UNIT IV 8 BIT MICROCONTROLLER- H/W ARCHITECTURE, INSTRUCTION SET AND PROGRAMMING

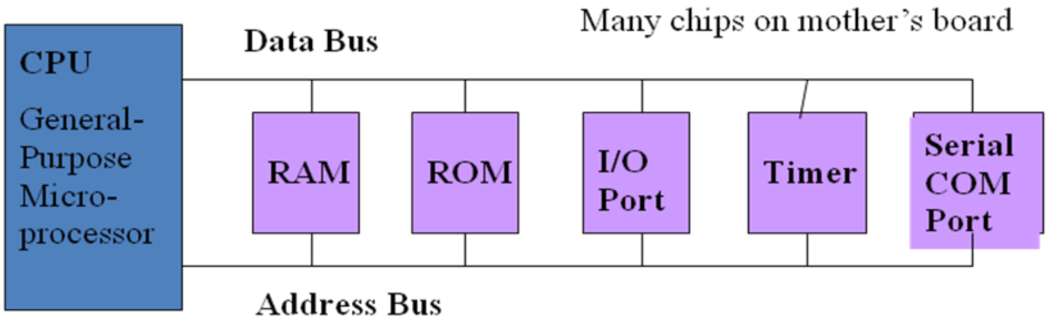
1. INTRODUCTION TO 8051 MICRO-CONTROLLER

The necessary tools for a microprocessor/controller are:

- CPU: Central Processing Unit
- I/O: Input /Output
- Bus: Address bus & Data bus
- Memory: RAM & ROM
- Timer
- Interrupt
- Serial Port
- Parallel Port

General-purpose microprocessor:

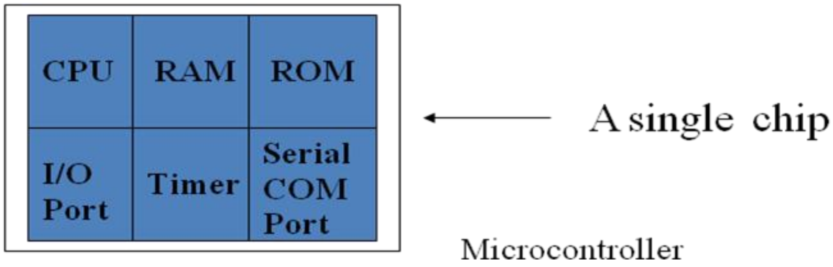
- CPU for Computers: CPU only
- No RAM, ROM, I/O on CPU chip itself
- Needs many ICs to implement a small system
- Example : Intel’s x86, Motorola’s 680x0



General-Purpose Microprocessor System

Microcontroller:

- A smaller computer
- On-chip RAM, ROM, I/O ports...
- Example : Motorola’s 6811, Intel’s 8051, Zilog’s Z8 and PIC 16X



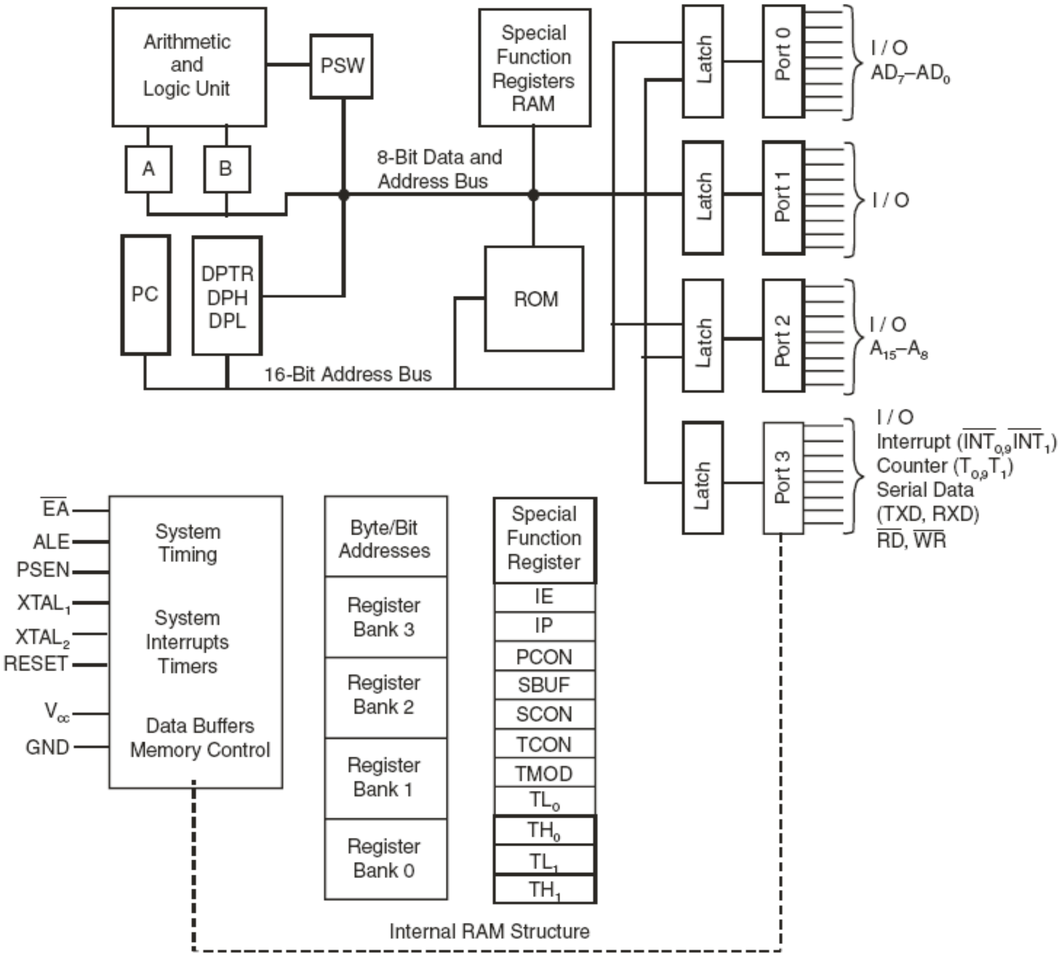
Microprocessor vs. Microcontroller:

Microprocessor	Microcontroller
CPU is stand-alone, RAM, ROM, I/O, timer are separate	CPU, RAM, ROM, I/O and timer are all on a single chip
Designer can decide on the amount of ROM, RAM and I/O ports.	Fix amount of on-chip ROM, RAM, I/O ports
Expansive	For applications in which cost, power and space are critical
General-purpose	Single-purpose

2. THE INTERNAL ARCHITECTURE OF 8051 MICROCONTROLLER.

Features:

- 4K bytes internal ROM
- 128 bytes internal RAM
- Four 8-bit I/O ports (P0 - P3).
- Two 16-bit timers/counters
- One serial interface
- only 1 On chip oscillator (external crystal)
- 6 interrupt sources (2 external , 3 internal, Reset)
- 64K external code (program) memory(only read)PSEN
- 64K external data memory(can be read and write) by RD,WR
- Code memory is selectable by EA (internal or external)
- External memory can be interfaced when data and code storage capacity exceed size of the internal memory.



ALU - Arithmetic Logical Unit: This unit is used for the arithmetic calculations.
A-Accumulator: This register is used for arithmetic operations. This is also bit addressable and 8 bit register.
B-Register: This register is used in only two instructions MUL AB and DIV AB. This is also bit addressable and 8 bit register.

