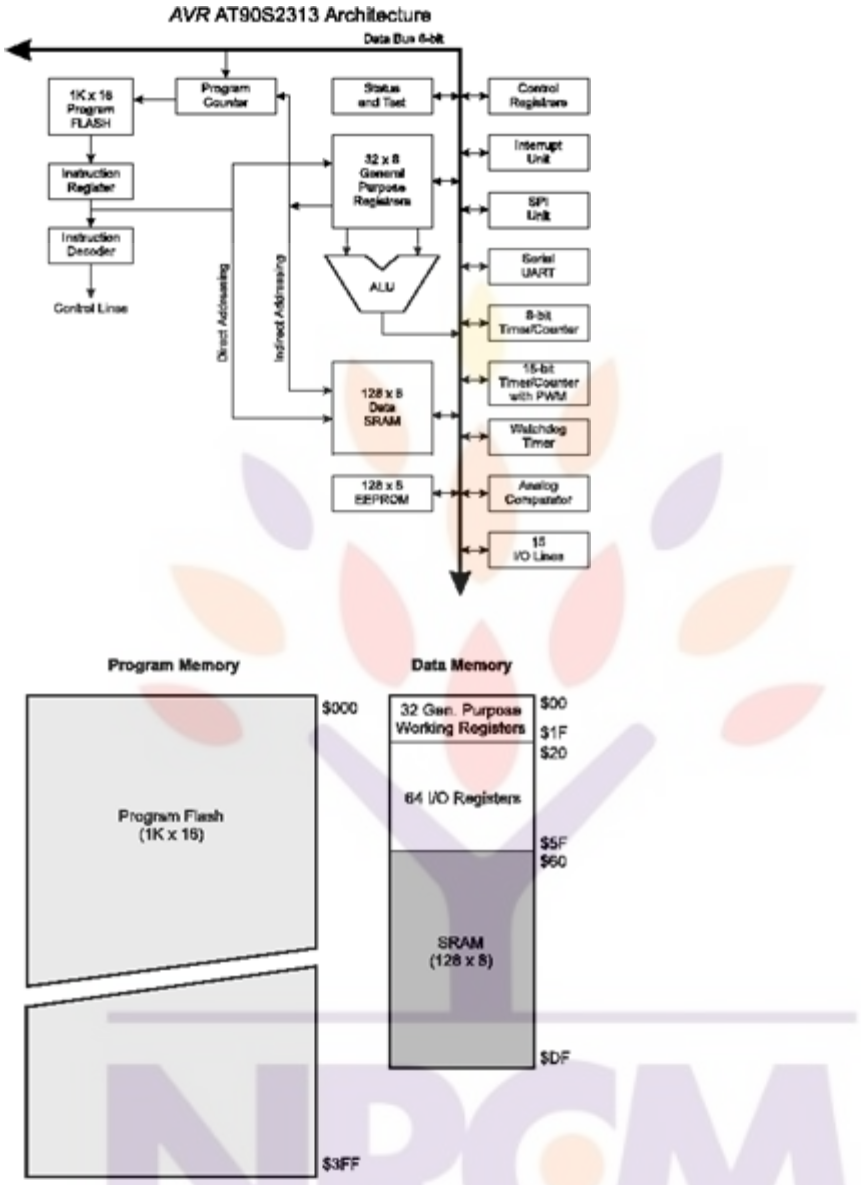


Fig 8.3. Architectural Overview

The fast-access Register File concept contains 32 x 8-bit general purpose working registerswith a single clock cycle access time. This means that during one single clockcycle, one ALU (Arithmetic Logic Unit) operation is executed. Two operands are outputfrom the Register File, the operation is executed, and the result is stored back in the
Register File – in one clock cycle.

Six of the 32 registers can be used as three 16-bit indirect address register pointers forData Space addressing – enabling efficient address calculations. One of the threeaddress pointers is also used as the address pointer for the constant table look-up function.These added function registers are

the 16-bit X-register, Y-register, and Z-register.The ALU supports arithmetic and logic functions between registers or between a constantand a register. Single register operations are also executed in the ALU.

In addition to the register operation, the conventional memory addressing modes can beused on the Register File as well. This is enabled by the fact that the Register File isassigned the 32 lowermost Data Space addresses ($00 - $1F), allowing them to beaccessed as though they were ordinary memory locations.

A flexible interrupt module has its control registers in the I/O space with an additionalGlobal Interrupt Enable bit in the Status Register. All the different interrupts have a separateInterrupt Vector in the Interrupt Vector table at the beginning of the programmemory. The different interrupts have priority in accordance with their Interrupt Vectorposition. The lower the Interrupt Vector address, the higher the priority.

8.5. Register file

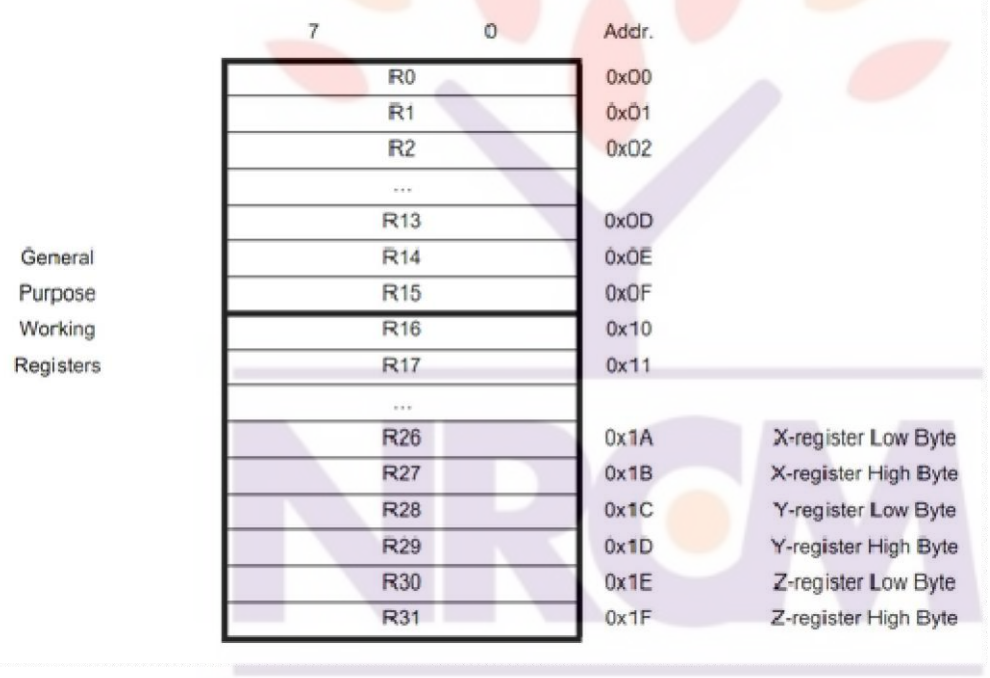


Fig 8.4 AT90S2313 Register file

All the register operating instructions in the instruction set have direct and single-cycleaccess to all registers. The only exception is the five constant arithmetic and logicinstructions SBCI, SUBI, CPI, ANDI, ORI between a constant and a register and the LDIinstruction for load immediate constant data. These instructions apply to the second halfof the registers in the Register File (R16..R31). The general SBC, SUB, CP, AND, OR,and all other operations between two registers or on a single register apply to the entireRegister File.

Each register is also assigned a data memory address, mappingthem directly into the first 32 locations of the user Data Space. Although the RegisterFile is not physically implemented as SRAM locations, this memory organization providesgreat flexibility in access of the registers, as the X-, Y-, and Z-registers can be setto index any register in the file.

X-register, Y-register, and Zregister

The registers R26..R31 have some added functions to their general purpose usage.These registers are the address pointers for indirect addressing of the Data Space.

