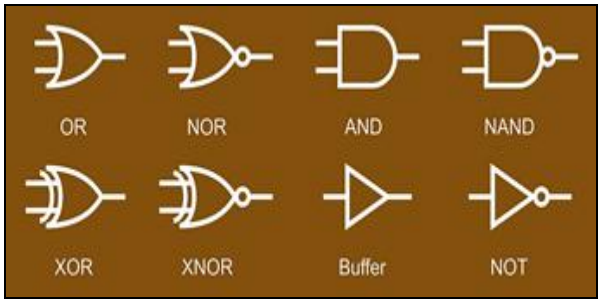
CHAPTER – 1
Basics

A) Logic Gates (AND, OR, NOT, NAND, NOR, EX-OR): Review of all logic gates; AND, OR, NOT, NAND, NOR, EX-OR & their truth tables. Appropriate combinations of gates result into an amazing & innovative logical configuration.

Basic Logic Gates



B) Bit, Nibble and Byte:

Bit: The smallest unit of data in a computer is called bit.

Nibble: Half a byte that is four bits is called a nibble.

Byte: Eight bits forms a byte.

C) 1's Complement & 2's Complement; their Significance: Let's consider a number in Hexadecimal ystem: $(FB)_{16}$ = $(1111\ 1011)_2$. Further let's consider complement of its binary form i.e. $(0000\ 0100)_2$ let add 1 to this number i.e. $(0000\ 0101)_2$. $(04)_{16}$ and $(05)_{16}$ are 1's complement and 2's complement of the number $(FB)_{16}$.

The alternative way is FF - FB = 04 is 1's complement and 04 + 01 = 05 is two's complement of $(FB)_{16}$. Further let's add FB & 05 =100. If we neglect 1, the sum of a number and its 2's complement is "ZERO" but we know addition of a number to its negative only can lead to zero, hence 2's complement of a number is negative of the number. Computer does not subtract, it always adds, hence

subtraction of number exactly means, addition of its 2's complement.

D) Number Systems (Binary, Octal, Decimal & Hexadecimal): In digital, we normally deal with four number systems of arithmetic Binary, Octal , Decimal and Hexadecimal. The commonly used number system by all of us is decimal, while the binary number system is used by computers.

Number Systems & Equivalence

Decimal (10)	Octal (8)	Hexadecimal (16)	Binary (2)
0	0	0	0000 0000
1	1	1	0000 0001
2	2	2	0000 0010
3	3	3	0000 0011
4	4	4	0000 0100
5	5	5	0000 0101
6	6	6	0000 0110
7	7	7	0000 0111
8	10	8	0000 1000
9	11	9	0000 1001
10	12	A	0000 1010
11	13	B	0000 1011
12	14	C	0000 1100
13	15	D	0000 1101
14	16	E	0000 1110
15	17	F	0000 1111
16	20	10	0001 0000
17	21	11	0001 0001
18	22	12	0001 0010

Refer the table above, it is obvious that:

- $(10)_{10}=(12)_8=(A)_{16}=(1010)_2$; $(15)_{10}=(17)_8=(F)_{16}=(1111)_2$; And so on…….

E) Sign-Magnitude: Refer the numbers 11100011 and 01100011. If the MSB is assigned for sign then, (1 1100011) $=(-63)_{16}$ and (0 1100011) = $(+63)_{16}$. In a decimal number system, a (+) sign is used to denote positive number and (-) sign is used to denote negative number. In digital system, the sign of the binary number is also represented by 0 & 1. In signed binary number system, the numbers also have the

sign, MSB '0' is to represent positive number and '1' is to represent negative number.

F) Parity and its Significance: Let's consider a number in hexadecimal system $(F2)_{16} = (1111\ 0010)_2$. The number of 1s in its binary equivalent is "5", hence, the number F2, is with odd parity. Similarly, let's consider another number $(B5)_{16} = (1011\ 0101)_2$ is again with an odd parity. While $(C3)_{16} = (1100\ 0011)_2$ is with an even parity. $\Longrightarrow$ Number of 1s in the binary equivalent of a hexadecimal number, decides the parity of the number.

G) LSB and MSB of a Binary Number & its Significance: The LSB and MSB of binary number has its own significance altogether. Refer the numbers $(0010)_2$, $(0100)_2$, $(0110)_2$, $(1000)_2$ …… the even numbers are those whose LSB is '0'. Refer the numbers $(0011)_2$, $(0101)_2$, $(0111)_2$ ……, the odd numbers are those whose LSB is one. Similarly, if MSB is '1' the number is negative and when it is '0' it is positive.

H) Binary Digits: The microprocessor operates on binary digits, 0 and 1, also known as bits. Bit is an abbreviation for the term binary digit. These digits are represented in terms of electrical voltages in machine: Generally, 0 represents low voltage level & 1 represents high voltage level. Digits 0 and 1 are also synonymous with low & high respectively.

CHAPTER – 2

µP 8051: Hardware & Software

Microcontroller, a computer on a single IC, contains a processor core, ROM, RAM & I/O pins dedicated to performing various tasks. Microcontrollers are generally used in projects and embedded systems. Some examples of µCs are 8051, AVR, PIC etc.

Even though there are many high-level languages that are currently in demand but assembly language programming is popularly used in many applications. It can be used for direct hardware manipulations & can also be used to write the 8051 programming code efficiently, with less number of clock cycles.

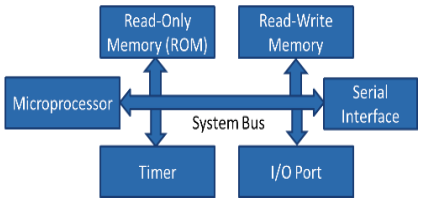
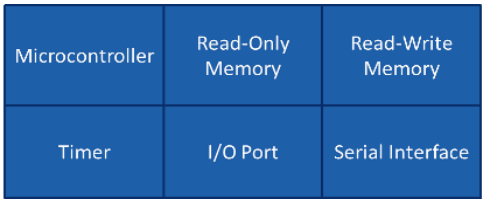
The designers must have sufficient knowledge about the hardware of particular microcontroller before programming it. Microcontrollers can understand only binary language in the form of 0s or 1s; an assembler converts the assembly language to binary and stores it in the microcontroller memory to perform the specific task.

The architecture of the Central Processing Unit (CPU) operates with the capacity to function from "Instruction Set Architecture" to where it was designed. The architectural design of the CPU is "Reduced Instruction Set Computing" (RISC) and "Complex Instruction Set Computing" (CISC). CISC has the capacity to perform multi-step operations or addressing modes within one instruction set. It is the CPU design where one instruction performs several low-level operations. For an instance, memory storage, loading from memory and an arithmetic operation. Reduced instruction set computing is a Central Processing Unit design strategy which based on the vision that basic instruction set gives a great performance when combined with a microprocessor architecture which has the capacity to perform the instructions by using some microprocessor cycles per instruction. This article discusses the difference between the RISC & CISC architecture.

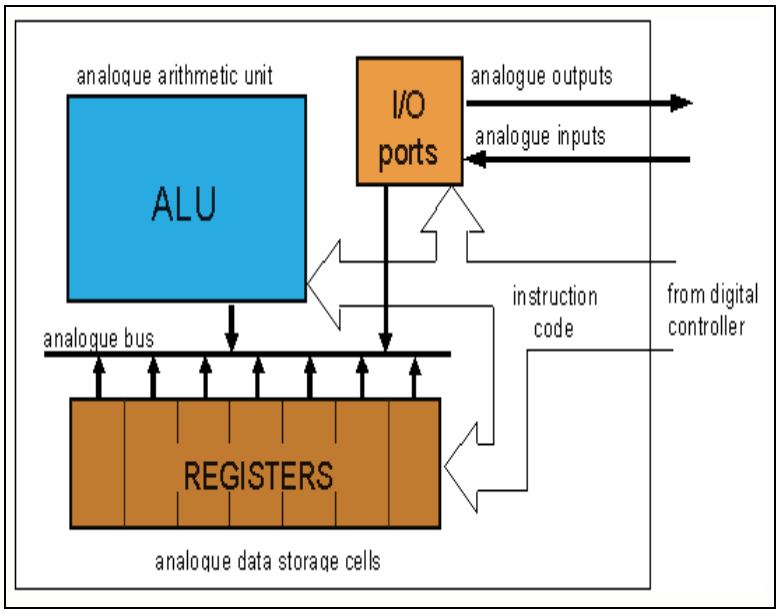
RISC & CISC Architecture

RISC	CISC
RISC: Reduced Instruction Set	CISC: Complex Instruction Set
Instructions corresponds to one clock cycle. The average clock cycle per instruction is 1.5	Instructions corresponds to that multiple clocks for execution. The average clock cycle per instruction varies from 2-15.
Performance is optimized with focus on software	Performance is optimized with focus on hardware.
NO memory & needs separate hardware to implement instructions.	It has a memory unit to implement complex instructions.
The instruction set is reduced i.e. it has only a few instructions in the instruction set, very primitive.	The instruction set has a variety of different instructions that can be used for complex operations.
The instruction set has a variety of different instructions that can be used for complex operations.	Different addressing modes & can thus be used to represent higher-level programming language statements more efficiently.
Multiple register sets	Single register set
Highly pipelined	Less pipelined
Complexity lies with the compiler	Complexity lies in the micro-program
Execution time is very less	Execution time is very high
Code expansion; a problem	Code expansion; not a problem
Decoding is simple.	Decoding is complex
RISC microprocessors are Alpha, ARC, ARM, AVR, MIPS, PA-RISC, PIC, SPARC.	Examples of CISC processors are the System/360, VAX, PDP-11, Motorola 68000, AMD & Intel x 86 CPUs.
RISC architecture is used in HEA video processing & image processing.	CISC architecture is used in LEA such as security systems, home automation, etc.

Comparative Study of Microprocessor and Microcontroller

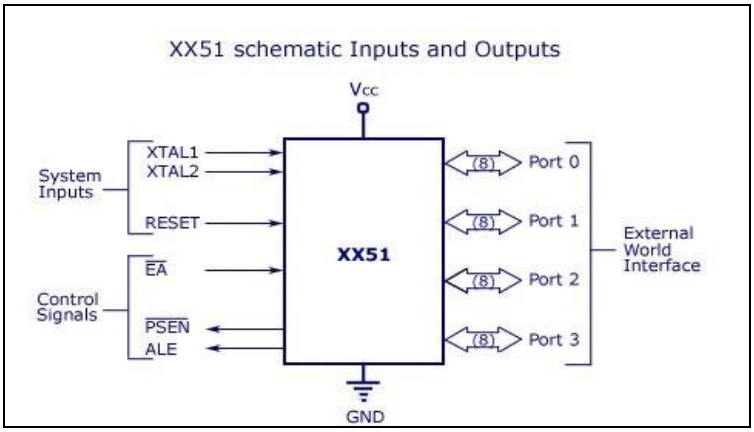
Microprocessor	Microcontroller
	
µP: Heart of Computer system.	µC: Heart of Embedded system.
Memory & I/O are connected externally	µC has processor along with internal memory & I/O components
I/O has to be connected externally circuit becomes large.	Memory & I/O are present internally, the circuit is compact.
Cannot be used in compact systems & hence inefficient	Can be used in compact systems & hence it is efficient
Less cost effective	Cost effective
Due to external components, power consumption is high; not suitable to use with devices running on stored power like batteries.	Since external components are low, total power consumption is less & can be used with devices running on stored power like batteries.
No power saving features.	Idle mode & power saving mode.
Components being external, each instruction will need external operation, hence is relatively slower.	Components are internal hence the operations are internal instruction & so speed is fast.
Smaller number of registers, hence more operations are memory based.	Large number of registers, hence the programs are easier to write.
Based on von Neumann architecture; program & data are stored in same memory module	Based on Harvard architecture where program memory & Data memory are stored separate
Used in personal computers	Used mainly in MP3 players & WM

The most popular microcontroller 8051 belongs to the MCS-51 family of microcontrollers by Intel. Following the success of 8051, many other semiconductor manufacturers has released microcontrollers under their own brand name but using the MCS-51 core.

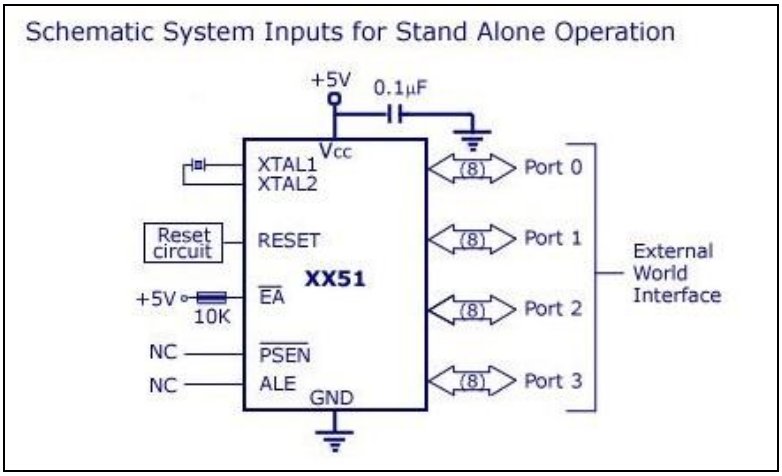


- Packaging: The most commonly used is Dual Inline Package (40 pins); popularly known as DIP. 8051 is also available in QFP (Quad Flat Package), TQFP (Thin Quad Flat Package), PQFP (Plastic Quad Flat Package) etc. For understanding pin diagram, we have used a 40 pin DIP IC as model.
- Architecture: It's possible to explain microcontroller architecture to a great detail, but we are limiting the scope of this article to internal architecture, pin configuration, program memory & data memory organization. In general, all microcontrollers in MCS-51 families are represented by XX51, where XX can take values like 80, 89 etc. The basic architecture remains same for the MCS-51 family.