PC-Program Counter:

- Points to the address of next instruction to be executed from ROM
- It is 16 bit register means the 8051 can access program address from 0000H to FFFFH. A total of 64KB of code.
- Initially PC has 0000H
- PC is incremented after each instruction.

ROM in 8051

- 4KB on chip ROM is available.
- Max ROM space is 64 KB because 16 bit address line is available in 8051.
- Starting address for ROM is 0000H (because PC which points the ROM is 16 bit wide).

Flag Bits and PSW Register:

→ Used to indicate the Arithmetic condition of Accumulator (ACC).

→ Flag register in 8051 is called as program status word (PSW). This special function register PSW is also bit addressable and 8 bit wide means each bit can be set or reset independently.

PSW0.7	PSW0.6	PSW0.5	PSW0.4	PSW0.3	PSW0.2	PSW0.1	PSW0.0
CY	AC	F0	RS1	RS0	OV	—	P

FLAG Register

There are four flags in 8051

- **P** → **Parity flag** → PSW 0.0
  1 – odd number of 1 in ACC
  0 – even number of 1 in ACC
- **OV(PSW 0.2)** → **overflow flag** → this is used to detect error in signed arithmetic operation. This is similar to carry flag but difference is only that carry flag is used for unsigned operation.
-

RS1(PSW0.4)	**RS0(PSW0.3)**	**Register Bank Select**
0	0	Bank 0
0	1	Bank 1
1	0	Bank 2
1	1	Bank 3

for selecting Bank 1, we use following commands

SETB PSW0.3 (means RS0=1)
CLR PSW0.4 (means RS1=0)

Initially by default always Bank 0 is selected.

- **F0** → user definable bit
- **AC** → **Auxiliary carry flag** → when carry is generated from D3 to D4, it is set to 1, it is used in BCD arithmetic.

$$\begin{array}{rcccccccccl}
 & & & & \boxed{1} & & & 1 & & & \\
 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & & \text{(0EH)} \\
+ & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & & \text{(5AH)} \\
\hline
 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & & \text{(38H)} \\
\hline
\end{array}$$

Since carry is generated from D3 to D4, so AC is set.

- **CY** → **carry flag** → Affected after 8 bit addition and subtraction. It is used to detect error in unsigned arithmetic opr. We can also use it as single bit storage.

  SETB C → for cy = 1
  CLR C → for cy = 0

Structure of RAM or 8051 Register Bank and Stack:

- 128 byte RAM is available in 8051.
- Address range of RAM is 00H to 7FH.
- In MC8051, 128 byte visible or user accessible RAM is available which is shown in figure.
- Extra 128B RAM which is not user accessible. 80H to FFH used for storage of SFR (special function register)

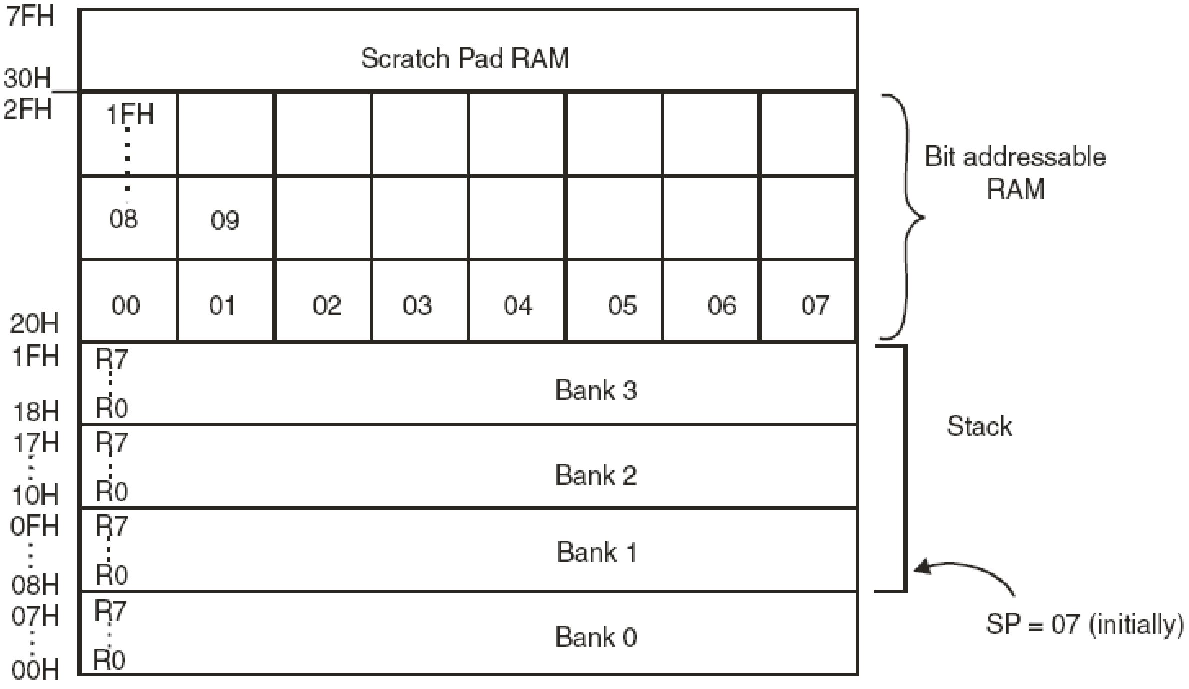


Fig. 1.6 Structure of RAM

Four Register Banks:

- There are four register banks, in each register bank there are eight 8 bit register available from R0 to R7
- By default Bank 0 is selected. For Bank 0, R0 has address 00H and R1 has 07H
- For selecting banks we use RS0 and RS1 bit of PSW.

Stack in 8051: RAM locations from 08H to 1FH can be used as stack. Stack is used to store the data temporarily. Stack is last in first out (LIFO)

Stack pointer (SP):

- 8bit register
- It indicates current RAM address available for stack or it points the top of stack.
- Initially by default at 07H because first location of stack is 08H.
- After each PUSH instruction the SP is incremented by one while in Microprocessor after PUSH instruction SP is decremented.
- After each POP instruction the SP is decremented.

DPTR → Data Pointer in 8051

→ 16 bit register, it is divided into two parts DPH and DPL.

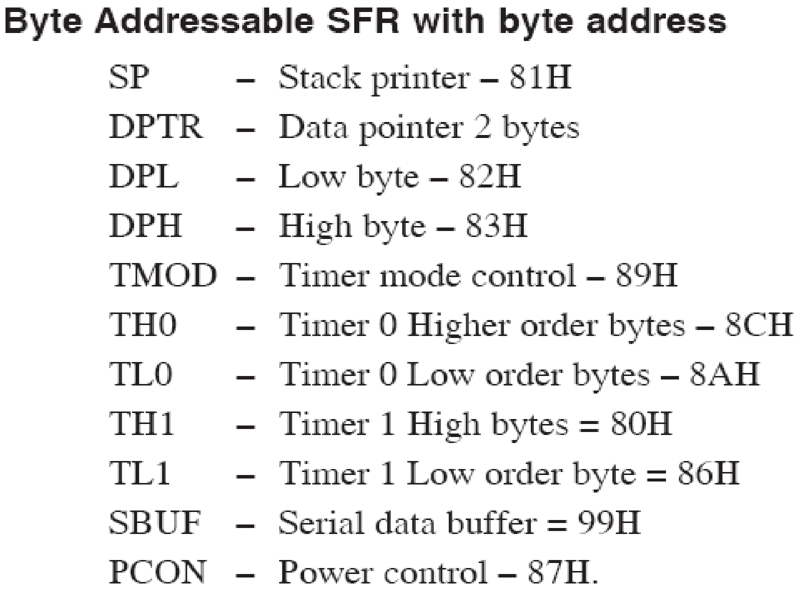
→ DPH for Higher order 8 bits, DPL for lower order 8 bits.