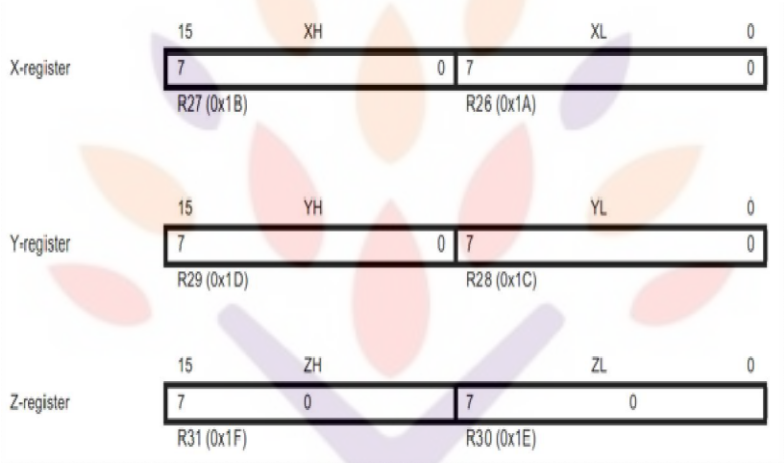


Fig 8.5. x,y, z registers

In the different addressing modes these address registers have functions as fixed displacement,automatic increment and decrement (see the descriptions for the differentinstructions).

ALU – Arithmetic LogicUnit

The high-performance AVR ALU operates in direct connection with all the 32 generalpurpose working registers. Within a single clock cycle, ALU operations between registersin the Register File are executed. The ALU operations are divided into three maincategories – arithmetic, logical, and bit functions.

In-SystemProgrammableFlash Program Memory

The AT90S2313 contains 2K bytes On-chip In-System Programmable Flash memory forprogram storage. Since all instructions are 16- or 32-bit words, the Flash is organized as1K x 16. The Flash memory has an endurance of at least 1,000 write/erase cycles.The AT90S2313 Program Counter (PC) is 10 bits wide, thus addressing the 1,024 programmemory addresses.

See page 60 for a detailed description on Flash data downloading. See page 10 for thedifferent addressing modes.

EEPROM Data Memory The AT90S2313 contains 128 bytes of EEPROM data memory. It is organized as a separatedata space in which single bytes can be read and written. The EEPROM has anendurance of at least 100,000 write/erase cycles.

8.6. Memory access and instruction execution

The AVR CPU is driven by the System Clock Ø, directly generated from the externalclock crystal for the chip. No internal clock division is used.This is the basicpipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique resultsfor functions per cost, functions per clocks and functions per power-unit.

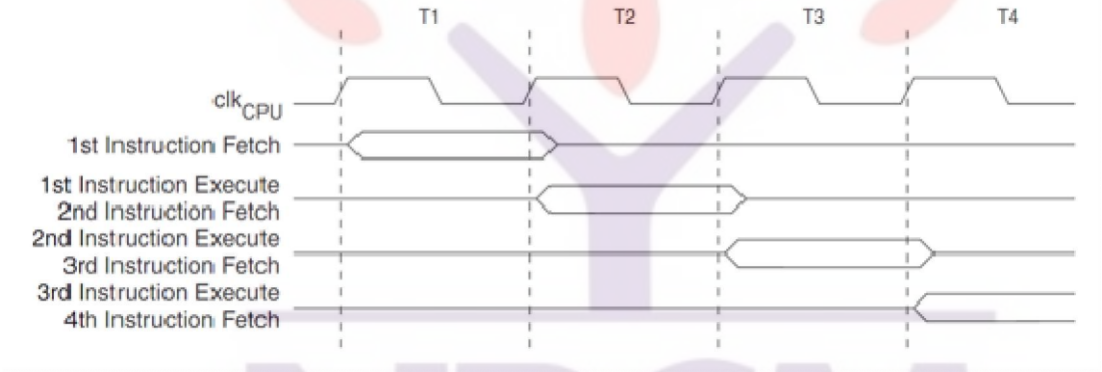


Fig 8.6. Parallel instruction fetch and instruction execution

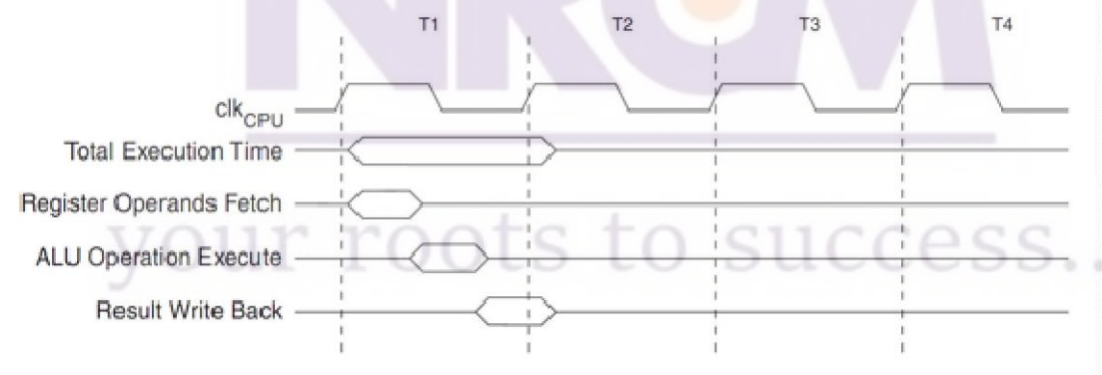


Fig 8.7. Single cycle memory operation

8.7. I/ O Memory

AT90S2313 I/O Space(1)

Address Hex	Name	Function
$3F ($5F)	SREG	Status Register
$3D ($5D)	SPL	Stack Pointer Low
$3B ($5B)	GIMSK	General Interrupt MaSK Register
$3A ($5A)	GIFR	General Interrupt Flag Register
$39 ($59)	TIMSK	Timer/Counter Interrupt MaSK Register
$38 ($58)	TIFR	Timer/Counter Interrupt Flag Register
$35 ($55)	MCUCR MCU	general Control Register
$33 ($53)	TCCR0	Timer/Counter 0 Control Register
$32 ($52)	TCNT0	Timer/Counter 0 (8-bit)
$2F ($4F)	TCCR1A	Timer/Counter 1 Control Register A
$2E ($4E)	TCCR1B	Timer/Counter 1 Control Register B
$2D ($4D)	TCNT1H	Timer/Counter 1 High Byte
$2C ($4C)	TCNT1L	Timer/Counter 1 Low Byte
$2B ($4B)	OCR1AH	Output Compare Register 1 High Byte
$2A ($4A)	OCR1AL	Output Compare Register 1 Low Byte
$25 ($45)	ICR1H T/C 1	Input Capture Register High Byte
$24 ($44)	ICR1L T/C 1	Input Capture Register Low Byte
$21 ($41)	WDTCR	Watchdog Timer Control Register
$1E ($3E)	EEAR	EEPROM Address Register
$1D ($3D)	EEDR	EEPROM Data Register
$1C ($3C)	EECR	EEPROM Control Register
$18 ($38)	PORTB	Data Register, Port B
$17 ($37)	DDRB	Data Direction Register, Port B
$16 ($36)	PINB	Input Pins, Port B
$12 ($32)	PORTD	Data Register, Port D
$11 ($31)	DDRD	Data Direction Register, Port D
$10 ($30)	PIND	Input Pins, Port D
$0C ($2C)	UDR	UART I/O Data Register
$0B ($2B)	USR	UART Status Register
$0A ($2A)	UCR	UART Control Register
$09 ($29)	UBRR	UART Baud Rate Register
$08 ($28)	ACSR	Analog Comparator Control and Status

All AT90S2313 I/O and peripherals are placed in the I/O space. The I/O locations areaccessed by the IN and OUT instructions transferring data between the 32 general purposeworking registers and the I/O space. I/O Registers within the address range $00 -$1F are directly bit-accessible using the SBI and CBI instructions. In these registers, thevalue of single bits can be checked by

using the SBIS and SBIC instructions. When using the I/O specific commands INand OUT, the I/O addresses $00 - $3F must be used. When addressing I/O Registers asSRAM, $20 must be added to this address.
For compatibility with future devices, reserved bits should be written to zero if accessed.
Reserved I/O memory addresses should never be written.Some of the Status Flags are cleared by writing a logical “1” to them. Note that the CBIand SBI instructions will operate on all bits in the I/O Register, writing a “1” back into anyflag read as set, thus clearing the flag. The CBI and SBI instructions work with registers$00 to $1F only.

