UNIT – 5

8051 MICROCONTROLLER

What is a Microcontroller?

A Microcontroller is a programmable digital processor with necessary peripherals. Both microcontrollers and microprocessors are complex sequential digital circuits meant to carry out job according to the program / instructions. Sometimes analog input/output interface makes a part of microcontroller circuit of mixed mode (both analog and digital nature).
A microcontroller can be compared to a Swiss knife with multiple functions incorporated in the same IC.

Fig. 5.1 A Microcontroller compared with a Swiss knife

Microcontrollers Vs Microprocessors

1. A microprocessor requires an external memory for program/data storage. Instruction execution requires movement of data from the external memory to the microprocessor or vice versa. Usually, microprocessors have good computing power and they have higher clock speed to facilitate faster computation.
2. A microcontroller has required on-chip memory with associated peripherals. A microcontroller can be thought of a microprocessor with inbuilt peripherals.
3. A microcontroller does not require much additional interfacing ICs for operation and it functions as a standalone system. The operation of a microcontroller is multipurpose, just like a Swiss knife.
4. Microcontrollers are also called embedded controllers. A microcontroller clock speed is limited only

to a few tens of MHz Microcontrollers are numerous and many of them are application specific.

Development/Classification of microcontrollers (Invisible)

Microcontrollers have gone through a silent evolution (invisible). The evolution can be rightly termed as silent as the impact or application of a microcontroller is not well known to a common user, although microcontroller technology has undergone significant change since early 1970's. Development of some popular microcontrollers is given as follows.

Intel 4004	4 bit (2300 PMOS trans, 108 kHz)	1971
Intel 8048	8 bit	1976
Intel 8031	8 bit (ROM-less)	.
Intel 8051	8 bit (Mask ROM)	1980
Microchip PIC16C64	8 bit	1985
Motorola 68HC11	8 bit (on chip ADC)	.
Intel 80C196	16 bit	1982
Atmel AT89C51	8 bit (Flash memory)	.
Microchip PIC 16F877	8 bit (Flash memory + ADC)	.

Development of microprocessors (Visible)

Microprocessors have undergone significant evolution over the past four decades. This development is clearly perceptible to a common user, especially, in terms of phenomenal growth in capabilities of personal computers. Development of some of the microprocessors can be given as follows.

Intel 4004	4 bit (2300 PMOS transistors)	1971
Intel 8080 8085	8 bit (NMOS) 8 bit	1974
Intel 8088 8086	16 bit 16 bit	1978
Intel 80186 80286	16 bit 16 bit	1982

Intel 80386	32 bit (275000 transistors)	1985
Intel 80486 SX DX	32 bit 32 bit (built in floating point unit)	1989
Intel 80586 I MMX Celeron II III IV	64 bit	1993 1997 1999 2000
Z-80 (Zilog)	8 bit	1976
Motorola Power PC 601 602 603	32-bit	1993 1995

Some of the microcontrollers of 8051 family are given as follows:

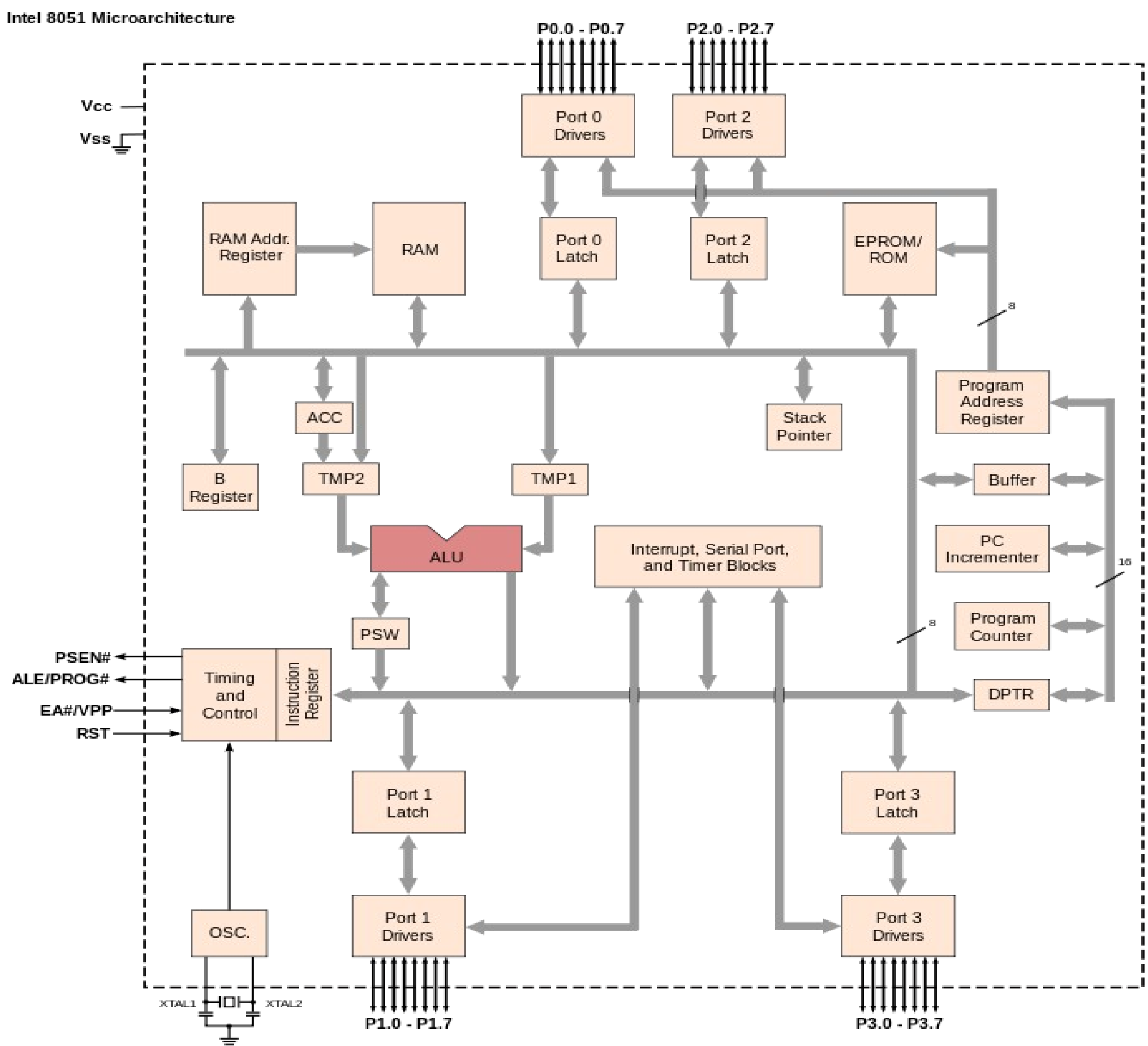
DEVICE	ON-CHIP DATA MEMORY (bytes)	ON-CHIP PROGRAM MEMORY (bytes)	16-BIT TIMER/COUNTER	NO. OF VECTORED INTERUPTS	FULL DUPLEX I/O
8031	128	None	2	5	1
8032	256	none	2	6	1
8051	128	4k ROM	2	5	1
8052	256	8k ROM	3	6	1
8751	128	4k EPROM	2	5	1
8752	256	8k EPROM	3	6	1
AT89C51	128	4k Flash Memory	2	5	1
AT89C52	256	8k Flash memory	3	6	1

Various features of 8051 microcontroller are given as follows:

- 8-bit CPU
- 16-bit Program Counter
- 8-bit Processor Status Word (PSW)
- 8-bit Stack Pointer
- Internal RAM of 128bytes
- Special Function Registers (SFRs) of 128 bytes
- 32 I/O pins arranged as four 8-bit ports (P0 - P3)
- Two 16-bit timer/counters : T0 and T1
- Two external and three internal vectored interrupts
- One full duplex serial I/O

ARCHITECTURE OF 8051 MICROCONTROLLER:

It is 8-bit microcontroller, means MC 8051 can Read, Write and Process 8 bit data. This is mostly used microcontroller in the robotics, home appliances like mp3 player, washing machines, electronic iron and industries. Mostly used blocks in the architecture of 8051 are as follows:



1. Oscillator and clock generator:

All operations in a microcontroller are synchronized by the help of an oscillator clock. The oscillator clock generates the clock pulses by which all internal operations are synchronized. A resonant network connected through pins XTAL1 and XTAL2 forms up an oscillator. For this purpose a quartz crystal and capacitors are employed. The crystal run at specified maximum and minimum frequencies typically at 1 MHz to 16 MHz.

2. ALU:

It is 8 bit unit. It performs arithmetic operation as addition, subtraction, multiplication, division, increment and decrement. It performs logical operations like AND, OR and EX-OR. It manipulates 8 bit and 16 bit data. It calculates address of jump locations in relative branch instruction. It performs compare, rotate and compliment operations. It consists of Boolean processor which performs bit, set, test, clear and compliment. 8051 micro controller contains 34 general purpose registers or working registers.2 of them are called math registers A & B and 32 are bank of registers.

a. Accumulator(A-reg):

It is 8 bit register. Its address is E0H and it is bit and byte accessible. Result of arithmetic & logic operations performed by ALU is accumulated by this register. Therefore it is called accumulator register. It is used to store 8 bit data and to hold one of operand of ALU units during arithmetical and logical operations. Most of the instructions are carried out on accumulator data. It is most versatile of 2 CPU registers.

b. B-register:

It is special 8 bit math register. It is bit and byte accessible. It is used in conjunction with A register as I/P operand for ALU. It is used as general purpose register to store 8 bit data.

c. PSW:

It is 8 bit register. Its address is D0H and It is bit and byte accessible. It has 4 conditional flags or math flags which sets or resets according to condition of result. It has 3 control flags, by setting or resetting bit required operation or function can be achieved. The format of flag register is as shown below:

d. FLAG:

1. Carry Flag(CY): During addition and subtraction any carry or borrow is generated then carry flag is set otherwise carry flag resets. It is used in arithmetic, logical, jump, rotate and Boolean operations.

2. Auxiliary carry flag(AC): If during addition and subtraction any carry or borrow is generated from lower 4 bit to higher 4 bit then AC sets else it resets. It is used in BCD arithmetic operations.

3. Overflow flag(OV): If in signed arithmetic operations result exceeds more than 7 bit than OV flag sets else resets.It is used in signed arithmetic operations only.

4. Parity flag(P): If in result, even no. Of ones "1" are present than it is called even parity and parity flag sets. In result odd no. Of ones "1"are present than it is called odd parity and parity flag resets.

ii. CONTROL FLAGS:

1. FO: It is user defined flag. The user defines the function of this flag. The user can set ,test n clear this flag through software.

2. RS1 and RS0: These flags are used to select bank of register by resetting those flags which are as shown in table :

3. Program counter (PC): The Program Counter (PC) is a 2-byte address which tells the 8051 where the next instruction to execute is found in memory. It is used to hold 16 bit address of internal RAM, external RAM or external ROM locations. When the 8051 is initialized PC always starts at 0000h and is incremented each time an instruction is executed. It is important to note that PC isnt always incremented by one and never decremented.