→ DPTR, DPH, DPL these all are SFRs in 8051.

Special Function Register

→ In RAM scratch pad, there is extra 128 byte RAM which is used to store the SFRs

→ Addresses range from 80H to FFH



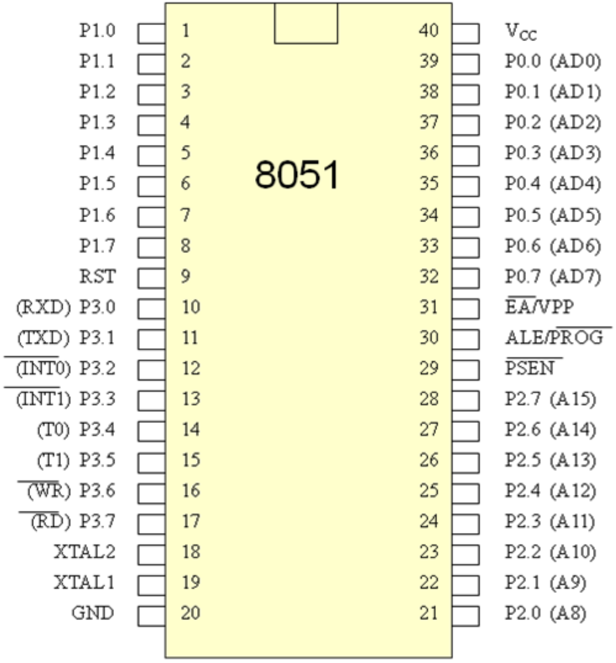
Byte Addressable SFR with byte address

SP	– Stack printer – 81H
DPTR	– Data pointer 2 bytes
DPL	– Low byte – 82H
DPH	– High byte – 83H
TMOD	– Timer mode control – 89H
TH0	– Timer 0 Higher order bytes – 8CH
TL0	– Timer 0 Low order bytes – 8AH
TH1	– Timer 1 High bytes = 80H
TL1	– Timer 1 Low order byte = 86H
SBUF	– Serial data buffer = 99H
PCON	– Power control – 87H.

I/O PORT: P0, P1, P2, P3

If neither external memory nor serial communication system are used then 4 ports with in total of 32 input/output pins are available for connection to peripheral environment. Each bit within these ports affects the state and performance of appropriate pin of the microcontroller. Thus, bit logic state is reflected on appropriate pin as a voltage (0 or 5 V) and vice versa, voltage on a pin reflects the state of appropriate port bit.

3. Pin Diagram:



- Vcc（pin 40）：
  - Vcc provides supply voltage to the chip.
  - The voltage source is +5V.
- GND（pin 20）：ground
- XTAL1 and XTAL2（pins 19,18）
- Using a quartz crystal oscillator we can observe the frequency on the XTAL2 pin.
- /EA（pin 31）：external access
  - There is no on-chip ROM in 8031 and 8032 .
  - The /EA pin is connected to GND to indicate the code is stored externally.
  - /PSEN ＆ ALE are used for external ROM.
  - For 8051, /EA pin is connected to Vcc.
  - "/" means active low.
- /PSEN（pin 29）：program store enable
  - This is an output pin and is connected to the OE pin of the ROM.
- ALE（pin 30）：address latch enable
  - It is an output pin and is active high.
  - 8051 port 0 provides both address and data.
  - The ALE pin is used for de-multiplexing the address and data by connecting to the G pin of the 74LS373 latch.
- I/O port pins
  - The four ports P0, P1, P2, and P3.
  - Each port uses 8 pins.
  - All I/O pins are bi-directional.

4. THE MEMORY ORGANIZATION OF 8051 MICROCONTROLLER

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 at most a 4 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 of external memory.

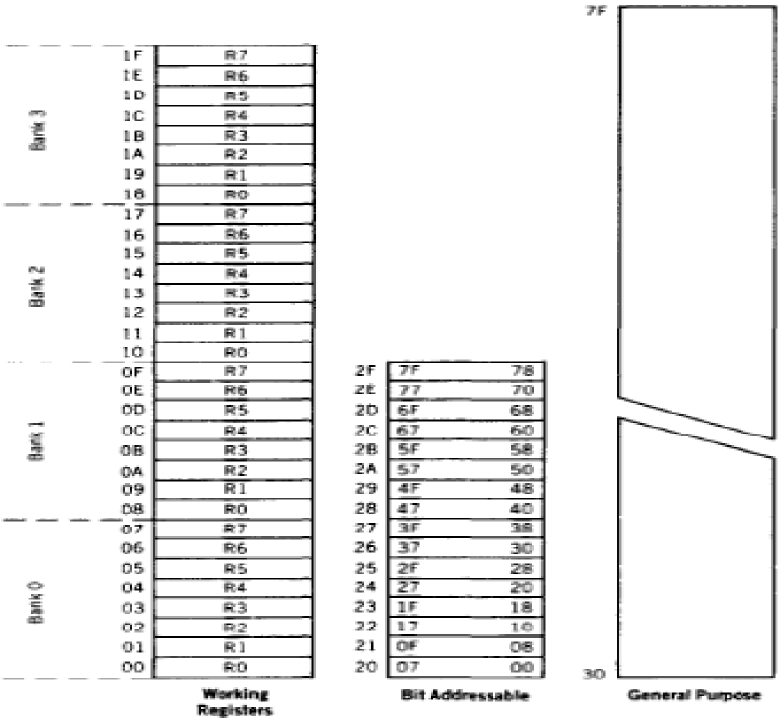
Program Memory

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.

Data Memory

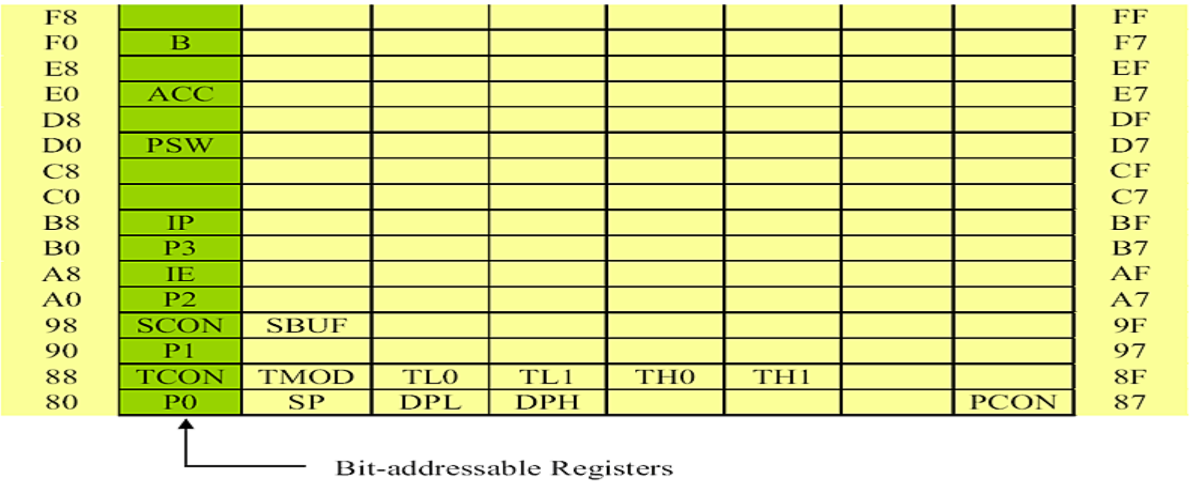
As already mentioned, Data Memory is used for temporarily storing data and intermediate results created and used during the operation of the microcontroller. Besides, RAM memory built in the 8051 family includes many registers such as hardware counters and timers, input/output ports, serial data buffers etc which are called special function registers. There are 256 RAM locations (addresses range from 0-FFh). The addresses range from 80h to FFh is used to represent the special function registers. 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 is general purpose.