Schematic & Features:

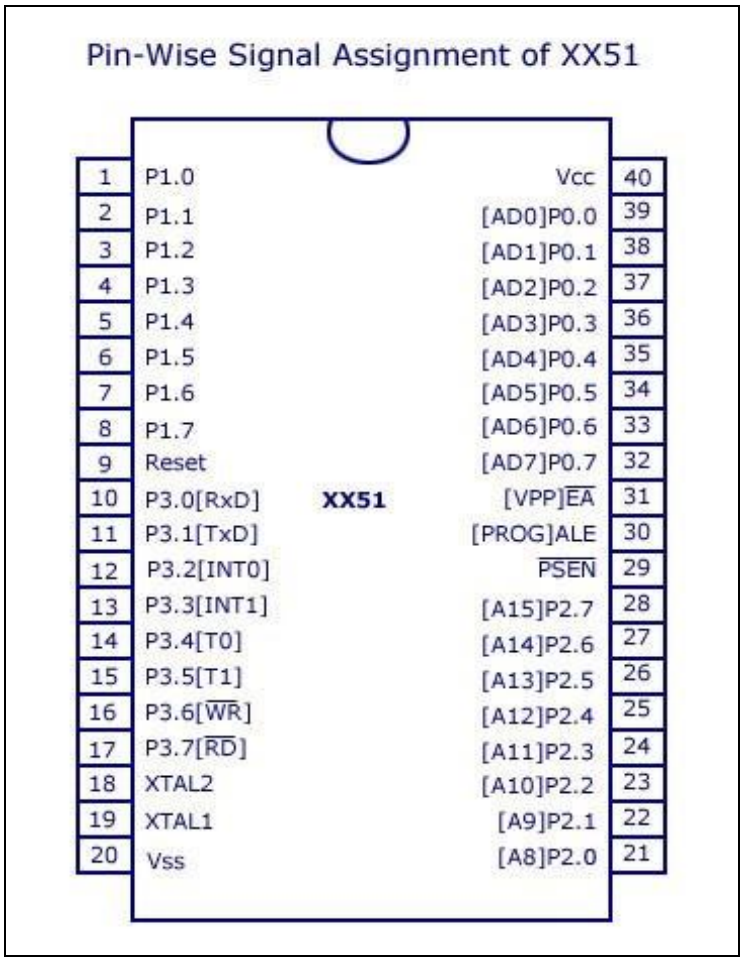


The general schematic diagram of 8051 microcontroller is shown above. We can see three system inputs, three control signals and four ports (for external interfacing). V_{CC} power supply and ground are with their usual significance. Let's understand each in detail. The first and most important of this is power, marked as V_{CC} with a GND (ground potential). XTAL 1 and XTAL 2 are for the system clock inputs from crystal clock circuit. RESET input is required to initialize microcontroller to default/desired values and to make a new start. There are control signals, EA, PSEN and ALE. These signals known as External Access (EA), Program Store Enable (PSEN) and Address Latch Enable (ALE); used for external memory interfacing.



As mentioned above, control signals are used for external memory interfacing. If there is no requirement of external memory interfacing then, EA pin is pulled high (connected to V_{cc}); two others PSEN & ALE are left alone. You can also see a 0.1µF decoupling capacitor connected to V_{cc} (to avoid HF oscillations at input). There are four ports numbered 0,1,2,3; called as Port 0, Port 1, Port 2 and Port 3 which are used for external interfacing of devices like DAC, ADC, 7-segment display, LED etc. Each port has 8 I/O lines and they all are bit programmable.

8051 Pin Diagram & Description:



For describing pin diagram and pin configuration of 8051, let's consider a 40 pin DIP.

Pin-40 (Vcc): Named as V_{CC} is the main power source; +5V DC. You may note some pins are designated with two signals (shown in brackets).

Pins 32-39 (Port0): Known as Port 0 (P0.0 to P0.7): In addition to serving as I/O port, lower order address and data bus signals are multiplexed with this port (to serve the purpose of external memory interfacing). This is a bi-directional I/O port (the only one in 8051) and external pull up resistors are required to function this port as I/O.

Pin-31 (EA): EA/External Access input is used to enable or disallow external memory interfacing. If there is no external memory requirement, this pin is connected it to V cc.

Pin-30 (ALE): Address Latch Enable is used to de-multiplex the address-data signal of port 0 (for external memory interfacing.) Two ALE pulses are available for each machine cycle.

Pin- 29 (PSEN): PSEN or Program Store Enable is used to read signal from external program memory.

Pins- 21-28 (Port 2): Known as Port 2 (P 2.0 to P 2.7); in addition to serving as I/O port, higher order address bus signals are multiplexed with this quasi bi-directional port.

Pin 20 (Vss): Represents ground (0 V) connection.

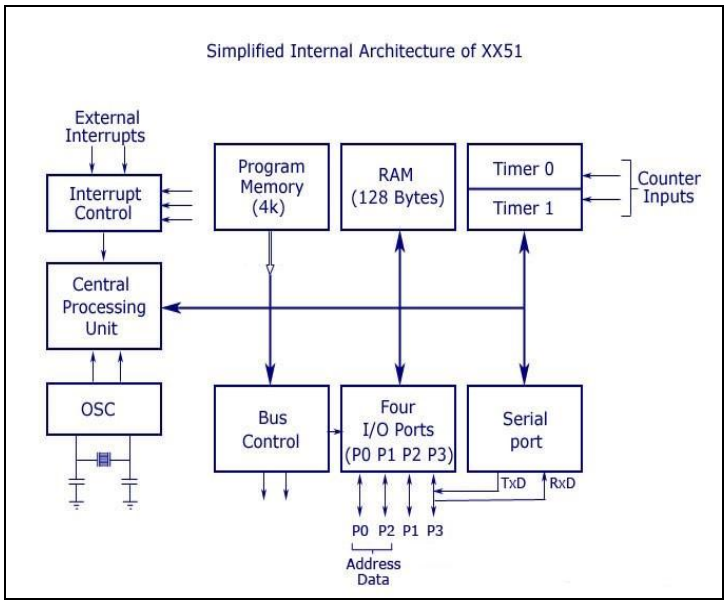
Pins 18 and 19: Use for interfacing an external crystal to provide system clock.

Pins 10–17 (Port 3): This port also serves some other functions like interrupts, timer input, control signals for external memory interfacing RD and WR, serial communication signals RxD and TxD etc. This is a quasi-bi-directional port with internal pull up.

Pin 9 (RESET): As explained before RESET pin is used to set the µC 8051 to its initial values, while the microcontroller is working or at the initial start of application. The RESET pin must be set high for two machine cycles.

Pins 1–8 (Port 1): Unlike other ports, this port does not serve any other functions. Port 1 is an internally pulled up, quasi-bi-directional I/O port.

8051 Internal Architecture: Just refer the diagram below and you observe it carefully. The system bus connects all the support devices with the central processing unit. 8051 system bus composes of an 8-bit data bus and a 16-bit address bus and bus control signals. From the figure, you can understand that all other devices like program memory, ports, data memory, serial interface, interrupt control, timers, and the central processing unit are all interfaced together through the system bus. RxD and TxD (serial port input and output) are interfaced with port 3.

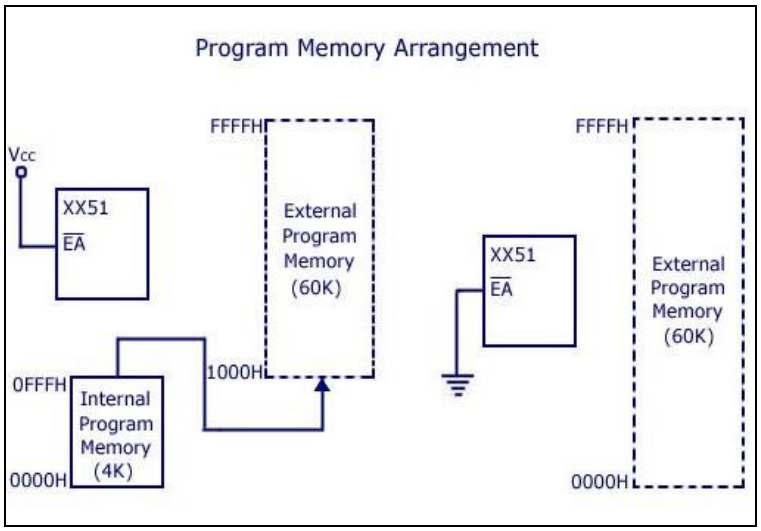


8051 Memory Organization: Before going deep into the memory architecture of 8051, let's talk a little bit about two variations available for the same. They are Princeton architecture and Harvard architecture. Princeton architecture treats address memory and data memory as a single unit (does not distinguish between two) whereas Harvard architecture treats program memory and data memory as separate entities. Thus, Harvard architecture demands address, data and control bus for accessing them separately whereas Princeton architecture does not demand any such separate bus. 8051 micro controllers are based on Harvard architecture and microprocessor 8085 is based on Princeton architecture.

Program Memory & Data Memory:

Program Memory Organization: Now let's dive into the program memory organization of 8051. It has an internal program of 4KB size and if needed an external memory can be added (by interfacing) of size 60KB maximum. So, in total 64KB size memory is available for 8051 micro controllers. By default, the External Access (EA) pin should be connected to Vcc so that instructions are fetched from internal memory initially. When the limit of internal memory (4KB) is crossed, control will automatically move to external memory to fetch remaining instructions. If the programmer wants to fetch instruction from external memory only (by passing the internal memory), then the programmer must connect External Access (EA) pin to ground (GND).

You may already know that 8051 has a special feature of locking the program memory (internal) and hence protecting against software piracy. This feature is enabling by program lock bits. Once these bits are programmed, contents of internal memory cannot be accessed using an external circuitry. However, locking the software is not possible if external memory is also used to store the software code. Only internal memory can be locked and protected. Once locked, these bits can be unlocked only by a memory-erase operation, which in turn will erase the programs in internal memory too. 8051 is capable of pipelining. Pipelining makes a processor capable of fetching the next instruction while executing previous instruction. It's something like multi-tasking, doing more than one operation at a time. µC 8051 is capable of fetching first byte of the next instruction while executing the previous instruction.

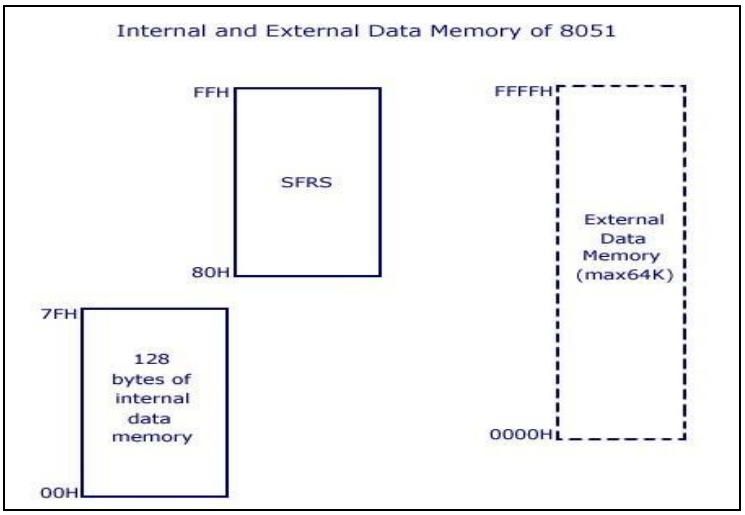


Data Memory Organization: In the MCS-51 family, 8051 has 128 bytes of internal data memory and it allows interfacing external data memory of maximum size up to 64KB. So, the total size of data memory in 8051 can be up to 64KB (external) +128 bytes (internal). To get more idea observe the diagram carefully. So, there are three divisions of the data memory:

1) Register banks

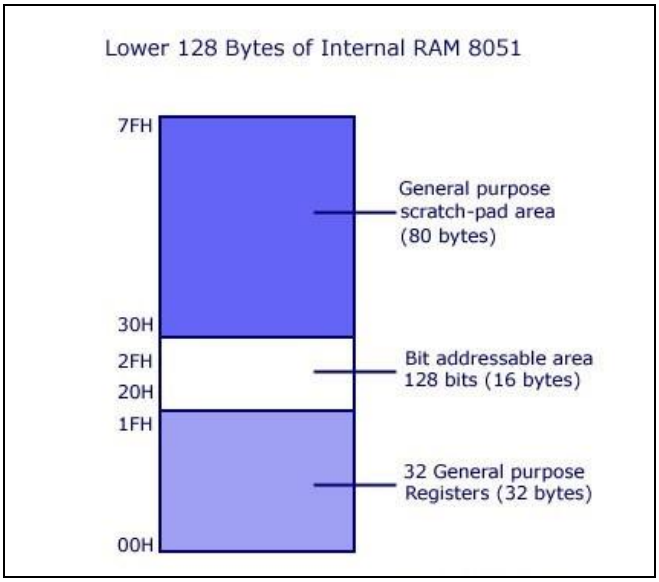
2) Bit addressable area

3) Scratch pad area.