4. Data pointer register (DTPR): It is a 16 bit register used to hold address of external or internal RAM where data is stored or result is to be stored. It is used to store 16 bit data. It is divided into2- 8bit registers, DPH-data pointer higher order (83H) and DPL-data pointer lower order (82H). Each register can be used as general purpose register to store 8 bit data and can also be used as memory location. DPTR does not have single internal address. It functions as Base register in base relative addressing mode and in-direct jump.

5. Stack pointer (SP): It is 8-bit register. It is byte addressable. Its address is 81H. It is used to hold the internal RAM memory location addresses which are used as stack memory. When the data is to be placed on stack by push instruction, the content of stack pointer is incremented by 1, and when data is retrieved from stack, content of stack of stack pointer is decremented by 1.

iii. Special function Registers(SFR): The 8051 microcontroller has 11 SFR divided in 4 groups:
A. Timer/Counter register: 8051 microcontroller has 2-16 bit Timer/counter registers called Timer-reg-T0 And Timer/counter Reg-T1.Each register is 16 bit register divide into lower and higher byte register as shown below: These register are used to hold initial no. of count. All of the 4 register are byte addressable.

1. Timer control register: 8051 microcontroller has two 8-bit timer control register i.e. TMOD and TCON register. TMOD Register: it is 8-bit register. Its address is 89H. It is byte addressable. It used to select mode and control operation of time by writing control word.

2. TCON register: It is 8-bit register. Its address is 88H. It is byte addressable. Its MSB 4-bit are used to control operation of timer/ counter and LSB 4-bit are used for external interrupt control.

B. Serial data register: 8051 micro controller has 2 serial data register viz. SBUF and SCON.

1. Serial buffer register (SBUF): it is 8-bit register. It is byte addressable .Its address is 99H. It is used to hold data which is to be transferred serially.

2. Serial control register (SCON): it is 8-bit register. It is bit/byte addressable. Its address is 98H. The 8-bit loaded into this register controls the operation of serial communication.

C. Interrupt register: 8051 µC has 2 8-bit interrupt register.

1. Interrupt enable register (IE): it is 8-bit register. It is bit/byte addressable. Its address is A8H.it is used to enable and disable function of interrupt.

2. Interrupt priority register (IP): It is 8-bit register. It is bit/byte addressable. Its address is B8H. it is used to select low or high level priority of each individual interrupts.

D. Power control register (PCON): it is 8-bit register. It is byte addressable .Its address is 87H. Its bits are used to control mode of power saving circuit, either idle or power down mode and also one bit is used to modify baud rate of serial communication.

Internal RAM

Internal RAM has memory 128-byte. See 8051 hardware for further internal RAM design. Internal RAM is organized into three distinct areas: 32 bytes working registers from address 00h to 1Fh 16 bytes bit addressable occupies RAM byte address 20h to 2Fh, altogether 128 addressable bits General purpose RAM from 30h to 7Fh.

Internal ROM

Data memory and program code memory both are in different physical memory but both have the same addresses. An internal ROM occupied addresses from 0000h to 0FFFh. PC addresses program codes from 0000h to 0FFFh. Program addresses higher than 0FFFh that exceed the internal ROM capacity will cause 8051 architecture to fetch codes bytes from external program memory.

28 bytes of Internal RAM Structure (lower address space)

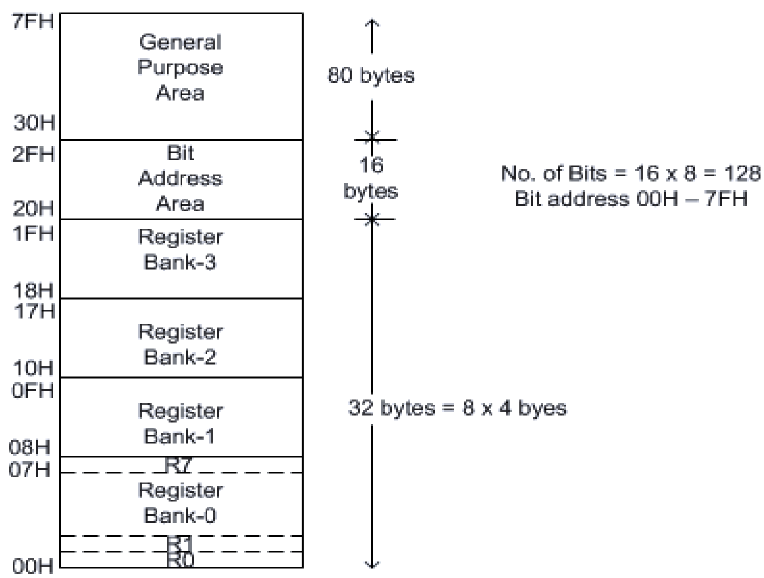


Fig 5.3: Internal RAM Structure

The lower 32 bytes are divided into 4 separate banks. Each register bank has 8 registers of one byte each. A register bank is selected depending upon two bank select bits in the PSW register. Next 16bytes are bit addressable. In total, 128bits (16X8) are available in bitaddressable area. Each bit can be accessed and modified by suitable instructions. The bit addresses are from 00H (LSB of the first byte in 20H) to 7FH (MSB of the last byte in 2FH). Remaining 80bytes of RAM are available for general purpose.

Internal Data Memory and Special Function Register (SFR) Map

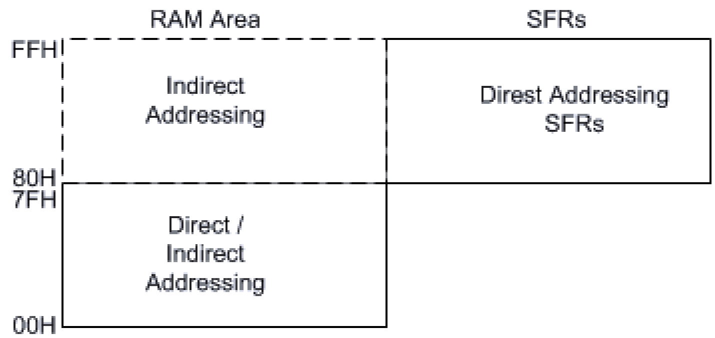


Fig 5.4 : Internal Data Memory Map

The special function registers (SFRs) are mapped in the upper 128 bytes of internal data memory address. Hence there is an address overlap between the upper 128 bytes of data RAM and SFRs. Please note that the upper 128 bytes of data RAM are present only in the 8052 family. The lower128 bytes of RAM (00H - 7FH) can be accessed both by direct or indirect addressing while the upper 128 bytes of RAM (80H - FFH) are accessed by indirect addressing.The SFRs (80H - FFH) are accessed by direct addressing only. This feature distinguishes the upper 128 bytes of memory from the SFRs, as shown in fig 5.4.

SFR Map

The set of Special Function Registers (SFRs) contains important registers such as Accumulator, Register B, I/O Port latch registers, Stack pointer, Data Pointer, Processor Status Word (PSW) and various control registers. Some of these registers are bit addressable (they are marked with a * in the diagram below). The detailed map of various registers is shown in the following figure.

Address

F8H								
F0H	B*							
E8H								
E0H	ACC*							
D8H								
D0H	PSW*							
C8H	(T2CON)*		(RCAP2L)	(RCAP2H)	(TL2)	(TH2)		
C0H								
B8H	IP*							
B0H	P3*							

A8H	IE*							
A0H	P2*							
98H	SCON*	SBUF						
90H	P1*							
88H	TCON*	TMOD	TL0	TL1	TH0	TH1		
80H	P0*	SP	DPL	DPH				PCON

Fig 5.5: SFR Map

It should be noted hat all registers appearing in the first column are bit addressable. The bit address of a bit in the register is calculated as follows.
Bit address of 'b' bit of register 'R' is
Address of register 'R' + b
where $0 \leq b \leq 7$

Processor Status Word (PSW) Address=D0H

CY	AC	F0	RS1	RS0	OV	-	P

Fi g 5.6: Processor Status Word

PSW register stores the important status conditions of the microcontroller. It also stores the bank select bits (RS1 & RS0) for register bank selection.

Power saving modes of operation :

8051 has two power saving modes. They are -

1. Idle Mode
2. Power Down mode.

The two power saving modes are entered by setting two bits IDL and PD in the special function register (PCON) respectively.

The structure of PCON register is as follows.

PCON: Address 87H

SMOD				GF1	GFO	PD	IDL

The schematic diagram for 'Power down' mode and 'Idle' mode is given as follows:

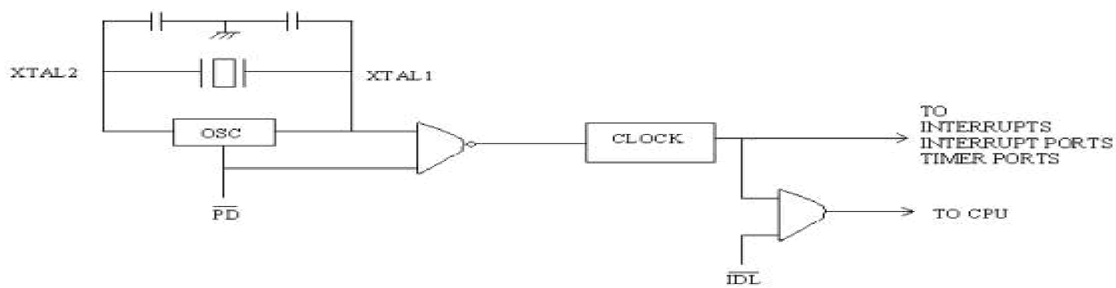


Fig 12.2 Schematic diagram for Power Down and Idle mode implementation

Idle Mode

Idle mode is entered by setting IDL bit to 1 (i.e., $\overline{IDL}$ =0). The clock signal is gated off to CPU, but not to the interrupt, timer and serial port functions. The CPU status is preserved entirely. SP, PC, PSW, Accumulator and other registers maintain their data during IDLE mode. The port pins hold their logical states they had at the time Idle was initiated. ALE and $\overline{PSEN}$ are held at logic high levels.

Ways to exit Idle Mode:

1. Activation of any enabled interrupt will clear PCON.0 bit and hence the Idle Mode is exited. The program goes to the Interrupt Service Routine (ISR). After RETI is executed at the end of the ISR, the next instruction will start from the one following the instruction that enabled Idle Mode.
2. A hardware reset exits the idle mode. The CPU starts from the instruction following the instruction that invoked the 'Idle' mode.

Power Down Mode:

The Power down Mode is entered by setting the PD bit to 1. The internal clock to the entire microcontroller is stopped (frozen). However, the program is not dead. The Power down Mode is exited (PCON.1 is cleared to 0) by Hardware Reset only. The CPU starts from the next instruction where the Power down Mode was invoked. Port values are not changed/ overwritten in power down mode. V_{cc} can be reduced to as low as 2V in Power Down mode. However, V_{cc} has to be restored to normal value before Power Down mode is exited.