5. Special Function Registers (SFRs)

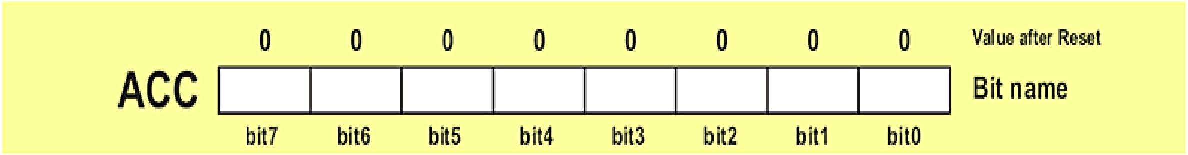
Special Function Registers (SFRs) are a sort of control table used for running and monitoring the operation of the microcontroller. Each of these registers as well as each bit they include, has its name, address in the scope of RAM and precisely defined purpose such as timer control, interrupt control, 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 are intensionally 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

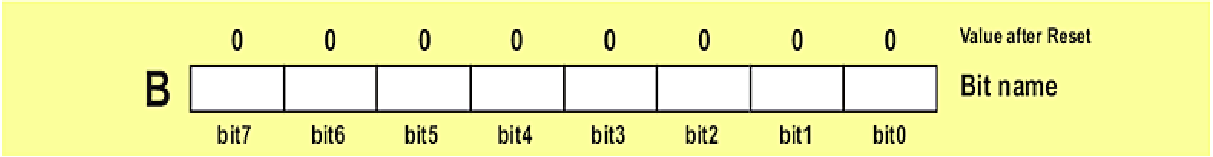
A Register (Accumulator)



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

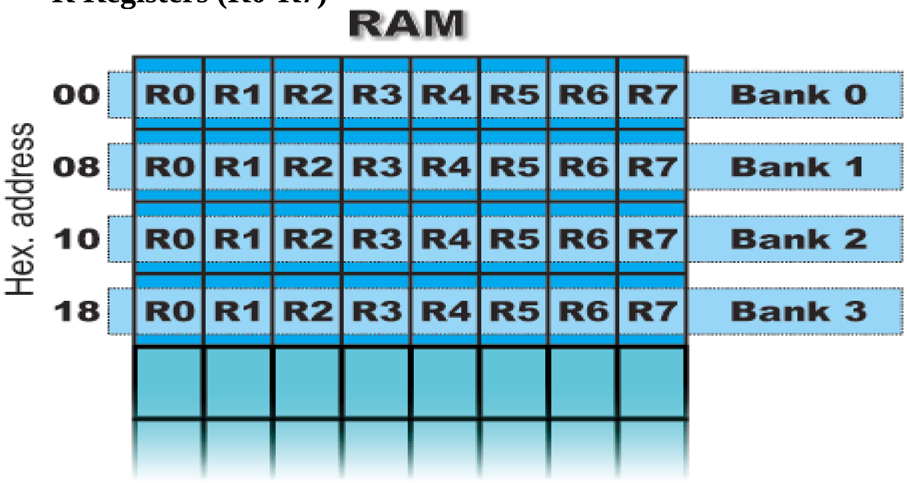
Multiplication and division can be performed only upon numbers stored in the A and B registers. All other instructions in the program can use this register as a spare accumulator (A).



Note

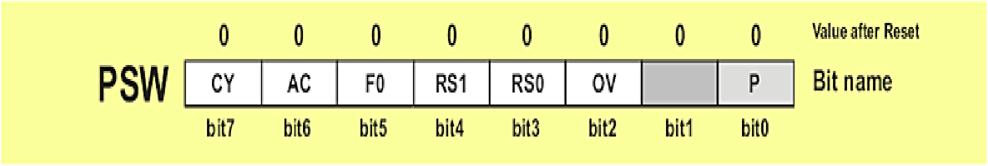
During the process of writing a program, each register is called by its name so that their exact addresses are not of importance for the user. During compilation, their names will be automatically replaced by appropriate addresses.

R Registers (R0-R7)



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.

Program Status Word (PSW) Register



PSW									
	0	0	0	0	0	0	0	0	Value after Reset
	CY	AC	F0	RS1	RS0	OV		P	Bit name
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	

PSW register is one of the most important SFRs. It contains several status bits that reflect the current state of the CPU. Besides, this register contains Carry bit, Auxiliary Carry, two register bank select bits, Overflow flag, parity bit and user-definable status flag.

P - Parity bit. If a number stored in the accumulator is even then this bit will be automatically set (1), otherwise it will be cleared (0). It is mainly used during data transmit and receive via serial communication.

- Bit 1. This bit is intended to be used in the future versions of microcontrollers.

OV Overflow occurs when the result of an arithmetical operation is larger than 255 and cannot be stored in one register. Overflow condition causes the OV bit to be set (1). Otherwise, it will be cleared (0).

RS0, RS1 - Register bank select bits. These two bits are used to select one of four register banks of RAM. By setting and clearing these bits, registers R0-R7 are stored in one of four banks of RAM.

RS1	**RS2**	**Space in RAM**
0	0	Bank0 00h-07h
0	1	Bank1 08h-0Fh
1	0	Bank2 10h-17h
1	1	Bank3 18h-1Fh

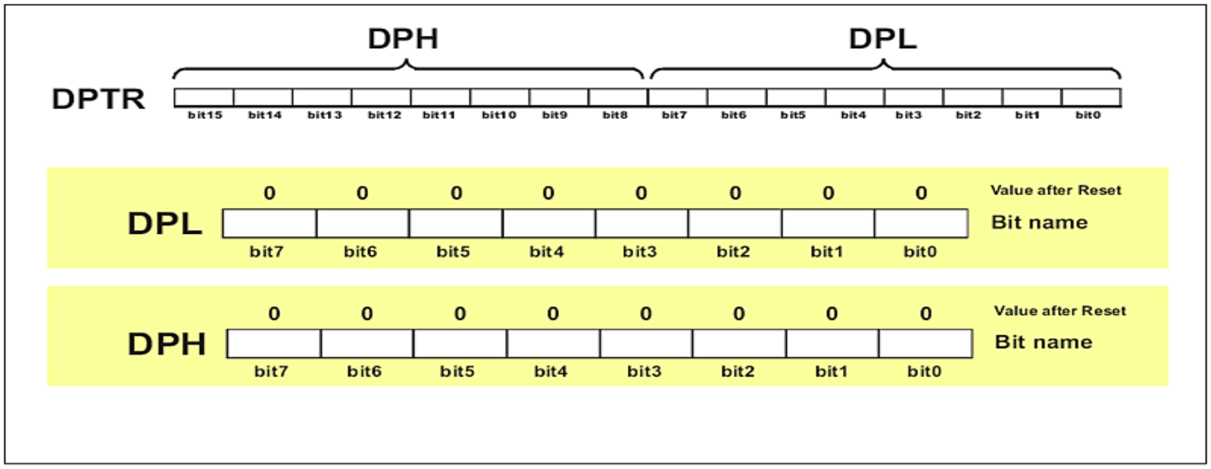
F0 - Flag 0. This is a general-purpose bit available for use.