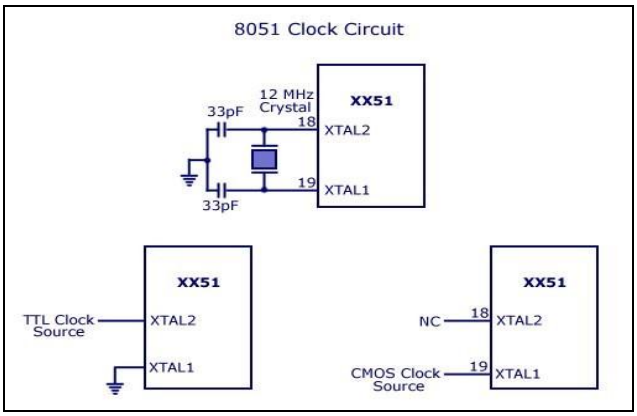
Register banks form the lowest 32 bytes on internal memory and there are four register banks designated as #0, #1, #2 and #3. Each bank has eight registers which are designated as R0, R1…R7. At a time only one register bank is selected for operations and the registers inside the selected bank are accessed using mnemonics R0…R1… etc. Other registers can be accessed simultaneously only by direct addressing. Registers are used to store data or operands during executions. By default, register bank #0 is selected (after a system reset).

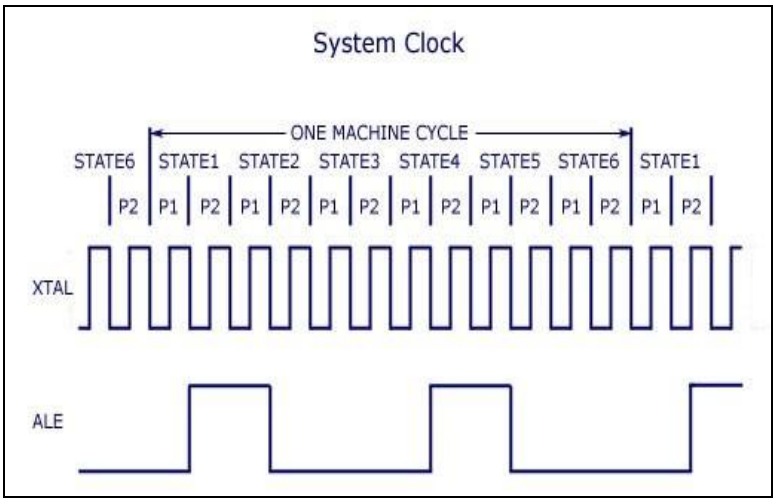
The bit addressable area of 8051 is usually used to store bit variables. The bit addressable area is formed by the 16-bytes next to register banks. They are designated from address 20H to 2FH (total 128 bits). Each bit can be accessed from 00H to 7FH within these 128 bits from 20H to 2FH. Bit addressable area is mainly used to store bit variables from application program, like status of an output device like LED or Motor (ON/OFF) etc. We need only a bit to store this status and using a complete byte addressable area for storing this is really bad programming practice, since it results in wastage of memory.



The scratch pad area is the upper 80 bytes which is used for general purpose storage. Scratch pad area is from 30H to 7FH and this includes stack too.

8051 System Clock: An 8051-clock circuit is shown below. In general cases, a quartz crystal is used to make the clock circuit. The connection is shown in figure below and note the connections to XTAL 1 and XTAL 2. In some cases, external clock sources are used and you can see the various connections. Clock frequency limits (maximum and minimum) may change from device to device. Standard practice is to use 12MHz frequency. If serial communications are involved there then it's best to use 11.0592 MHz frequency.





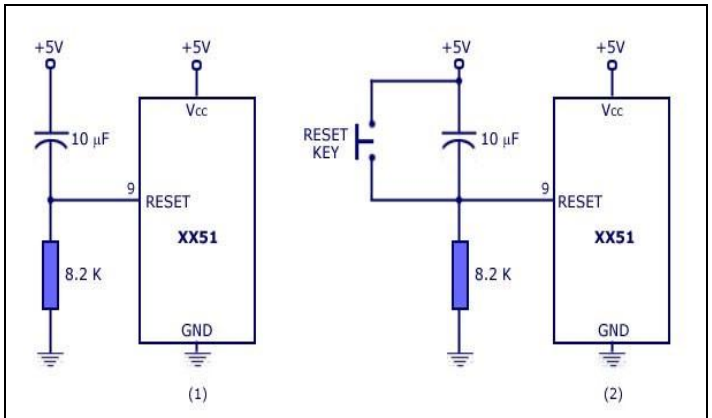
Take a look at the above machine cycle waveform. One complete oscillation of the clock source is called a pulse. Two pulses form a state and six states forms one machine cycle. Also note that, two pulses of ALE are available for 1 machine cycle.

8051 Reset Circuit:

8051 can be reset in two ways

1) Power-on reset; resets µC 8051 when power is turned ON.
2) Manual reset; in which a reset happens only when a push button is pressed manually.

   Two different reset circuits are shown below. A reset doesn't affect contents of internal RAM. For reset to happen, the reset input pin (pin 9) must be active high for at least 2-machine cycles. During a reset operation Program counter is cleared and it starts from 00H, register bank #0 is selected as default, Stack pointer is initialized to 07H, all ports are written with FFH. The assembly language is a low-level programming language used to write program code in terms of mnemonics. Assembly programming language is popularly used in many applications. It can be used for direct hardware manipulations. It is also used to write the 8051-programming code efficiently with a smaller number of clock cycles by consuming less memory compared to the other high-level languages.



(1) Power-on Reset Circuit and (2) With Manual Reset Option

Addressing Modes:

1. Immediate Addressing Mode: In this type, the operand is specified in the instruction along with the op-code. In simple way, it means data is provided in instruction itself. Ex: MOV A, #05H; Where MOV stands for move, # represents

immediate data, 05H is the data. It means the immediate data 05H, provided in instruction is moved into “A” register.

2. Register Addressing Mode: Here the operand in contained in the specific register of microcontroller. The user must provide the name of register from where the operand/data need to be fetched. The permitted registers are A, R_7-R_0 of each register bank. Ex: MOV A, R_0; content of R_0 register is copied into Accumulator.
3. Direct Addressing Mode: In this mode, the direct address of memory location is provided in instruction to fetch the operand. Only internal RAM and SFR's address can be used in this type of instruction. Example: MOV A, 30H; Content of RAM address 30H is copied into Accumulator.
4. Register Indirect Addressing Mode: Here the address of memory location is indirectly provided by a register. The '@' sign indicates that the register holds the address of memory location i.e. fetch the content of memory location whose address is provided in register. Ex: MOV A, @R_0; Copy the content of memory location whose address is given in R_0 register.
5. Indexed Addressing Mode: This addressing mode is basically used for accessing data from look up table. Here the address of memory is indexed i.e. added to form the actual address of memory. Ex: MOVC A, @A+DPTR; here 'C' means Code. Here the content of “A” register is added with content of DPTR and the resultant is the address of memory location from where the data is copied to “A” register.

To program any microcontroller, first you need to learn and understand its instruction set. Instruction set contains a set of instructions that is available for the programmer to write any kind of program. So, first of all, one need to learn all available instructions, how an instruction works, how the execution of an instruction affects the microcontroller (affecting the registers, PSW, stack etc.) & the way it is used in a program. Once the instruction set is learned, you can start playing with them to write the best possible programs with minimum possible bytes.

μC 8051belongs to MCS-51 family of micro controllers. This basically means any 8051-variant microcontroller (that comes under the MCS-51 family) made by any other manufacturer must use the same set of instructions made for MCS-51. So, the "instruction decoder" part of all micro controllers under MCS-51 family is same.

Example: Atmel's AT89C2051 is one such micro controller that falls under MCS-51 family. So, a program written for Intel 8051 can be used to run AT89C2051 too except you may have to make slight modifications to match hardware disparities.

1. Data transfer Operations: MOV, MOVX, PUSH, XCH, XCHD
2. Arithmetic Operations: ADDC, SUBB, MUL, DIV, INC, DA A
3. Branching Operations: LJMP, AJMP, SJMP, LCALL, ACALL JZ, JNZ, CJNE, DJNZ, JMP, NOP, RET, RETI
4. Logical Operations: ANL, XRL, CPL, RL, RLC, RR, SWAP
5. Boolean Operations: SETB, CLR, JB, JNB, JBC, JC

CHAPTER – 3

Understanding Instruction Set of 8051

A. Data Transfer Instructions:

Mnemonic	Instruction code								Code	Explanation
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
MOV A, Rn	1	1	1	0	1	n_2	n_1	n_0	E8~EF	(A)←(Rn)
MOV A, direct	1	1	1	0	0	1	0	1	E5	(A)←(direct)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
MOV A, @Ri	1	1	1	0	0	1	1	i	E6~E7	(A)←((Rn))
MOV A, #data	0	1	1	1	0	1	0	0	74	(A)←(data)
	D7	D6	D5	D4	D3	D2	D1	D0	Byte 2	
MOV Rn, A	1	1	1	1	1	n_2	n_1	n_0	F8~FF	(Rn)←(A)
MOV Rn, direct	0	1	1	1	1	n_2	n_1	n_0	A8~AF	(Rn)←(direct)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
MOV Rn, #data	0	1	1	1	1	n_2	n_1	n_0	78~7F	(Rn)←#data
	D7	D6	D5	D4	D3	D2	D1	D0	Byte 2	
MOV direct, A	1	1	1	1	0	1	0	1	F5	(direct) ←(A)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
MOV direct, Rn	1	0	0	0	1	n_2	n_1	n_0	88~8F	(direct) ←(Rn)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	

MOV direct, direct	1	0	0	0	0	1	0	1	85	(direct) ←(direct)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 3	
MOV direct, @Ri	1	0	0	0	0	1	1	i	86~87	(direct)←((Ri))
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
MOV direct, #data	0	1	1	1	0	1	0	1	75	(direct) ←#data
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
	D7	D6	D5	D4	D3	D2	D1	D0	Byte 3	
MOV @Ri, A	1	1	1	1	0	1	1	i	F6~F7	((Ri)) ← A
MOV @Ri, direct	1	0	1	0	0	1	1	i	A6~A7	((Ri)) ← (direct)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
MOV @Ri, #data	0	1	1	1	0	1	1	i	76~77	((Ri)) ←#data
	D7	D6	D5	D4	D3	D2	D1	D0	Byte 2	
MOV DPTR, #data-16	1	0	0	1	0	0	0	0	90	(DPTR)←#data
	D7	D6	D5	D4	D3	D2	D1	D0	Byte 1	(DPH) ← $\#data_{15\sim8}$
	D7	D6	D5	D4	D3	D2	D1	D0	Byte 2	(DPL) ←$\#data_{7\sim0}$
MOVC A, @A + DPTR	1	0	0	1	0	0	1	1	93	A←((A) +(DPTR))
MOVC A, @A + PC	1	0	0	0	0	0	1	1	83	A ← ((A) + (PC))
MOVX A, @Ri	1	1	1	0	0	0	1	i	E2~E3	A←((Ri)) Ext Memory
MOVX A, @DPTR	1	1	1	0	0	0	0	0	E0	A←((DPTR))
MOVX @Ri, A	1	1	1	1	0	0	1	i	F2~F3	((Ri)) ← A
MOVX @DPTR, A	1	1	1	1	0	0	0	0	F0	((DPTR)) ←A
PUSH direct	1	1	0	0	0	0	0	0	C0	(SP)←(SP)+1
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	((SP)) ←(direct)

POP direct	1	1	0	1	0	0	0	0	D0	(direct)← ((SP))
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	(SP)←(SP)-1
XCH A, Rn	1	1	0	0	1	n_2	n_1	n_0	C8~CF	(A)↔(Rn)
XCH A,	1	1	0	0	0	1	0	1	C5	(A)↔(direct)
Direct	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
XCH A, @Ri	1	1	0	0	0	1	1	i	C6~C7	(A)↔((Ri))
XCHD A, @Ri	1	1	0	1	0	1	1	i	D6~D7	

B. Arithmetic Instructions:

Mnemonic	Instruction code								Hex code	Explanation
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
ADD A, Rn	0	0	1	0	1	n_2	n_1	n_0	28~2F	(A)←(A)+(Rn)
ADD A, direct	0	0	1	0	0	1	0	1	25	(A)←(A)+(direct)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
ADD A, @Ri	0	0	1	0	0	1	1	i	26~27	(A)←(A)+((Ri))
ADD A,	0	0	1	0	0	1	0	0	24	(A)←(A)+ #data
#data	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Byte 2	
ADDC A, Rn	0	0	1	1	1	n_2	n_1	n_0	38~3F	(A)←(A)+(C)+(Rn)
ADDC A, direct	0	0	1	1	0	1	0	1	35	(A)←(A)+(C)+
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	(direct)
ADDC A, @Ri	0	0	1	1	0	1	1	i	36~37	(A)←(A)+(C)+((Ri))
ADDC A,	0	0	1	1	0	1	0	0	34	(A)←(A)+(C)+ #data
#data	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Byte 2	

