UNIT-2

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.

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

Block Diagram

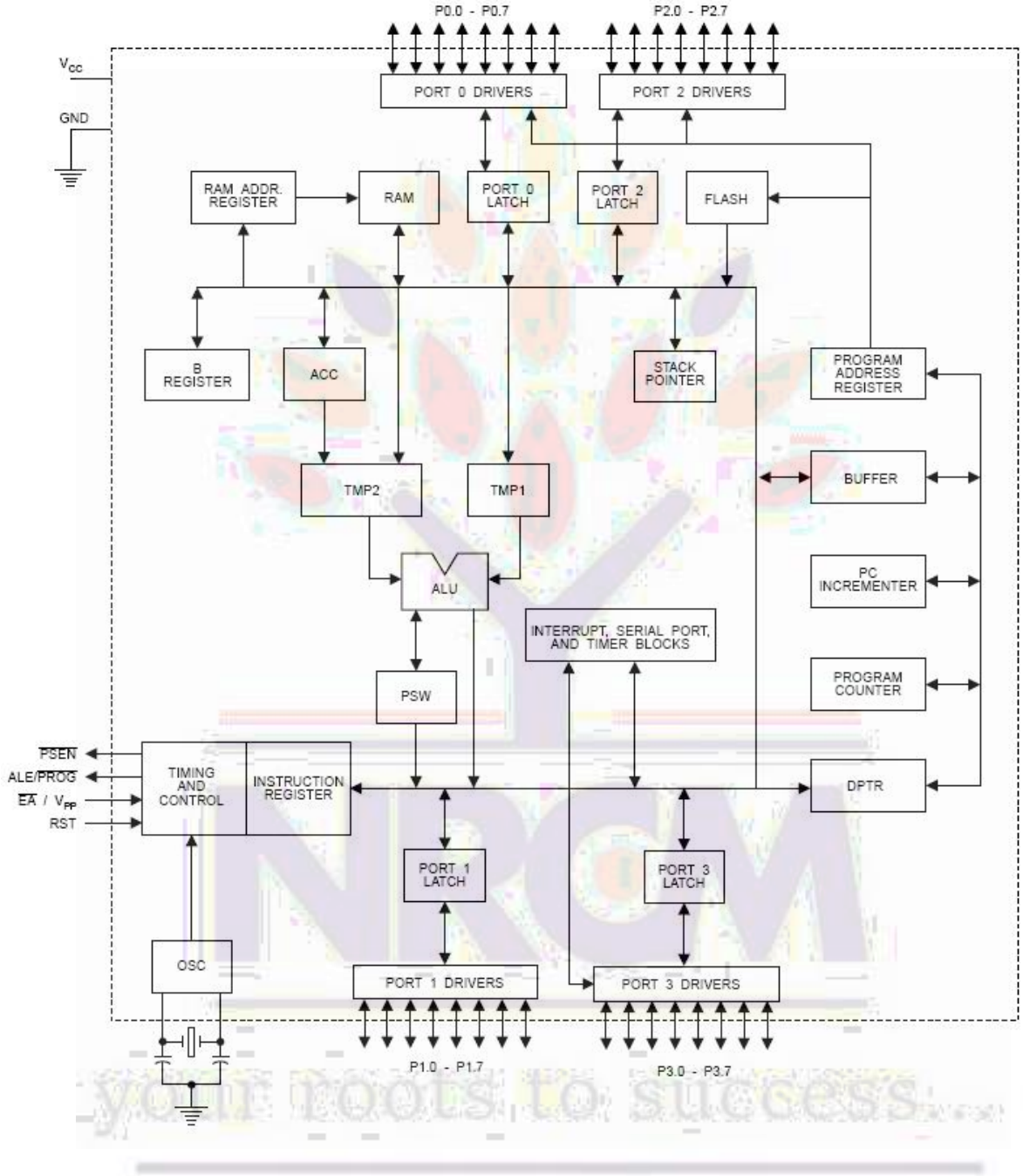


Fig 6.1. 8051 Architecture

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

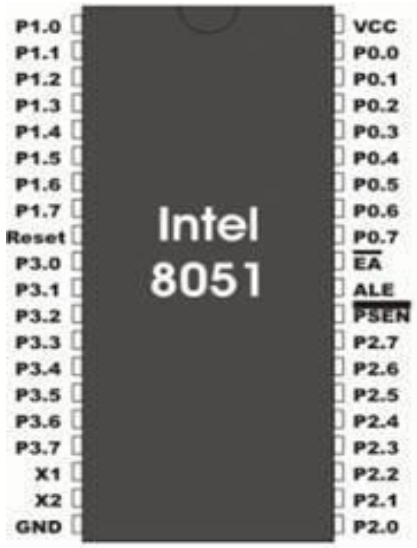


Fig 6.2. 8051 pin diagram

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

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

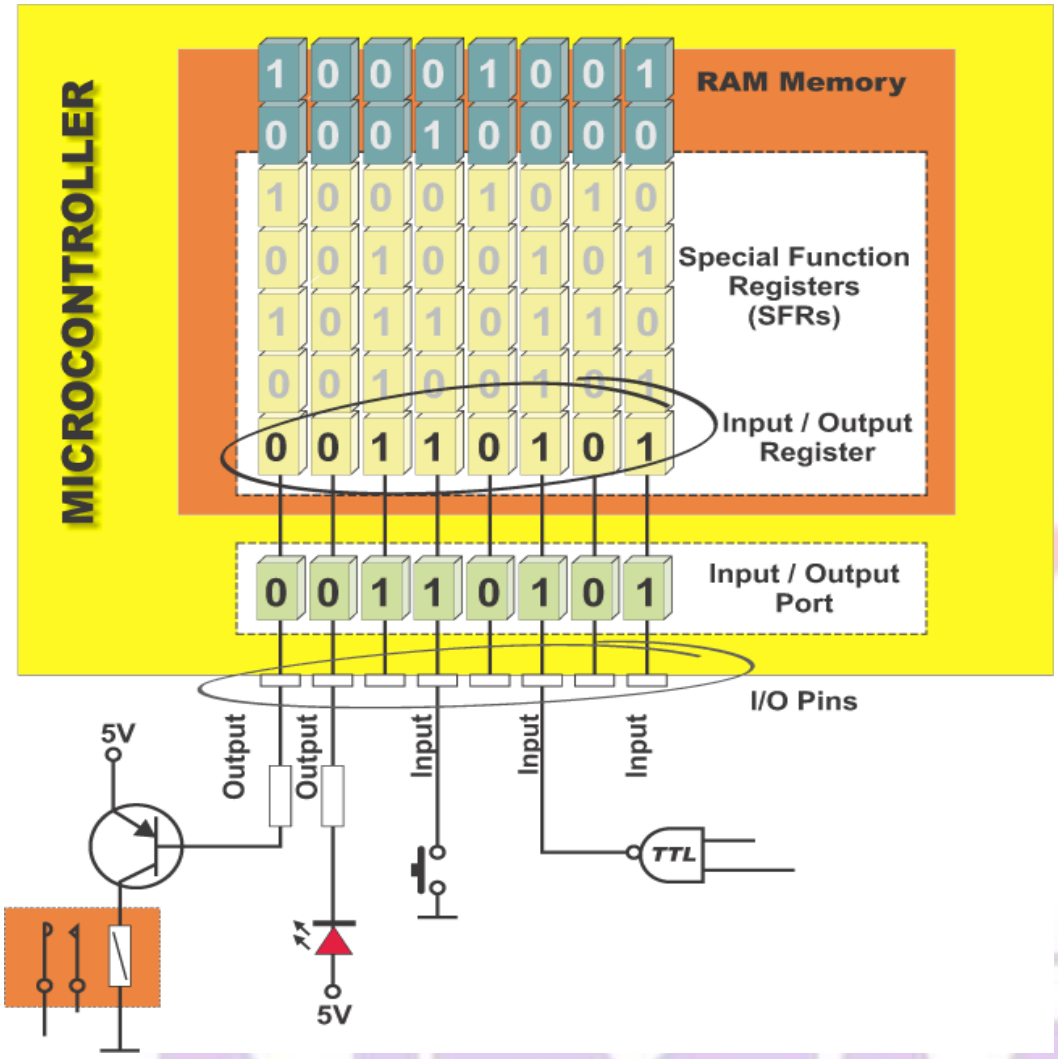
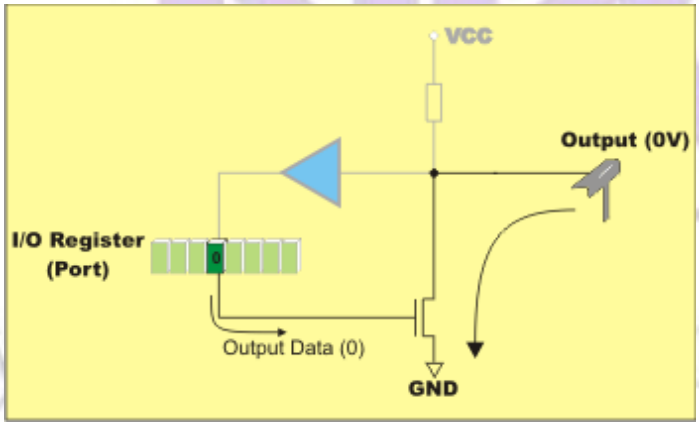


Fig 6.3. input/ output pin structure

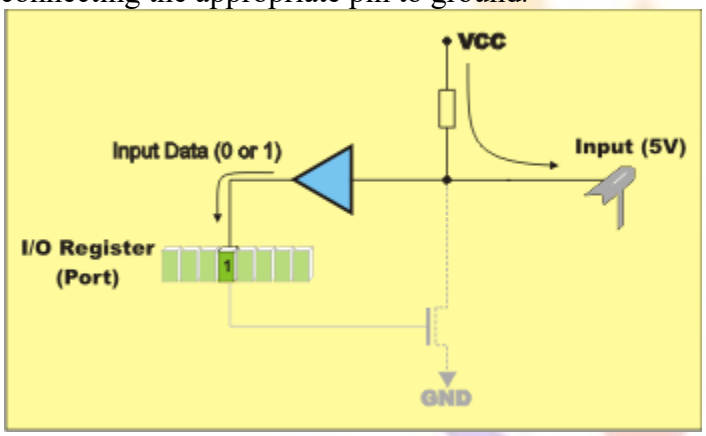


Input /Output (I/O)pin

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

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.