AC - Auxiliary Carry Flag is used for BCD operations only.

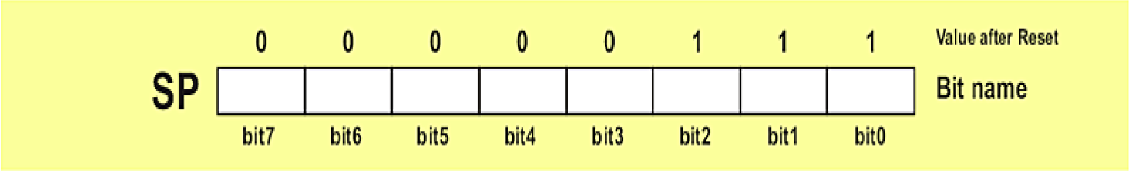
CY - Carry Flag is the (ninth) auxiliary bit used for all arithmetical operations and shift instructions.

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.

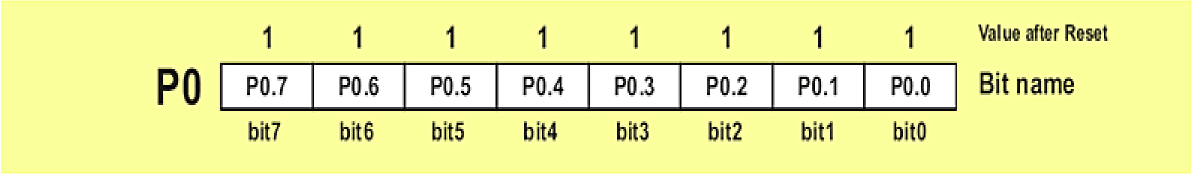

DPH
DPL
DPTR
bit15 bit14 bit13 bit12 bit11 bit10 bit9 bit8 bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
DPL
0 0 0 0 0 0 0 0
Value after Reset
Bit name
bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0
DPH
0 0 0 0 0 0 0 0
Value after Reset
Bit name
bit7 bit6 bit5 bit4 bit3 bit2 bit1 bit0

Stack Pointer (SP) Register



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



	1	1	1	1	1	1	1	1	Value after Reset
P0	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	Bit name
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	

If neither external memory nor serial communication system are used then 4 ports with in total of 32 input/output pins are available for connection to peripheral environment. Each bit within these ports affects the state and performance of appropriate pin of the microcontroller. Thus, bit logic state is reflected on appropriate pin as a voltage (0 or 5 V) and vice versa, voltage on a pin reflects the state of appropriate port bit.

As mentioned, port bit state affects performance of port pins, i.e. whether they will be configured as inputs or outputs. If a bit is cleared (0), the appropriate pin will be configured as an output, while if it is set (1), the appropriate pin will be configured as an input. Upon reset and power-on, all port bits are set (1), which means that all appropriate pins will be configured as inputs.

In short...

I/O ports are directly connected to the microcontroller pins. Accordingly, logic state of these registers can be checked by voltmeter and vice versa, voltage on the pins can be checked by inspecting their bits!

6. Port Architecture and Operation:

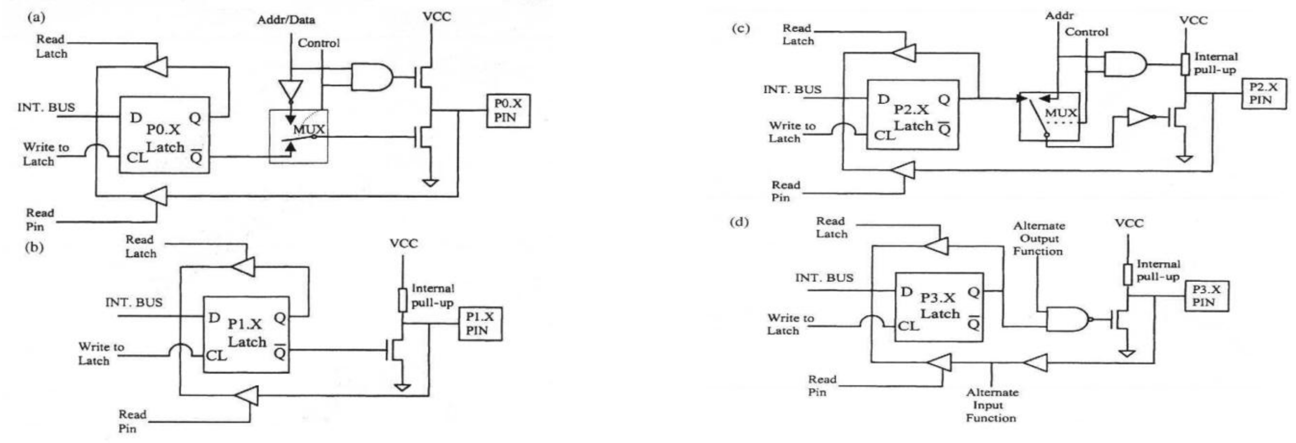
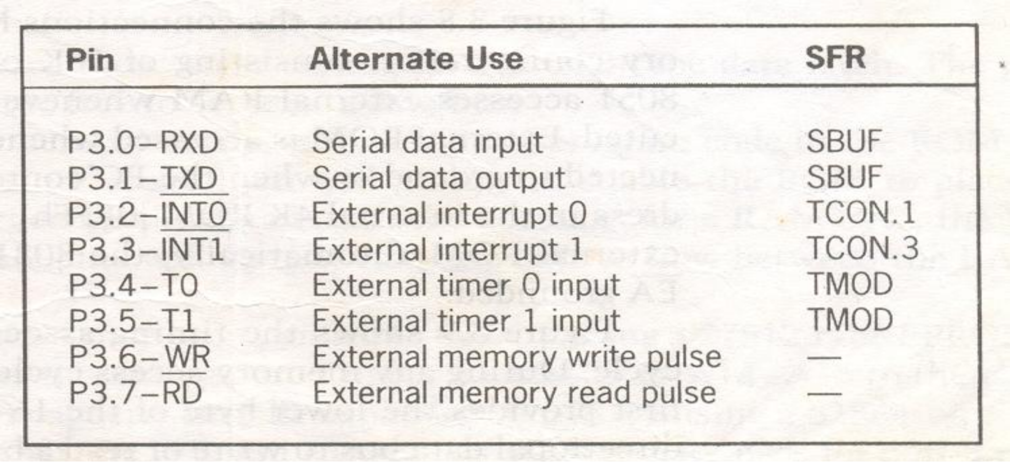
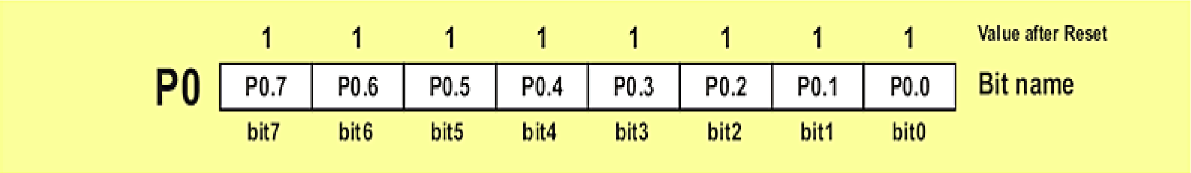


Figure 6.4 8XC51FX port bit latches and I/O buffers. (a) Port 0 bit. (b) Port 1 bit. (c) Port 2 bit. (d) Port 3 bit. (Redrawn with permission of Intel.)