Status Register – SREG The AVR Status Register (SREG) at I/O space location $3F ($5F) is defined as:

• Bit 7 – I: Global Interrupt Enable
The Global Interrupt Enable bit must be set (one) for the interrupts to be enabled. Theindividual interrupt enable control is then performed in separate control registers. If theGlobal Interrupt Enable bit is cleared (zero), none of the interrupts are enabled independentof the individual interrupt enable settings. The I-bit is cleared by hardware after aninterrupt has occurred, and is set by the RETI instruction to enable subsequentinterrupts.

• Bit 6 – T: Bit Copy Storage
The Bit Copy instructions BLD (Bit Load) and BST (Bit Store) use the T-bit as sourceand destination for the operated bit. A bit from a register in the Register File can be copiedinto T by the BST instruction, and a bit in T can be copied into a bit in a register in theRegister File by the BLD instruction.
• Bit 5 – H: Half-carry Flag
The Half-carry Flag H indicates a Half-carry in some arithmetic operations. See theInstruction Set description for detailed information.

• Bit 4 – S: Sign Bit, S = N □□V
The S-bit is always an exclusive or between the Negative Flag N and the Two’s ComplementOverflow Flag V. See the Instruction Set description for detailed information.

• Bit 3 – V: Two’s Complement Overflow Flag
The Two’s Complement Overflow Flag V supports two’s complement arithmetic. Seethe Instruction Set description for detailed information.

• Bit 2 – N: Negative Flag
The Negative Flag N indicates a negative result after the different arithmetic and logicoperations. See the Instruction Set description for detailed information.

• Bit 1 – Z: Zero Flag

The Zero Flag Z indicates a zero result after the different arithmetic and logic operations.See the Instruction Set description for detailed information.

• Bit 0 – C: Carry Flag

The Carry Flag C indicates a Carry in an arithmetic or logic operation. See the InstructionSet description for detailed information.Note that the Status Register is not automatically stored when entering an interrupt routineand restored when returning from an interrupt routine. This must be handled bysoftware.

Stack Pointer – SP An 8-bit register at I/O address $3D ($5D) forms the Stack Pointer of the AT90S2313. 8bits are used to address the 128 bytes of SRAM in locations $60 - $DF.The Stack Pointer points to the data SRAM stack area where the Subroutine and InterruptStacks are located. This stack space in the data SRAM must be defined by theprogram before any subroutine calls are executed or interrupts are enabled. The StackPointer must be set to point above $60. The Stack Pointer is decremented by 1 whendata is pushed onto the stack with the PUSH instruction, and it is decremented by 2when an address is pushed onto the stack with subroutine calls and interrupts. TheStack Pointer is incremented by 1 when data is popped from the stack with the POPinstruction, and it is incremented by 2 when an address is popped from the stack withreturn from subroutine RET or return from interrupt RETI.

8..8.Timer/Counters

The AT90S2313 provides two general purpose Timer/Counters – one 8-bit T/C and one16-bit T/C. The Timer/Counters have individual prescaling selection from the same 10-bit prescaling timer. Both Timer/Counters can either be used as a timer with an internalclock time base or as a counter with an external pin connection that triggers thecounting.

NRCM
your roots to success...

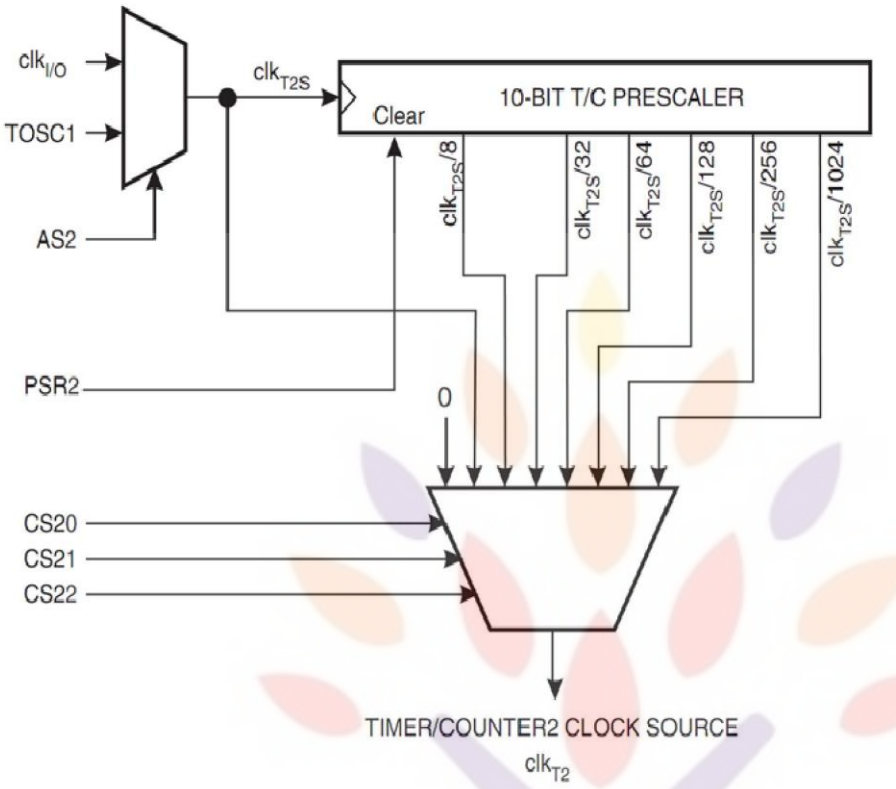


Fig 8.8. Timer/ Counter prescalar

The four different prescaled selections are: CK/8, CK/64, CK/256, and CK/1024, whereCK is the Oscillator clock. For the two Timer/Counters, added selections such as CK,external clock source and stop can be selected as clock sources.

8-bit Timer/Counter0

The 8-bit Timer/Counter0 can select clock source from CK, prescaled CK or an externalpin. In addition, it can be stopped as described in the specification for theTimer/Counter0 Control Register (TCCR0). The Overflow Status Flag is found in theTimer/Counter Interrupt Flag Register (TIFR). Control signals are found in theTimer/Counter0 Control Register (TCCR0). The interrupt enable/disable settings forTimer/Counter0 are found in the Timer/Counter Interrupt Mask Register (TIMSK).When Timer/Counter0 is externally clocked, the external signal is synchronized with theOscillator frequency of the CPU. To assure proper sampling of the external clock, theminimum time between two external clock transitions must be at least one internal CPUclock period. The external clock signal is sampled on the rising edge of the internal CPUclock.

The 8-bit Timer/Counter0 features both a high-resolution and a high-accuracy usagewith the lower prescaling opportunities. Similarly, the high prescaling opportunities makethe Timer/Counter0 useful for lower speed functions or exact timing functions with infrequentactions

Bit	7	6	5	4	3	2	1	0	
$33 ($53)	–	–	–	–	–	CS02	CS01	CS00	TCCR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

• Bits 7..3 – Res: Reserved Bits

These bits are reserved bits in the AT90S2313 and always read zero.

• Bits 2,1,0 – CS02, CS01, CS00: Clock Select0, Bit 2,1 and 0

The Clock Select0 bits 2, 1 and 0 define the prescaling source of Timer/Counter0.