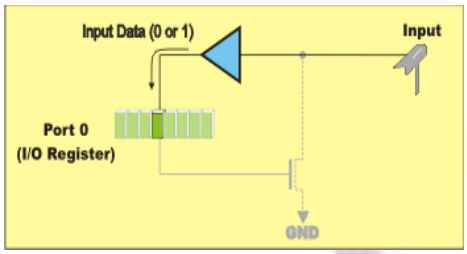
Input pin

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.

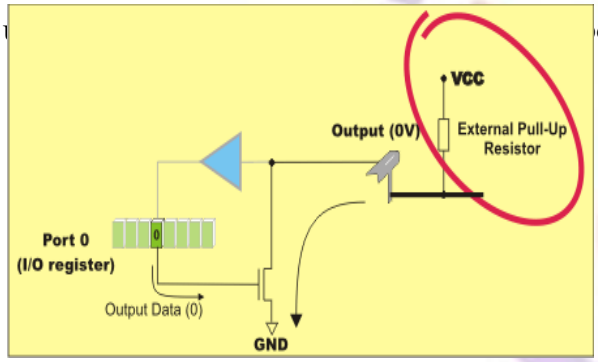
Port 0

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 otential.

- 

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?

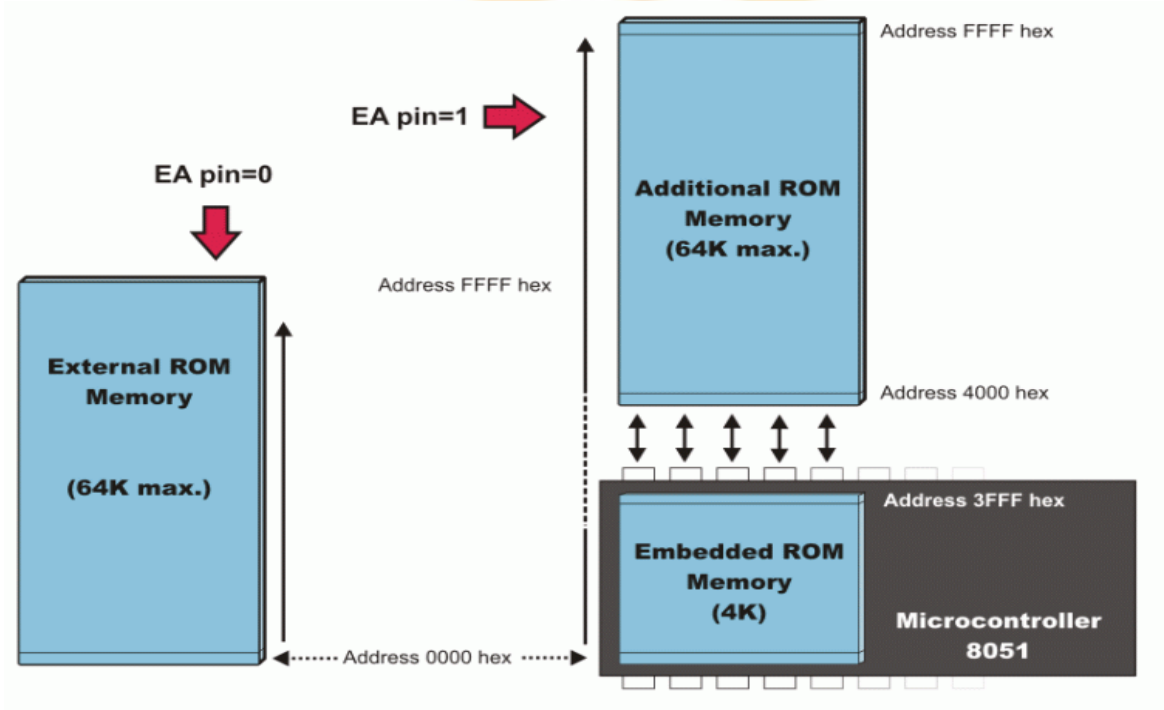


Fig 6.3. 8051 program memory

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.

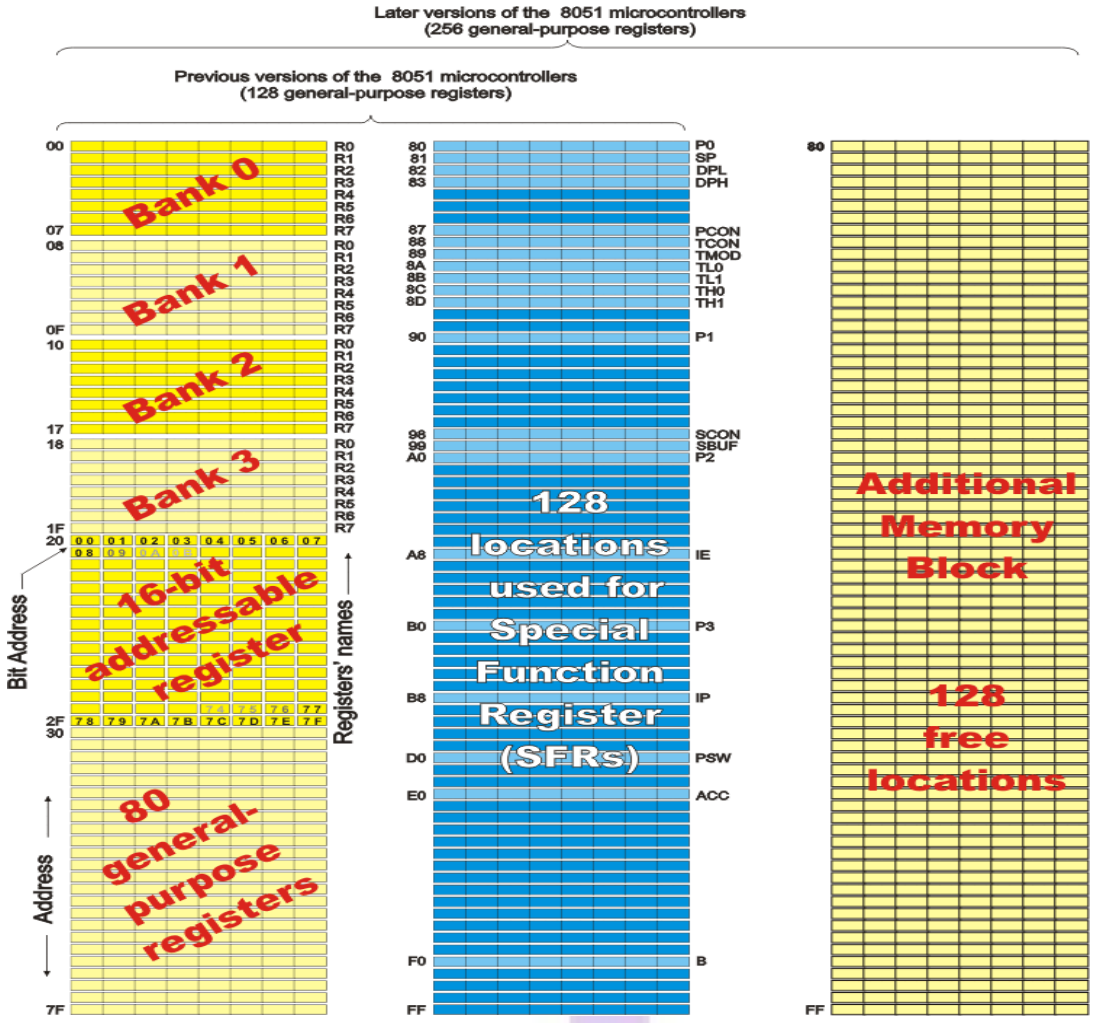


Fig 6.4. Data memory organization

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.

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

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 memory space. The address operand in this example is formed by adding the content of the DPTR 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, arithmetic instructions, logical instructions, program control instructions, and some special instructions such as the rotate instructions.

Data Transfer

Many computer operations are concerned with moving data from one location to another. 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 are used in accessing external memory data. The MOV instruction has the following format:

MOV *destination <- source*

The instruction copies *(copy* is a more accurate word than *move)* data from a defined source 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. The DPTR 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 can also be used for indirect addressing of external memory but the address range is limited to the lower 256 bytes of memory (28 = 256 bytes).

The MOVX instruction is used to access the external memory (X indicates external memory 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 where register A can be used to provide an offset in the address specification. Like the MOVX instruction all moves must be done through register A. The following sequence 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 SP contains 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 is incremented.

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

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

XCH

The above move instructions copy data from a source location to a destination location, leaving the source data unaffected. A special XCH (exchange) instruction will actually swap the data between source and destination, effectively changing the source data. Immediate addressing may not be used with XCH. XCH instructions must use register A. XCHD is a special case of the exchange instruction where just the 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 the arithmetic 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, as follows:

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

The AC (auxiliary) flag is set to 1 if there is a carry out of bit position 3 of the accumulator, 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 often required to add 16 bit numbers, or 24 bit numbers etc. This leads to the use of multiple 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 significant byte etc. This addition process is done at 8-bit precision steps to achieve multi precision arithmetic. The ADDC instruction is used to include the carry bit in the addition process. Example instructions using ADDC are:

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

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

Subtraction

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

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

Increment/Decrement

The increment (INC) instruction has the effect of simply adding a binary 1 to a number while a decrement (DEC) instruction has the effect of subtracting a binary 1from a number. The increment and decrement instructions can use the addressing modes: direct, indirect and register. The flags C, AC, and OV are **not** affected by the increment 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 overflow from FFFFh to 0000h. The DPTR register cannot be decremented using a DEC instruction (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 is relatively fast since multiplication

and division are implemented as single instructions. If better precision, or indeed, if floating point arithmetic is required then special software routines need to be written. For the MUL or DIV instructions the A and 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 the high 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 register A.