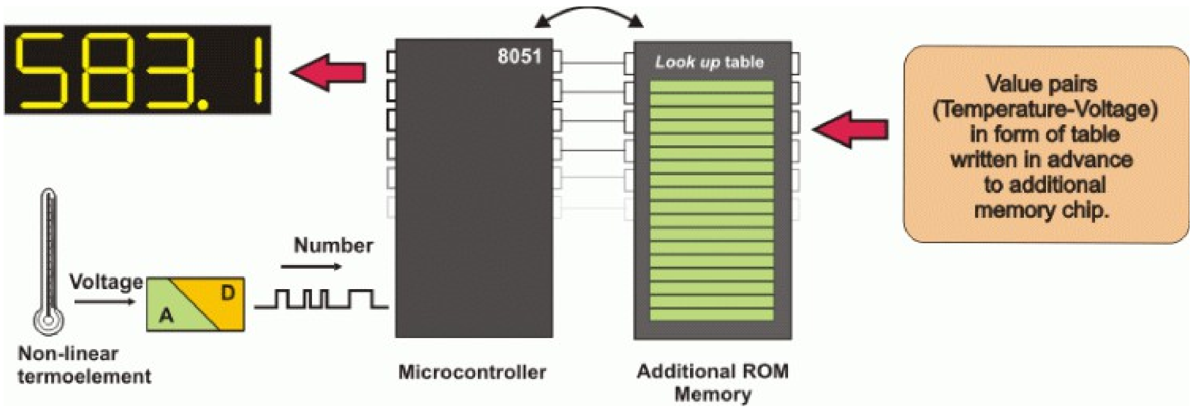
MEMORY ORGANISATION:

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

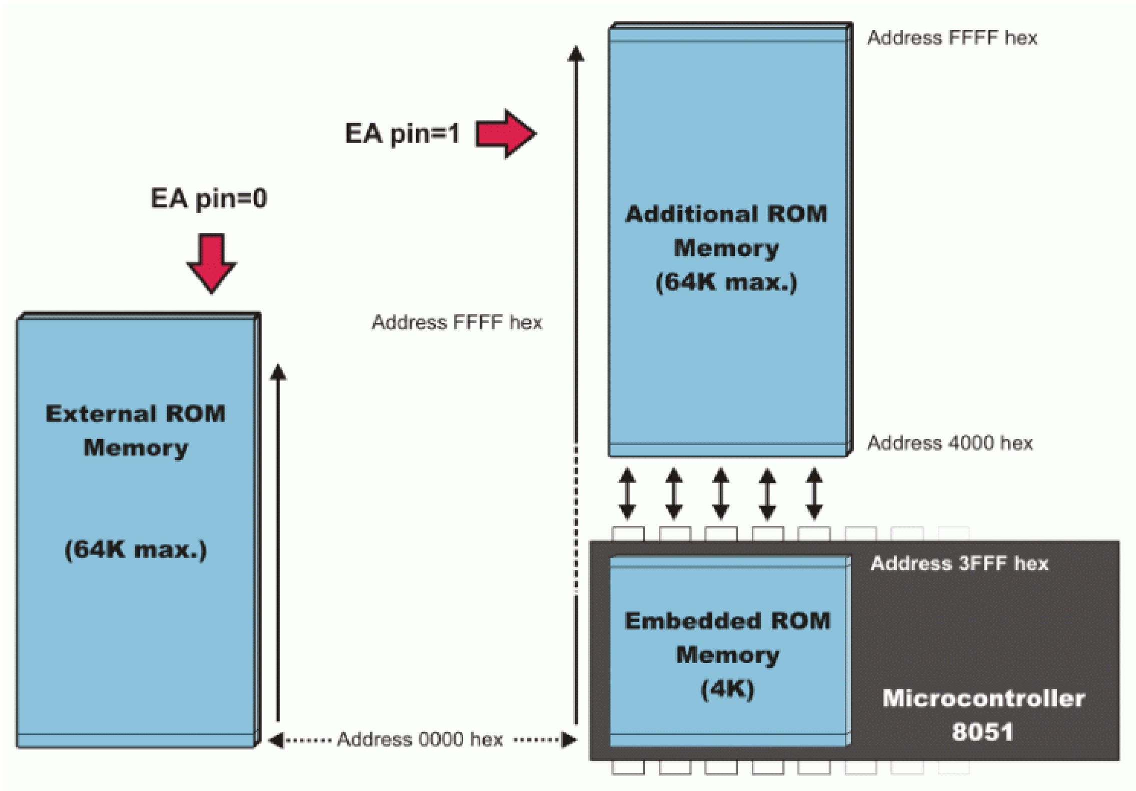
All 8051 microcontrollers have a 16-bit addressing bus and are capable of addressing 64 kb memory. It is neither a mistake nor a big ambition of engineers who were working on basic core development. It is a matter of smart memory organization which makes these microcontrollers a real “programmers' goody“.

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 (similar to logarithmic tables).



How does the microcontroller handle external memory depends on the EA pin logic state:



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

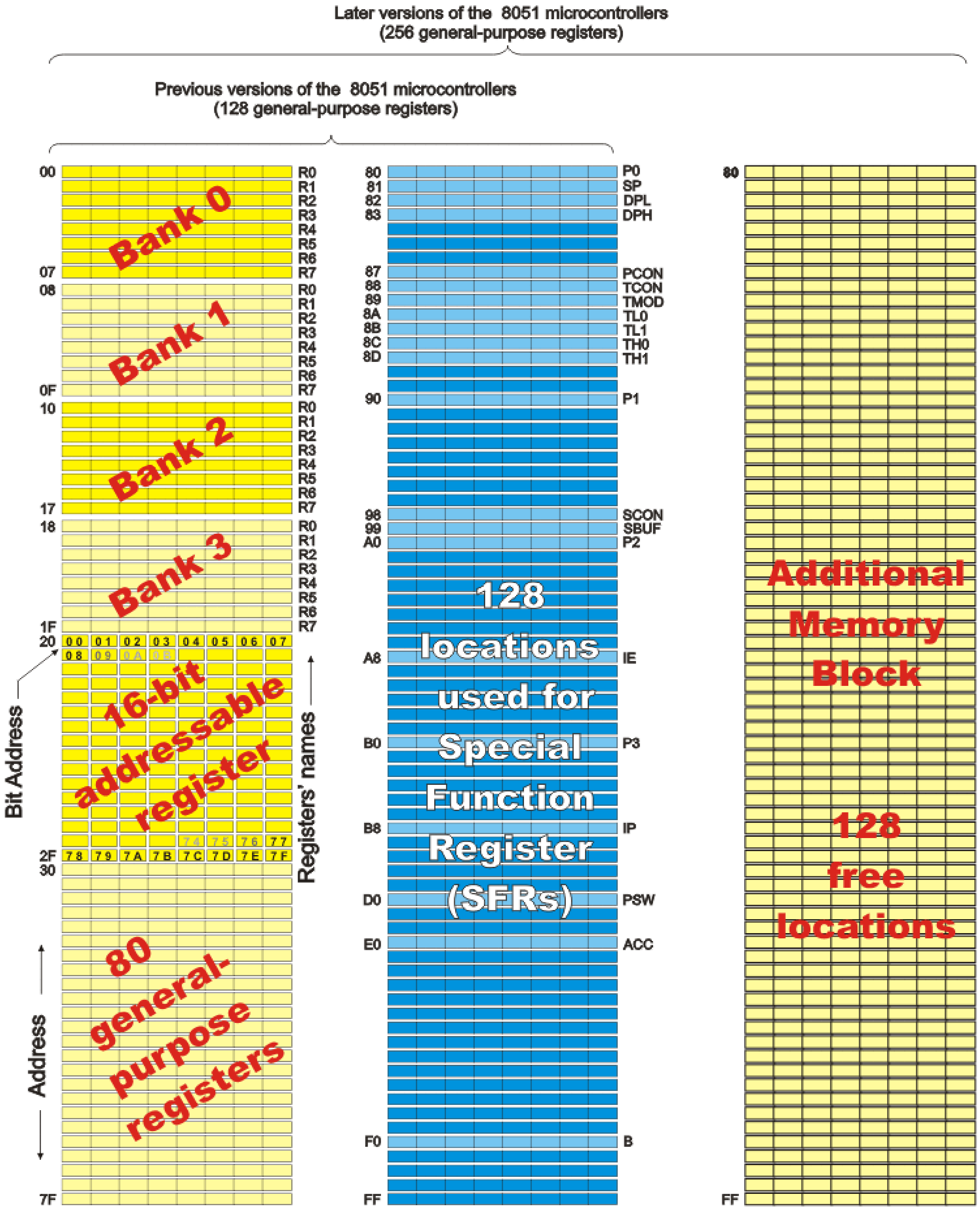
As already mentioned, Data Memory is used for temporarily storing data and intermediate results created and used during the operation of the microcontroller. Besides, RAM memory built in the 8051 family includes many registers such as hardware counters and timers, input/output ports, serial data buffers 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 occupy addresses 2Fh-7Fh, i.e. 80 locations, and does not have any special functions or features.

Additional RAM

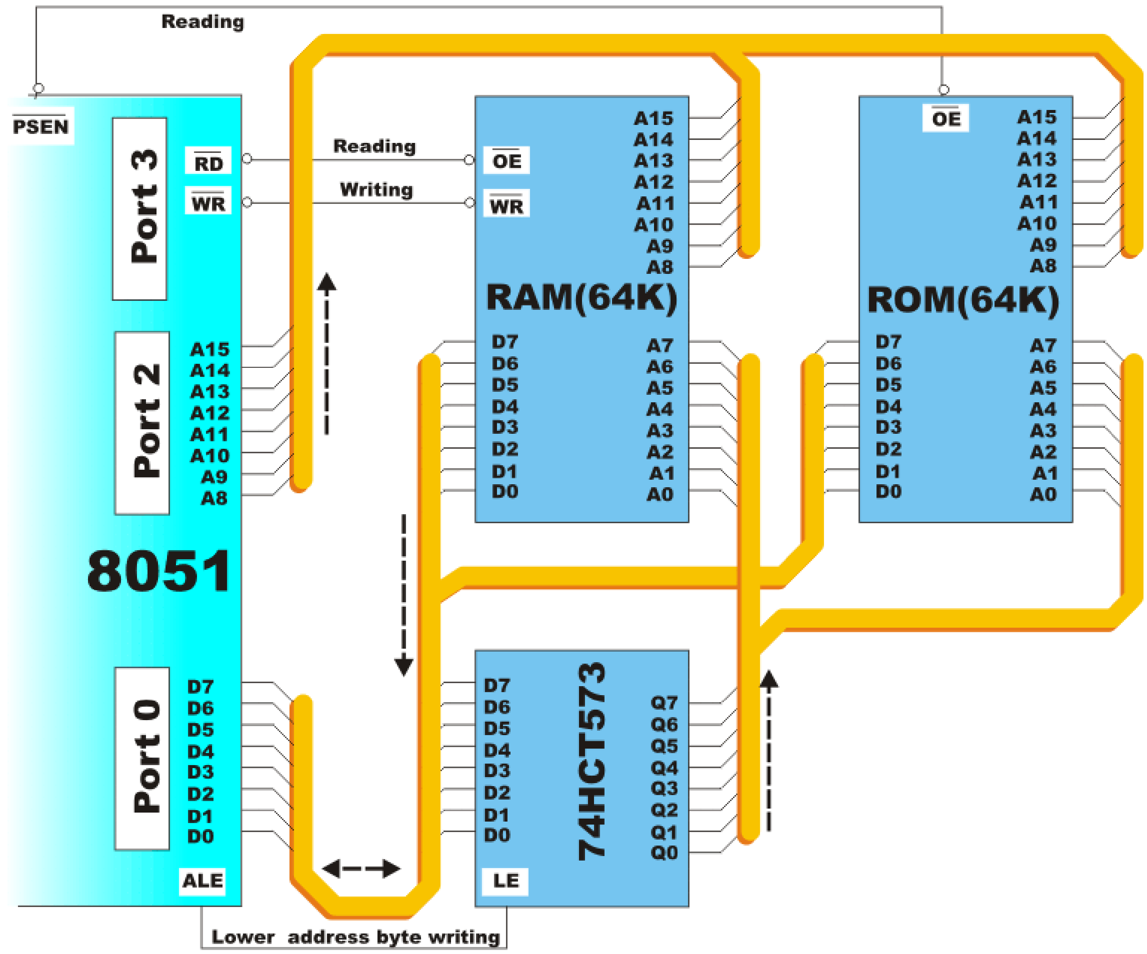
In order to satisfy the programmers' constant hunger for Data Memory, the manufacturers decided to embed an additional memory block of 128 locations into the latest versions of the 8051 microcontrollers. However, it's not as simple as it seems to be… The problem is that electronics performing addressing 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.