Table 7. Clock 0 Prescale Select

CS02	CS01	CS00	Description
0	0	0	Stop, the Timer/Counter0 is stopped.
0	0	1	CK
0	1	0	CK/8
0	1	1	CK/64
1	0	0	CK/256
1	0	1	CK/1024
1	1	0	External Pin T0, falling edge
1	1	1	External Pin T0, rising edge

Fig 8.9. TCCR0 Register

• Bits 7..3 – Res: Reserved Bits

These bits are reserved bits in the AT90S2313 and always read zero.

• Bits 2,1,0 – CS02, CS01, CS00: Clock Select0, Bit 2,1 and 0

The Clock Select0 bits 2, 1, and 0 define the prescaling source of Timer/Counter0.

Timer/Counter0 – TCNT0

The Timer/Counter0 is realized as an up-counter with read and writes access. If theTimer/Counter0 is written and a clock source is present, the Timer/Counter0 continuescounting in the timer clock cycle following the write operation.

16-bit Timer/Counter1

The 16-bit Timer/Counter1 can select clock source from CK, prescaled CK or an externalpin. In addition, it can be stopped as described in the specification for the Timer/Counter1 Control Register (TCCR1B). The different Status Flags (Overflow, CompareMatch and Capture Event) and control signals are found in the Timer/CounterInterrupt Flag Register (TIFR). The interrupt

enable/disable settings for Timer/Counter1are found in the Timer/Counter Interrupt Mask Register (TIMSK).When Timer/Counter1 is externally clocked, the external signal is synchronized with theOscillator frequency of the CPU. To assure proper sampling of the external clock, theminimum time between two external clock transitions must be at least one internal CPUclock period. The external clock signal is sampled on the rising edge of the internal CPUclock.

The 16-bit Timer/Counter1 features both a high-resolution and a high-accuracy usagewith the lower prescaling opportunities. Similarly, the high prescaling opportunitiesmakes the Timer/Counter1 useful for lower speed functions or exact timing functionswith infrequent actions.The Timer/Counter1 supports an Output Compare function using the Output CompareRegister 1A (OCR1A) as the data source to be compared to the Timer/Counter1 contents.The Output Compare functions include optional clearing of the counter oncompare matches, and actions on the Output Compare pin 1 on compare matches.

Timer/Counter1 can also be used as an 8-, 9-, or 10-bit Pulse Width Modulator. In thismode the counter and the OCR1 Register serve as a glitch-free standalone PWM withcentered pulses

Timer/Counter1 Control Register A – TCCR1A

• Bits 7, 6 – COM1A1, COM1A0: Compare Output Mode1, Bits 1 and 0

The COM1A1 and COM1A0 control bits determine any output pin action following acompare match in Timer/Counter1. Any output pin actions affect pin OC1 (Output Comparepin 1) (PB3). This is an alternative function to the I/O port, and the correspondingdirection control bit must be set (one) to control an output pin.

Compare 1 Mode Select(

COM1A1	COM1A0	Description
0	0	Timer/Counter1 disconnected from output pin OC1
0	1	Toggle the OC1 output line.
1	0	Clear the OC1 output line (to zero).
1	1	Set the OC1 output line (to one).

• Bits 5..2 – Res: Reserved Bits

These bits are reserved bits in the AT90S2313 and always read zero.

• Bits 1, 0 – PWM11, PWM10: Pulse Width Modulator Select Bits

PWM11	PWM10	Description
0	0	PWM operation of Timer/Counter1 is disabled
0	1	Timer/Counter1 is an 8-bit PWM
1	0	Timer/Counter1 is a 9-bit PWM
1	1	Timer/Counter1 is a 10-bit PWM

NIPCM your roots to success...

Timer/Counter1 Control
Register B – TCCR1B

• Bit 7 – ICNC1: Input Capture1 Noise Canceller (4 CKs)
When the ICNC1 bit is cleared (zero), the input capture trigger noise canceller function isdisabled. The input capture is triggered at the first rising/falling edge sampled on the ICP(input capture pin) as specified. When the ICNC1 bit is set (one), four successive samples are measured on the ICP (input capture pin), and all samples must be high/low according to the input capture trigger specification in the ICES1 bit. The actual sampling frequency is the XTAL clock frequency.

• Bit 6 – ICES1: Input Capture1 Edge Select
While the ICES1 bit is cleared (zero), the Timer/Counter1 contents are transferred to theInput Capture Register (ICR1) on the falling edge of the input capture pin (ICP). While the ICES1 bit is set (one), the Timer/Counter1 contents are transferred to the Input CaptureRegister (ICR1) on the rising edge of the input capture pin (ICP).

• Bits 5, 4 – Res: Reserved Bits
These bits are reserved bits in the AT90S2313 and always read zero.

• Bit 3 – CTC1: Clear Timer/Counter1 on Compare Match
When the CTC1 control bit is set (one), the Timer/Counter1 is reset to $0000 in the clockcycle after a compareA match. If the CTC1 control bit is cleared, Timer/Counter1 continuescounting and is unaffected by a compare match. Since the compare match isdetected in the CPU clock cycle following the match, this function will behave differentlywhen a prescaling higher than 1 is used for the timer. When a prescaling of 1 is used,and the Compare A Register is set to C, the timer will count as follows if CTC1 is set:
... | C-2 | C-1 | C | 0 | 1 |...

When the prescaler is set to divide by 8, the timer will count like this:
... | C-2, C-2, C-2, C-2, C-2, C-2, C-2, C-2 | C-1, C-1, C-1, C-1, C-1, C-1, C-1, C-1 | C, 0, 0, 0, 0, 0, 0, 0 |...
In PWM mode, this bit has no effect.

• Bits 2,1,0 – CS12, CS11, CS10: Clock Select1, Bits 2, 1 and 0
The Clock Select1 bits 2, 1, and 0 define the prescaling source of Timer/Counter1.

Clock 1 Prescale Select

CS12	CS11	CS10	Description
0	0	0	Stop, the Timer/Counter1 is stopped.
0	0	1	CK
0	1	0	CK/8

0	1	1	CK/64
1	0	0	CK/256
1	0	1	CK/1024
1	1	0	External Pin T1, falling edge
1	1	1	External Pin T1, rising edge

The Stop condition provides a Timer Enable/Disable function. The CK down dividedmodes are scaled directly from the CK Oscillator clock. If the external pin modes areused for Timer/Counter1, transitions on PD5/(T1) will clock the counter even if the pin isconfigured as an output. This feature can give the user software control of the counting.

Timer/Counter1 – TCNT1Hand TCNT1L

This 16-bit register contains the precaled value of the 16-bit Timer/Counter1. Toensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary register(TEMP). This temporary register is also used when accessing OCR1A and ICR1. Ifthe main program and interrupt routines perform access to registers using TEMP, interruptsmust be disabled during access from the main program or interrupts if interruptsare re-enabled.

• TCNT1 Timer/Counter1 Write:

When the CPU writes to the high byte TCNT1H, the written data is placed in theTEMP Register. Next, when the CPU writes the low byte TCNT1L, this byte of datais combined with the byte data in the TEMP Register, and all 16 bits are written tothe TCNT1 Timer/Counter1 Register simultaneously. Consequently, the high byteTCNT1H must be accessed first for a full 16-bit register write operation.

• TCNT1 Timer/Counter1 Read:

When the CPU reads the low byte TCNT1L, the data of the low byte TCNT1L is sentto the CPU and the data of the high byte TCNT1H is placed in the TEMP Register.When the CPU reads the data in the high byte TCNT1H, the CPU receives the datain the TEMP Register. Consequently, the low byte TCNT1L must be accessed firstfor a full 16-bit register read operation. The Timer/Counter1 is realized as an up or up/down (in PWM mode) counter with readand write access. If Timer/Counter1 is written to and a clock source is selected, theTimer/Counter1 continues counting in the timer clock cycle after it is preset with the writtenvalue.

Timer/Counter1 Output
Compare Register A –OCR1AH and OCR1AL

The Output Compare Register is a 16-bit read/write register.The Timer/Counter1 Output Compare Register contains the data to be continuouslycompared with Timer/Counter1. Actions on compare matches are specified in theTimer/Counter1 Control and Status Registers.