Decimal Adjust (Special)

The 8051 performs all arithmetic in binary numbers (i.e. it does not support BCD arithmetic). If two BCD numbers are added then the result can be adjusted by using the 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 on the data. Most microcomputers provide a set of Boolean instructions that act on byte level data. However, the 8051 (somewhat uniquely) additionally provides Boolean instruction 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), a general register, or a direct address. Status flags are not affected by these logical operations (unless PSW is directly manipulated). Example instructions are:

ANL A, #55h ; AND each bit in A with corresponding bit in number 55h, leaving the 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 address is 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 flag can easily be tested using a branch (jump) instruction to quickly establish program flow 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 examination of 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-order nibble using the instruction:
SWAP A

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 by the 16-bit operand in the jump instruction. Because a 16-bit address is used the instruction 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 number and 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 used with 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 address boundary (a 2K page)There is also a generic JMP instruction supported by many 8051 assemblers. The assembler 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 automatically pushed onto the stack of the PC. When the program returns from the subroutine the PC contents is returned from the stack so that the program can resume operation from the point where the LCALL was made. The return from subroutine is achieved using the RET instruction, which simply pops the PC back from the stack.

ACALL

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

Program control using conditional jumps

Most 8051 jump instructions use an 8-bit destination address, based on relative addressing, i.e. addressing within the range –128 to +127 bytes. When using a conditional jump instruction the programmer can simply specify a program label or a full 16-bit address for the conditional jump instruction's destination. 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 the magnitudes are not equal.

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)

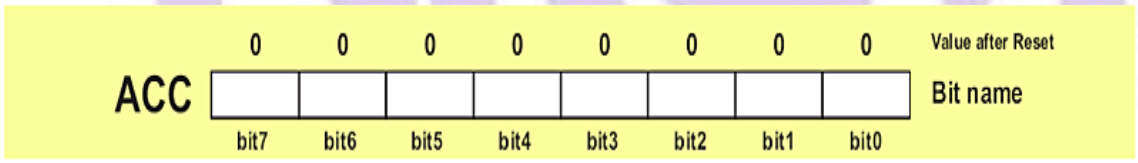


Fig 7.2. 8051 Accumulator

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).

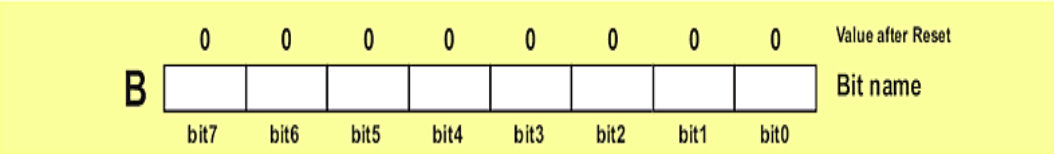


Fig 7.3. 8051 b register

R Registers (R0-R7)

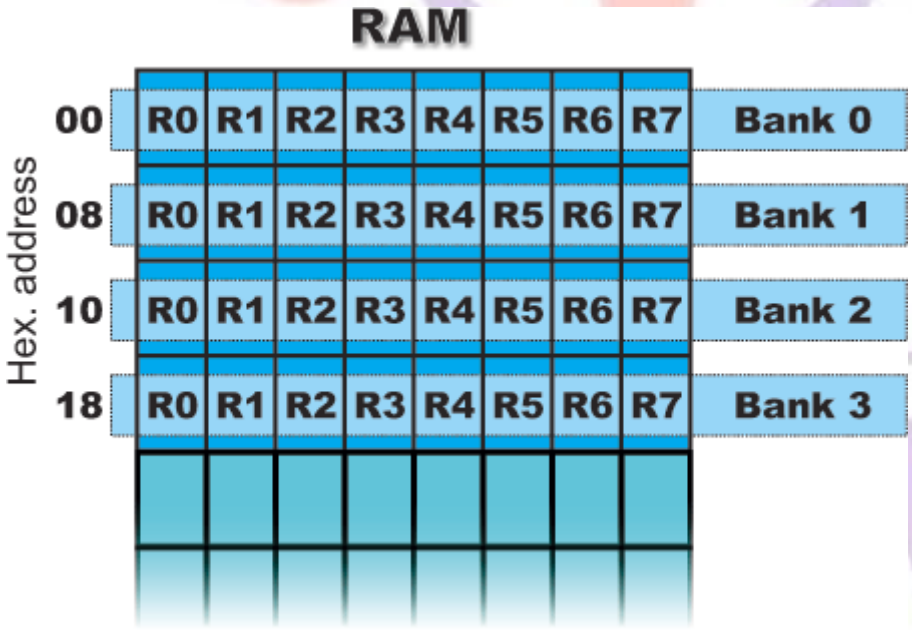


Fig 7.4. 8051 Register banks

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

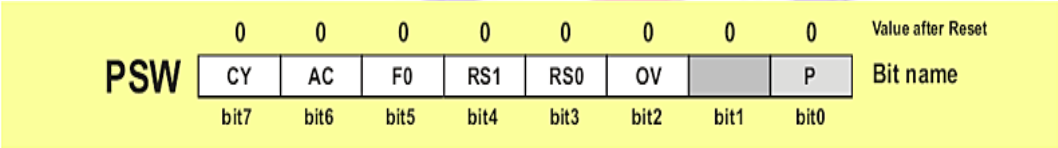


Fig 7.5. 8051 PSW

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	SPACEIN 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.

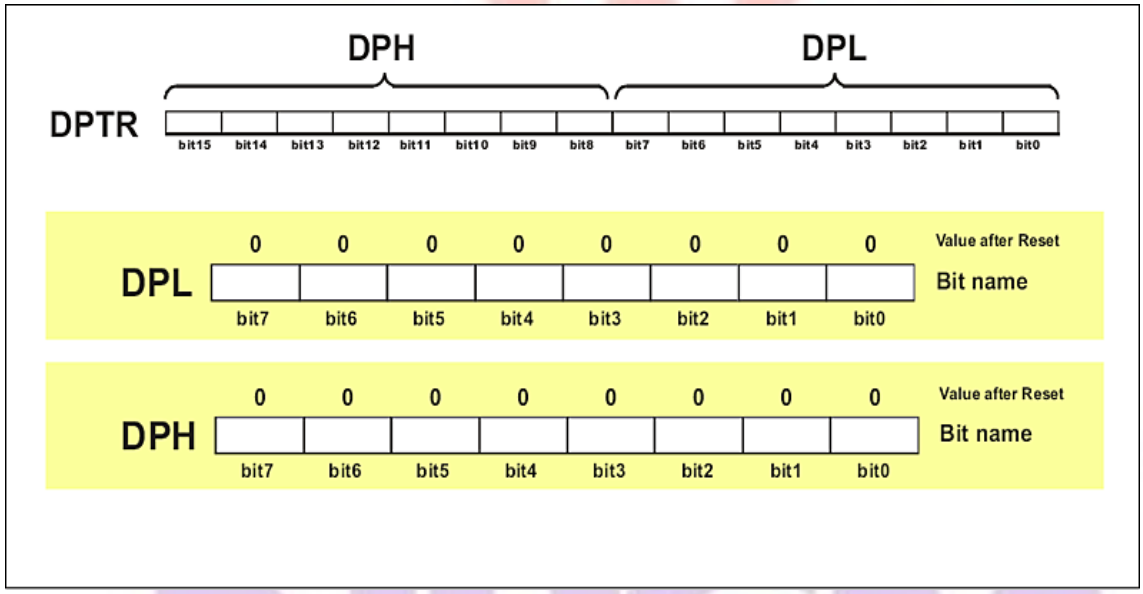


Fig 7.6. 8051 Data Pointer

Stack Pointer (SP) Register

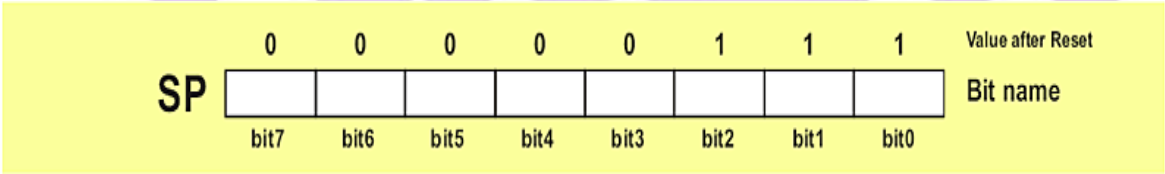


Fig 7.7. 8051 Stack Pointer

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 within 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.

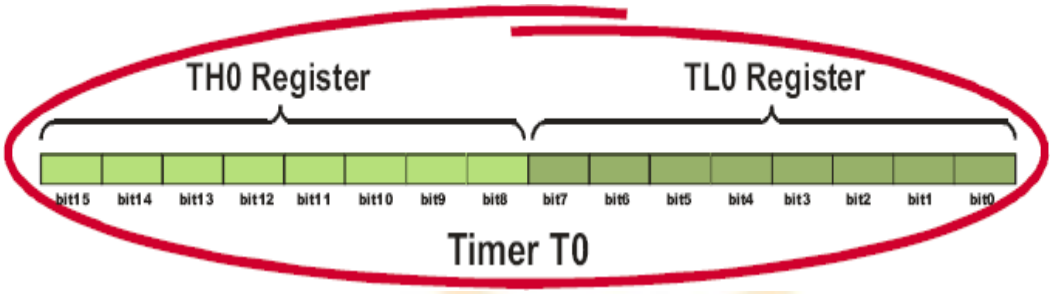
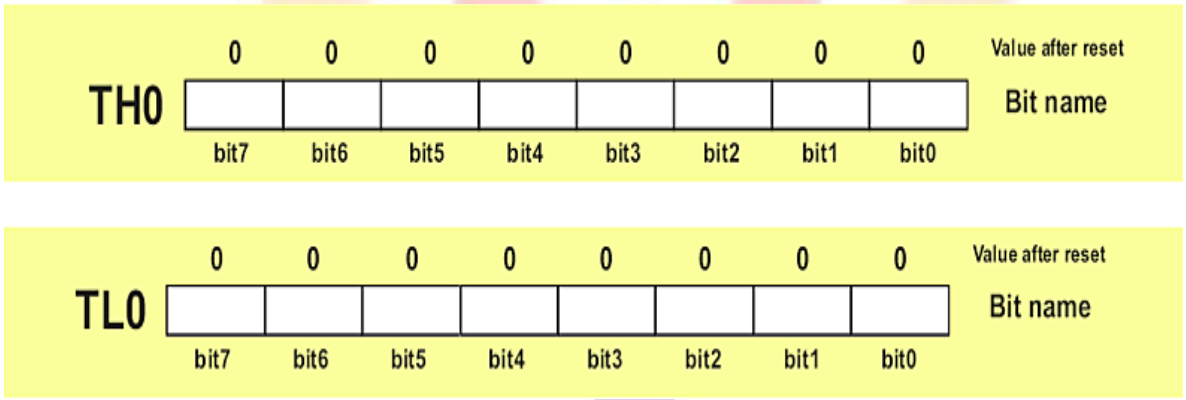


Fig 7.8. Timer-0 register

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.