Memory expansion

In case memory (RAM or ROM) built in the microcontroller is not sufficient, it is possible to add two external memory chips with capacity of 64Kb each. P2 and P3 I/O ports are used for their addressing and data transmission.



From the user's point of view, everything works quite simply when properly connected because most operations are performed by the microcontroller itself. The 8051 microcontroller has two pins for data read RD#(P3.7) and PSEN#. The first one is used for reading data from external data memory (RAM), while the other is used for reading data from external program memory (ROM). Both pins are active low. A typical example of memory expansion by adding RAM and ROM chips (Hardward architecture), is shown in figure above.

Even though additional memory is rarely used with the latest versions of the microcontrollers, we will describe in short what happens when memory chips are connected according to the previous schematic. The whole process described below is performed automatically.

When the program during execution encounters an instruction which resides in external memory (ROM), the microcontroller will activate its control output ALE and set the first 8 bits of address (A0-A7) on P0. IC circuit 74HCT573 passes the first 8 bits to memory address pins.
A signal on the ALE pin latches the IC circuit 74HCT573 and immediately afterwards 8 higher bits of address (A8-A15) appear on the port. In this way, a desired location of additional program memory is addressed. It is left over to read its content.

Port P0 pins are configured as inputs, the PSEN pin is activated and the microcontroller reads from memory chip.
Similar occurs when it is necessary to read location from external RAM. Addressing is performed in the same way, while read and write are performed via signals appearing on the control outputs RD (is short for read) or WR (is short for write).

Interfacing External Memory

If external program/data memory are to be interfaced, they are interfaced in the following way.

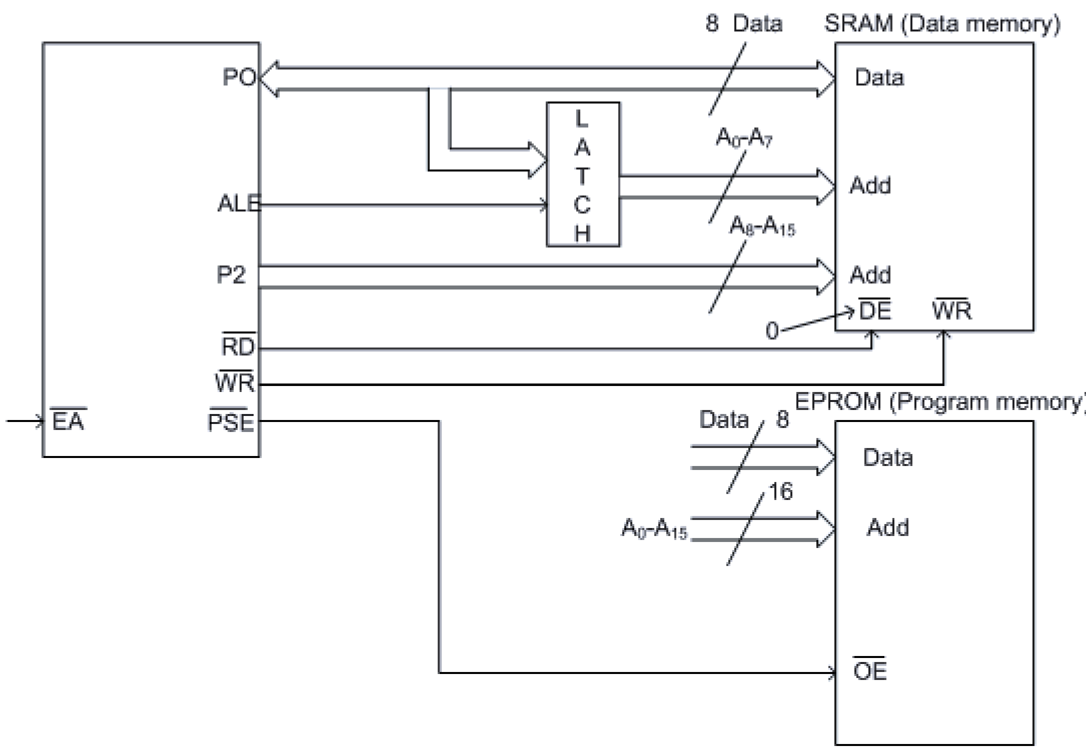


Fig 6.1: Circuit Diagram for Interfacing of External Memory

External program memory is fetched if either of the following two conditions are satisfied.

1. $\overline{EA}$ (Enable Address) is low. The microcontroller by default starts searching for program from external program memory.
2. PC is higher than FFFH for 8051 or 1FFFH for 8052.

$\overline{PSEN}$ tells the outside world whether the external memory fetched is program memory or data memory. $\overline{EA}$

is user configurable. $\overline{PSEN}$ is processor controlled.

8051 INSTRUCTION SET AND PROGRAMMING:

8051 Addressing Modes

8051 has four addressing modes.

1. Immediate Addressing: Data is immediately available in the instruction.
For example -

ADD A, #77; Adds 77 (decimal) to A and stores in A

ADD A, #4DH; Adds 4D (hexadecimal) to A and stores in A

MOV DPTR, #1000H; Moves 1000 (hexadecimal) to data pointer

2. Register Addressing: This way of addressing accesses the bytes in the current register bank. Data is available in the register specified in the instruction. The register bank is decided by 2 bits of Processor Status Word (PSW).
For example-

ADD A, R0; Adds content of R0 to A and stores in A

3. Direct Addressing: The address of the data is available in the instruction.
For example -

MOV A, 088H; Moves content of SFR TCON (address 088H)to A

4. Register Indirect Addressing: The address of data is available in the R0 or R1 registers as specified in the instruction.
For example -

MOV A, @R0 moves content of address pointed by R0 to A

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

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

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

External Code Addressing:
Sometimes we may want to store non-volatile data into the ROM e.g. look-up tables. Such data may require reading the code memory. This may be done as follows -

MOVC A, @A+DPTR; Moves content of address pointed by A+DPTR to A

MOVC A, @A+PC; Moves content of address pointed by A+PC to A

INSTRUCTION SET OF 8051:

The C8051 instructions are divided into five functional groups:

1. Arithmetic operations
2. Logical operations
3. Data transfer operations
4. Boolean variable operations
5. Program branching operation

Arithmetic Operations:

OPCODE	OPERAND	DESCRIPTION	NO. OF BYTES
ADD	A,Rn	Add register to Accumulator	1
ADD	A,direct	Add direct byte to Accumulator	2
ADD	A,@Ri	Add indirect RAM to Accumulator	1
ADD	A,#data	Add immediate data to Accumulator	2
ADDC	A,Rn	Add register to Accumulator with Carry	1
ADDC	A,direct	Add direct byte to Accumulator with Carry	2
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry	1
ADDC	A,#data	Add immediate data to Acc with Carry	2
SUBB	A,Rn	Subtract Register from Acc with borrow	1
SUBB	A,direct	Subtract direct byte from Acc with borrow	2
SUBB	A,@Ri	Subtract indirect RAM from ACC with borrow	1
SUBB	A,#data	Subtract immediate data from Acc with borrow	2
INC	A	Increment Accumulator	1
INC	Rn	Increment register	1

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

Port-0 Pin Structure

Port -0 has 8 pins (P0.0-P0.7).
The structure of a Port-0 pin is shown in fig 6.2.

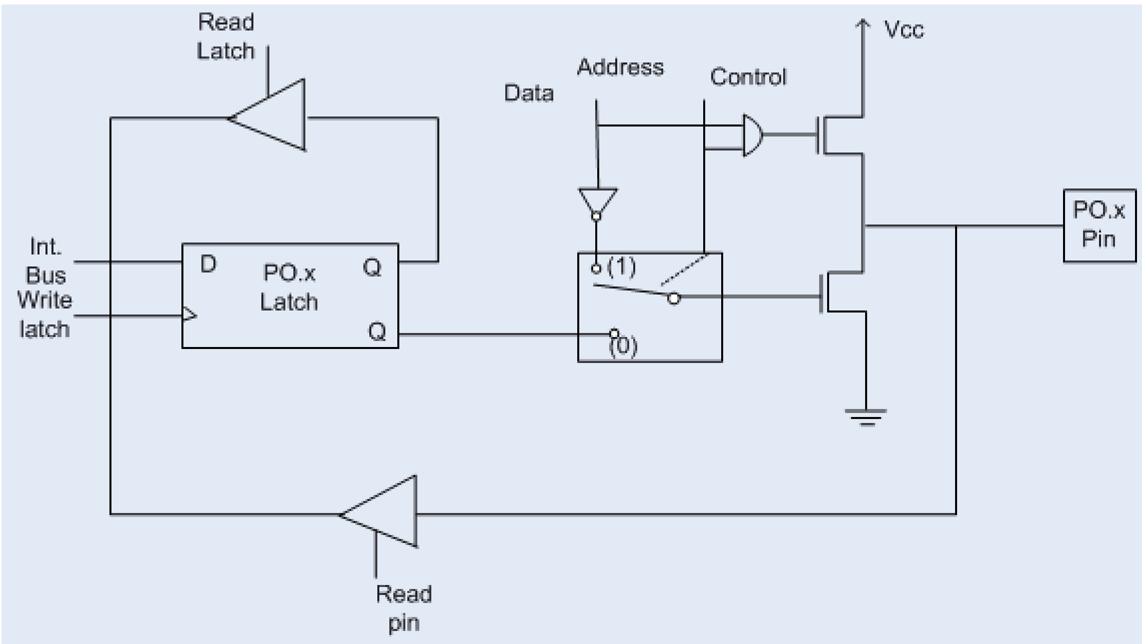


Fig 6.2: Port-0 Structure

Port-0 can be configured as a normal bidirectional I/O port or it can be used for address/data interfacing for accessing external memory. When control is '1', the port is used for address/data interfacing. When the control is '0', the port can be used as a normal bidirectional I/O port.