Since the Output Compare Register (OCR1A) is a 16-bit register, a temporary registerTEMP is used when OCR1A is written to ensure that both bytes are updated simultaneously.When the CPU writes the high byte, OCR1AH, the data is temporarily stored inthe TEMP Register. When

the CPU writes the low byte, OCR1AL, the TEMP Register issimultaneously written to OCR1AH. Consequently, the high byte OCR1AH must be writtenfirst for a full 16-bit register write operation.
The TEMP Register is also used when accessing TCNT1, and ICR1. If the main programand interrupt routines perform access to registers using TEMP, interrupts must bedisabled during access from the main program or interrupts if interrupts are re-enabled.

Timer/Counter1 Input Capture Register – ICR1H and ICR1L

The Input Capture Register is a 16-bit read-only register.When the rising or falling edge (according to the input capture edge setting [ICES1]) ofthe signal at the input capture pin (ICP) is detected, the current value of theTimer/Counter1 is transferred to the Input Capture Register (ICR1). At the same time,the Input Capture Flag (ICF1) is set (one).Since the Input Capture Register (ICR1) is a 16-bit register, a temporary register TEMPis used when ICR1 is read to ensure that both bytes are read simultaneously. When theCPU reads the low byte ICR1L, the data is sent to the CPU and the data of the high byteICR1H is placed in the TEMP Register. When the CPU reads the data in the high byteICR1H, the CPU receives the data in the TEMP Register. Consequently, the low byteICR1L must be accessed first for a full 16-bit register read operation.The TEMP Register is also used when accessing TCNT1 and OCR1A. If the main programand interrupt routines perform access to registers using TEMP, interrupts must bedisabled during access from the main program or interrupts if interrupts are re-enabled.

8.9.UART

The AT90S2313 features a full duplex (separate Receive and Transmit Registers) UniversalAsynchronous Receiver and Transmitter (UART). The main features are:
• Baud Rate Generator that can Generate a Large Number of Baud Rates (bps)
• High Baud Rates at Low XTAL Frequencies
• 8 or 9 Bits Data
• Noise Filtering
• Overrun Detection
• Framing Error Detection
• False Start Bit Detection
• Three separate Interrupts on TX Complete, TX Data Register Empty and RX Complete

your roots to success...

Data Transmission A block schematic of the UART transmitter is shown in Figure 8.10.Data transmission is initiated by writing the data to be transmitted to the UART I/O DataRegister (UDR). Data is transferred from UDR to the Transmit Shift Register when:
• A new character has been written to UDR after the stop bit from the previouscharacter has been shifted out. The Shift Register is loaded immediately.

• A new character has been written to UDR before the stop bit from the previouscharacter has been shifted out. The Shift Register is loaded when the stop bit of thecharacter currently being transmitted has been shifted out.

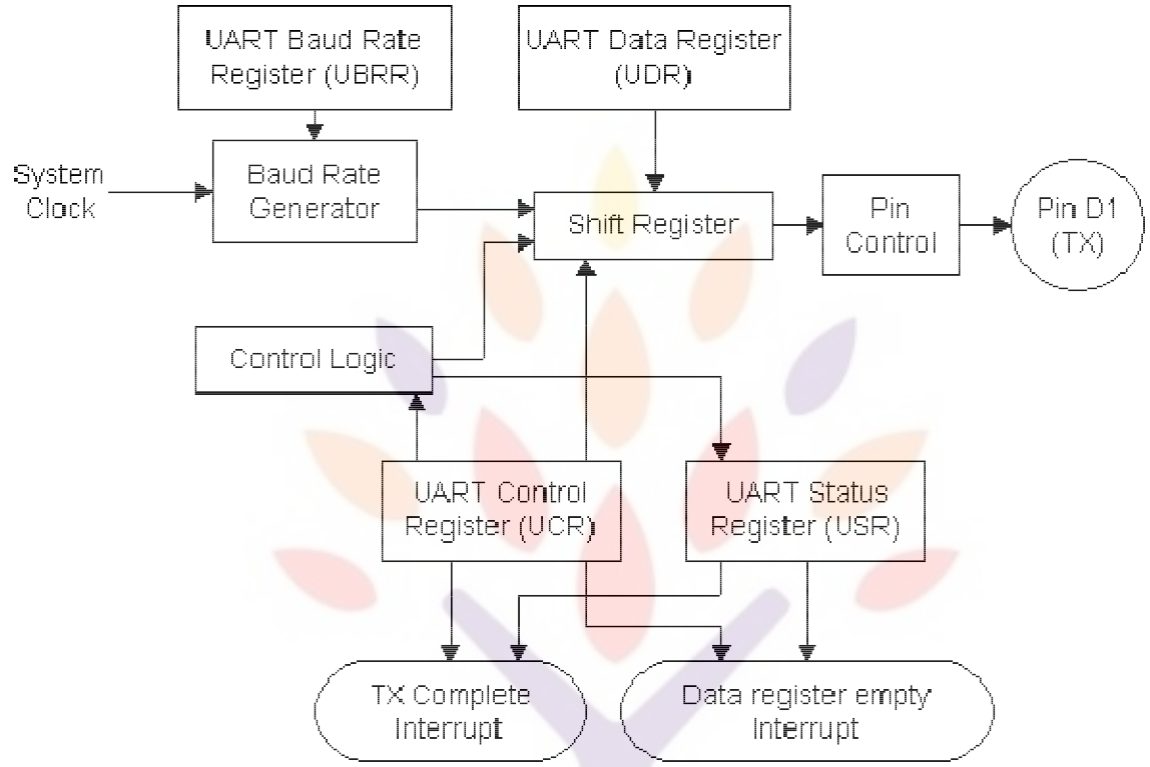


Fig 8.10. UART Transmitter

If the 10(11)-bit Transmitter Shift Register is empty, data is transferred from UDR to theShift Register. At this time the UDRE (UART Data Register Empty) bit in the UART Status Register (USR) is set. When this bit is set (one), the UART is ready to receive thenext character. At the same time as the data is transferred from UDR to the 10(11)-bitShift Register, bit 0 of the Shift Register is cleared (start bit) and bit 9 or 10 is set (stopbit). If 9-bit data word is selected (the CHR9 bit in the UART Control Register [UCR] isset), the TXB8 bit in UCR is transferred to bit 9 in the Transmit Shift Register.On the Baud Rate clock following the transfer operation to the Shift Register, the start bitis shifted out on the TXD pin. Then follows the data, LSB first. When the stop bit hasbeen shifted out, the Shift Register is loaded if any new data has been written to theUDR during the transmission. During loading, UDRE is set. If there is no new data in theUDR Register to send when the stop bit is shifted out, the UDRE Flag will remain setuntil UDR is written again. When no new data has been written, and the stop bit hasbeen present on TXD for one bit length, the TX Complete Flag (TXC) in USR is set.The TXEN bit in UCR enables the UART transmitter when set (one). When this bit iscleared (zero), the PD1 pin can be used for general I/O. When TXEN is set, the UARTTransmitter will be connected to PD1, which is forced to be an output pin regardless ofthe setting of the DDD1 bit in DDRD