Port 0- Alternate function: Multiplexed Address/data bus (AD0-AD7)
Port 1- Alternate function: Only I/O Ports
Port 2- Alternate function: Higher order address bus (A8-A15)
Port 0- Alternate function: Multiple functions as given in the following table



Pin	Alternate Use	SFR
P3.0–RXD	Serial data input	SBUF
P3.1–TXD	Serial data output	SBUF
P3.2–INT0	External interrupt 0	TCON.1
P3.3–INT1	External interrupt 1	TCON.3
P3.4–T0	External timer 0 input	TMOD
P3.5–T1	External timer 1 input	TMOD
P3.6–WR	External memory write pulse	—
P3.7–RD	External memory read pulse	—



	1	1	1	1	1	1	1	1	Value after Reset
P0	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0	Bit name
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	

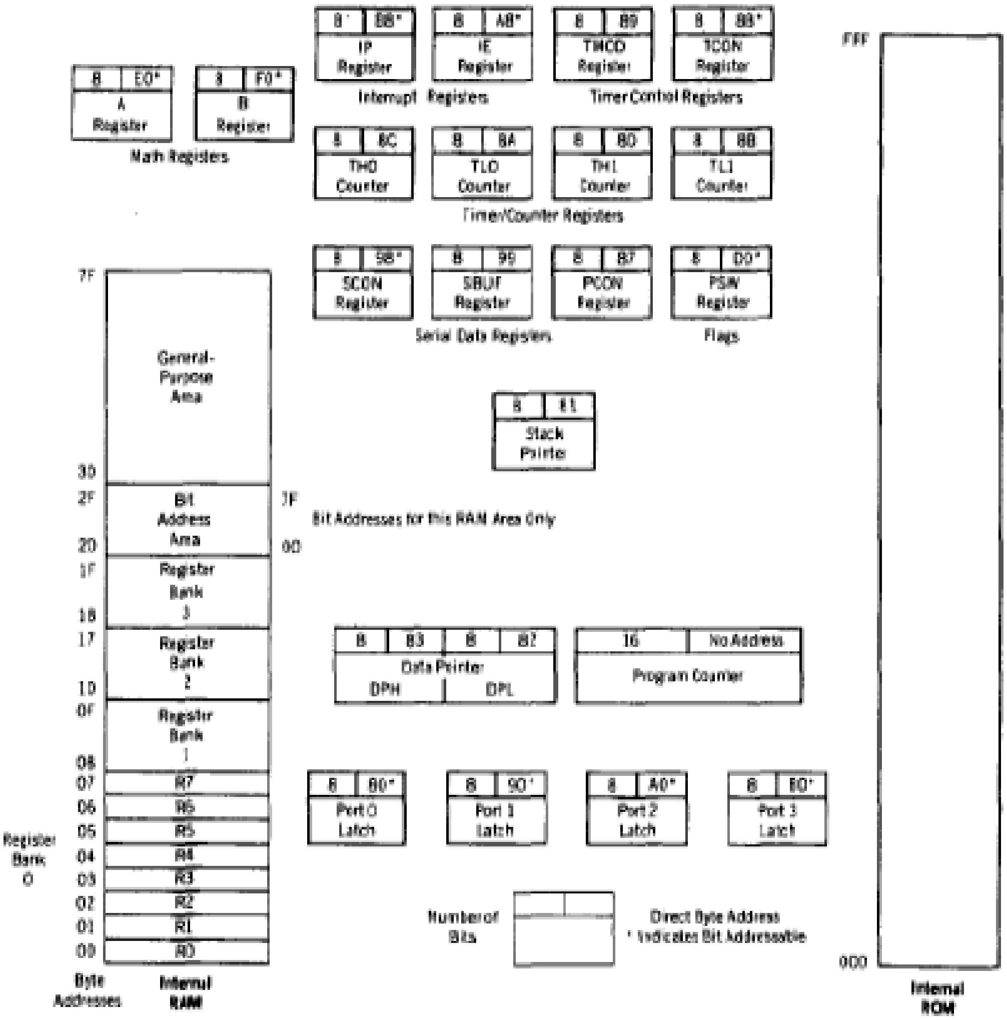
If neither external memory nor serial communication system are used then 4 ports with in total of 32 input/output pins are available for connection to peripheral environment. Each bit within these ports affects the state and performance of appropriate pin of the microcontroller. Thus, bit logic state is reflected on appropriate pin as a voltage (0 or 5 V) and vice versa, voltage on a pin reflects the state of appropriate port bit.

As mentioned, port bit state affects performance of port pins, i.e. whether they will be configured as inputs or outputs. If a bit is cleared (0), the appropriate pin will be configured as an output, while if it is set (1), the appropriate pin will be configured as an input. Upon reset and power-on, all port bits are set (1), which means that all appropriate pins will be configured as inputs.

In short...

I/O ports are directly connected to the microcontroller pins. Accordingly, logic state of these registers can be checked by voltmeter and vice versa, voltage on the pins can be checked by inspecting their

11. THE PROGRAMMER'S MODEL OF 8051 MICROCONTROLLER.



Refer explanation of 8051 Architecture and Special Function Registers

12.ADDRESSING MODES OF 8051

An "addressing mode" refers to how you are addressing a given memory location. In summary, the addressing modes are as follows, with an example of each:

Immediate Addressing	MOV A,#20h
Internal Direct Addressing	MOV A,30h
Internal Indirect Addressing	MOV A,@R0
External indirect	MOVX A,@DPTR
Code Indirect	MOVC A,@A+DPTR

Each of these addressing modes provides important flexibility.

The addressing modes in 8051 are as follows:

Register addressing	:	e.g., **MOV A,R3**
Direct byte addressing	:	e.g., **ADD A,50H**
Register indirect addressing	:	e.g., **MOV A,@R0**
Immediate addressing	:	e.g., **MOV A,#52H**
Register Specific	:	e.g., **SWAP A**
Index	:	e.g., **MOVC A,@A+DPTR**

Immediate Addressing: Immediate addressing is so-named because the value to be stored in memory immediately follows the operation code in memory. That is to say, the instruction itself dictates what value will be stored in memory.For example, the instruction: **MOV A,#20h**

This instruction uses Immediate Addressing because the Accumulator will be loaded with the value that immediately follows; in this case 20 (hexidecimal). Immediate addressing is very fast since the value to be loaded is included in the instruction. However, since the value to be loaded is fixed at compile-time it is not very flexible.

Direct Addressing: Direct addressing is so-named because the value to be stored in memory is obtained by directly retrieving it from another memory location. For example: **MOV A,30h**

This instruction will read the data out of Internal RAM address 30 (hexidecimal) and store it in the Accumulator. Direct addressing is generally fast since, although the value to be loaded is not included in the instruction, it is quickly accessible since it is stored in the 8051s Internal RAM. It is also much more flexible than Immediate Addressing since the value to be loaded is whatever is found at the given address--which may be variable. Also, it is important to note that when using direct addressing any instruction which refers to an address between 00h and 7Fh is referring to Internal Memory. Any instruction which refers to an address between 80h and FFh is referring to the SFR control registers that control the 8051 microcontroller itself.