Let us assume that control is '0'. When the port is used as an input port, '1' is written to the latch. In this situation both the output MOSFETs are 'off'. Hence the output pin floats. This high impedance pin can be pulled up or low by an external source. When the port is used as an output port, a '1' written to the latch again turns 'off' both the output MOSFETs and causes the output pin to float. An external pull-up is required to output a '1'. But when '0' is written to the latch, the pin is pulled down by the lower MOSFET. Hence the output becomes zero.

When the control is '1', address/data bus controls the output driver MOSFETs. If the address/data bus (internal) is '0', the upper MOSFET is 'off' and the lower MOSFET is 'on'. The output becomes '0'. If the address/data bus is '1', the upper transistor is 'on' and the lower transistor is 'off'. Hence the output is '1'. Hence for normal address/data interfacing (for external memory access) no pull-up resistors are required.

Port-0 latch is written to with 1's when used for external memory access.

Port-1 Pin Structure

Port-1 has 8 pins (P1.1-P1.7) .The structure of a port-1 pin is shown in fig 6.3.

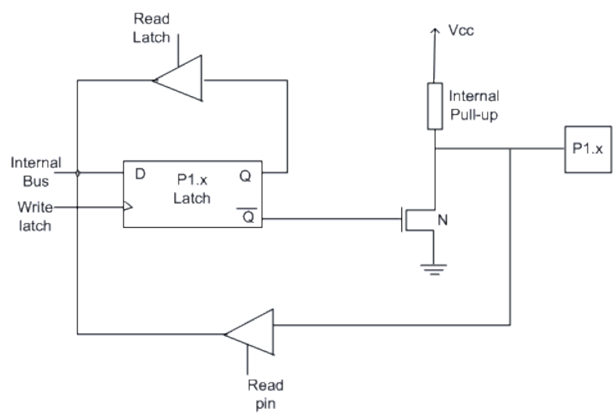


Fig 6.3 Port 1 Structure

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

PORT 2 Pin Structure

Port-2 has 8-pins (P2.0-P2.7) . The structure of a port-2 pin is shown in fig 6.4.

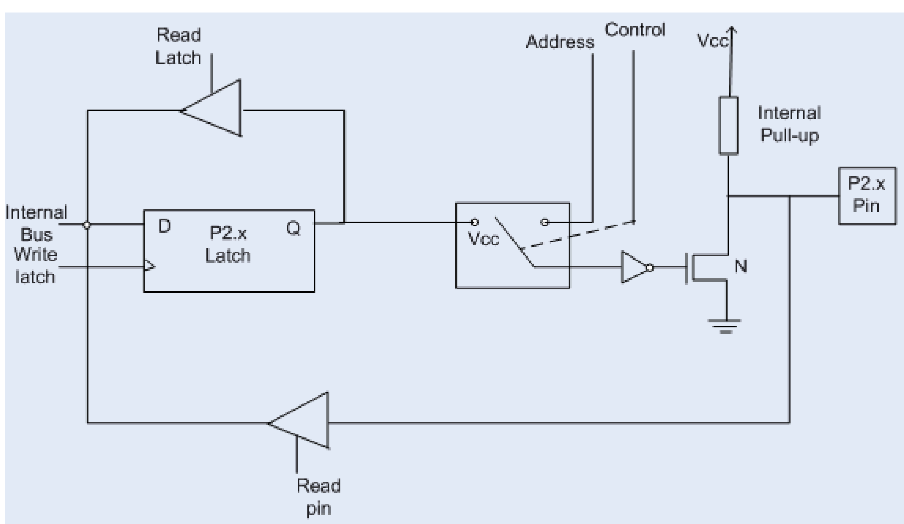


Fig 6.4 Port 2 Structure

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

PORT 3 Pin Structure

Port-3 has 8 pin (P3.0-P3.7) . Port-3 pins have alternate functions. The structure of a port-3 pin is shown in fig 6.5.

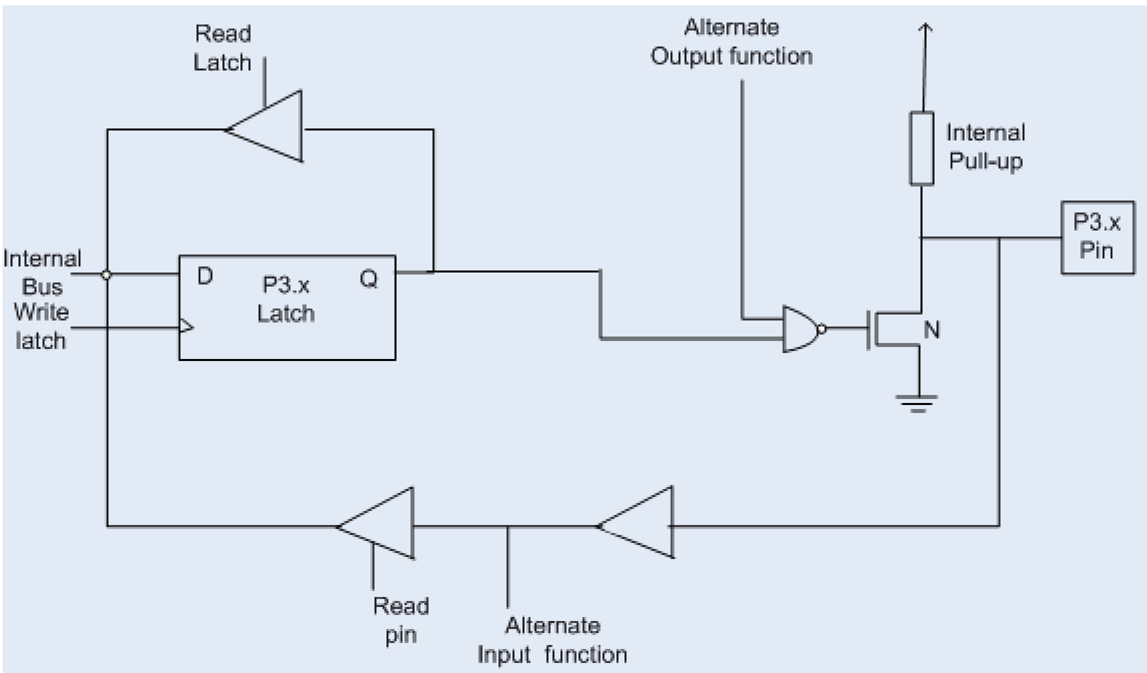


Fig 6.5 Port 3 Structure

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

Alternate functions of Port-3 pins are -

P3.0	RxD
P3.1	TxD
P3.2	$\overline{INT0}$
P3.3	$\overline{INT1}$
P3.4	T0

P3.5	T1
P3.6	$\overline{WR}$
P3.7	$\overline{RD}$

Note:

1. Port 1, 2, 3 each can drive 4 LS TTL inputs.
2. Port-0 can drive 8 LS TTL inputs in address /data mode. For digital output port, it needs external pull-up resistors.
3. Ports-1,2and 3 pins can also be driven by open-collector or open-drain outputs.
4. Each Port 3 bit can be configured either as a normal I/O or as a special function bit.

Reading a port (port-pins) versus reading a latch

There is a subtle difference between reading a latch and reading the output port pin.

The status of the output port pin is sometimes dependant on the connected load. For instance if a port is configured as an output port and a '1' is written to the latch, the output pin should also show '1'. If the output is used to drive the base of a transistor, the transistor turns 'on'.

If the port pin is read, the value will be '0' which is corresponding to the base-emitter voltage of the transistor.

Reading a latch: Usually the instructions that read the latch, read a value, possibly change it, and then rewrite it to the latch. These are called "read-modify-write" instructions. Examples of a few instructions are-

ORL P2, A; P2 <-- P2 or A

MOV P2.1, C; Move carry bit to PX.Y bit.

In this the latch value of P2 is read, is modified such that P2.1 is the same as Carry and is then written back to P2 latch.

Reading a Pin: Examples of a few instructions that read port pin, are-

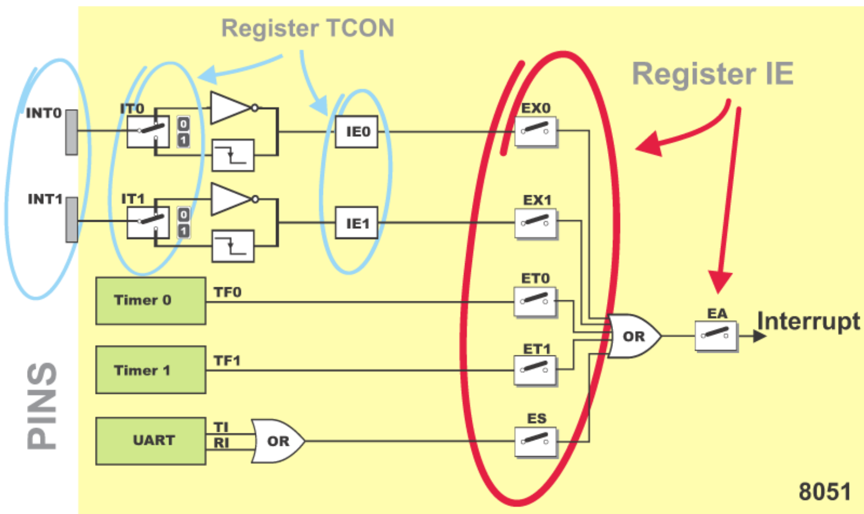
MOV A, P0 ; Move port-0 pin values to A

MOV A, P1; Move port-1 pin values to A

INTERRUPTS OF 8051:

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.



IE Register (Interrupt Enable)

IE	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	
	0	X	0	0	0	0	0	0	Value after Reset
	EA		ET2	ES	ET1	EX1	ET0	EX0	Bit name

- **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 master. When a reset request arrives, everything is stopped and the microcontroller restarts.
2. Interrupt priority 1 can be disabled by Reset only.
3. 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).

IP									
	X	X	0	0	0	0	0	0	Value after Reset
			PT2	PS	PT1	PX1	PT0	PX0	Bit name
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	

- **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:

1. Instruction in progress is ended.
2. The address of the next instruction to execute is pushed on the stack.
3. Depending on which interrupt is requested, one of 5 vectors (addresses) is written to the program counter in accordance to the table below:
4.

INTERRUPT SOURCE	**VECTOR (ADDRESS)**
IE0	3 h
TF0	B h
TF1	1B h
RI, TI	23 h

All addresses are in hexadecimal format

5. These addresses store appropriate subroutines processing interrupts. Instead of them, there are usually jump instructions specifying locations on which these subroutines reside.
6. When an interrupt routine is executed, the address of the next instruction to execute is poped from the stack to the program counter and interrupted program resumes operation from where it left off.

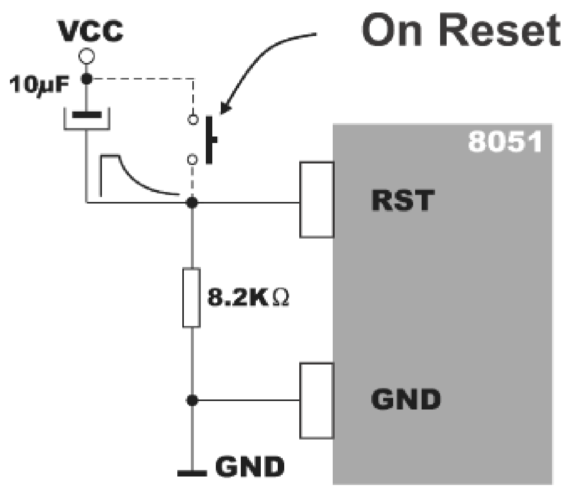
In short

From the moment an interrupt is enabled, the microcontroller is on alert all the time. When an interrupt request arrives, the program execution is stopped, electronics recognizes the source and the program “jumps” to the appropriate address (see the table above). This address usually stores a jump instruction specifying the start of appropriate subroutine. Upon its execution, the 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.