Data Reception

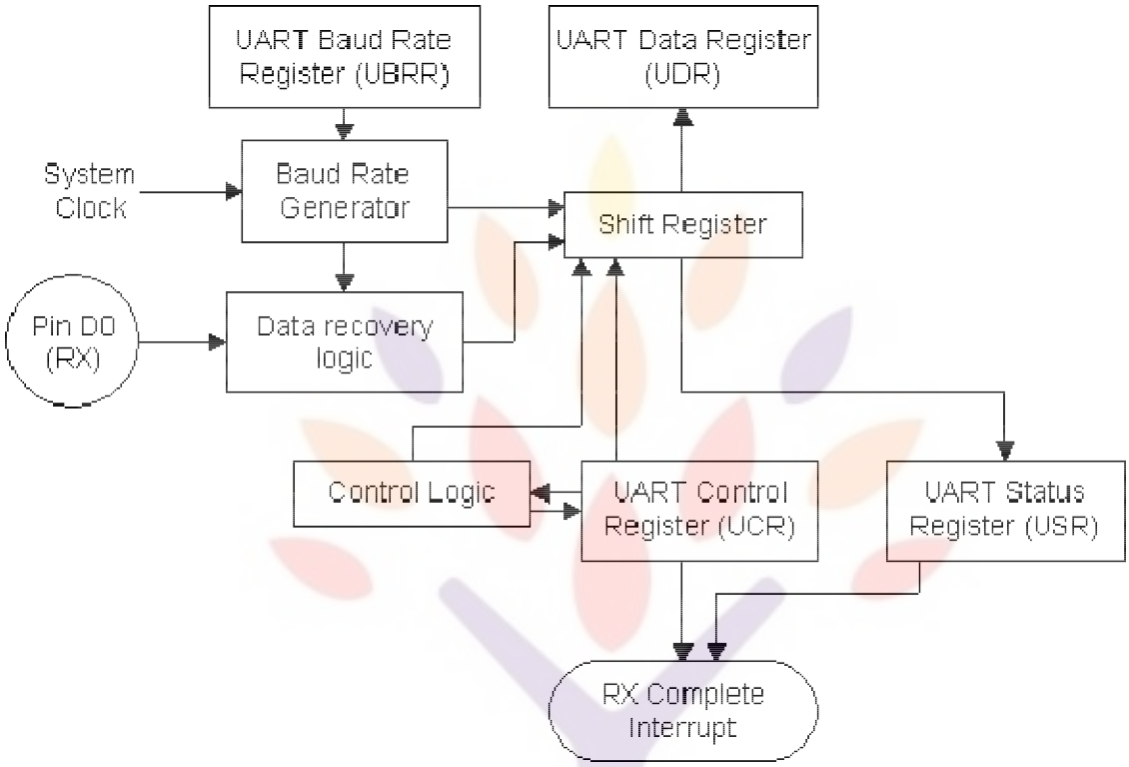


Fig 8.11. UART receiver

The Receiver front-end logic samples the signal on the RXD pin at a frequency of 16times the baud rate. While the line is idle, one single sample of logical "0" will be interpretedas the falling edge of a start bit, and the start bit detection sequence is initiated.Let sample 1 denote the first zero-sample. Following the 1-to-0 transition, the receiversamples the RXD pin at samples 8, 9 and 10. If two or more of these three samples are found to be logical "1"s, the start bit is rejected as a noise spike and the receiver startslooking for the next 1-to-0 transition.

If, however, a valid start bit is detected, sampling of the data bits following the start bit isperformed. These bits are also sampled at samples 8, 9 and 10. The logical value foundin at least two of the three samples is taken as the bit value. All bits are shifted into theTransmitter Shift Register as they are sampled.When the stop bit enters the Receiver, the majority of the three samples must be "1" toaccept the stop bit. If two or more samples are logical "0"s, the Framing Error (FE) Flagin the UART Status Register (USR) is set. Before reading the UDR Register, the usershould always check the FE bit to detect Framing Errors.Whether or not a valid stop bit is detected at the end of a character-reception cycle, thedata is transferred to UDR and the RXC Flag in USR is set. UDR is in fact two physicallyseparate registers; one for transmitted data and one for received data. When UDR isread, the Receive Data Register is accessed, and

when UDR is written, the TransmitData Register is accessed. If 9-bit data word is selected (the CHR9 bit in the UART ControlRegister [UCR] is set), the RXB8 bit in UCR is loaded with bit 9 in the Transmit ShiftRegister when data is transferred to UDR.If, after having received a character, the UDR Register has not been read since the lastreceive, the OverRun (OR) Flag in UCR is set. This means that the last data byte shiftedinto the Shift Register could not be transferred to UDR and has been lost. The OR bit is buffered and is updated when the valid data byte in UDR is read. Thus, the user should always check the OR bit after reading the UDR Register in order to detect any overrunsif the baud rate is high or CPU load is high.
When the RXEN bit in the UCR Register is cleared (zero), the receiver is disabled. Thismeans that the PD0 pin can be used as a general I/O pin. When RXEN is set, the UARTReceiver will be connected to PD0, which is forced to be an input pin regardless of thesetting of the DDD0 bit in DDRD. When PD0 is forced to input by the UART, thePORTD0 bit can still be used to control the pull-up resistor on the pin.When the CHR9 bit in the UCR Register is set, transmitted and received characters arenine bits long plus start and stop bits. The ninth data bit to be transmitted is the TXB8 bitin UCR Register. This bit must be set to the wanted value before a transmission is initiatedby writing to the UDR Register. The ninth data bit received is the RXB8 bit in theUCR Register.

UART Control

The UART I/O Data Register –UDR

The UDR Register is actually two physically separate registers sharing the same I/Oaddress. When writing to the register, the UART Transmit Data Register is written.When reading from UDR, the UART Receive Data Register is read.

UART Status Register – USR

The USR Register is a read-only register providing information on the UART status.

• Bit 7 – RXC: UART Receive Complete

This bit is set (one) when a received character is transferred from the Receiver ShiftRegister to UDR. The bit is set regardless of any detected framing errors. When theRXCIE bit in UCR is set, the UART Receive Complete interrupt will be executed whenRXC is set (one). RXC is cleared by reading UDR. When interrupt-driven data reception is used, the UART Receive Complete Interrupt routine must read UDR in order to clearRXC, otherwise a new interrupt will occur once the interrupt routine terminates.

• Bit 6 – TXC: UART Transmit Complete

This bit is set (one) when the entire character (including the stop bit) in the TransmitShift Register has been shifted out and no new data has been written to UDR. This flagis especially useful in half-duplex communications interfaces, where a transmitting applicationmust enter Receive mode and free the communications bus immediately aftercompleting the transmission.

When the TXCIE bit in UCR is set, setting of TXC causes the UART Transmit Completeinterrupt to be executed. TXC is cleared by hardware when executing the correspondinginterrupt handling vector. Alternatively, the TXC bit is cleared (zero) by writing a logical“1” to the bit.

• Bit 5 – UDRE: UART Data Register Empty

This bit is set (one) when a character written to UDR is transferred to the Transmit ShiftRegister. Setting of this bit indicates that the transmitter is ready to receive a new characterfor transmission.

When the UDRIE bit in UCR is set, the UART Transmit Complete interrupt is executedas long as UDRE is set. UDRE is cleared by writing UDR. When interrupt-driven datatransmittal is used, the UART Data Register Empty Interrupt routine must write UDR inorder to clear UDRE, otherwise a new interrupt will occur once the interrupt routineterminates.UDRE is set (one) during reset to indicate that the transmitter is ready.

• Bit 4 – FE: Framing Error

This bit is set if a Framing Error condition is detected (i.e., when the stop bit of an incomingcharacter is zero).The FE bit is cleared when the stop bit of received data is one.

• Bit 3 – OR: OverRun

This bit is set if an OverRun condition is detected (i.e., when a character already presentin the UDR Register is not read before the next character has been shifted into theReceiver Shift Register). The OR bit is buffered, which means that it will be set once thevalid data still in UDRE is read.The OR bit is cleared (zero) when data is received and transferred to UDR.

• Bits 2..0 – Res: Reserved Bits

These bits are reserved bits in the AT90S2313 and will always read as zero.

UART Control Register – UCR

• Bit 7 – RXCIE: RX Complete Interrupt Enable

When this bit is set (one), a setting of the RXC bit in USR will cause the Receive CompleteInterrupt routine to be executed provided that global interrupts are enabled.

• Bit 6 – TXCIE: TX Complete Interrupt Enable

When this bit is set (one), a setting of the TXC bit in USR will cause the Transmit CompleteInterrupt routine to be executed provided that global interrupts are enabled.