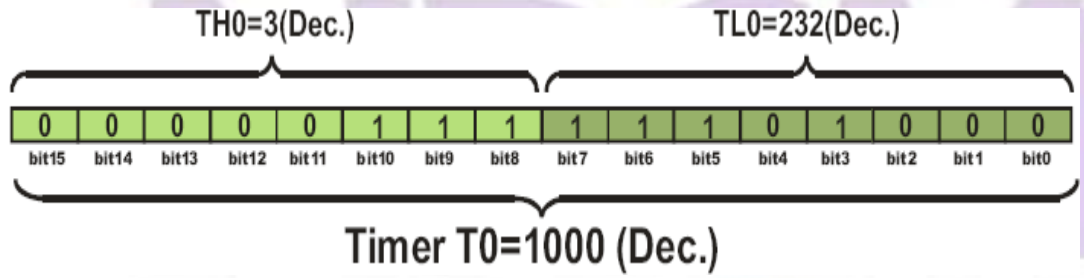


Formula used to calculate values in these two registers is very simple:

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.

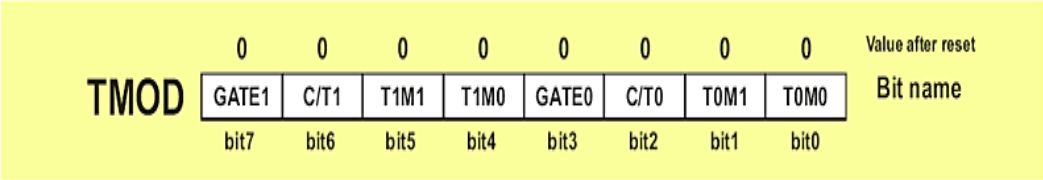


Fig 7.9. 8051 TMOD Register

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.

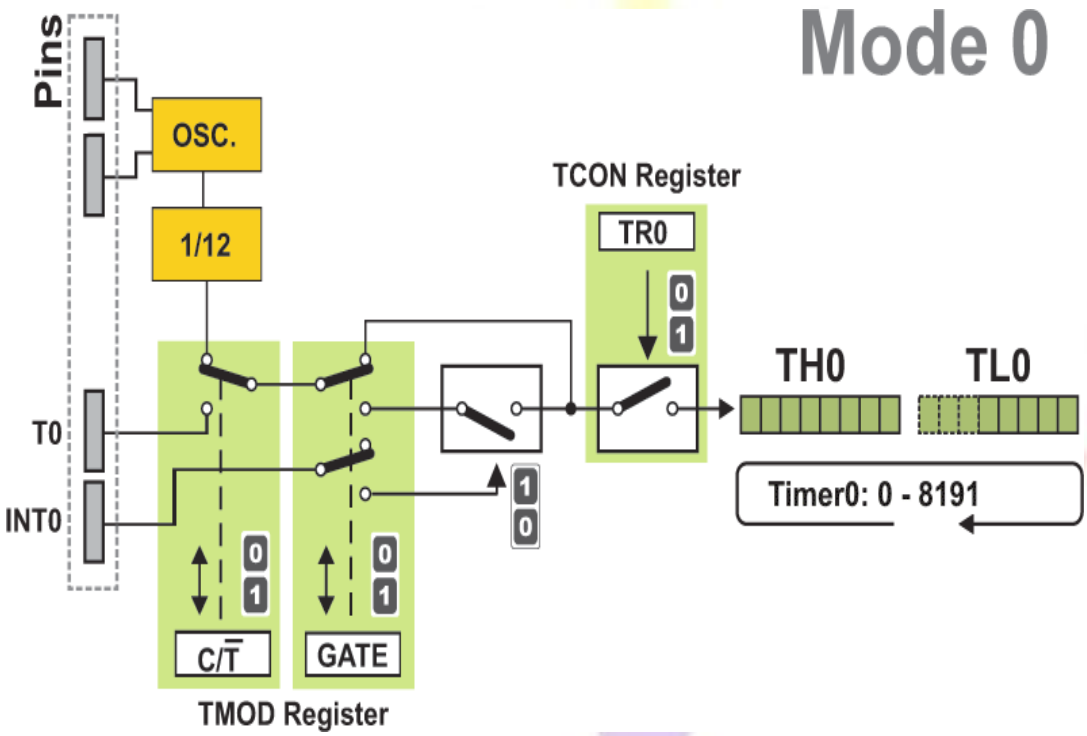


fig 7.10. Timer-0 in Mode -0

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.

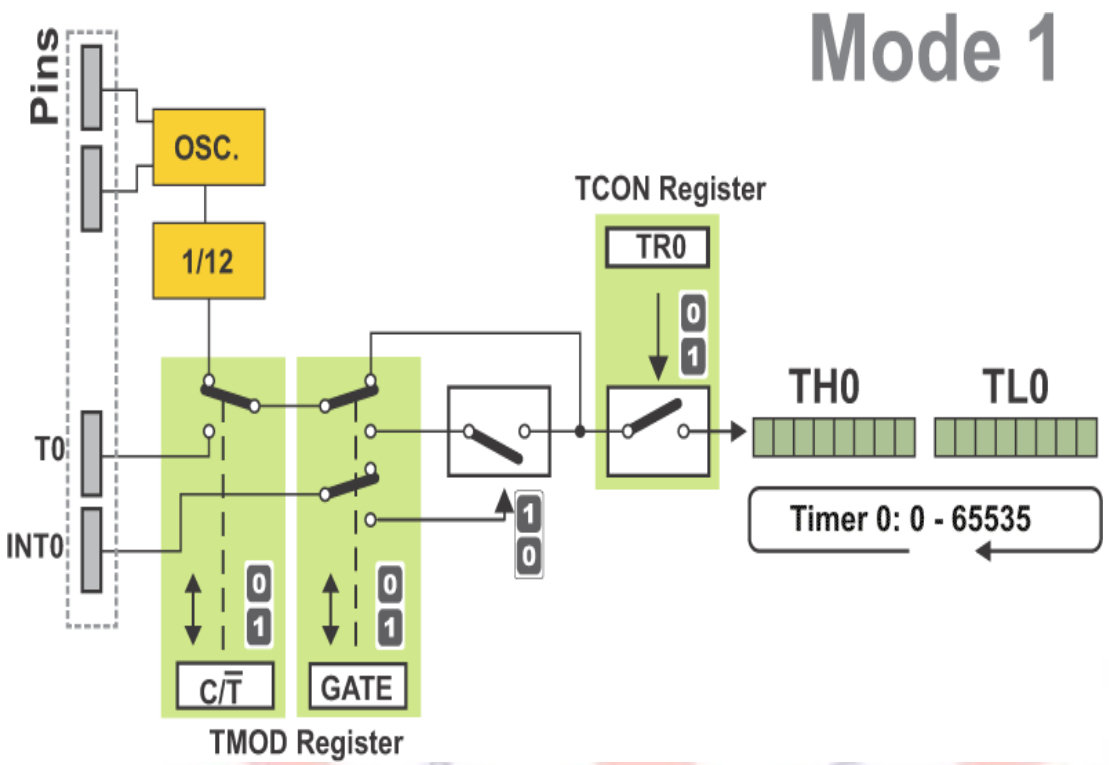


Fig 7.11. Timer -0 in Mode -1

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.

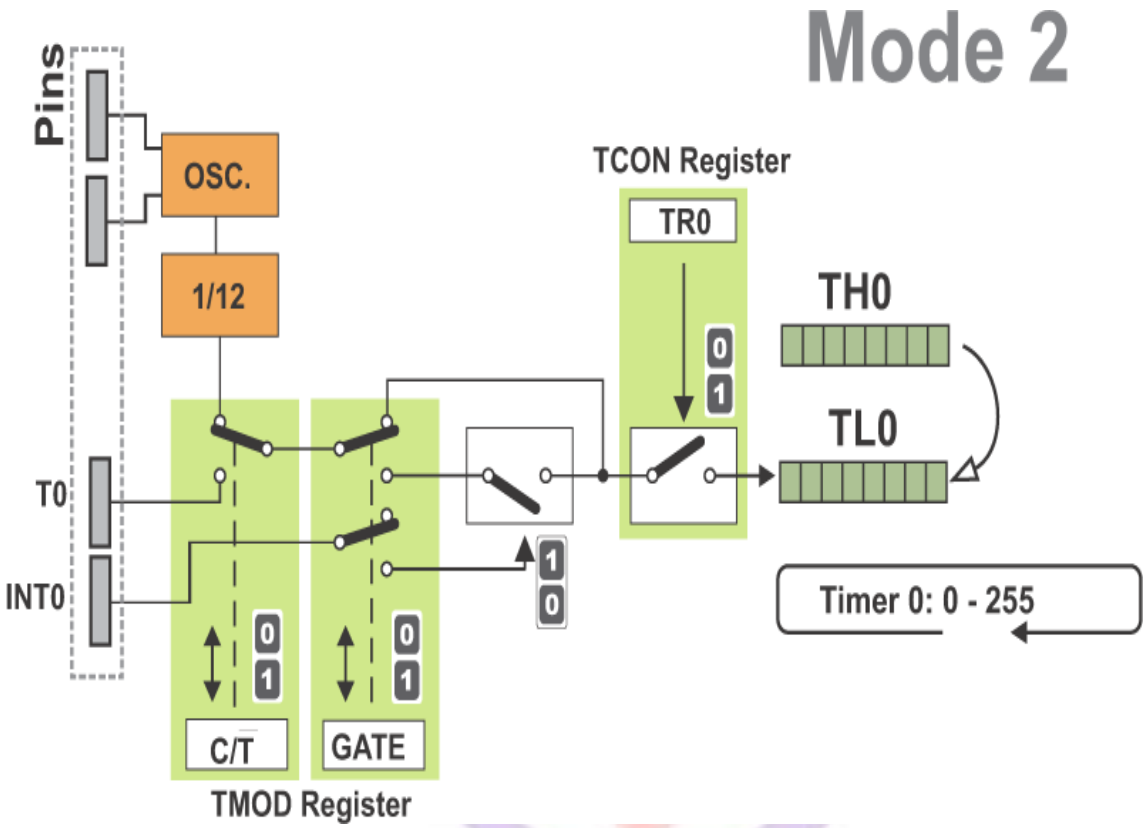


Fig 7.12. Timer-0 in Auto Reload Mode

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.