SUBB A, Rn	1	0	0	1	1	n_2	n_1	n_0	98~9F	(A)←(A)-(C)-(Rn)
SUBB A, direct	1	0	0	1	0	1	0	1	95	(A)←(A)-(C)-(direct)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
SUBB A, @Ri	1	0	0	1	0	1	1	i	96~97	(A)←(A)-(C)-((Ri))
SUBB A, #data	1	0	0	1	0	1	0	0	94	(A)←(A)-(C)- #data
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Byte 2	
INC A	0	0	0	0	0	1	0	0	04	(A)←(A)+1
INC Rn	0	0	0	0	1	n_2	n_1	n_0	08~0F	(Rn)←(Rn)+1
INC direct	0	0	0	0	0	1	0	1	05	(direct)←(direct)+1
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
INC @Ri	0	0	0	0	0	1	1	i	06~07	((Ri)) ← ((Ri))+1
INC DPTR	1	0	1	0	0	0	1	1	A3	(DPTR)←(DPTR)+1
DEC A	0	0	0	1	0	1	0	0	14	(A)←(A)-1
DEC Rn	0	0	0	1	1	n_2	n_1	n_0	18~1F	(Rn)←(Rn)-1
DEC direct	0	0	0	1	0	1	0	1	15	(direct)←(direct)-1
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
DEC @Ri	0	0	0	1	0	1	1	i	16~17	((Ri)) ← ((Ri))-1
MUL AB	1	0	1	0	0	1	0	0	A4	$(B_{15\sim8}),(A_{7\sim0})$ ←(A)×(B)
DIV AB	1	0	0	0	0	1	0	0	84	$(A_{15\sim8}),(B_{7\sim0})$ ←(A)/(B)
DAA	1	1	0	1	0	1	0	0	D4	Decimal adjust accumulator

C. Logical Instructions:

Mnemonics	Instruction Code								Hex code	Explanation
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
ANL A, Rn	0	1	0	1	1	n_2	n_1	n_0	58~5F	(A)←(A) AND (Rn)
ANL A, direct	0	1	0	1	0	1	0	1	55	(A)←(A) AND
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	(direct)
ANL A, @Ri	0	1	0	1	0	1	1	i	56~57	(A)←(A) AND ((Ri))
ANL A, #data	0	1	0	1	0	1	0	0	54	(A)←(A) AND
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Byte 2	#data
ANL direct, A	0	1	0	1	0	0	1	0	52	(direct)←(A) AND
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	(direct)
ANL direct, #data	0	1	0	1	0	0	1	1	53	(direct)←(direct)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	AND #data
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Byte 3	
ORL A, Rn	0	1	0	0	1	n_2	n_1	n_0	48~4F	(A)←(A) OR (Rn)
ORL A, direct	0	1	0	0	0	1	0	1	45	(A)←(A) OR
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	(direct)
ORL A, @Ri	0	1	0	0	0	1	1	i	46~47	(A)←(A) OR ((Ri))
ORL A, #data	0	1	0	0	0	1	0	0	44	(A)←(A) OR
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Byte 2	#data
ORL direct, A	0	1	0	0	0	0	1	0	42	(direct)←(A) OR
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	(direct)
ORL direct, #data	0	1	0	0	0	0	1	1	43	(direct)←(direct)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	OR #data
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Byte 3	

XRL A, Rn	0	1	1	0	1	n_2	n_1	n_0	68~6F	(A)←(A) XOR (Rn)
XRL A, direct	0	1	1	0	0	1	0	1	65	(A)←(A) XOR
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	(direct)
XRL A, @Ri	0	1	1	0	0	1	1	i	66~67	(A)←(A) XOR ((Ri))
XRL A, #data	0	1	1	0	0	1	0	0	64	(A)←(A) XOR
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Byte 2	#data
XRL direct, A	0	1	1	0	0	0	1	0	62	(direct)←(A) XOR
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	(direct)
XRL direct, #data	0	1	1	0	0	0	1	1	63	(direct)←(direct)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	XOR #data
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Byte 3	
CLR A	1	1	1	0	0	1	0	0	E4	(A)←0
CPL A	1	1	1	1	0	1	0	0	F4	(A)←(Ã)
RL A	0	0	1	0	0	0	1	1	23	Content of acc. rotated left by 1 bit ($A_7 \rightarrow A_0$)
RLC A	0	0	1	1	0	0	1	1	33	Content of acc. rotated left by 1 bit with content of carry bit ($A_7 \rightarrow C \rightarrow A_0$)
RR A	0	0	0	0	0	0	1	1	03	Content of acc. rotated right by 1 bit ($A_7 \rightarrow A_0$)
RRC A	0	0	0	1	0	0	1	1	13	Content of acc. rotated right by 1 bit with content of carry bit ($A_7 \rightarrow C \rightarrow A_0$)
SWAP A	1	1	0	0	0	1	0	0	C4	(A_3~A_0) ↔ (A_7~A_4)

D. Control Transfer Instructions:

Mnemonics	Instruction Code								Hex code	Explanation
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
ACALL addr 11									Byte 1	Program will jump within 2k memory
	AA	A9	A8	1	0	0	0	1	Byte 2	
	A7	A6	A5	A4	A3	A2	A1	A0		
LCALL addr 16	0	0	0	1	0	0	1	0	12	Program will jump within 64k memory
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 3	
RET	0	0	1	0	0	0	1	0	22	
RET1	0	0	1	1	0	0	1	0	32	
AJMP addr 11										
	AA	A9	A8	1	0	0	0	1	Byte 1	
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
LJMP addr 16	0	0	0	0	0	0	1	0	02	
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 1	
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
SJMP rel	1	0	0	0	0	0	0	0	80	
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 1	
JMP @A+DPTR	0	1	1	1	0	0	1	1	73	(PC)←(PC)+(A)
JZ rel	0	1	1	0	0	0	0	0	60	If (A)=0, then
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 1	(PC)←(PC)+(rel)
JNZ rel	0	1	1	1	0	0	0	0	70	If (A)≠0, then
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 1	(PC)←(PC)+(rel)

JC rel	0	1	0	0	0	0	0	0	40	If (C)=1, then
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 1	(PC)←(PC)+(rel)
JNC rel	0	1	0	1	0	0	0	0	50	If (C)=0, then
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 1	(PC)←(PC)+(rel)
JB bit, rel	0	0	1	0	0	0	0	0	20	If (bit)=1, then
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 1	(PC)←(PC)+(rel)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
JNB bit, rel	0	0	1	0	0	0	0	0	30	If (bit)=0, then
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 1	(PC)←(PC)+(rel)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
JBC bit, rel	0	0	0	1	0	0	0	0	10	If bit=1,
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 1	then bit → (0)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	then
										(PC)←(PC)+(rel)
CJNE A, direct, rel	1	0	1	1	0	1	0	1	B5	If (direct) <(A) then (C)← (0)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 1	&
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	(PC)←(PC)+(rel)
										If (direct) >(A) then (C)← (1)
										&
										(PC)←(PC)+(rel)
CJNE A, #data, rel	1	0	1	1	0	1	0	0	B4	If (data) <(A) then (C)← (0) &
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 1	(PC)←(PC)+(rel)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	If (data) >(A) then (C)← (1) &
										(PC)←(PC)+(rel)

CJNE Rn, #data, rel	1	0	1	1	1	n_2	n_1	n_0	B8~BF
									Byte 1
									Byte 2
										If (data) <(Rn) then (C)←(0)
										&
										(PC)←(PC)+(rel)
										If (data) >(Rn) then (C)←(1)
										&
										(PC)←(PC)+(rel)

	A7	A6	A5	A4	A3	A2	A1	A0		
	A7	A6	A5	A4	A3	A2	A1	A0		

CJNE @Ri, #data, rel	1	0	1	1	0	1	1	i	B6~B7
									Byte 1
									Byte 2
										If (data) <((Ri)) then (C)←(0)
										& (PC)←(PC)+(rel)
										If (data) >((Ri)) then (C)←(1)
										& (PC)←(PC)+(rel)

	A7	A6	A5	A4	A3	A2	A1	A0		
	A7	A6	A5	A4	A3	A2	A1	A0		
DJNZ Rn, rel	1	1	0	1	1	n_2	n_1	n_0	D8~DF Byte 1	(Rn)←(Rn)-1 Then if Rn≠0 Then (PC)←(PC)+(rel)
	A7	A6	A5	A4	A3	A2	A1	A0		
DJNZ direct, rel	1	1	0	1	1	n_2	n_1	n_0	D5 Byte 1 Byte 2	(direct)←(direct)-1 Then if direct≠0 Then (PC)←(PC)+(rel)
	A7	A6	A5	A4	A3	A2	A1	A0		
	A7	A6	A5	A4	A3	A2	A1	A0		
NOP	0	0	0	0	0	0	0	0	00	No operation

E. Bit Control Instructions:

Mnemonics	Instruction Code								Hex code	Explanation
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
CLR C	1	1	0	0	0	0	1	1	C3	(C)←(0)
CLR bit	1	1	0	0	0	0	1	0	C2	(bit)←(0)
SETB C	1	1	0	1	0	0	1	1	D3	(C)←(1)
SETB bit	1	1	0	1	0	0	1	0	D2	(bit)←(1)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
CPL C	1	0	1	1	0	0	1	1	B3	(C)←(c̄)
CPL bit	1	0	1	1	0	0	1	0	B2	(bit)←(b̄it)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
ANL C, bit	1	0	0	0	0	0	1	0	82	(C)←(C) AND (bit)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
ORL C, bit	0	1	1	1	0	0	1	0	72	(C)←(C) OR (bit)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
MOV C, bit	1	0	1	0	0	0	1	0	A2	(C)←(bit)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	
MOV bit, C	1	0	0	1	0	0	1	0	92	(bit)←(C)
	A7	A6	A5	A4	A3	A2	A1	A0	Byte 2	

CHAPTER – 4

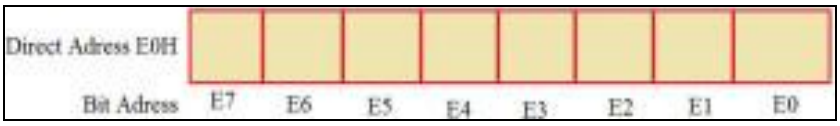
8051: Special Function Registers

The 8051 Microcontroller Special Function Registers are used to program and control different peripherals like Timers, Serial Port, I/O Ports etc. In fact, by appropriate use of the SFRs, one can assess or change the operating mode of the μC 8051

1. Math / CPU Registers: A and B
2. Status Register: PSW (Program Status Word)
3. Pointer Registers: DPTR (Data Pointer: DPL, DPH) and SP (Stack Pointer)
4. I/O Port Latches: P0 (Port 0), P1 (Port 1), P2 (Port 2) and P3 (Port 3)
5. Peripheral Control Registers: PCON, SCON, TCON, TMOD, IE and IP
6. Peripheral Data Registers: TL0, TH0, TL1, TH1 and SBUF

CPU or Math Registers:

A or Accumulator (ACC): The Accumulator or Register A is the most important and most used 8051 Microcontroller SFRs. The Register A is located at the address E0H in the SFR memory space. The Accumulator is used to hold the data for almost all the arithmetic & logic operations. The accumulator is both bit and byte addressable.



Example: Using A Register:

The Assembly program for subtraction used with an Accumulator

Org 0000h

MOV R0, #09h

MOV A, #03h (1byte data)

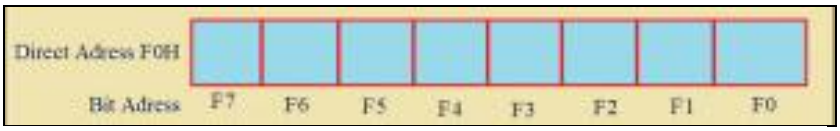
SUBB A, 01h (1byte data)

END

Operations with aaccumulator:

- Arithmetic: Addition, Subtraction, Multiplication & Div.
- Logical: AND, OR, NOT etc.
- Data Transfer (between 8051 & External Memory)

B (Register B): The B Register is used along with the ACC in Multiplication and Division operations. These two operations are performed on data that are stored only in Registers A and B. During Multiplication Operation, one of the operands (multiplier or multiplicand) is stored in B Register and also the higher byte of the result. In case of Division Operation, the B Register holds the divisor and also the remainder of the result. It can also be used as a general-purpose register for normal operations and is often used as an auxiliary register by programmers to store temporary results. Register B is located at the address F0H of the SFR address space.



The B-register is a bit as well as byte-addressable register. You can access 1-bit or all 8-bits by a physical address F0H.

Example Using B-Register:

The Assembly program for multiplication used with a B-Register

Org0000h

MOVA, #09h

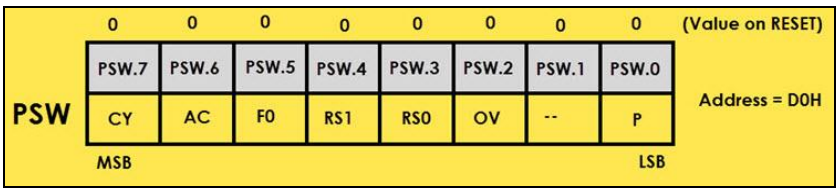
MOVB, #03h

MUL A, B (Final value stored in A)

END

Program Status Word (PSW): The PSW is also called as flag register and is one of the important SFRs. The PSW register consists of flag bits, which help the programmer in checking the condition of the result and also to make decisions. Flags are 1-bit storage elements that store and indicate the nature of the result

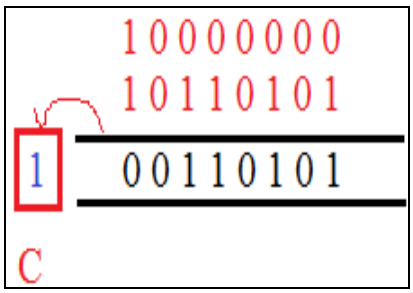
that is generated by execution of certain instructions. The following image shows the contents of the PSW Register.