• Bit 5 – UDRIE: UART Data Register Empty Interrupt Enable

When this bit is set (one), a setting of the UDRE bit in USR will cause the UART DataRegister Empty Interrupt routine to be executed provided that global interrupts areenabled.

• Bit 4 – RXEN: Receiver Enable

This bit enables the UART Receiver when set (one). When the Receiver is disabled, theRXC, OR and FE Status Flags cannot become set. If these flags are set, turning offRXEN does not cause them to be cleared.

• Bit 3 – TXEN: Transmitter Enable

This bit enables the UART Transmitter when set (one). When disabling the Transmitter while transmitting a character, the Transmitter is not disabled before the character in theShift Register plus any following character in UDR has been completely transmitted.

• Bit 2 – CHR9: 9 Bit Characters

When this bit is set (one), transmitted and received characters are nine bits long plusstart and stop bits. The ninth bit is read and written by using the RXB8 and TXB8 bits inUCR, respectively. The ninth data bit can be used as an extra stop bit or a parity bit.

• Bit 1 – RXB8: Receive Data Bit 8

When CHR9 is set (one), RXB8 is the ninth data bit of the received character.

• Bit 0 – TXB8: Transmit Data Bit 8

When CHR9 is set (one), TXB8 is the ninth data bit in the character to be transmitted.

Baud Rate Generator The baud rate generator is a frequency divider that generates baud rates according tothe following equation:

• BAUD = Baud Rate = $f_{CK} / 16(UBRR + 1)$
• fCK = Crystal Clock frequency
• UBRR = Contents of the UART Baud Rate Register (UBRR) (0 - 255)

For standard crystal frequencies, the most commonly used baud rates can be generatedby using the UBRR settings in Table 15. UBRR values that yield an actual baud rate differingless than 2% from the target baud rate, are boldfaced in the table. However, usingbaud rates that have more than 1% error is not recommended. High error ratings giveless noise resistance.

MICROPROCESSORS VIVA AND INTERVIEW QUESTIONS

1) How many bit 8086 microprocessor is?
2) What is the size of data bus of 8086?
3) What is the size of address bus of 8086?
4) What is the max memory addressing capacity of 8086?
5) Which are the basic parts of 8086?
6) What are the functions of BIU?
7) What are the functions of EU?
8) How many pin IC 8086 is?
9) What IC8086 is?
10) What is the size of instruction queue in 8086?
11) What is the size of instruction queue in 8088?
12) Which are the registers present in 8086?
13) What do you mean by pipelining in 8086?
14) How many 16 bit registers are available in 8086?
15) Specify addressing modes for any instruction?
16) What do you mean by assembler directives?
17) What .model small stands for?
18) What is the supply requirement of 8086?
19) What is the relation between 8086 processor frequency & crystal frequency?
20) Functions of Accumulator or AX register?
21) Functions of BX register?
22) Functions of CX register?
23) Functions of DX register?
24) How Physical address is generated?
25) Which are pointers present in this 8086?
26) Which is by default pointer for CS/ES?
27) How many segments present in it?
28) What is the size of each segment?
29) Basic difference between 8085 and 8086?
30) Which operations are not available in 8085?

your roots to success...

31) What are the flags in 8086?

In 8086 Carry flag, Parity flag, Auxiliary carry flag, Zero flag, Overflow flag, Trace flag, Interrupt flag, Direction flag, and Sign flag.

32) What are the various interrupts in 8086?

Maskable interrupts, Non-Maskable interrupts.

33) What is meant by Maskable interrupts?

An interrupt that can be turned off by the programmer is known as Maskable interrupt.

34) What is Non-Maskable interrupts?

An interrupt which can be never be turned off (ie.disabled) is known as Non-Maskable interrupt.

35) Which interrupts are generally used for critical events?

Non-Maskable interrupts are used in critical events. Such as Power failure, Emergency, Shut off etc.,

36)Give examples for Maskable interrupts?

RST 7.5, RST6.5, RST5.5 are Maskable interrupts

37) Give example for Non-Maskable interrupts?

Trap is known as Non-Maskable interrupts, which is used in emergency condition.

38) What is the Maximum clock frequency in 8086?

5 Mhz is the Maximum clock frequency in 8086.

39) What are the various segment registers in 8086?

Code, Data, Stack, Extra Segment registers in 8086.

40) Which Stack is used in 8086?

FIFO (First In First Out) stack is used in 8086.In this type of Stack the first stored information is retrieved first.

41) What are the address lines for the software interrupts?

RST 0 0000 H
RST1 0008 H
RST2 0010 H
RST3 0018 H
RST4 0020 H
RST5 0028 H
RST6 0030 H
RST7 0038 H

NRCM
our roots to success...

42) What is SIM and RIM instructions?

SIM is Set Interrupt Mask. Used to mask the hardware interrupts.
RIM is Read Interrupt Mask. Used to check whether the interrupt is Masked or not.

43) Which is the tool used to connect the user and the computer?

Interpreter is the tool used to connect the user and the tool.

44) What is the position of the Stack Pointer after the PUSH instruction?

The address line is 02 less than the earlier value.

45) What is the position of the Stack Pointer after the POP instruction?

The address line is 02 greater than the earlier value.

46) Logic calculations are done in which type of registers?

Accumulator is the register in which Arithmetic and Logic calculations are done.

47) What are the different functional units in 8086?

Bus Interface Unit and Execution unit, are the two different functional units in 8086.

48) Give examples for Micro controller?

Z80, Intel MSC51 &96, Motorola are the best examples of Microcontroller.

49) What is meant by cross-compiler?

A program runs on one machine and executes on another is called as cross-compiler.

your roots to success...

50) What are the address lines for the hardware interrupts?

RST 7.5 003C H
RST 6.5 0034 H
RST 5.5 002C H
TRAP 0024 H

51) Which Segment is used to store interrupt and subroutine return address registers?

Stack Segment in segment register is used to store interrupt and subroutine return address registers.

52) Which Flags can be set or reset by the programmer and also used to control the operation of the processor?

Trace Flag, Interrupt Flag, Direction Flag.

53) What does EU do?

Execution Unit receives program instruction codes and data from BIU, executes these instructions and store the result in general registers.

54) Which microprocessor accepts the program written for 8086 without any changes?

8088 is that processor.

55) What is the difference between 8086 and 8088?

The BIU in 8088 is 8-bit data bus & 16- bit in 8086.Instruction queue is 4 byte long in 8088and 6 byte in 8086.

56) What is the difference between min mode and max mode of 8086?
57) What is the difference between near and far procedure?
58) What is the difference between Macro and procedure?
59) What is the difference between instructions RET & IRET?
60) What is the difference between instructions MUL & IMUL?
61) What is the difference between instructions DIV & IDIV?
62) What is difference between shifts and rotate instructions?
63) Which are strings related instructions?
64) Which are addressing modes and their examples in 8086?
65) What does u mean by directives?
66) What does u mean by Prefix?
67) What .model small means?
68) Difference between small, medium, tiny, huge?
69) What is dd, dw, db?
70) Interrupts in 8086 and there function.
71) What is the function of 01h of Int 21h?
72) What is the function of 02h of Int 21h?

EDOM
Your roots to success...