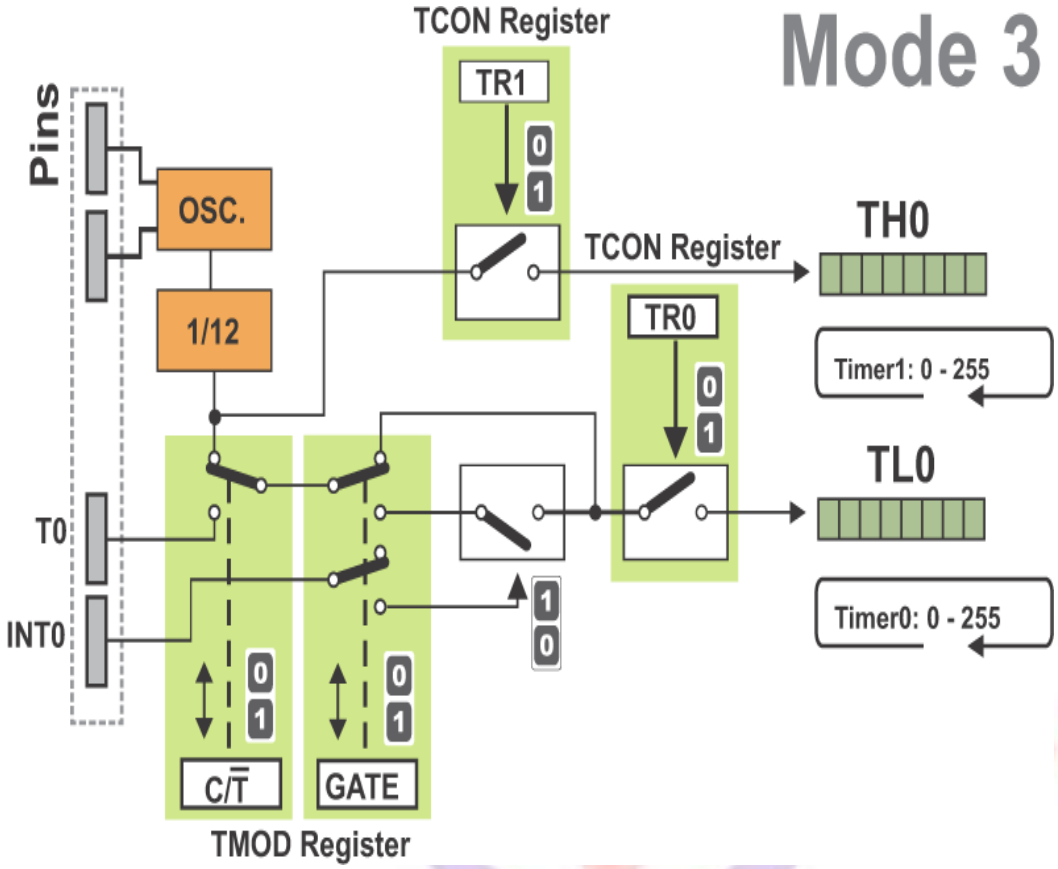


Fig 7.13. Timer -0 in Split Timer Mode

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. for interrupt

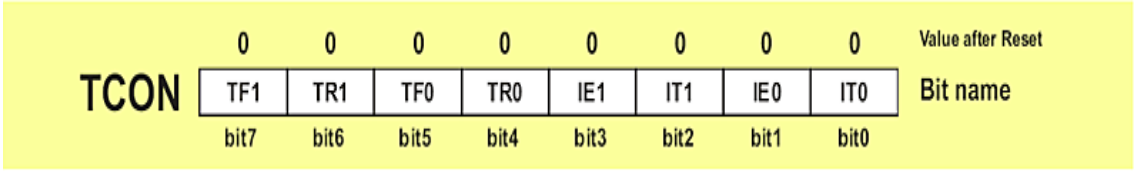


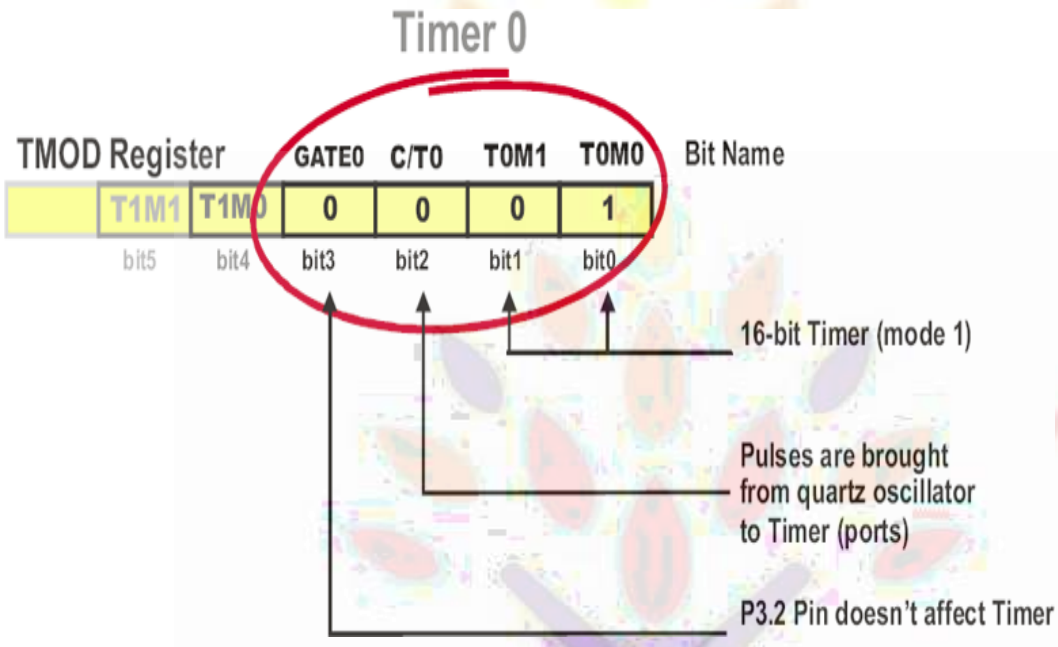
Fig 7.14. 8051 TCON Register

- **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.

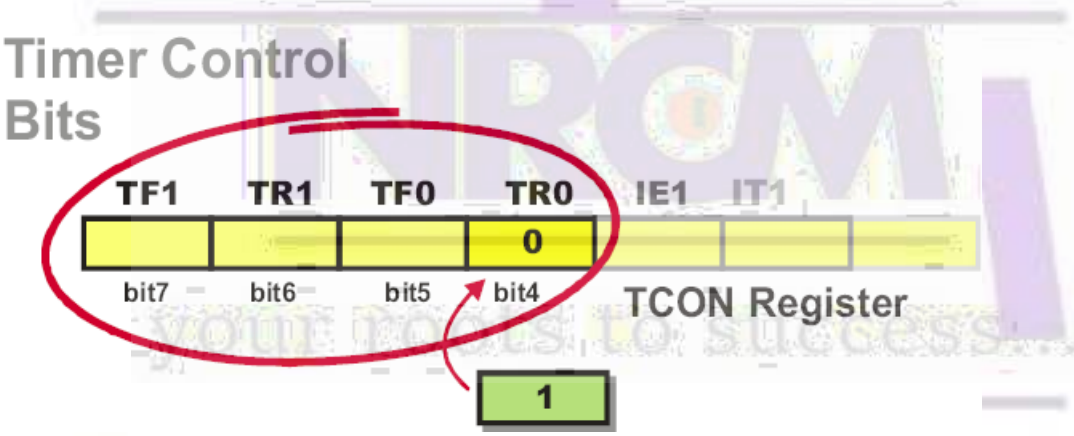
- **1** - Timer 0 is enabled.
- **0** - Timer 0 is disabled.

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

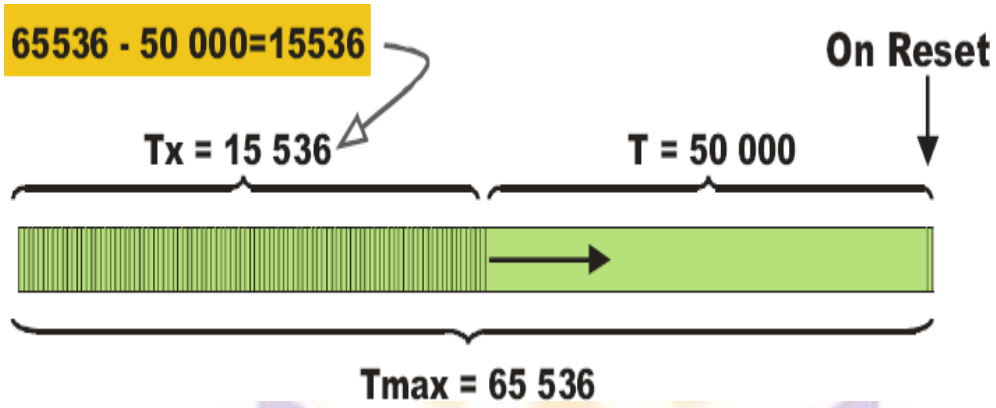
In case of negligence, such an error in counting (255 pulses) may occur for not so obvious but quite logical reason. The lower byte is correctly read (255), but at the moment the program counter was about to read the higher byte TH0, an overflow occurred and the contents of both registers have been changed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

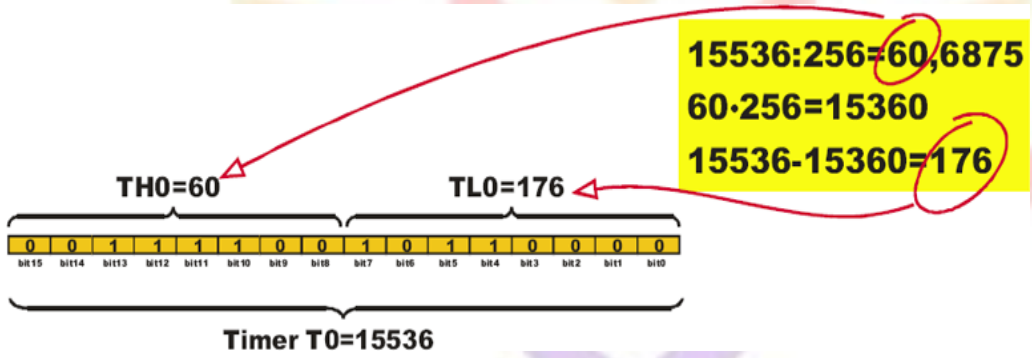
Timer 0 Overflow Detection

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?