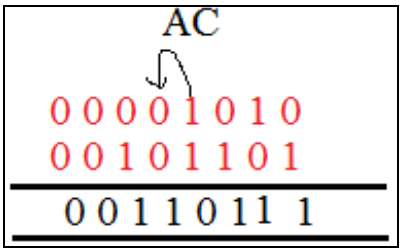
BIT	SYMBOL	FLAG NAME			DESCRIPTION
7	C or CY	Carry			Used in Arithmetic, Logic & Boolean Operations
6	AC	Auxiliary Carry			Used in BCD Arithmetic
5	F0	Flag 0			General Purpose User Flag
4	RS1	Register Bank Selection Bit 1			
3	RS0	Register Bank Selection Bit 1			
		RS1	RS0	Bank	
		0	0	Bank 0	
		0	1	Bank 1	
		1	0	Bank 2	
		1	1	Bank 3	
2	OV	Overflow			Used in Arithmetic Operations
1	--	Reserved			May be used as a General Purpose Flag
0	P	Parity			Set to 1 if A has odd # of 1's; otherwise Reset

Example: Understanding PSW

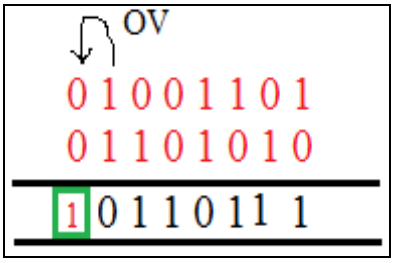
Carry Flag (C): The location of the carry flag is D7. This carry flag is affected when the bit is generated from the 7th position. When C=0 ; carry bit resets, C=1; carry bit sets.



Auxiliary Flag (AC): The location of the auxiliary carry bit is D5. This auxiliary carry is affected when a bit is generated from the 3rd position to the 4th position. AC= 0 auxiliary is reset, AC=1 auxiliary is set.



Overflow Flag (OV): The address of the overflow flag bit is D2. When a bit is generated from the 6th position to the 7th position, then the overflow flag is affected. OV=0 overflow flag bit resets, OV=1 overflow flag bit set



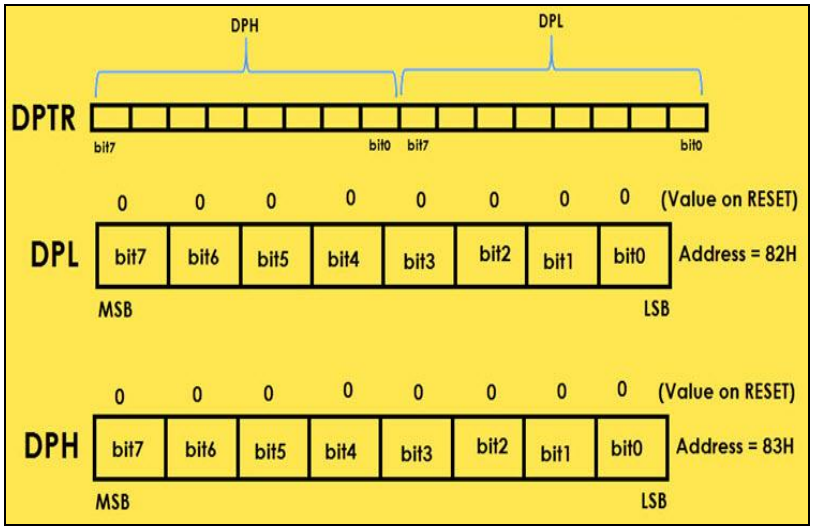
Parity Flag (P): The address of the parity flag bit is D0. While performing arithmetic operations, if the parity of the resultant byte is ODD this bit will SET and if parity is EVEN this bit will reset.

- **SFR Memory Address:**

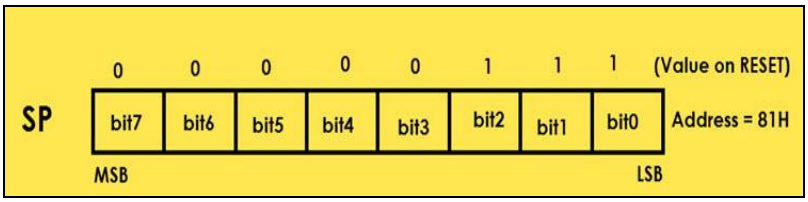
Direct byte address	MSB							LSB	Special function reg. symbol
$(F0)_H$	F7	F6	F5	F4	F3	F2	F1	F0	B
$(E0)_H$	E7	E6	E5	E4	E3	E2	E1	E0	ACC
$(D0_H$	D7	D6	D5	D4	D3	D2	D1	D0	PSW
$(B8)_H$	BF	BE	BD	BC	BB	BA	B9	B8	IP
$(B0)_H$	B7	B6	B5	B4	B3	B2	B1	B0	P3
$(A8)_H$	AF	AE	AD	AC	AB	AA	A9	A8	IE
$(A0)_H$	A7	A6	A5	A4	A3	A2	A1	A0	P2
$(99)_H$	Not bit Addressable								SBUF
$(98)_H$	SMO	SM1	SM2	REN	TBB	RBB	TI	RI	SCON
	9F	9E	9D	9C	9B	9A	99	98	
$(90)_H$	97	96	95	94	93	92	91	90	P1
$(8D)_H$	Not bit-addressable								TH1
$(8C)_H$	Not bit-addressable								TH0
$(8B)_H$	Not bit-addressable								TL1
$(8A)_H$	Not bit-addressable								TL0
$(89)_H$	Not bit-addressable								TMOD
$(88)_H$	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	TCON
	8F	8E	8D	8C	8B	8A	89	88	
$(87)_H$	Not bit-addressable								PCON
$(83)_H$	Not bit-addressable								DPH
$(82)_H$	Not bit-addressable								DPL
$(81)_H$	Not bit-addressable								SP
$(80)_H$	87	86	85	84	83	82	81	80	PO

Pointer Registers:

1. Data Pointer (DPTR – DPL & DPH): The data pointer is a 16-bit register and is physically the combination of DPL (Data Pointer Low) and DPH (Data Pointer High) SFRs. The Data Pointer can be used as a single 16-bit register (as DPTR) or two 8-bit registers (as DPL and DPH). DPTR doesn't have a physical memory address but the DPL (Lower byte of DPTR) and DPH (Higher byte of DPTR) have separate addresses in the SFR memory space. DPL = 82H and DPH = 83H.The DPTR register is used by the programmer addressing external memory (Program – ROM or Data – RAM).

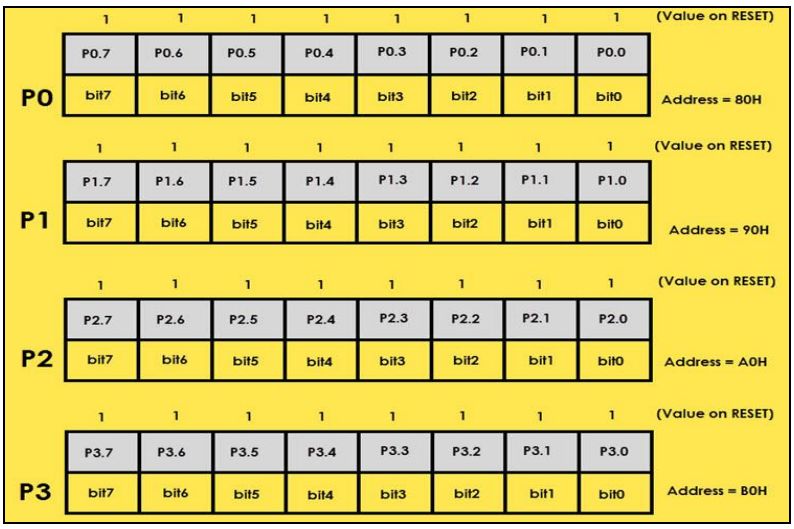


2. Stack Pointer (SP): SP or stack pointer points out to the top of the stack and it indicates the next data to be accessed. Stack pointer can be accessed using PUSH, POP, CALL and RET instructions. The stack pointer is an 8-bit register and upon reset, the stack pointer is initialized with 07H. When writing a new data byte into the stack, the SP (Stack Pointer) is automatically incremented by 1 and the new data is written at an address SP+1. When microcontroller is reading data from stack, the data is retrieved from the address in SP and after that the SP is decremented by 1 (SP-1).



3. I/O Port Registers (P0, P1, P2 and P3): The 8051 microcontroller four ports which can be used as input and/or output. These four ports are P0, P1, P2 and P3. Each Port has a corresponding register with same names (the port registers are also P0, P1, P2 and P3). The addresses of the port registers are, P0: 80H, P1: 90H, P2: A0H and P2: B0H. Each bit in these SFRs corresponds to one physical pin in the 8051 microcontrollers. All these port registers are both bit

addressable and byte addressable. Writing 1 or 0 on a port register bit will reflect as an appropriate voltage (5V and 0V) on the corresponding pin. If a port bit is SET (declared as 1), the corresponding port pin will be configured as input and similarly if a port bit is CLEARED (declared as 0), the corresponding port pin is configured as output. Upon reset, all the port bits are SET (1) and hence, all the port pins are configured as inputs.

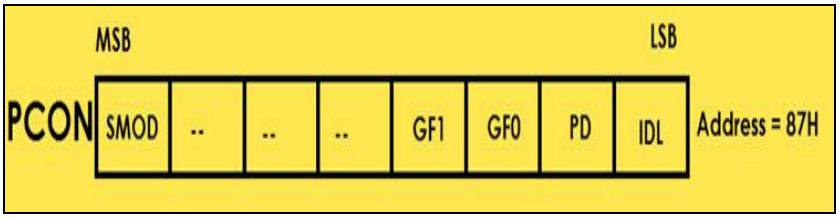


Example: Using Ports

Assembly program to toggle LEDs of Port 0

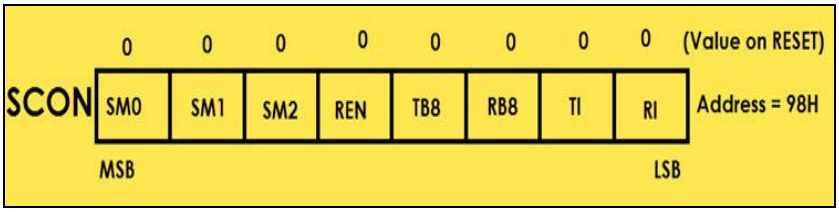
```
ORG0000h
RETURN: MOVP0, #00h
ACALL DEL1
MOVP0, #0FF
ACALL DEL1
SJMP RETURN
DEL1:MOVR2, #200
DEL: DJNZR0, #230
DJNZ R2, DEL
RET
END
```

4. PCON (Peripheral Control Register): The PCON or power control register, as the name suggests is used to control the 8051 microcontroller's power modes and is located at 87H of the SFR memory space. Using two bits in the PCON register, the microcontroller can be set to idle mode and power down mode. During idle mode, the microcontroller will stop the clock signal to the ALU (CPU) but it is given to other peripherals like Timer, Serial, Interrupts, etc. In order to terminate the idle mode, you have to use an interrupt or hardware reset.

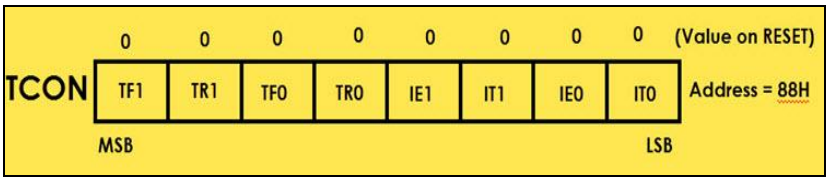


In the power down mode, the oscillator will be stopped and the power will be reduced to 2V. To terminate the power down mode, you have to use the hardware reset. Apart from these two, the PCON register can also be used for few additional purposes. The SMOD bit in the PCON register is used to control the baud rate of the serial port. There are two general purpose flag bits in the PCON register, which can be used by the programmer during execution.

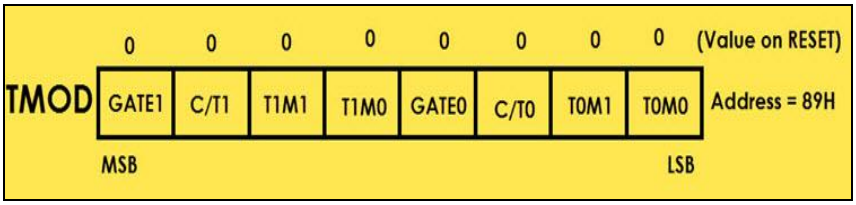
5. SCON (Serial Control): The serial control or SCON SFR is used to control the 8051 microcontroller's serial port. It is located at the address of 98H. Using SCON, you can control the operation modes of the serial port, Baud rate of the serial port and send or receive data using serial port. SCON register also consists of bits that are automatically SET when a byte of data is transmitted or received.



6. TCON (Timer Control): Timer control or TCON register is used to start or stop the timers of 8051 microcontroller. It also contains bits to indicate if the timers get overflowed. The TCON SFR also consists of interrupt related bits.



7. TMOD (Timer Mode):The TMOD or Timer Mode register or SFR is used to set the ooperating modes of the timers T0 and T1. The lower four bits are used to configure timer0 and the higher four bits are used to configure timer1.



M1, M0:

- M0 and M1 select the timer mode.
- There are three modes: 0,1 and 2.
- Mode 0 is a 13-bit timer
- Mode 1 is a 16-bit timer
- Mode 2 is an 8-bit timer

C/T (Clock/Timer)

- This bit in the TMOD register is used to decide whether the timer is used as a delay generator or an event counter.
- If C/T = 0, then it is used as a timer for time delay generation.
- The clock source for the time delay is the crystal frequency of the μC 8051

If C/T = 1 then it is used as event counter.

Clock Source

- Every timer needs a clock pulse to tick.
- When C/T = 0, the crystal frequency attached to the 8051 is

the source of the clock for the timer.