Indirect Addressing: Indirect addressing is a very powerful addressing mode which in many cases provides an exceptional level of flexibility. Indirect addressing is also the only way to access the extra 128 bytes of Internal RAM found on an 8052. Indirect addressing appears as follows: **MOV A,@R0**

This instruction causes the 8051 to analyze the value of the R0 register. The 8051 will then load the accumulator with the value from Internal RAM which is found at the address indicated by R0. For example, lets say R0 holds the value 40h and Internal RAM address 40h holds the value 67h. When the above instruction is executed the 8051 will check the value of R0. Since R0 holds 40h the 8051 will get the value out of Internal RAM address 40h (which holds 67h) and store it in the Accumulator. Thus, the Accumulator ends up holding 67h. Indirect addressing always refers to Internal RAM; it never refers to an SFR. Thus, in a prior example we mentioned that SFR 99h can be used to write a value to the serial port. Thus one may think that the following would be a valid solution to write the value 1 to the serial port:

MOV R0,#99h ;Load the address of the serial port
MOV @R0,#01h ;Send 01 to the serial port -- **WRONG!!**

This is not valid. Since indirect addressing always refers to Internal RAM these two instructions would write the value 01h to Internal RAM address 99h on an 8052. On an 8051 these two instructions would produce an undefined result since the 8051 only has 128 bytes of Internal RAM.

External Direct: External Memory is accessed using a suite of instructions by "External Direct" addressing. There are only two commands that use External Direct addressing mode:

MOVX A,@DPTR

MOVX @DPTR,A

Both commands utilize DPTR. In these instructions, DPTR must first be loaded with the address of external memory. Once DPTR holds the correct external memory address, the first command will move the contents of that external memory address into the Accumulator. The second command will do the opposite: it will allow to write the value of the Accumulator to the external memory address pointed to by DPTR.

External Indirect

External memory can also be accessed using a form of indirect addressing. This form of addressing is usually only used in relatively small projects that have a very small amount of external RAM. An example of this addressing mode is:

MOVX @R0,A

Once again, the value of R0 is first read and the value of the Accumulator is written to that address in External RAM. Since the value of @R0 can only be 00h through FFh the project would effectively be limited to 256 bytes of External RAM. There are relatively simple hardware/software tricks that can be implemented to access more than 256 bytes of memory using External Indirect addressing; however, it is usually easier to use External Direct addressing if your project has more than 256 bytes of External RAM.

13.INSTRUCTION SET OF 8051 MICROCONTROLLER

DATA TRANSFER INSTRUCTIONS:

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

Data Transfer Instructions			
Mnemonic	**Description**	**Byte**	**Cycle**
MOV A,Rn	Moves the register to the accumulator	1	1
MOV A,direct	Moves the direct byte to the accumulator	2	2
MOV A,@Ri	Moves the indirect RAM to the accumulator	1	2
MOV A,#data	Moves the immediate data to the accumulator	2	2
MOV Rn,A	Moves the accumulator to the register	1	2
MOV Rn,direct	Moves the direct byte to the register	2	4
MOV Rn,#data	Moves the immediate data to the register	2	2
MOV direct,A	Moves the accumulator to the direct byte	2	3
MOV direct,Rn	Moves the register to the direct byte	2	3
MOV direct,direct	Moves the direct byte to the direct byte	3	4

MOV direct,@Ri	Moves the indirect RAM to the direct byte	2	4
MOV direct,#data	Moves the immediate data to the direct byte	3	3
MOV @Ri,A	Moves the accumulator to the indirect RAM	1	3
MOV @Ri,direct	Moves the direct byte to the indirect RAM	2	5
MOV @Ri,#data	Moves the immediate data to the indirect RAM	2	3
MOV DPTR,#data	Moves a 16-bit data to the data pointer	3	3
MOVC A,@A+DPTR	Moves the code byte relative to the DPTR to the accumulator (address=A+DPTR)	1	3
MOVC A,@A+PC	Moves the code byte relative to the PC to the accumulator (address=A+PC)	1	3
MOVX A,@Ri	Moves the external RAM (8-bit address) to the accumulator	1	3-10
MOVX A,@DPTR	Moves the external RAM (16-bit address) to the accumulator	1	3-10
MOVX @Ri,A	Moves the accumulator to the external RAM (8-bit address)	1	4-11
MOVX @DPTR,A	Moves the accumulator to the external RAM (16-bit address)	1	4-11
PUSH direct	Pushes the direct byte onto the stack	2	4
POP direct	Pops the direct byte from the stack/td>	2	3
XCH A,Rn	Exchanges the register with the accumulator	1	2
XCH A,direct	Exchanges the direct byte with the accumulator	2	3
XCH A,@Ri	Exchanges the indirect RAM with the accumulator	1	3
XCHD A,@Ri	Exchanges the low-order nibble indirect RAM with the accumulator	1	3

ARITHMETIC INSTRUCTIONS

Arithmetic instructions perform several basic operations such as addition, subtraction, division, multiplication etc. After execution, the result is stored in the first operand. For example:

`ADD A,R1` - The result of addition (A+R1) will be stored in the accumulator.

Mnemonic	**Description**
ADD A,Rn	Adds the register to the accumulator
ADD A,direct	Adds the direct byte to the accumulator
ADD A,@Ri	Adds the indirect RAM to the accumulator
ADD A,#data	Adds the immediate data to the accumulator
ADDC A,Rn	Adds the register to the accumulator with a carry flag
ADDC A,direct	Adds the direct byte to the accumulator with a carry flag
ADDC A,@Ri	Adds the indirect RAM to the accumulator with a carry flag
ADDC A,#data	Adds the immediate data to the accumulator with a carry flag
SUBB A,Rn	Subtracts the register from the accumulator with a borrow
SUBB A,direct	Subtracts the direct byte from the accumulator with a borrow

SUBB A,@Ri	Subtracts the indirect RAM from the accumulator with a borrow
SUBB A,#data	Subtracts the immediate data from the accumulator with a borrow
INC A	Increments the accumulator by 1
INC Rn	Increments the register by 1
INC Rx	Increments the direct byte by 1
INC @Ri	Increments the indirect RAM by 1
DEC A	Decrements the accumulator by 1
DEC Rn	Decrements the register by 1
DEC Rx	Decrements the direct byte by 1
DEC @Ri	Decrements the indirect RAM by 1
INC DPTR	Increments the Data Pointer by 1
MUL AB	Multiplies A and B
DIV AB	Divides A by B
DA A	Decimal adjustment of the accumulator according to BCD code

LOGIC INSTRUCTIONS:

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

Mnemonic	**Description**
ANL A,Rn	AND register to accumulator
ANL A,direct	AND direct byte to accumulator
ANL A,@Ri	AND indirect RAM to accumulator
ANL A,#data	AND immediate data to accumulator
ANL direct,A	AND accumulator to direct byte
ANL direct,#data	AND immediae data to direct register
ORL A,Rn	OR register to accumulator
ORL A,direct	OR direct byte to accumulator
ORL A,@Ri	OR indirect RAM to accumulator
ORL direct,A	OR accumulator to direct byte
ORL direct,#data	OR immediate data to direct byte
XRL A,Rn	Exclusive OR register to accumulator
XRL A,direct	Exclusive OR direct byte to accumulator
XRL A,@Ri	Exclusive OR indirect RAM to accumulator
XRL A,#data	Exclusive OR immediate data to accumulator
XRL direct,A	Exclusive OR accumulator to direct byte
XORL direct,#data	Exclusive OR immediate data to direct byte
CLR A	Clears the accumulator
CPL A	Complements the accumulator (1=0, 0=1)

SWAP A	Swaps nibbles within the accumulator
RL A	Rotates bits in the accumulator left
RLC A	Rotates bits in the accumulator left through carry
RR A	Rotates bits in the accumulator right
RRC A	Rotates bits in the accumulator right through carry

ROTATE AND SWAP INSTRUCTIONS:

SWAP A	Swaps nibbles within the accumulator
RL A	Rotates bits in the accumulator left
RLC A	Rotates bits in the accumulator left through carry
RR A	Rotates bits in the accumulator right
RRC A	Rotates bits in the accumulator right through carry

SWAP A - Swaps nibbles within the accumulator

Description: A nibble refers to a group of 4 bits within one register (bit0-bit3 and bit4-bit7). This instruction interchanges high and low nibbles of the accumulator.