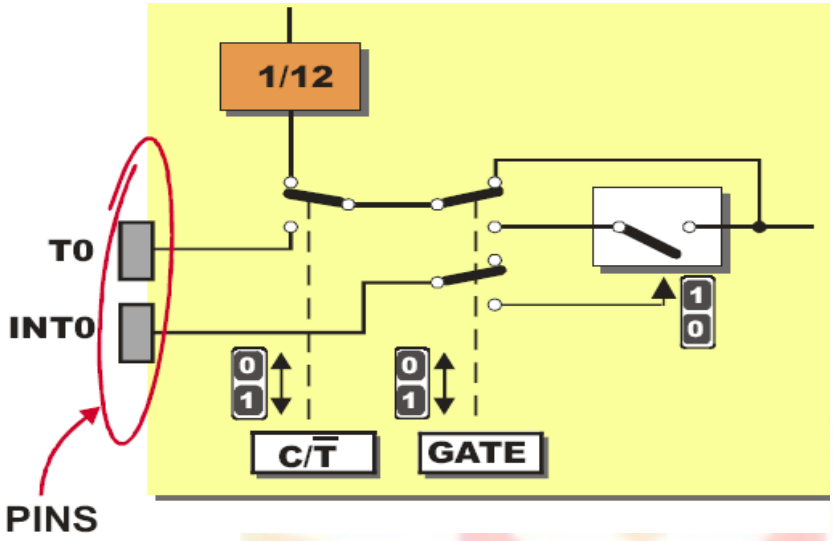


Fig 7.14. Internal operation of the timer

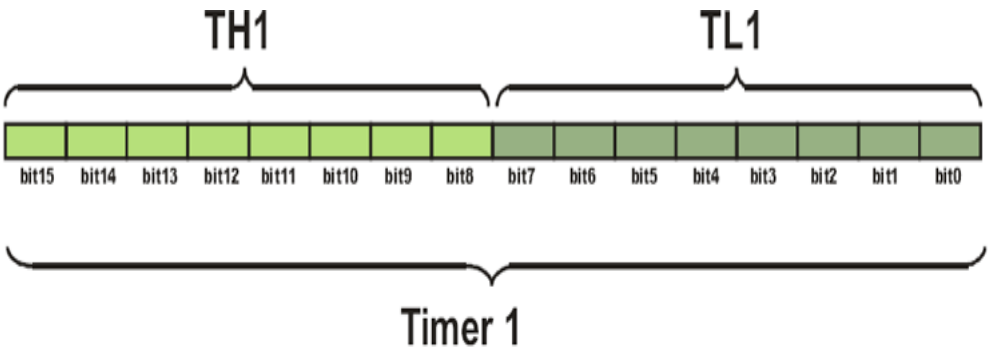
Suppose it is necessary to measure the duration of an operation, for example how long a device has been turned on? Look again at the figure illustrating the timer and pay attention to the function of the GATE0 bit of the TMOD register. If it is cleared then the state of the P3.2 pin doesn't affect timer 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.

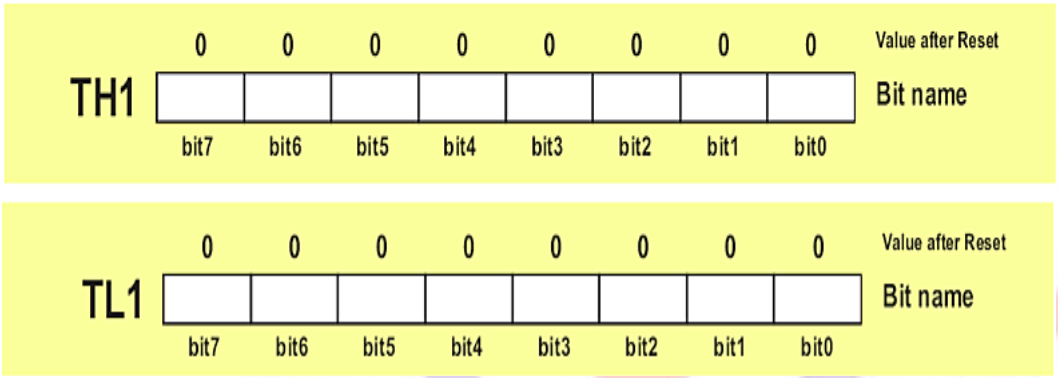
How to count up pulses?

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.

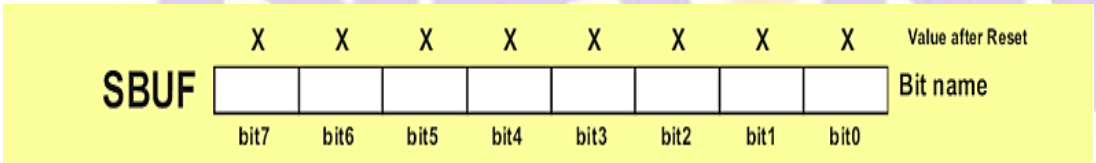


Fig 7.15. 8051 SBUF register

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

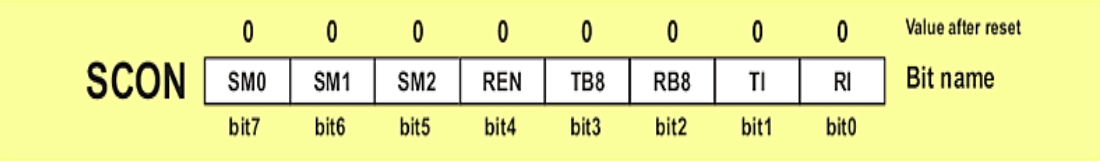


Fig 7.16. Serial Control Register

- **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	BAUDRATE
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

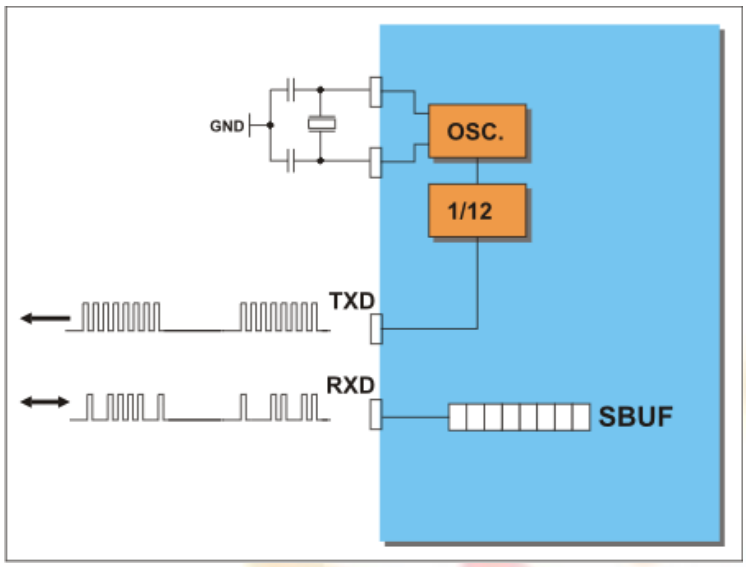
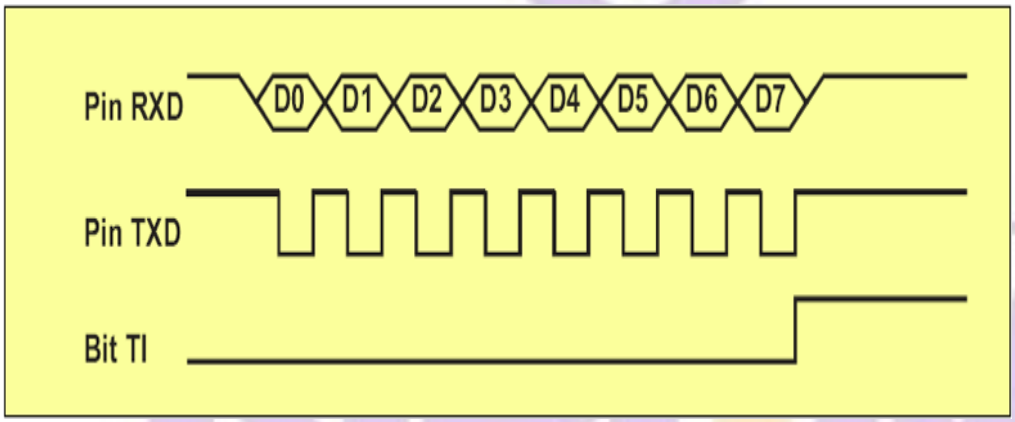