- This means that the size(frequency) of the crystal frequency attached to the 8051 also decides the speed at which the 8051 timer ticks.
- The frequency for the timer is always $[1/12]^{th}$ the frequency of the crystal attached to the 8051.

GATE

- Every timer has a means of starting and stopping. Some timers do this by software, some by hardware and some have both software and hardware controls.
- The timer in the 8051 have both. The start and stop of the timer are controlled by way of software by the TR (Timer Start) bits TR0 and TR1.
- This is achieved by the instructions SETB TR1 and CLR TR1 for timer 1 and SETB TR0 and CLR TR0 for timer 0. The SETB instruction starts it and it is stopped by the CLR instruction.
- These instructions start and Stop the Timers as long as GATE = 0, in the TMODregister.
- The hardware way of starting and stopping the timer by an external source is achieved by making GATE =1
- When GATE = 0, means that that no external hardware is needed to start and stop the timers.
- In using software to start and stop the timer where GATE = 0, all we need are the instructions SET TRx and CLRTRx.

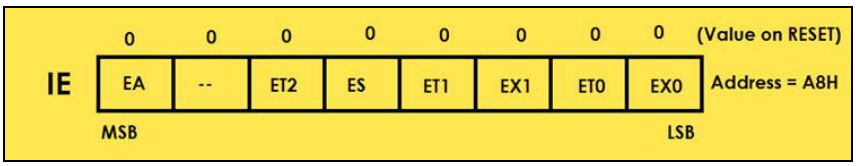
TIMERs @ µP 8051:

- The 8051 has two timers: Timer0 and Timer1.
- They can be used either as timers or as event counters.
- Both Timer 0 and Timer 1 are 16 bits wide.
- Since the 8051 has an 8-bit architecture, each 16-bit timer is accessed as two separate registers of low byte and high byte.
- Timer 0 register: The 16-bit register of Timer 0 is accessed as low byte and high byte.
- The low byte register is called TL0 (Timer 0 Low byte) and

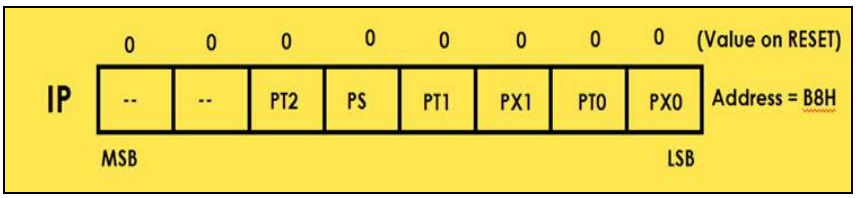
the high byte register is referred to as TH0 (Timer 0 highbyte).

- Timer 1 is also 16-bits, and its 16-bit register is split into bytes referred to as TL1 (Timer 1 low byte) and TH1 (Timer 1 highbyte).
- These registers can be accessed like any other registers.
- For example, the instructions MOV TLO, #4FH moves the value 4FH into TL0, the low byte of Timer0.
- These registers can also be read like any other register for example MOV R5, TH0, which saves the TH0 (high byte of Timer 0) value in R5.
- Both Timers 0 and 1 use the same register called TMOD, to set the various timer operation modes.

8. IE (Interrupt Enable): The IE or interrupt enable register is used to enable or disable individual interrupts. If a bit is SET, the corresponding interrupt is enabled and if the bit is cleared, the respective interrupt is disabled. The Bit 7 of the IE register i.e. EA bit is used to enable or disable all the interrupts.

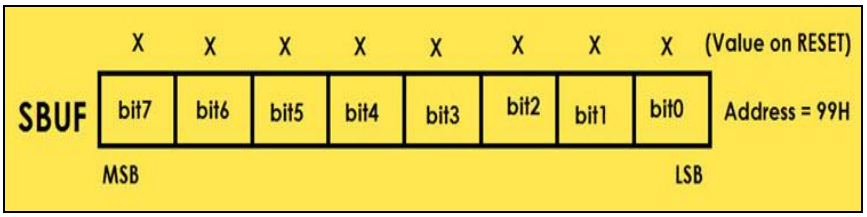


9. IP (Interrupt Priority): The IP or interrupt priority register is used to set the priority of the interrupt as high or low. If a bit is CLEARED, the corresponding interrupt is assigned low priority and if the bit is SET, the interrupt is assigned high priority.

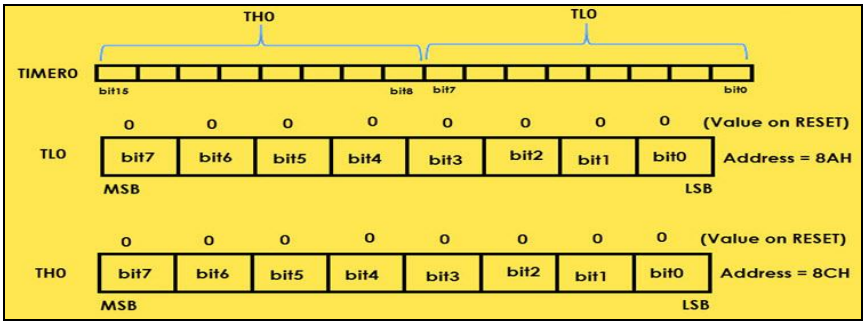


10. Peripheral Data Registers:

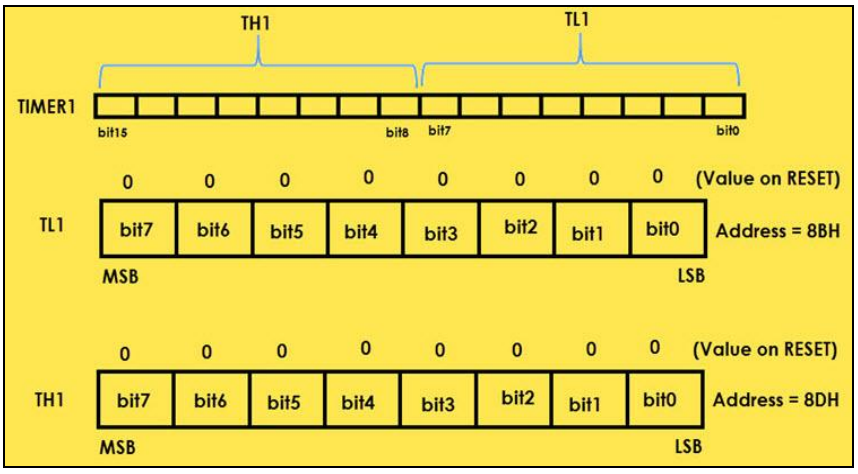
a. SBUF (Serial Data Buffer): The serial buffer or SBUF register is used to hold the serial data while transmission or reception.



b. TL0/TH0 (Timer 0 Low/High): The Timer 0 consists of two SFRs: TL0 and TH0. The TL0 is the lower byte and the TH0 is the higher byte and together they form a 16-bit Timer 0 Register.



c. TL1/TH1 (Timer 1 Low/High): The TL1 and TH1 are the lower and higher bytes of the Timer 0.



CHAPTER – 5

Fundamentals of Programming

D	H	O	B
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111

Q1. Find value of the (1111) B in D/H/O.

Q2. Express the result of addition of 67 & 75 in D/H/O.

Q3. Find the binary equivalent of (8453) H.

Q4. Find the binary equivalent of (7421) O.

Q5. (10101010111100001000) B = (X)O = (Y)H. Find X & Y

Q6. Find; AND, OR & XOR of 4E & FC in Hexadecimal System.

Q7. Find: 05 x 07 = ( )H.

Q8. Find 2's complement of (3C) H. Discuss it's significance.

Q9. Find the negative of the following numbers: 23, 45, 4D, 5C.

Q10. Find: 4678 + FBCD + 3DEF = ( ) H

Q11. Determine: 4DFCD – 3FFFF = ( ) H

Q12. Find Even/Odd/Positive/Negative from 4A, 6B, 8C, 7D.

Q13. (24)H x (13)H

Q14. (111) B x (11)B

CHAPTER – 6

Understanding PSW

The flag register in the 8051 is called the program status word **(PSW)** register. In this section, we will discuss various bits of this register and will understand the bits setting/resetting through some examples.

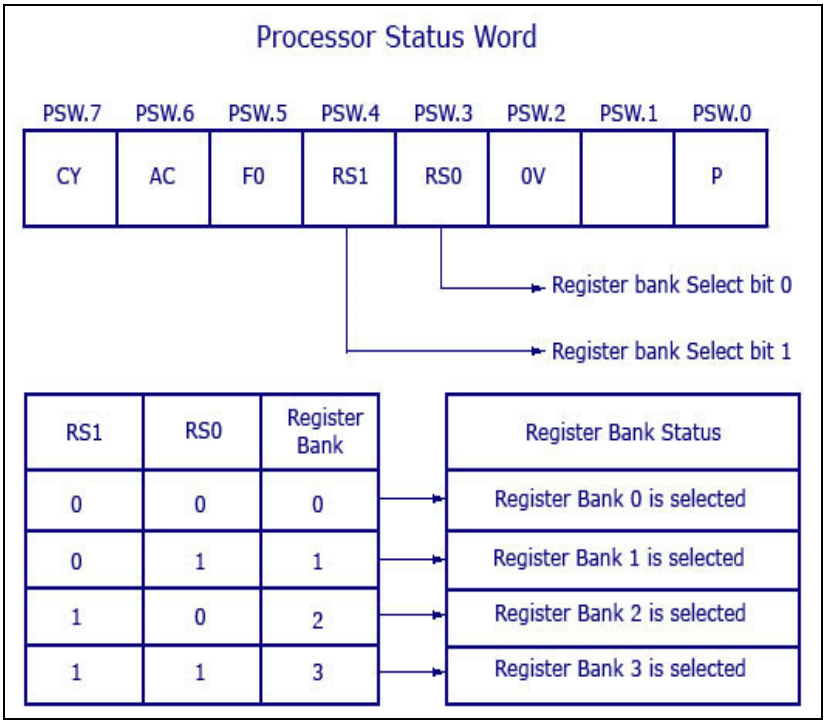
PSW (Program Status Word) register:

The program status word (PSW) register is an 8-bit register. It is also referred to as the **flag register**. Although the PSW register is 8-bits wide, only six bits of it are used by the 8051. The two unused bits are user-definable flags. Four of the flags are called **conditional flags**, meaning that they indicate some conditions that result after an instruction is executed. These four are **CY (carry), AC (auxiliary carry), P (parity), and OV (overflow).**

Refer the figure below, the bits PSW.3 and PSW.4 are designated as RS0 and RS1, respectively and are used to change the bank registers. The PSW.5 and PSW.1 bit are general-purpose status flag bits and can be used by the programmer for any purpose. In other words, they are user definable.

CY (Carry Flag): This flag is set whenever there is a carry out from the D7 bit. This flag bit is affected after an 8-bit addition or subtraction. It can also be set to 1 or 0 directly by an instruction such as “SETB C” and “CLR C” where “SETB C” stands for “set bit carry” and “CLR C” for “clear carry”.

AC (Auxiliary Carry Flag): If there is a carry from D3 to D4 during an ADD or SUB operation, this bit is set; otherwise, it is cleared. This flag is used by instructions that perform BCD (binary coded decimal) arithmetic.



P (Parity Flag): The parity flag reflects the number of 1s in the A (accumulator) register only. If A register contains an odd number of 1s, then P = 1 and P = 0, if A has an even number of 1s.

OV (Overflow Flag): This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit. In general, the carry flag is used to detect errors in unsigned arithmetic operations. The overflow flag is only used to detect errors in signed arithmetic operations.

Show the status of the CY, AC, and P flags after the addition of 38H and 2FH in the following instructions.

```
MOV A,#38H
ADD A,#2FH      ;after the addition A=67H, CY=0
```

Solution:

38	00111000
+ 2F	00101111
67	01100111

CY = 0 since there is no carry beyond the D7 bit.
AC = 1 since there is a carry from the D3 to the D4 bit.
P = 1 since the accumulator has an odd number of 1s (it has five 1s).

Show the status of the CY, AC, and P flags after the addition of 9CH and 64H in the following instructions.

```
MOV A,#9CH
ADD A,#64H      ;after addition A=00 and CY=1
```

Solution:

9C	10011100
+ 64	01100100
100	00000000

CY = 1 since there is a carry beyond the D7 bit.
AC = 1 since there is a carry from the D3 to the D4 bit.
P = 0 since the accumulator has an even number of 1s (it has zero 1s).

Show the status of the CY, AC, and P flags after the addition of 88H and 93H in the following instructions.

```
MOV A,#88H
ADD A,#93H      ;after the addition A=1BH,CY=1
```

Solution:

88	10001000
+ 93	10010011
11B	00011011

CY = 1 since there is a carry beyond the D7 bit.
AC = 0 since there is no carry from the D3 to the D4 bit.
P = 0 since the accumulator has an even number of 1s (it has four 1s).

CHAPTER – 7

Delays & Applications

"TO BE FAST IS NOT THE NEED ALWAYS!"

μC processes the data in micro seconds to whom it's slow peripherals cannot respond, moreover, human eye cannot respond to very fast variations.

Solution?