73) What is the function of 09h of Int 21h?
74) What is the function of 0Ah of Int 21h?
75) What is the function of 4ch of Int 21h?
76) What is the reset address of 8086?
77) What is the size of flag register in 8086? Explain all.
78) What is the difference between 08H and 01H functions of INT 21H?
79) Which is faster- Reading word size data whose starting address is at even or at odd address of memory in 8086?
80) Which are the default segment base: offset pairs?
81) Can we use SP as offset address holder with CS?
82) Which are the base registers in 8086?
83) Which is the index registers in 8086?
84) What do you mean by segment override prefix?
85) Whether micro reduces memory requirements?
86) What do you mean by macro?
87) What is diff between macro and procedure?
88) Types of procedure?
89) What TASM is?
90) What TLINK is?
91) What TD is?
92) What do u mean by assembler?
93) What do u mean by linker?
94) What do u mean by loader?
95) What do u mean by compiler?
96) What do u mean by emulator?
97) Stack related instruction?
98) .stack 100 means?
99) What do you mean by 20 dup (0)?
100) Which flags of 8086 are not present in 8085?
101) What is the size of flag register?
102) Can you perform 32 bit operation with 8086? How?
103) Whether 8086 is compatible with Pentium processor?
104) What is 8087? How it is different from 8086?
105) While accepting no. from user why u need to subtract 30 from that?
106) While displaying no. from user why u need to add 30 to that?
107) What are ASCII codes for nos. 0 to F?
108) How does U differentiate between positive and negative numbers?
109) What is range for these numbers?
110) Which no. representation system you have used?
111) What is LEA?
112) What is @data indicates in instruction- MOV ax, @data?
113) What is maximum size of the instruction in 8086?
114) Why we indicate FF as 0FF in program?
115) What is mul BX and div BX? Where result goes?
116) Where queue is present?
117) What is the advantage of using internal registers?

NRCM
your roots to success...

118) What is SI, DI and their functions?
119) Which are the pointers used in 8086 and their functions?
120) What is a type of queue in 8086?
121) What is minimum mode of 8086?
122) What is maximum mode of 8086?
123) Which are string instructions?
124) In string operations which is by default string source pointer?
125) In string operations which is by default string destination pointer?
126) what is segmentation?
127) how many bit processor does 8086?
128) how many address lines in 8086
129) how many data lines in 8086?
130) multiplexed lines in 8086?
131) over flow flag, interrupt flag ,direction flag, trap flag?
132) role of pointers ?
133) how 16 bit processor generates 20 bit addresses
134) instructions set of 8086
135) timing diagram of 8086
136) min/max mode working of 8086?
137) pin difference in min/max mode
138) interrupt structure in 8086?
139) how an interrupt is acknowledged?
140) how the cs:ip is working during interrupt
141) new cs:ip during interrupt
142) how the even odd address are assigned through 8086?

NRCM
your boots to success...

PROGRAMS:

1) What do you mean by assembler?
2) What do you mean by linker?
3) What do you mean by debugger?
4) What do you mean by compiler?
5) What do you mean by locator?
6) What do you mean by emulator?
7) When divide overflow error occurs?
8) What .startup stands for?
9) Explain the logic of array addition program.
10) Explain the logic of finding out negative nos. from an array of signed nos.

11) Explain the logic of code conversion (BCD to hex and hex to BCD) program.
12) Explain the logic of multiplication (by successive addition and shift and add method) program.
13) Explain the logic of non overlap and overlap block transfer program
14) Explain the logic of string related programs.
15) Which assembler directives are used with near procedure?
16) Which assembler directives are used with far procedure?

80386 (microprocessor):

1) What IC 80386 is?
2) How many pin IC 80836 is?
3) 80386 is how many bit processor?
4) What is the size of instruction queue in 80386?

INTERRUPTS:

1) What do you mean by interrupt?
2) Which are the hardware and software interrupts in 8086?
3) Mention the priority of interrupts in8086.
4) What is int1, int2, int3?
5) What do you mean by NMI interrupt?
6) What do you mean by IVT in 8086?
7) What is the size of IVT?
8) Where IVT is located?
9) Which steps 8086 follows to handle any interrupt?

INTERFACING:

1) What are the types of interfacing?
2) Compare memory interfacing and IO interfacing.
3) What are the types of IO interfacing?
4) What is the difference between direct and indirect IO interfacing?
5) What is the difference between memory mapped IO and IO mapped IO interfacing?

8255 (programmable peripheral interface) :

1) What IC 8255 is?
2) How many pin IC 8255 is?
3) Explain control word format of 82

INTERVIEW QUESTIONS:

1. What is a Microprocessor?

   Microprocessor is a program-controlled device, which fetches the instructions from memory, decodes and executes the instructions. Most Micro Processor are single- chip devices.

2. Give examples for 8 / 16 / 32 bit Microprocessor?

   8-bit Processor - 8085 / Z80 / 6800;
   16-bit Processor - 8086 / 68000 / Z8000;
   32-bit Processor - 80386 / 80486.

3. Why 8085 processor is called an 8 bit processor?

   Because 8085 processor has 8 bit ALU (Arithmetic Logic Review). Similarly 8086 processor has 16 bit ALU.

4. What is 1st / 2nd / 3rd / 4th generation processor?

   The processor made of PMOS / NMOS / HMOS / HCMOS technology is called 1st / 2nd / 3rd / 4th generation processor, and it is made up of 4 / 8 / 16 / 32 bits.

5. Define HCMOS?

   High-density n- type Complimentary Metal Oxide Silicon field effect transistor.

6. What does microprocessor speed depend on?

   The processing speed depends on DATA BUS WIDTH.

7. Is the address bus unidirectional?

   The address bus is unidirectional because the address information is always given by the Micro Processor to address a memory location of an input / output devices.

8. Is the data bus is Bi-directional?

   The data bus is Bi-directional because the same bus is used for transfer of data between Micro Processor and memory or input / output devices in both the direction.

NRCM your root to success...

9. What is the disadvantage of microprocessor?

It has limitations on the size of data. Most Microprocessor does not support floating-point operations.

10. What is the difference between microprocessor and microcontroller?

In Microprocessor more op-codes, few bit handling instructions. But in Microcontroller: fewer op-codes, more bit handling Instructions, and also it is defined as a device that includes micro processor, memory, & input / output signal lines on a single chip.

11. What is meant by LATCH?

Latch is a D- type flip-flop used as a temporary storage device controlled by a timing signal, which can store 0 or 1. The primary function of a Latch is data storage. It is used in output devices such as LED, to hold the data for display.

12. Why does microprocessor contain ROM chips?

Microprocessor contain ROM chip because it contain instructions to execute data.

13. What is the difference between primary & secondary storage device?

In primary storage device the storage capacity is limited. It has a volatile memory.
In secondary storage device the storage capacity is larger. It is a nonvolatile memory.
Primary devices are: RAM / ROM.
Secondary devices are: Floppy disc / Hard disk.

14. Difference between static and dynamic RAM?

Static RAM: No refreshing, 6 to 8 MOS transistors are required to form one memory cell, Information stored as voltage level in a flip flop.
Dynamic RAM: Refreshed periodically, 3 to 4 transistors are required to form one memory cell, Information is stored as a charge in the gate to substrate capacitance.

15. What is interrupt?

Interrupt is a signal send by external device to the processor so as to request the processor to perform a particular work.

NRCM your roots to success...

16.What is cache memory?

Cache memory is a small high-speed memory. It is used for temporary storage of data & information between the main memory and the CPU (center processing unit). The cache memory is only in RAM.

16. What is called .Scratch pad of computer.?

Cache Memory is scratch pad of computer.

17. Which transistor is used in each cell of EPROM?

Floating .gate Avalanche Injection MOS (FAMOS) transistor is used in each cell of EPROM.

18. Differentiate between RAM and ROM?

RAM: Read / Write memory, High Speed, Volatile Memory. ROM: Read only memory, Low Speed, Non Voliate Memory.

19. What is a compiler?

Compiler is used to translate the high-level language program into machine code at a time. It doesn.t require special instruction to store in a memory, it stores automatically. The Execution time is less compared to Interpreter.

20. Which processor structure is pipelined?

All x86 processors have pipelined structure.

21. What is flag?

Flag is a flip-flop used to store the information about the status of a processor and the status of the instruction executed most recently

22. What is stack?

Stack is a portion of RAM used for saving the content of Program Counter and general purpose registers.

23. Can ROM be used as stack?

ROM cannot be used as stack because it is not possible to write to ROM.

NRCM your roots to success...

24. What is NV-RAM?

Nonvolatile Read Write Memory, also called Flash memory. It is also know as shadow RAM.

NRCM
your roots to success...