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…



Microcontrollers normally operate at very high speed. The use of 12 Mhz quartz crystal enables 1.000.000 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:

1. External interrupt sensitive to the signal level should be enabled (for example INT0).
2. 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
```

What is going on? 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.

Serial ports on 8051:

Serial Interface

The serial port of 8051 is full duplex, i.e., it can transmit and receive simultaneously.

The register SBUF is used to hold the data. The special function register SBUF is physically two registers. One is, write-only and is used to hold data to be transmitted out of the 8051 via TXD. The other is, read-only and holds the received data from external sources via RXD. Both mutually exclusive registers have the same address 099H.

Serial Port Control Register (SCON)

Register SCON controls serial data communication.
Address: 098H (Bit addressable)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Mode select bits

SM0	SM1	Mode
0	0	Mode 0
0	1	Mode 1
1	0	Mode 2
1	1	Mode 3

SM2: multi processor communication bit
REN: Receive enable bit
TB8: Transmitted bit 8 (Normally we have 0-7 bits transmitted/received)
RB8: Received bit 8
TI: Transmit interrupt flag
RI: Receive interrupt flag

Power Mode control Register

Register PCON controls processor powerdown, sleep modes and serial data bandrate. Only one bit of PCON is used with respect to serial communication. The seventh bit (b7)(SMOD) is used to generate the baud rate of serial communication.

Address: 87H

b7							b0
SMOD	—	—	—	GF1	GF0	PD	IDL

SMOD: Serial baud rate modify bit
GF1: General purpose user flag bit 1
GF0: General purpose user flag bit 0
PD: Power down bit
IDL: Idle mode bit

Data Transmission

Transmission of serial data begins at any time when data is written to SBUF. Pin P3.1 (Alternate function bit TXD) is used to transmit data to the serial data network. TI is set to 1 when data has been transmitted. This signifies that SBUF is empty so that another byte can be sent.

Data Reception

Reception of serial data begins if the receive enable bit is set to 1 for all modes. Pin P3.0 (Alternate function bit RXD) is used to receive data from the serial data network. Receive interrupt flag, RI, is set after the data has been received in all modes. The data gets stored in SBUF register from where it can be read.

Serial Data Transmission Modes:

Mode-0**:** In this mode, the serial port works like a shift register and the data transmission works synchronously with a clock frequency of f_{osc} /12. Serial data is received and transmitted through RXD. 8 bits are transmitted/ received aty a time. Pin TXD outputs the shift clock pulses of frequency f_{osc} /12, which is connected to the external circuitry for synchronization. The shift frequency or baud rate is always 1/12 of the oscillator frequency.

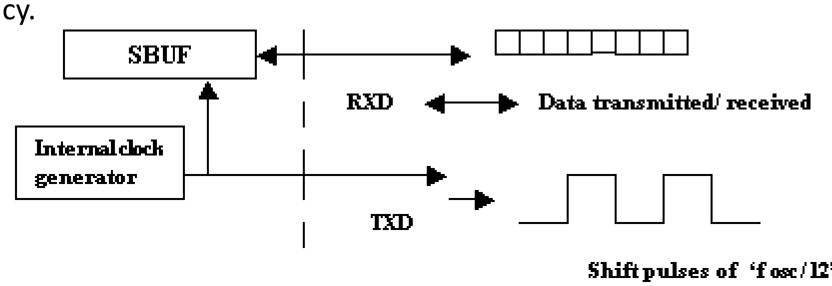


Fig 11.1 Data transmission/reception in Mode-0

Mode-1 (standard UART mode) **:**

In mode-1, the serial port functions as a standard Universal Asynchronous Receiver Transmitter (UART) mode. 10 bits are transmitted through TXD or received through RXD. The 10 bits consist of one start bit (which is usually '0'), 8 data bits (LSB is sent first/received first), and a stop bit (which is usually '1'). Once received, the stop bit goes into RB8 in the special function register SCON. The baud rate is variable.

The following figure shows the way the bits are transmitted/ received.

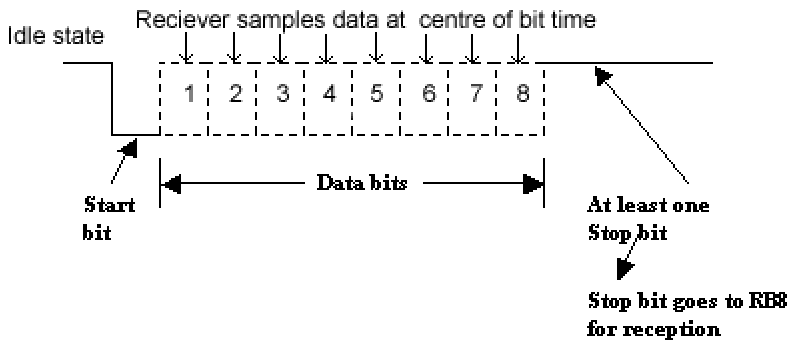


Fig 11.2 Data transmission format in UART mode

Bit time= $1/f_{baud}$