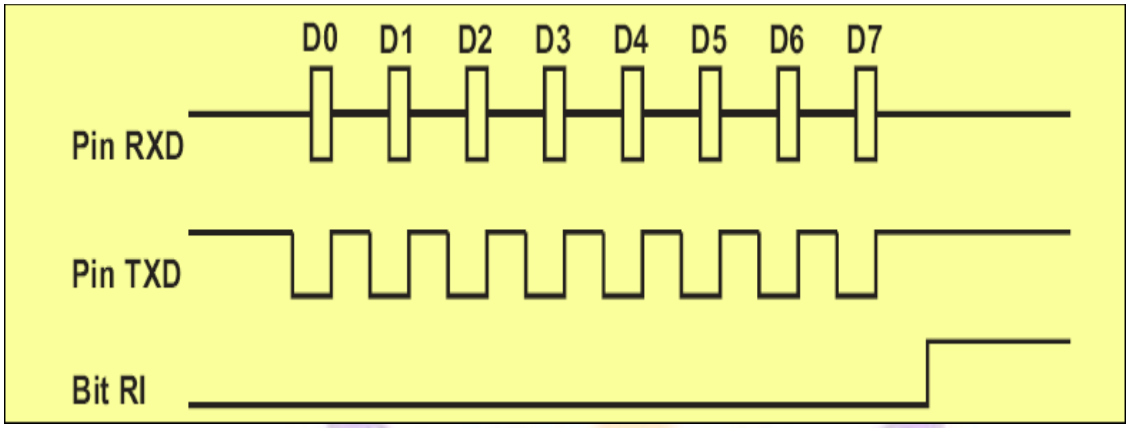
Fig 7.16. Serial communication overview

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

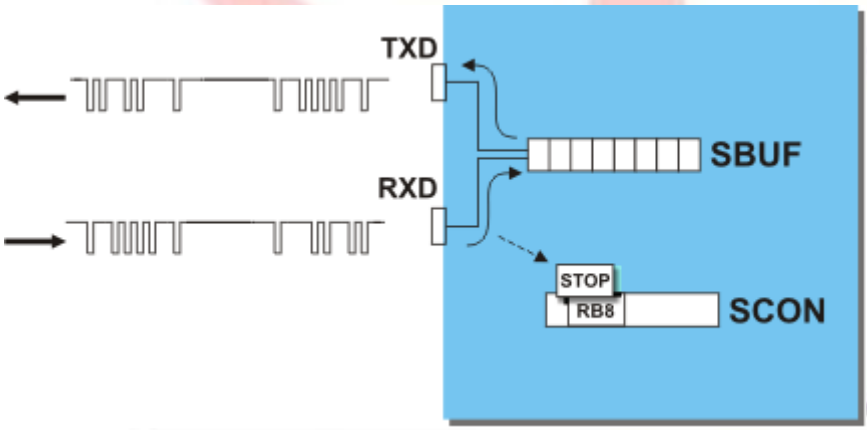
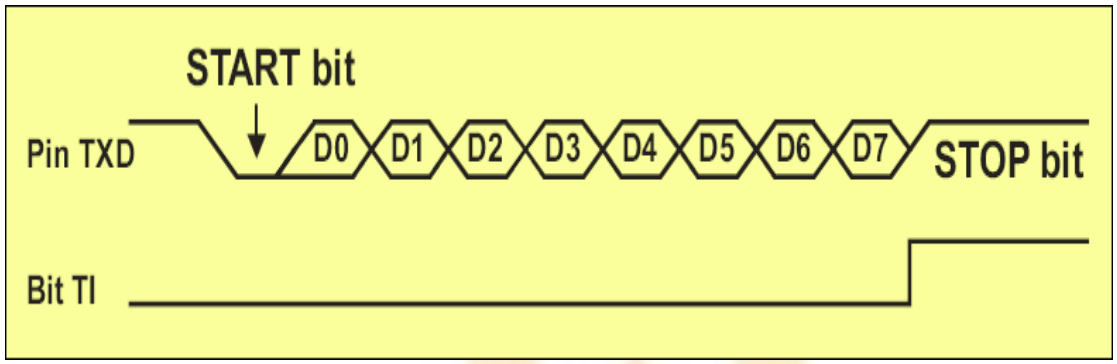


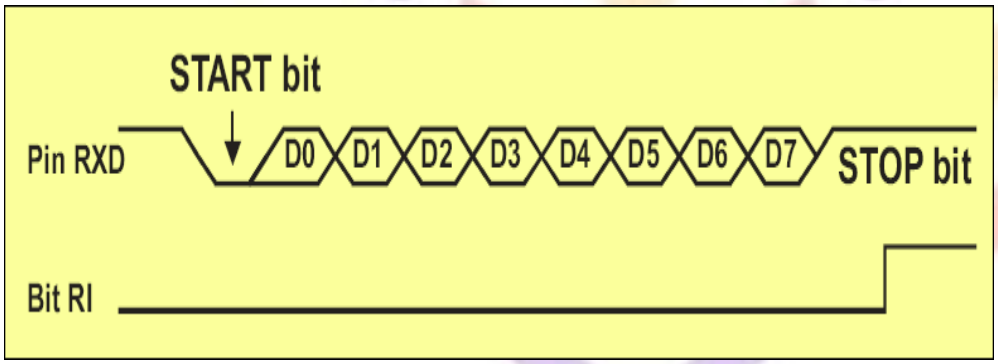
Fig 7.17. Serial communication in mode1

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

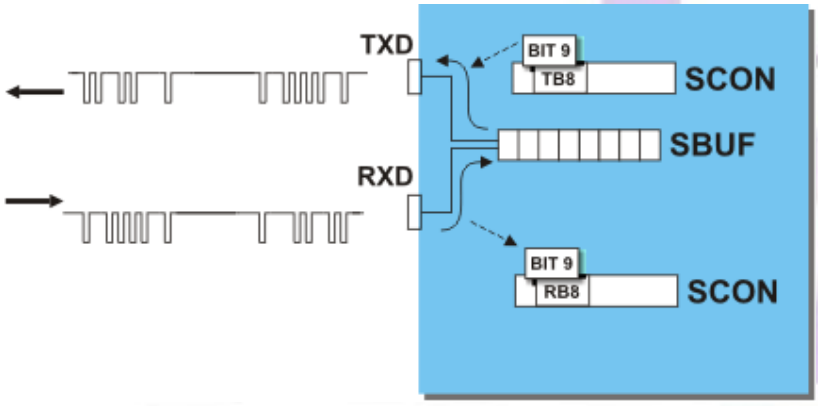
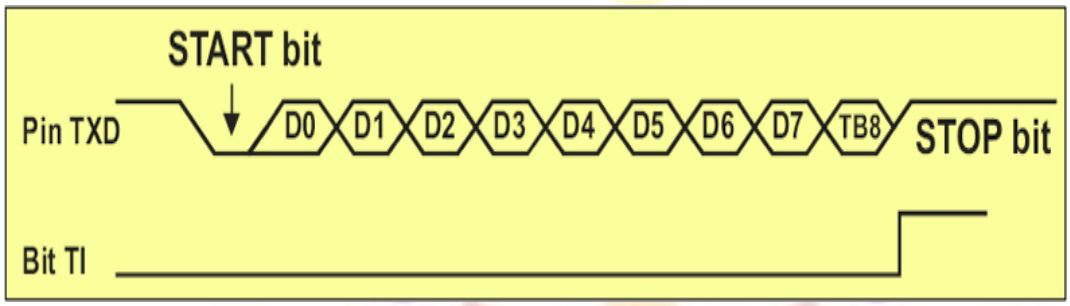


Fig 7.18. Serial communication in mode 2

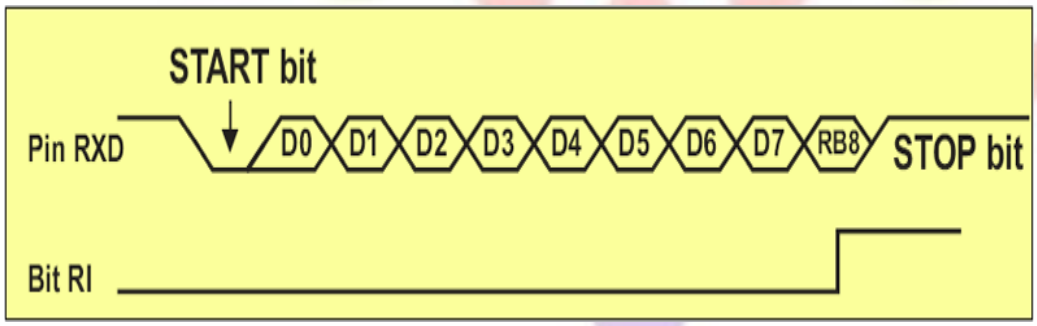
In mode 2, 11 bits are transmitted through the TXD pin or received through the RXD pin: a START bit (always 0), 8 data bits (LSB first), a programmable 9th data bit and a STOP bit (always 1). On transmit, the 9th data bit is actually the TB8 bit of the SCON register. This bit

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)	

essary formulas are specified in the table:

Timer 1 as a clock generator

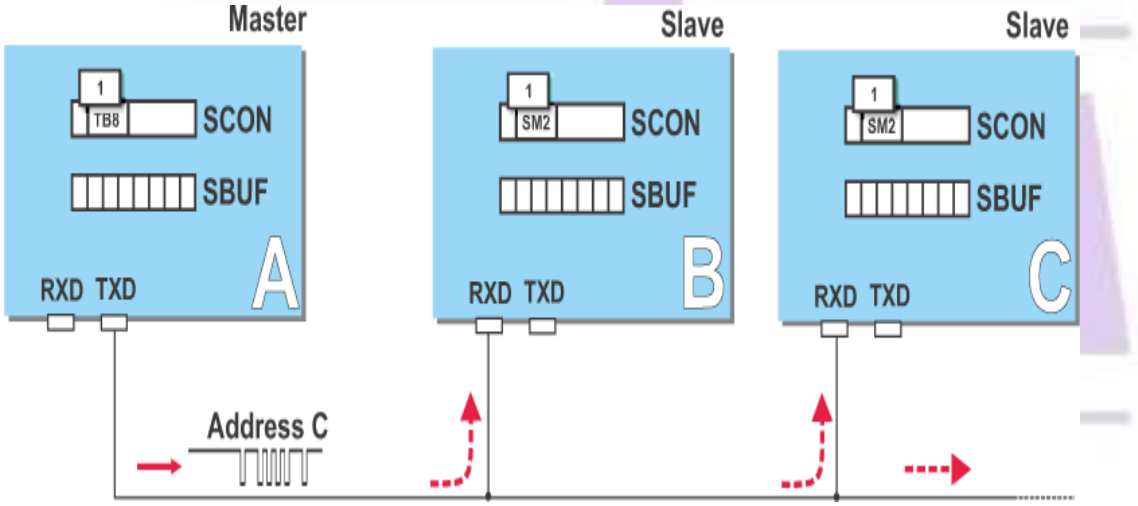
Timer 1 is usually used as a clock generator as it enables various baud rates to be easily set. The whole procedure is simple and is as follows:

- 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.