DELAYS

```
{
TOGLE: MOV P1, #01;    move 00000001 to the port P1
       CALL DELAY;     execute the delay
       MOV A, P1;      move P1 value to the accumulator
       CPL A;          complement A value
       MOV P1, A;      move 11111110 to the port1 register
       CALL DELAY;     execute the delay
       SJMP TOGLE
}
DELAY:      MOV R5, #10H; load register R5 with 10
TWO:        MOV R6, #200; load register R6 with 200
ONE:        MOV R7, #200; load register R7 with 200
   $        DJNZ R7; decrement R7 till it is zero
            DJNZ R6; decrement R6 till it is zero
            DJNZ R5; decrement R5 till it is zero
```

500μS time delay in μC 8051

```
MOV TMOD, #10H     ; select the timer mode by the registers
MOV TH1, #0FEH     ; store the delay time in TH1
MOV TL1, #32H      ; store the delay time in TL1
SETB TR1           ; to start the Timer
$ JNB TF1   $      ; decrement the value of the timer till it is zero
CLR TF1            ; clear the timer flag bit to stop the timer
CLR TF0            ; clear the overflow flag
```

CHAPTER – 8

Programming-I

Add & Store the Carry in R0:

ADDRESS	LABEL	OPCODE	MNEMONICS	COMMENT
2000	START	78 00	MOV R_0, #00	
2002		75 45 7F	MOV 45, #7F	
2005		74 AF	MOV A, #AF	
2007		25 45	ADD A, 45	
2009		50 01	JNC LOOP1	
200B		08	INC R_0	
200C	LOOP1	22	RET	

Subtract & Store the Borrow in R0:

ADDRESS	LABEL	OPCODE	MNEMONICS	COMMENT
2100	START	78 00	MOV R_0, #00	
2102		75 45 7F	MOV 45, #7F	
2105		74 AF	MOV A, #AF	
2107		95 45	SUBB A, 45	
2109		50 01	JNC LOOP1	
210B		08	INC R_0	
210C	LOOP1	22	RET	

Multiplication:

ADDRESS	LABEL	OPCODE	MNEMONICS	COMMENT
2200	START	74 7F	MOV A, #7F	
2202		75 F0 AF	MOV B, #AF	
2205		A4	MUL A, B	
2206		22	RET	

Division:

ADDRESS	LABEL	OPCODE	MNEMONICS	COMMENT
2300	START	74 7F	MOV A, #7F	
2302		75 F0 AF	MOV B, #AF	
2305		84	DIV A, B	
2306		22	RET	

Data Transfer:

ADDRESS	LABEL	OPCODE	MNEMONICS	COMMENT
2400	START	7A 0A	MOV R_2, #0A	
2402		78 20	MOV R_0, #20	
2404		79 30	MOV R_1, #30	
2406	LOOP1	E6	MOV A @ R_0	
2407		F7	MOV @R_1, A	
2408		08	INC R_0	
2409		09	INC R_1	
240A		DA FA	DJNZ R_2 LOOP1	
240C		22	RET	

Addition of numbers in a Block & Store the Carry in R1:

ADDRESS	LABEL	OPCODE	MNEMONICS	COMMENT
2500	START	79 00	MOV R_1, #00	
2502		7A 0A	MOV R_2, #0A	
2504		78 20	MOV R_0, #20	
2506		E6	MOV A, @ R_0	
2507	LOOP2	08	INC R_0	
2508		26	ADD A, @ R_0	
2509		50 01	JNC LOOP1	
250B		09	INC R_1	
250C	LOOP1	DA F9	DJNZ R_2 LOOP2	
250E		22	RET	

Even / Odd Numbers & Storing the Even/Odd numbers from Memory location #30:

ADDRESS	LABEL	OPCODE	MNEMONICS	COMMENT
2600	START	78 20	MOV R_0, # 20	
2602		79 30	MOV R_1, #30	
2604		7A 00	MOV R_2, #00	
2606		7B 0A	MOV R_3, #0A	
2608	LOOP1	E6	MOV A, @ R0	
2609		13	RRC A	
260A		40 04	JC LOOP 2	
260C		33	RLC A	
260D		F7	MOV @R_1, A	
260E		09	INC R_1	
260F		0A	INC R_2	
2610	LOOP2	08	INC R_0	
2611		DB F5	DJNZ R_3 LOOP1	
2613		22	RET	

Positive/Negative Numbers & Storing the Positive/Negative numbers from Memory location#30:

ADDRESS	LABEL	OPCODE	MNEMONICS	COMMENT
2700	START	78 20	MOV R_0, # 20	
2702		79 30	MOV R_1, #30	
2704		7A 00	MOV R_2, #00	
2706		7B 0A	MOV R_3, #0A	
2708	LOOP2	E6	MOV A, @ R0	
2709		33	RLC A	
270A		40 04	JC LOOP 1	
270C		13	RRC A	
270D		F7	MOV @R_1, A	
270E		0A	INC R_2	
270F		09	INC R_1	
2710	LOOP1	08	INC R_0	
2711		DB F5	DJNZ R_3 LOOP2	
2713		22	RET	

Find Maximum Number from a Set of Numbers:

ADDRESS	LABEL	OPCODE	MNEMONICS	COMMENT
2800	START	78 20	MOV R_0, #20	
2802		79 04	MOV R_1, #04	
2804		E6	MOV A, @ R_0	
2805	LOOP2	08	INC R_0	
2806		FA	MOV R_2, A	
2807		96	SUBB A, @R_0	
2808		50 03	JNC LOOP1	
280A		E6	MOV A, @R_0	
280B		80 01	SJMP LOOP3	
280D	LOOP1	EA	MOV A, R_2	
280E	LOOP3	D9 F5	DJNZ R_1 LOOP2	
2810		08	INC R_0	
2811		F6	MOV @R_0, A	
2812		22	RET	

Arrange the Numbers in Descending order:

ADDRESS	LABEL	OPCODE	MNEMONICS	COMMENT
2900	START	7A 04	MOV R_2, #04	
2902	LOOP4	78 20	MOV R_0, #20	
2904		79 04	MOV R_1, #04	
2906	LOOP3	E6	MOV A, @R_0	
2907		08	INC R_0	
2908		FB	MOV R_3, A	
2909		96	SUBB A, @R_0	
290A		50 08	JNC LOOP1	
290C		E6	MOV A, @R_0	
290D		18	DEC R_0	
290E		F6	MOV @R_0, A	
290F		08	INC R_0	
2910		EB	MOV A, R_3	
2911		F6	MOV @R_0, A	
2912		80 04	SJMP LOOP2	
2914	LOOP1	EB	MOV A, R_3	
2915		18	DEC R_0	
2916		F6	MOV @R_0, A	
2917		08	INC R_0	
2918	LOOP2	D9 EC	DJNZ R_1 LOOP3	
2919		DA E6	DJNZ R_2 LOOP4	
291B		22	RET	

Write an ALP to generate the cube of numbers from 1 to 6 and store them in consecutive memory locations starting from 30H

ADDERESS	MNEMONICS	OPCODE	COMMENT
6100	MOV Ro, #06H	78,06	
6102	MOV R1, #30H	79,30	
6104	$ MOV A, @R1	E7	
6105	MOV F0, A	F5, F0	
6107	MUL AB	A4	
6108	MOV F0, @R1	87, F0	
610A	MUL AB	A4	
610B	MOV @R1, A	F7	
610C	INC R1	09	
610D	DJNZ R0 $	D9, F5	
610F	RET	22	

Write an ALP to count and display hexadecimal numbers starting from 00H. The counter should come back from FFH to 00H.

ADDRESS	MNEMONICS	OPCODE`	COMMENT
6200	MOV A, #00H	74,00	
6202	$ MOV R5, #02H	7D, 02	
6204	LCALL CLRF	12,06,1D	
6207	MOV R3, A	FB	
6208	LCALL NOUT	12,05,9E	
620B	LCALL DELAY	12,01,14	
620E	INC A	04	
620F	SJMP $	80, F1	

Write an ALP to rotate a 32-bit register. Treat register R0, R1, R2 and R3 as a 32 bit register and rotate them one place to the left i.e bit 7 of R0 becomes bit 0 of R1, bit 7 of R1 become bit 0 of R2, bit 7 of R2 become bit 0 of R3 and bit 7 of R3 becomes bit 0 of R0.

R3	R2	R1	R0

ADDRESS	MNEMONICS	OPCODE	COMMENT
6000	MOV A, R0	E8	
6001	RLC A	33	
6002	MOV R0, A	F8	
6003	MOV A, R1	E9	
6004	RLC A	33	
6005	MOV R1, A	F9	
6006	MOV A, R2	EA	
6007	RLC A	33	
6008	MOV R2, A	FA	
6009	MOV A, R3	EB	
600A	RLC A	33	
600B	MOV R3, A	FB	
600C	MOV A, R0	E8	
600D	MOV E0, C	92, E0	
600F	MOV R0, A	F8	
6010	RET	22	

CHAPTER – 9

Programming-II

Through DPTR: Addition:

ADDRESS	MNEMONICS	OPCODE	COMMENT
2000	MOV DPTR, #2040	90 20 40	
2003	MOVX A, @DPTR	E0	
2004	MOV R0, A	F8	
2005	INC DPTR	A3	
2006	MOVX A, @DPTR	E0	
2007	ADD A, R0	28	
2008	INC DPTR	A3	
2009	MOVX @ DPTR, A	F0	
200A	RET	22	

Subtraction:

ADDRESS	MNEMONICS	OPCODE	COMMENT
2100	MOV DPTR, #2040	90 20 40	
2103	MOVX A, @DPTR	E0	
2104	MOV R0, A	F8	
2105	INC DPTR	A3	
2106	MOVX A, @DPTR	E0	
2107	CLR C	C3	
2108	SUBB A, R0	98	
2109	INC DPTR	A3	
210A	MOVX @DPTR, A	F0	
210B	RET	22	

Multiplication:

ADDRESS	MNEMONICS	OPCODE	COMMENT
2200	MOV DPTR, #2040	90 20 40	
2203	MOVX A, @DPTR	E0	
2204	MOV B, A	F5 F0	
2206	INC DPTR	A3	
2207	MOVX A, @DPTR	E0	
2208	MUL A, B	A4	
2209	INC DPTR	A3	
220A	MOVX @DPTR, A	F0	
220B	INC DPTR	A3	
220C	MOV A, B	E5 F0	
220E	MOVX @DPTR, A	F0	
220F	RET	22	

Division:

ADDRESS	MNEMONICS	OPCODE	COMMENT
2300	MOV DPTR, #2040	90 20 40	
2303	MOVX A, @DPTR	E0	
2304	MOV B, A	F5 F0	
2306	INC DPTR	A3	
2307	MOV A, @DPTR	E0	
2308	DIV A, B	84	
2309	INC DPTR	A3	
230A	MOVX @DPTR, A	F0	
230B	INC DPTR	A3	
230C	MOV A, B	E5 F0	
230E	MOVX @DPTR, A	F0	
230F	RET	22	

Square of the Numbers:

ADDRESS	LABEL	MNEMONICS	OP-CODE	COMMENT
2500		MOV R0, #06	78 06	
2502		MOV DPTR, #2040	90 20 40	
2505	@	MOV A, @DPTR	E0	
2506		MOV B, A	F5 F0	
2508		MUL A, B	A4	
2509		MOV @DPTR, A	F0	
250A		INC DPTR	A3	
250B		DJNZ R0 @	D8 F8	
250D		RET	22	

Cube of the Numbers:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENT
2600		MOV DPTR, #2040	90 20 40	
2603		MOV R0, #06	78 06	
2605	@	MOVX A, @DPTR	E0	
2606		MOV B, A	F5 FO	
2608		MUL A, B	A4	
2609		MOV B, A	F5 F0	
260B		MOV A, @DPTR	E0	
260C		MUL A, B	A4	
260D		MOV @DPTR, A	F0	
260E		INC DPTR	A3	
260F		DJNZ R0 @	D8 F4	
2611		RET	22	

Even Number:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENT
2700		MOV R0, #0A	78 0A	
2702		MOV R1, #00	79 00	
2704		MOV DPTR, #2040	90 20 40	
2707	$	MOV A, @DPTR	E0	
2708		RRC A	13	
2709		JC @	40 01	
270B		INC R1	09	
270C	@	INC DPTR	A3	
270D		DJNZ R0 $	D8 F8	
270F		MOVA, R1	E9	
2710		MOV@DPTR, A	F0	
2711		RET	22	

CHAPTER – 10

Waveform Generation

Waveform Generation: PPI 8255, µC 8051 & an 8-bit DAC

8255A is a parallel programmable interfacing I/O device. It can be used to transfer data under various conditions, from simple input-output to interrupt input-output. It is an I/O device designed to interface the CPU with its outside world such as ADC, DAC, keyboard etc.

It can be used with almost any **µP/µC**. It consists of three 8-bit, bidirectional I/O ports i.e. PORT A, PORT B and PORT C. We can assign ports as input or output functions.

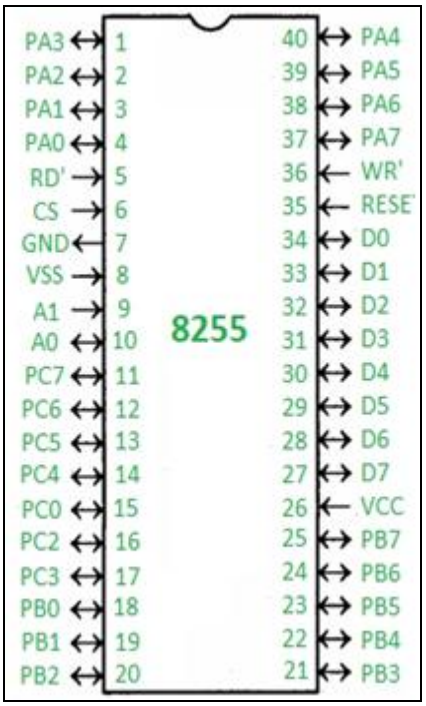
It consists of 40 pins and operates on +5V regulated power supply. Port C is further divided into two 4-bit ports i.e. port C_L (lower) and port C_U (upper), moreover, port C can work in either BSR (bit set rest) mode or in mode 0 of input-output mode of 8255. Port B can work in either mode 0 or in mode 1 of input-output mode. Port A can work either in mode 0, mode 1 or mode 2 of input-output mode. It has two control groups; Group A and Group B. Group A consist of port A and port C upper. Control group B consists of port C lower and port B. The function of these ports is defined by writing a control word in control register.