In receiving mode, data bits are shifted into the receiver at the programmed baud rate. The data word (8-bits) will be loaded to SBUF if the following conditions are true.

1. RI must be zero. (i.e., the previously received byte has been cleared from SBUF)
2. Mode bit SM2 = 0 or stop bit = 1.

After the data is received and the data byte has been loaded into SBUF, RI becomes one.

Mode-1 baud rate generation:

Timer-1 is used to generate baud rate for mode-1 serial communication by using overflow flag of the timer to determine the baud frequency. Timer-1 is used in timer mode-2 as an auto-reload 8-bit timer. The data rate is generated by timer-1 using the following formula.

$$f_{baud} = \frac{2^{SMOD}}{32} \times \frac{fosc}{12 \times [256-(TH1)]}$$

Where,

SMOD is the 7th bit of PCON register
f_{osc} is the crystal oscillator frequency of the microcontroller

It can be noted that f_{osc}/ (12 X [256- (TH1)]) is the timer overflow frequency in timer mode-2, which is the auto-reload mode.

If timer-1 is not run in mode-2, then the baud rate is,

$$f_{baud} = \frac{2^{SMOD}}{32} \times \text{(timer-1 overflow frequency)}$$

Timer-1 can be run using the internal clock, fosc/12 (timer mode) or from any external source via pin T1

(P3.5) (Counter mode).

Example: If standard baud rate is desired, then 11.0592 MHz crystal could be selected. To get a standard 9600 baud rate, the setting of TH1 is calculated as follows.

Assuming SMOD to be '0'

$$9600 = \frac{2^0}{32} \times \frac{11.0592 \times 10^6}{12 \times (256\text{-TH1})}$$

Or,

$$256\text{-TH1} = \frac{1}{32} \times \frac{11.0592 \times 10^6}{12 \times 9600} = 3$$

Or,

$$TH1 = 256 - 3 = 253 = FDH$$

In mode-1, if SM2 is set to 1, no receive interrupt (RI) is generated unless a valid stop bit is received.

Serial Data Mode-2 - Multiprocessor Mode :

In this mode 11 bits are transmitted through TXD or received through RXD. The various bits are as follows: a start bit (usually '0'), 8 data bits (LSB first), a programmable 9^{th} (TB8 or RB8)bit and a stop bit (usually '1').

While transmitting, the 9^{th} data bit (TB8 in SCON) can be assigned the value '0' or '1'. For example, if the information of parity is to be transmitted, the parity bit (P) in PSW could be moved into TB8. On reception of the data, the 9^{th} bit goes into RB8 in 'SCON', while the stop bit is ignored. The baud rate is programmable to either 1/32 or 1/64 of the oscillator frequency.

$$f_{baud} = (2^{SMOD}/64)\, f_{osc}.$$

Mode-3 - Multi processor mode with variable baud rate :

In this mode 11 bits are transmitted through TXD or received through RXD. The various bits are: a start bit (usually '0'), 8 data bits (LSB first), a programmable 9 th bit and a stop bit (usually '1').

Mode-3 is same as mode-2, except the fact that the baud rate in mode-3 is variable (i.e., just as in mode-1).

$$f_{baud} = (2^{SMOD}/32) * (f_{osc} / 12\,(256\text{-TH1}))\ .$$

This baud rate holds when Timer-1 is programmed in Mode-2.

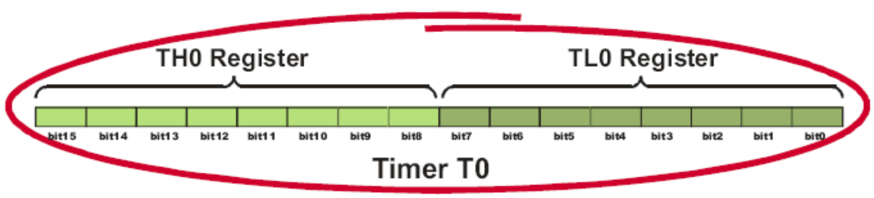
Counters and Timers:

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.

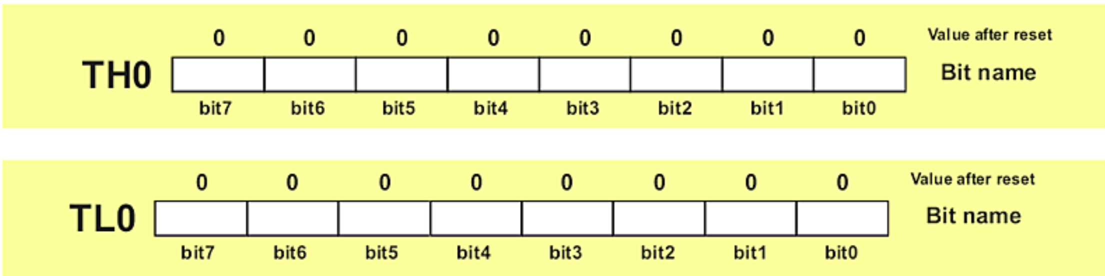
The 8051 microcontroller has 2 timers/counters called T0 and T1. As their names suggest, their main purpose is to measure time and count external events. Besides, they can be used for generating clock pulses to be used in serial communication, so called Baud Rate.

Timer T0

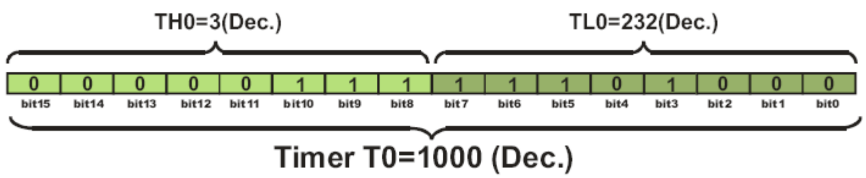
As seen in figure below, the timer T0 consists of two registers – TH0 and TL0 representing a low and a high byte of one 16-digit binary number.



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.



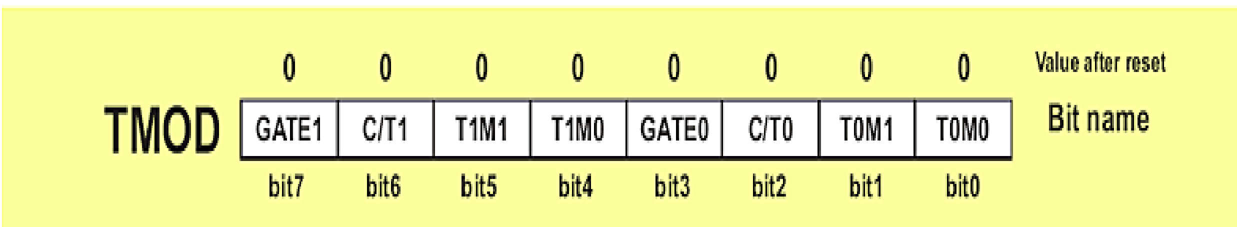
Formula used to calculate values in these two registers is very simple:
TH0 × 256 + TL0 = T
Matching the previous example it would be as follows:
3 × 256 + 232 = 1000