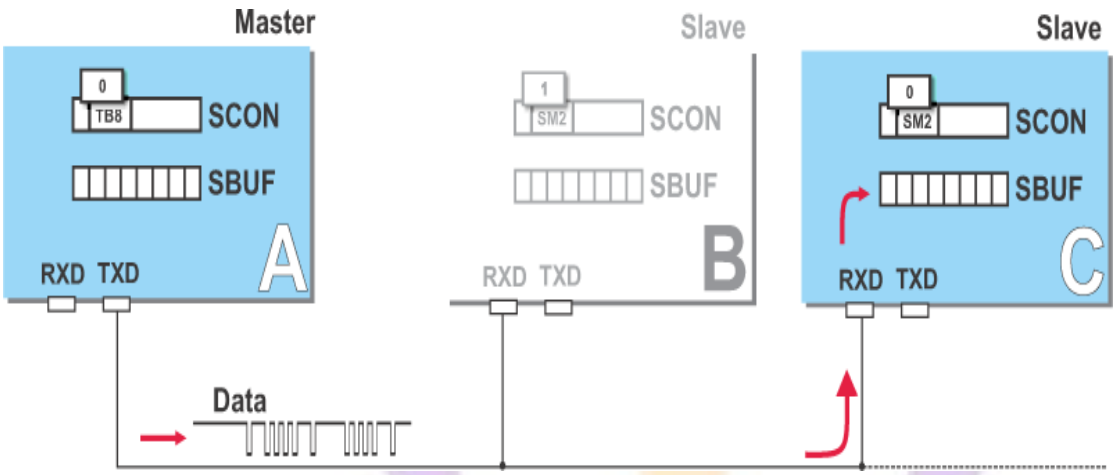
Multiprocessor Communication

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.

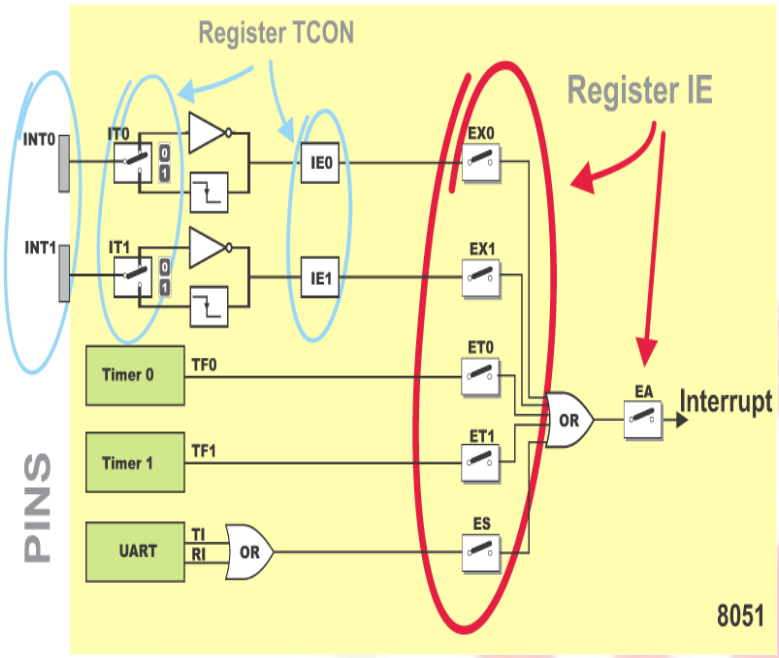


Fig 7.19. Interrupt sources of 8051

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).

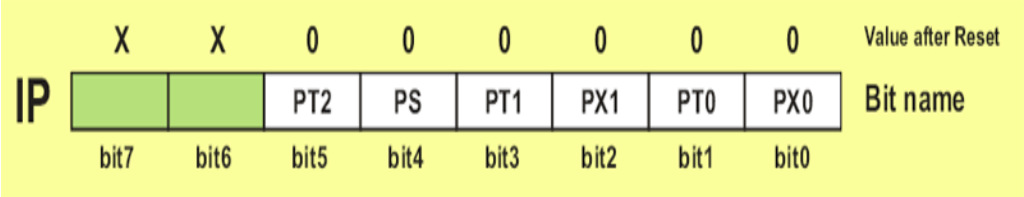


Fig 7.21. IP register of 8051

- **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

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.

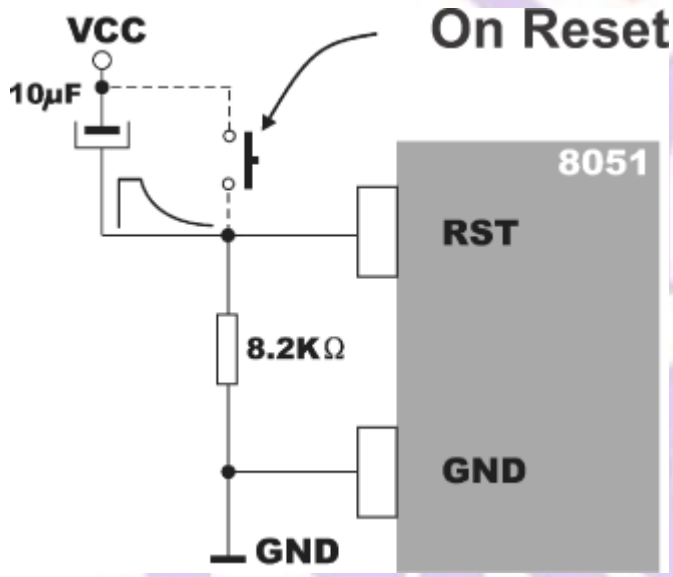


Fig 7.22. 8051 Reset

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):

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.*

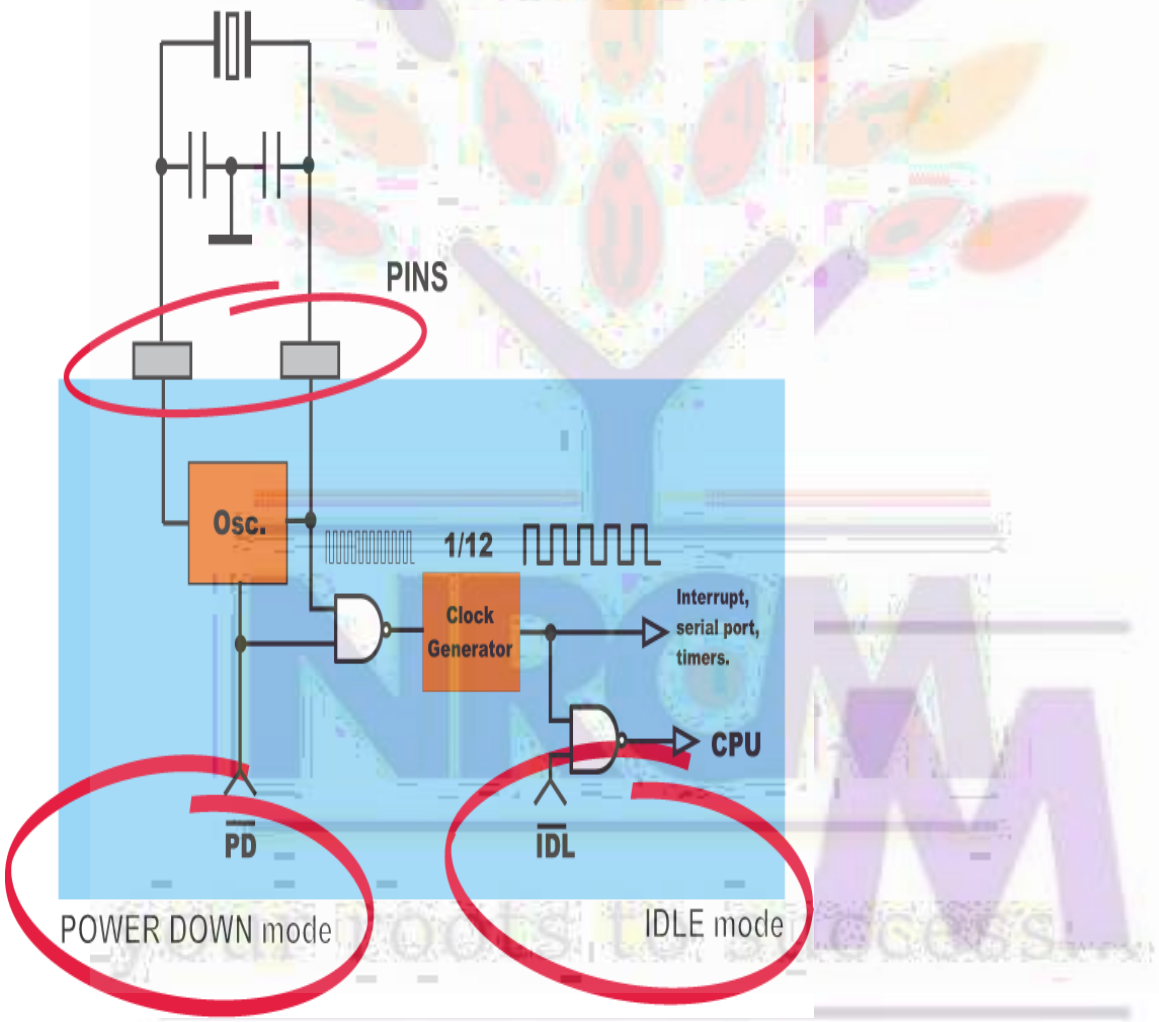


Fig 7.23. power down and idle modes of 8051