Operating Modes:

8255A has three different operating modes:

- Mode 0: In this mode, Port A and B is used as two 8-bit ports and Port C as two 4-bit ports. Each port can be programmed as either input port or output port. The outputs are latched and inputs are not in this mode. Ports do not possess interrupt capability.
- Mode 1: In this mode, Port A and B are used as 8-bit I/O ports. They can be configured as either input or output ports. Each port of this mode uses three lines taken from port C as handshake signals. Inputs and outputs both are latched.

- Mode 2: In this mode, Port A can be configured as the bidirectional port and Port B can be available in Mode 0 or Mode 1. Five signals from Port C are used as handshaking signals for Port A. The remaining three signals from Port C can be used either as simple I/O or as handshake for port B.

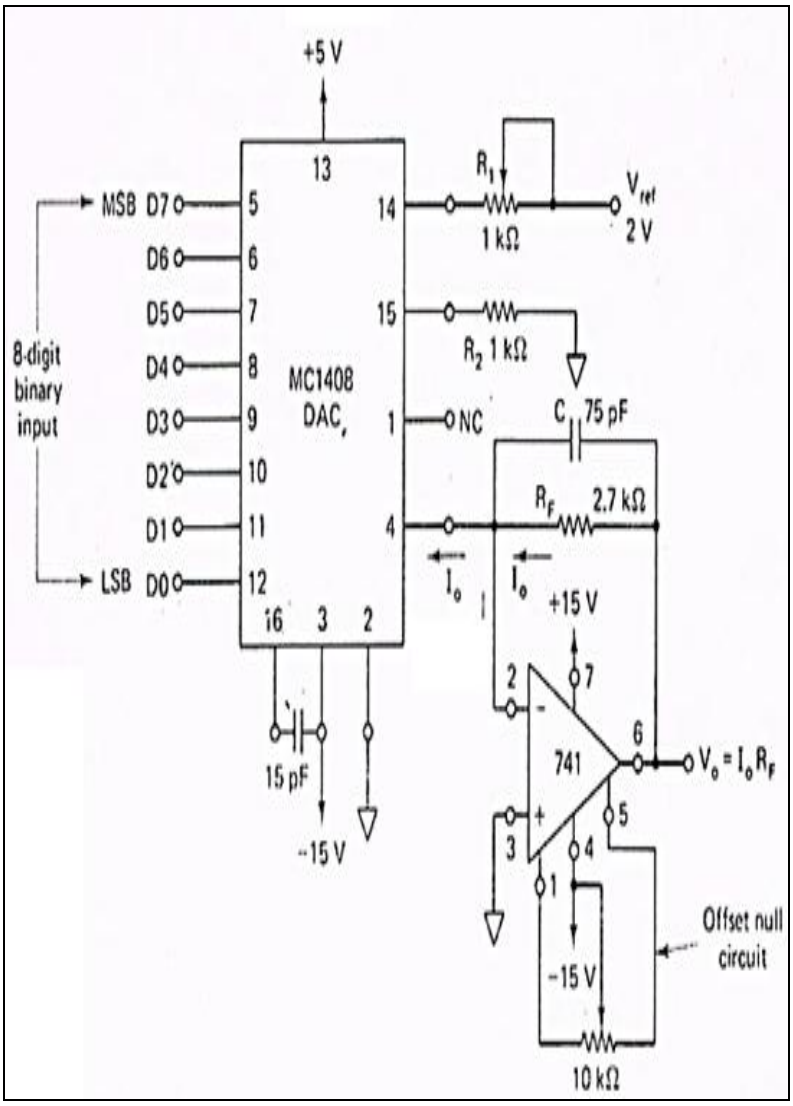


Pin Diagram:

- PA_0 – PA_7: Port A
- PB_0 – PB_7: Port B
- PC_0 – PC_7: Port C
- D_0 – D_7: Data Bus
- RESET: Reset input
- RD': Read input
- WR': Write input
- CS: Chip select
- A1 and A0: Address pins

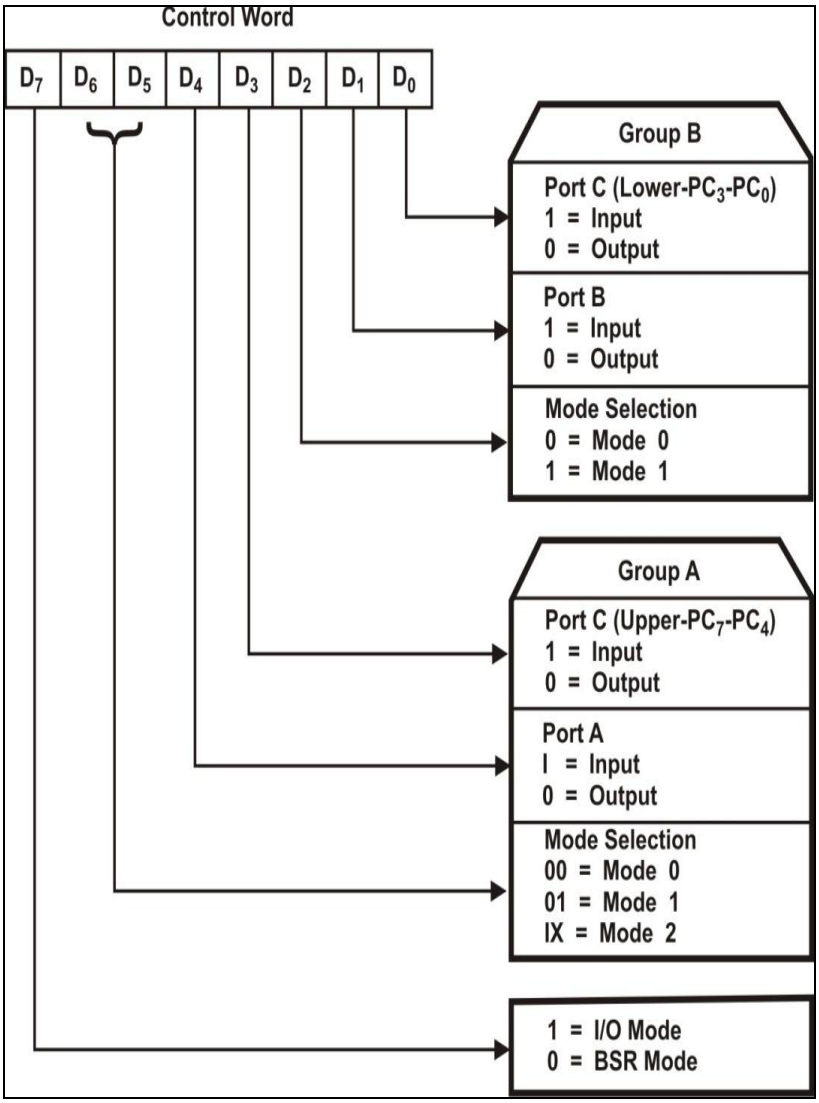
Addresses: (Varies from kit to kit)

Port A	$(80)_H$
Port B	$(81)_H$
Port C	$(82)_H$
Control Register	$(83)_H$


+5 V
13
MSB D7
D6
D5
D4
D3
D2
D1
LSB D0
8-digit binary input
5
6
7
8
9
10
11
12
MC1408 DAC
14
15
1
4
16
3
2
R_1
1 kΩ
V_{ref}
2 V
R_2 1 kΩ
NC
C 75 pF
R_F 2.7 kΩ
I_o
I_o
+15 V
2
7
741
6
$V_o = I_o R_F$
3
4
5
1
15 pF
-15 V
-15 V
Offset null circuit
10 kΩ

8-Bit DAC

I/O CONTROL WORD



Square Wave Generation:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENT
2800		MOV A, #80	74 80	
2802		MOV DPTR, #FF03	90 FF 03	
2805		MOV @ DPTR, A	F0	
2806	&	MOV DPTR, #FF00	90 FF 00	
2809		MOV A, #FF	74 FF	
280B		MOV @ DPTR, A	F0	
280C		MOV R1, #FF	79 FF	
280E	#	MOV R2, #FF	7A FF	
2810	@	DJNZ R2 @	DA FE	
2812		DJNZ R1 #	D9 FA	
2814		MOV A, #00	74 00	
2816		MOV @ DPTR, A	F0	
2817		MOV R1, #FF	79 FF	
2819	!	MOV R2, #FF	7A FF	
281B	$	DJNZ R2 $	DA FE	
281D		DJNZ R1!	D9 FA	
281F		SJMP &	80 E5	

1. Write a Control Word in BSR mode to SET PC.7?
2. Write a Control Word in BSR mode to RESET PC.0?
3. Write an ALP to blink an LED @ PC.6 with a delay?
4. Write a CW in I/O mode for all Ports as I/P Ports.
5. Write a CW in I/O mode for all Ports as O/P Ports.
6. Write an ALP in I/O mode to receive the data @ Port A & Port C & to display the addition at Port B.
7. Write an ALP in I/O mode to receive the data @ Port B & Port A & to display the ANDed data so received at Port C.
8. Write an ALP for Square/Triangular/Ramp/using 8051, 8255 and an ADC
9. Write various ALPs to glow/blink LEDs representing 8-bit LEDs using µC 8051, PPI 8255 & an 8-Bit DAC.
10. Step size and number of steps in an 8 – bit DAC.

CHAPTER – 11

Programming -III

1. Write an ALP to transfer content of source memory area, that begins from location $(XX)_H$ to the destination memory area that begins from location $(YY)_H$.
2. Write an ALP to find the minimum number from an array of 8-bit numbers stored from $(XX)_H$ to $(YY)_H$ and store it at the memory location $(ZZ)_H$.
3. Write an ALP to find the maximum number from an array of 8-bit numbers stored from $(XX)_H$ to $(YY)_H$ and store it at the memory location $(ZZ)_H$.
4. Write an ALP to find the number of Even/Odd numbers from an array of 8-bit numbers, stored from $(XX)_H$ to $(YY)_H$ and store it at the memory location $(ZZ)_H$.
5. Write an ALP by using appropriate monitor routine to implement flashing display of message "PHYSICS" with suitable delay.
6. Write an ALP to implement hex counter that starts counting from $(00)_H$ to $(FF)_H$ with suitable delay. The count should be displayed on 7-segment output console of the kit by using appropriate monitor subroutine.
7. Write an ALP to implement decimal counter that starts counting from $(00)_D$ to $(99)_D$ with suitable delay. The count should be displayed on 7-segment output console of the kit, by using appropriate monitor subroutine.
8. Write an ALP for:
   a. Flashing of LEDs.
   b. Running light effect on LEDs.
   c. Binary up counter display on LEDs.
9. Write an ALP to generate:
   a. Square wave.
   b. Triangular wave.
   c. Sawtooth wave.

10. Write an ALP to count and display hexadecimal number starting from $(00)_H$. The counter should come back from $(FF)_H$ to $(00)_H$.
11. Write an ALP to count and display decimals numbers from 0 to 99.
12. Write an ALP to generate the cubes of the numbers and store them in consecutive memory location starting from _____H.
13. Write an ALP to rotate a 32-bit register. Treat register R_0, R_1, R_2 and R_3 as a 32-bit register and rotate them one place to the left i.e., bit 7 of R_0 becomes bit0 of R_1, bit 7 of R_1 becomes bit 0 of R_2, bit 7 of R_2 becomes bit 0 of R_3 and bit 7 of R_3 becomes bit 0 of R_0.
14. Write an ALP to find a largest number in each of the two memory blocks and multiply them
    a. Block 1 starts from ____H and ends at ____H.
    b. Block 2 starts from ____H and ends at ____H.
    c. Store the result at ____H.
15. Four switch's A, B, C, D are connected to P1.7, P1.6, P1.5 and P1.4 resp and a relay driver at P1.0. Write an assembly language program that monitor these switches and will trip the relay only when the following conditions are satisfied:

$$Y = \overline{A}BC\overline{D} + \overline{A}B\overline{C}D + AB\overline{C}D + ABC\overline{D}$$

16. Write an ALP that will monitor 4 switches connected to (P1.7, P1.6, P1.5, P1.4) and drive 4 LEDs connected to (P1.3, P1.2, P1.1, P1.0) such that:
    a. LED 1 (P1.3) will be ON if any one switch is closed.
    b. LED 2 (P1.2) will be ON if any two switches are closed.
    c. LED 3 (P1.1) will be ON if any three switches are closed.
    d. LED 4 (P1.0) will be ON if all the switches are closed.
17. Write an ALP that will output ANY ONE of the following sequences on the LEDs. Assume the same port as above.
    a. 1000
    b. 0100
    c. 0010
    d. 0001
18. Write an ALP to monitor a light beam with the light sensor circuit and an appropriate buffer, connected to pin P1.0, the

display should show the count in hexadecimal, which indicates the number of interruptions that takes place.

19. Modify the above program for decimal counts.
20. Write a main program to display the message “ELECT1” on the display of the kit using appropriate sub-routine. Write an ISR program to display the message “PSWALU” on the display of the kit when an interrupt occurs. Use external interrupt INT0/INT1. (Refer kit manual to find the addresses of various utility programs).