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.



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

- **GATE0** enables and disables Timer 1 using a signal brought to the INT0 pin (P3.2):

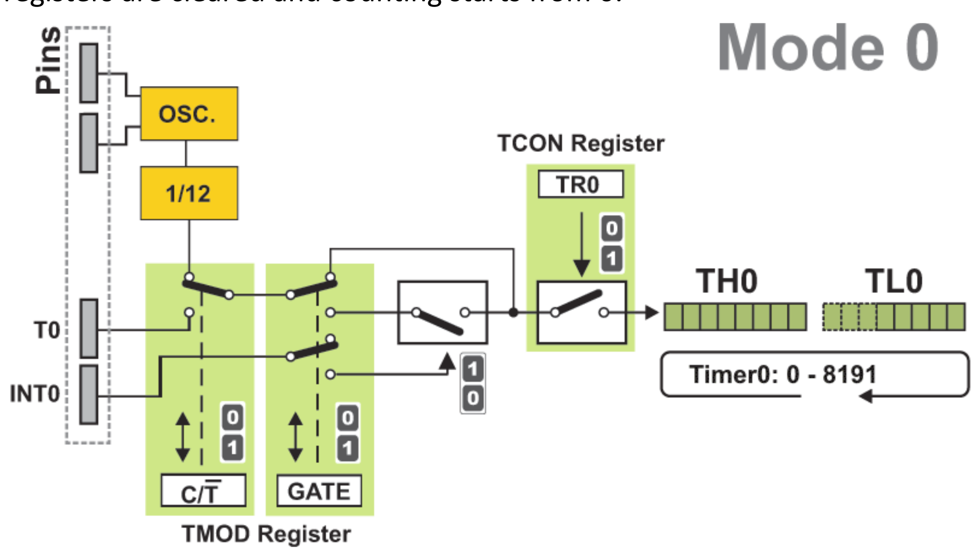
- **1** – Timer 0 operates only if the INT0 bit is set.
- **0** – Timer 0 operates regardless of the logic state of the INT0 bit.

- **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.

T0M1	T0M0	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

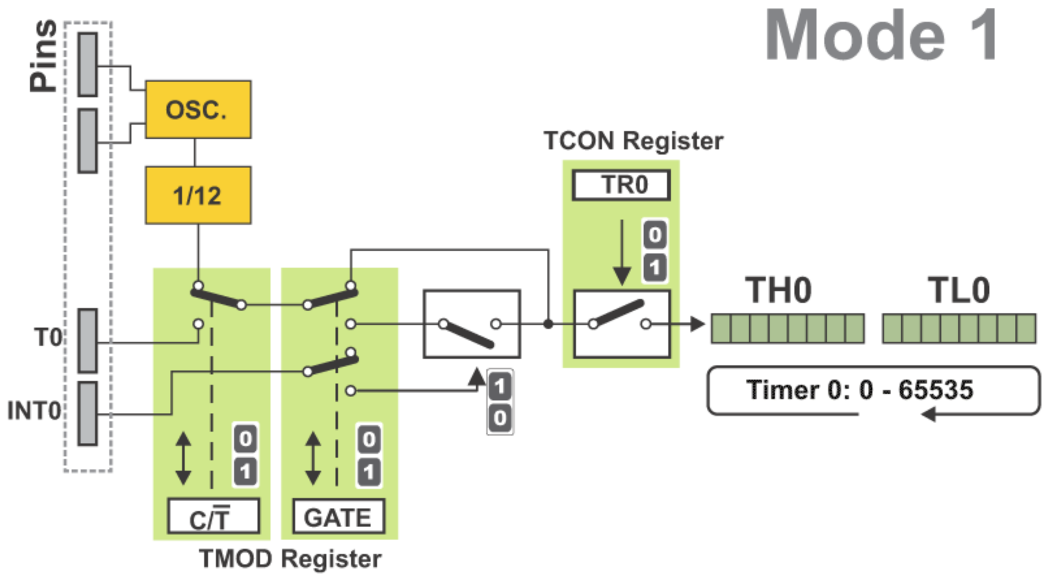
Timer 0 in mode 0 (13-bit timer)

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.



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.

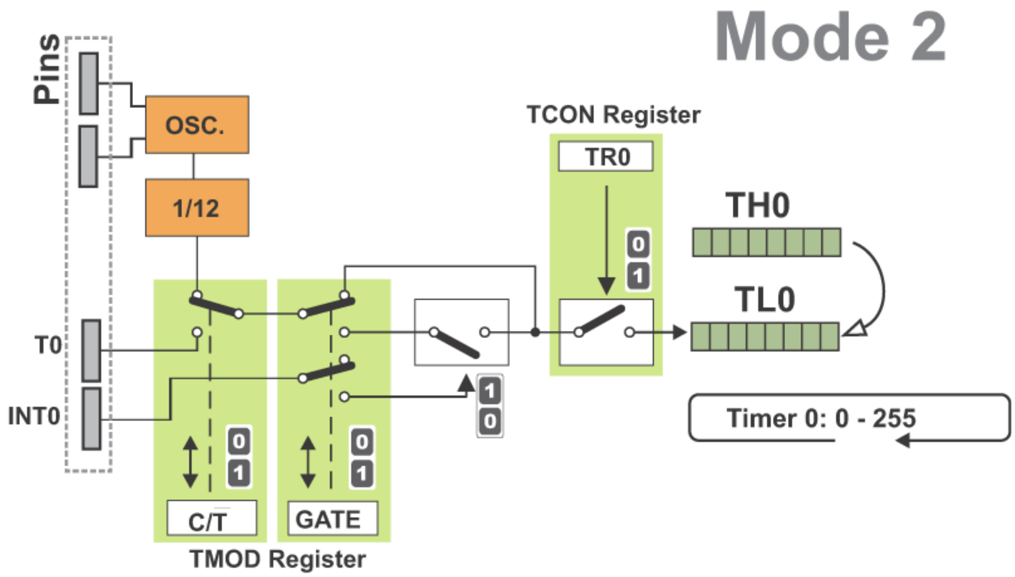


Timer 0 in mode 2 (Auto-Reload Timer)

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.

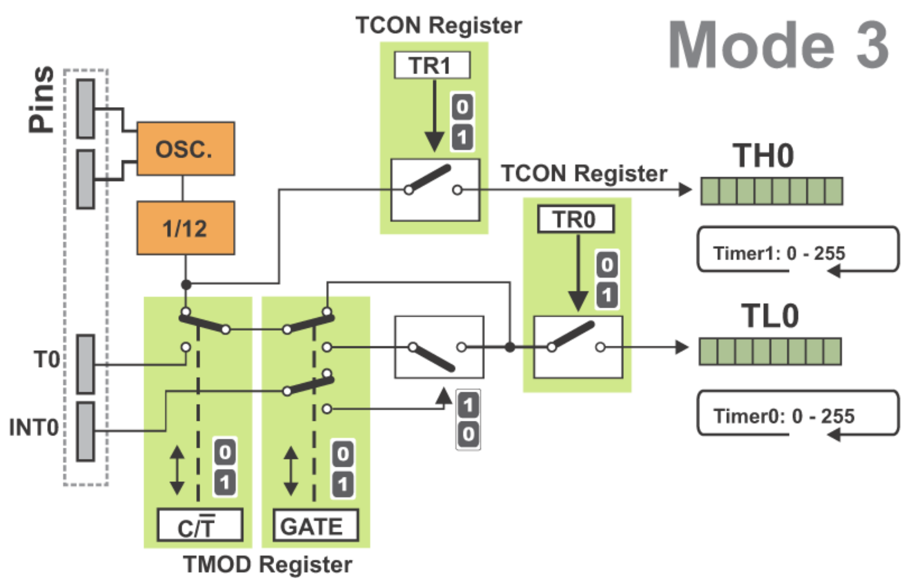
The following example shows the advantages of this mode. Suppose it is necessary to constantly count up 55 pulses generated by the clock.

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.



Timer 0 in Mode 3 (Split Timer)

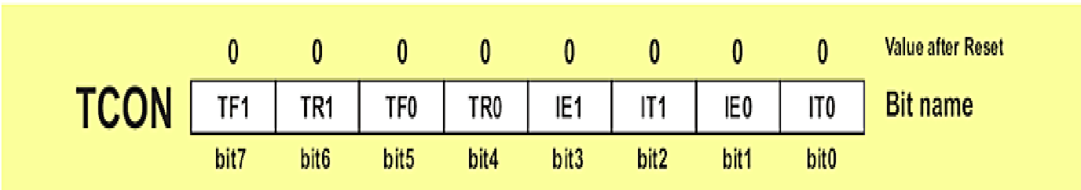
Mode 3 configures timer 0 so that registers TL0 and TH0 operate as separate 8-bit timers. In other words, the 16-bit timer consisting of two registers TH0 and TL0 is split into two independent 8-bit timers. This mode is provided for applications requiring an additional 8-bit timer or counter. The TL0 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.



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

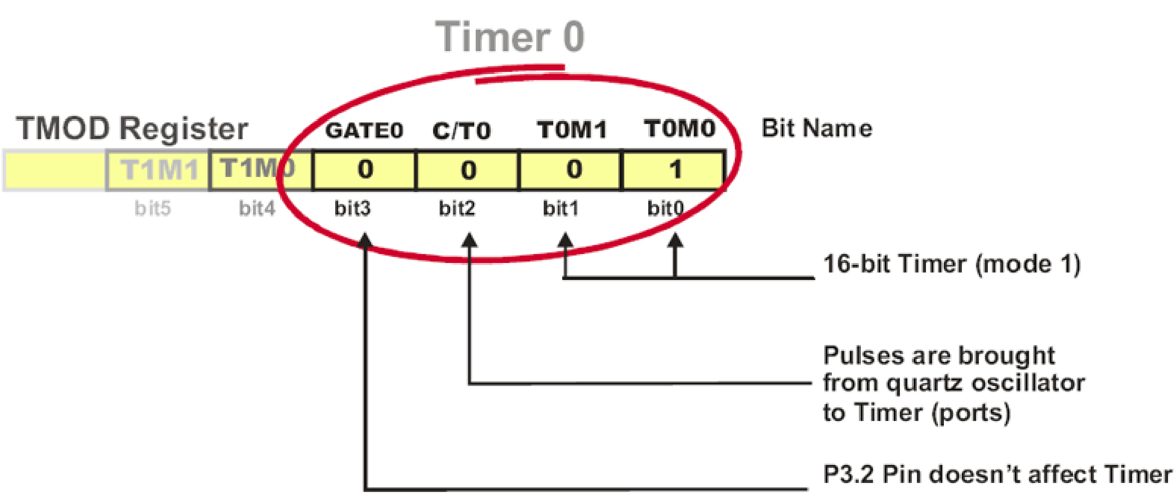
TCON register is also one of the registers whose bits are directly in control of timer operation. Only 4 bits of this register are used for this purpose, while rest of them is used for interrupt control to be discussed later.



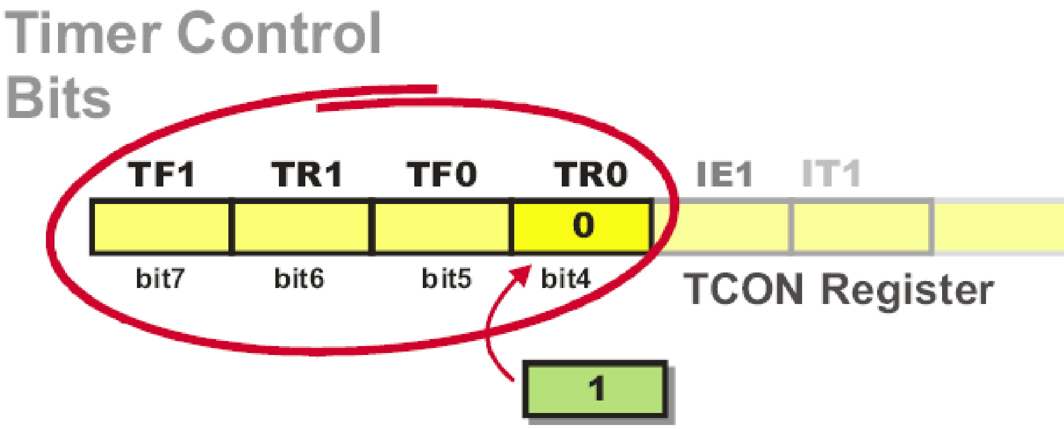
- **TF1** bit is automatically set on the Timer 1 overflow.
- **TR1** bit enables the Timer 1.
  - **1** – Timer 1 is enabled.
  - **0** – Timer 1 is disabled.
- **TF0** bit is automatically set on the Timer 0 overflow.
- **TR0** bit enables the timer 0.
  - **1** – Timer 0 is enabled.
  - **0** – Timer 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

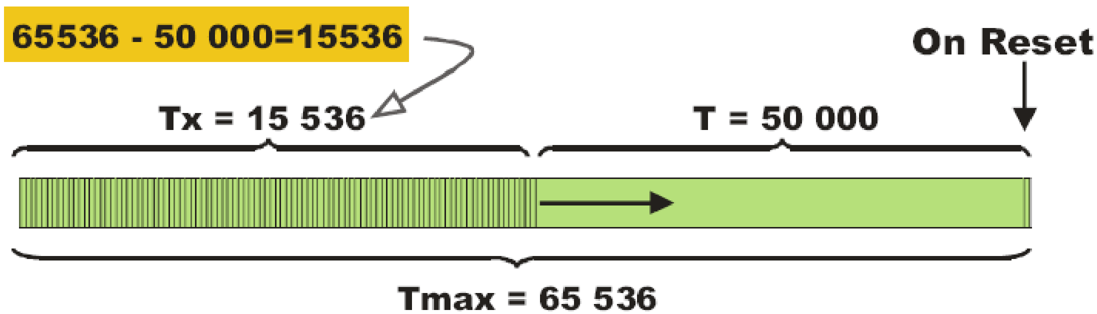
In case of negligence, such an error in counting (255 pulses) may occur for not so obvious but quite logical reason. The lower byte is correctly read (255), but at the moment the program counter was about to read the higher byte TH0, an overflow occurred and the contents of both registers have been changed (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.

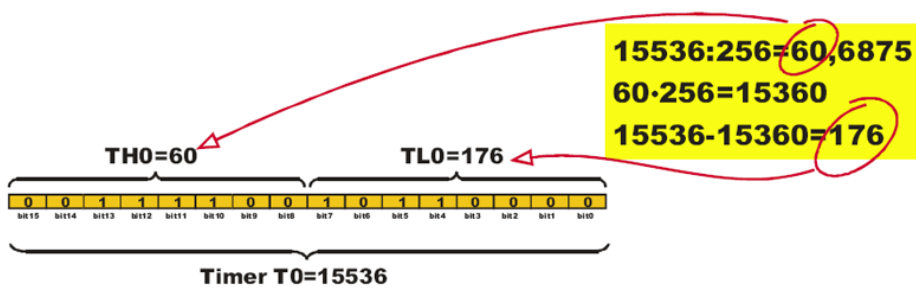
Timer 0 Overflow Detection

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:



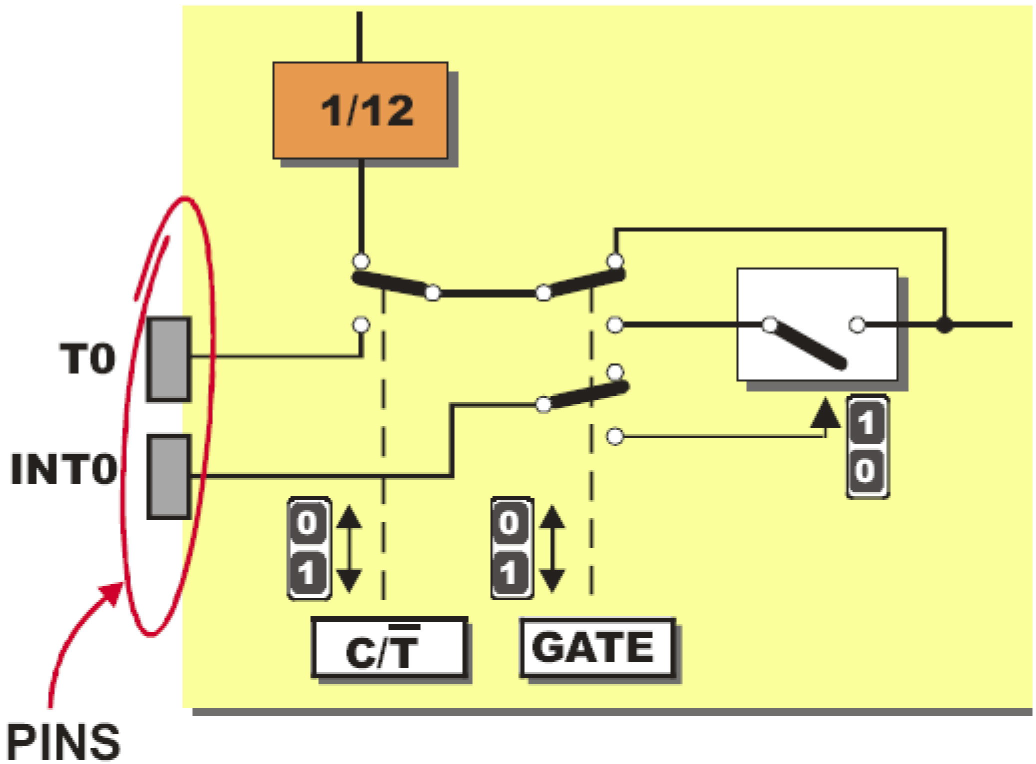
Then it should be written to the timer registers TH0 and TL0:



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?

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.

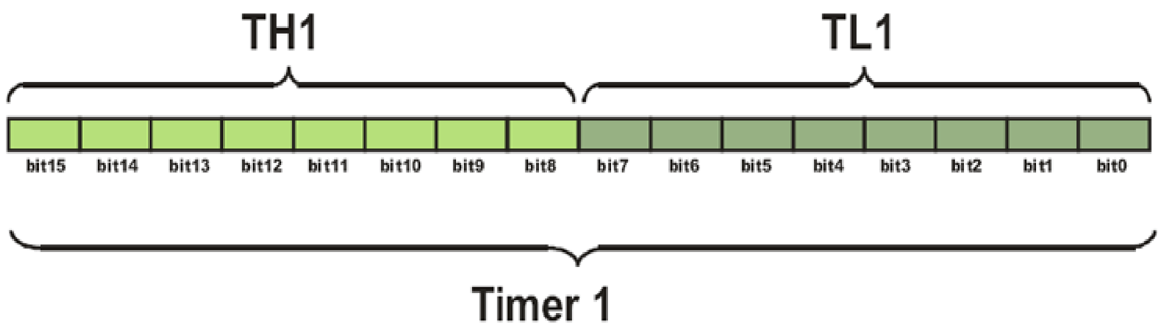


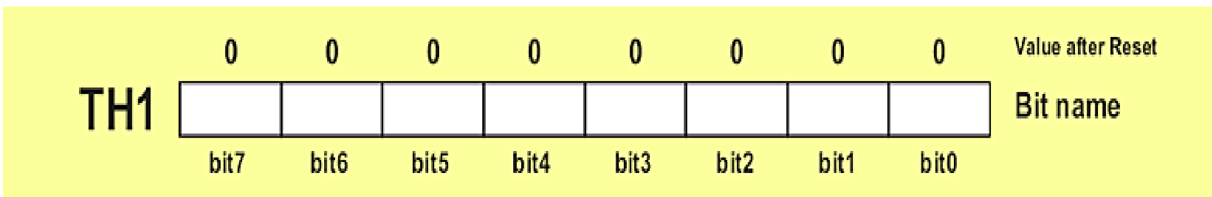
How to count up pulses?

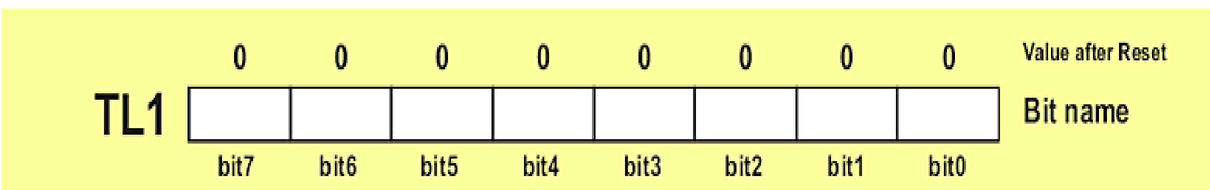
Similarly to the previous example, the answer to this question again lies in the TCON register. This time it's about the C/T0 bit. If the bit is cleared the timer counts pulses generated by the internal oscillator, i.e. measures the time passed. If the bit is set, the timer input is provided with pulses 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.


TH1
TL1
bit15 bit14 bit13 bit12 bit11 bit10 bit9 bit8 bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
Timer 1


0 0 0 0 0 0 0 0
Value after Reset
TH1
Bit name
bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0


0 0 0 0 0 0 0 0
Value after Reset
TL1
Bit name
bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0