Syntax: SWAP A;

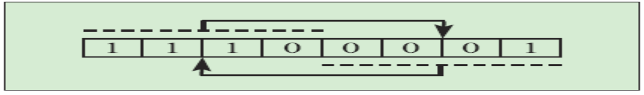
Byte: 1 (instruction code);

STATUS register flags: No flags are affected;

EXAMPLE:

Before execution: A=E1h (11100001)bin.

After execution: A=1Eh (00011110)bin.



RL A - Rotates the accumulator one bit left

Description: Eight bits in the accumulator are rotated one bit left, so that the bit 7 is rotated into the bit 0 position.

Syntax: RL A;

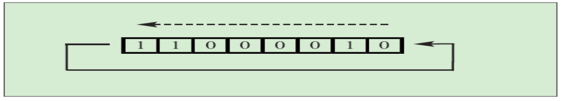
Byte: 1 (instruction code);

STATUS register flags: No flags affected;

EXAMPLE:

Before execution: A= C2h (11000010 Bin.)

After execution: A=85h (10000101 Bin.)



RLC A - Rotates the accumulator one bit left through the carry flag

Description: All eight bits in the accumulator and carry flag are rotated one bit left. After this operation, the bit 7 is rotated into the carry flag position and the carry flag is rotated into the bit 0 position.

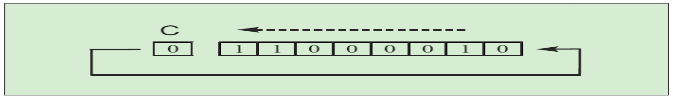
Syntax: RLC A;

Byte: 1 (instruction code);

STATUS register flags: C;

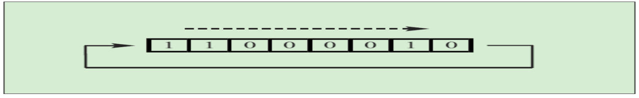
EXAMPLE:

Before execution: A= C2h (11000010 Bin.) C=0
After execution: A= 85h (10000100 Bin.) C=1



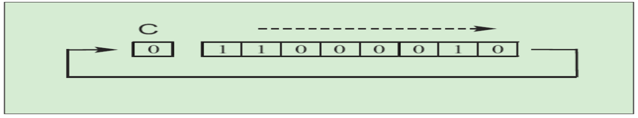
RR A - Rotates the accumulator one bit right

Description: All eight bits in the accumulator are rotated one bit right so that the bit 0 is rotated into the bit 7 position.
Syntax: RR A;
Byte: 1 (instruction code);
STATUS register flags: No flags affected;
EXAMPLE:
Before execution: A= C2h (11000010 Bin.)
After execution: A= 61h (01100001 Bin.)



RRC A - Rotates the accumulator one bit right through the carry flag

Description: All eight bits in the accumulator and carry flag are rotated one bit right. After this operation, the carry flag is rotated into the bit 7 position and the bit 0 is rotated into the carry flag position.
Syntax: RRC A;
Byte: 1 (instruction code);
STATUS register flags: C;
EXAMPLE:
Before execution: A= C2h (11000010 Bin.) C=0
After execution: A= 61h (01100001 Bin.) C=0



BRANCH INSTRUCTIONS OR CONTROL TRANSFER INSTRUCTIONS:

There are two kinds of branch instructions:
Unconditional jump instructions: a jump to a new program location is executed without any condition. Conditional jump instructions: a jump to a new program location is executed only if a specified condition is met. Otherwise, the program normally proceeds with the next instruction.

Program Control Instructions

The 8051 supports three kind of jump instructions:

1. LJMP
2. SJMP
3. 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 singe 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 16 bits 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.

Mnemonic	**Description**
ACALL addr11	Absolute subroutine call

LCALL addr16	Long subroutine call
RET	Returns from subroutine
RETI	Returns from interrupt subroutine
AJMP addr11	Absolute jump
LJMP addr16	Long jump
SJMP rel	Short jump (from –128 to +127 locations relative to the following instruction)
JC rel	Jump if carry flag is set. Short jump.
JNC rel	Jump if carry flag is not set. Short jump.
JB bit,rel	Jump if direct bit is set. Short jump.
JBC bit,rel	Jump if direct bit is set and clears bit. Short jump.
JMP @A+DPTR	Jump indirect relative to the DPTR
JZ rel	Jump if the accumulator is zero. Short jump.
JNZ rel	Jump if the accumulator is not zero. Short jump.
CJNE A,direct,rel	Compares direct byte to the accumulator and jumps if not equal. Short jump.
CJNE A,#data,rel	Compares immediate data to the accumulator and jumps if not equal. Short jump.
CJNE Rn,#data,rel	Compares immediate data to the register and jumps if not equal. Short jump.
CJNE @Ri,#data,rel	Compares immediate data to indirect register and jumps if not equal. Short jump.
DJNZ Rn,rel	Decrements register and jumps if not 0. Short jump.
DJNZ Rx,rel	Decrements direct byte and jump if not 0. Short jump.
NOP	No operation

14.PROGRAMMING

Here some simple programs of 8051 are given to understand the operation of different instructions and to understand the logic behind particular program. First the statement of the program that describes what should be done is given. Then the solution is given which describes the logic how it will be done and last the code is given with necessary comments.

Statement 1: - Exchange the content of FFh and FF00h

Solution: - Here one is internal memory location and other is memory external location. so first the content of ext memory location FF00h is loaded in acc. then the content of int memory location FFh is saved first and then content of acc is transferred to FFh. now saved content of FFh is loaded in acc and then it is transferred to FF00h.

```
MOV DPTR, #FF00H       ; take the address in DPTR
MOVX A, @DPTR          ; get the content of  FF00H  in A
MOV R0, FFH            ; save the content of  FFH  in R0
MOV FFH, A             ; move a to  FFH
MOV A, @R0             ; get content of  FFH  in A
MOVX @DPTR, A          ; move it to  FF00H
```

Statement 2: - Store the higher nibble of r7 in to both nibbles of r6

Solution: –first we shall get the upper nibble of r7 in r6. Then we swap nibbles of r7 and make OR operation with r6 so the upper and lower nibbles are duplicated

```
MOV A, R7            ; get the content in acc
ANL A, #F0H          ; mask lower bit
MOV R6, A            ; send it to r6
SWAP A               ; xchange upper and lower nibbles of acc
ORL A, R6            ; OR operation
MOV R6, A            ; finally load content in r6
```

Statement 3: - treat r6-r7 and r4-r5 as two 16 bit registers. Perform subtraction between them. Store the result in 20h (lower byte) and 21h (higher byte).

Solution: - first we shall clear the carry. Then subtract the lower bytes afterward then subtract higher bytes.

```
CLR C                ; clear carry
MOV A, R4            ; get first lower byte
SUBB A, R6           ; subtract it with other
MOV 20H, A           ; store the result
MOV A, R5            ; get the first higher byte
SUBB A, R7           ; subtract from other
MOV 21H, A           ; store the higher byte
```

Statement 4: - divide the content of r0 by r1. Store the result in r2 (answer) and r3 (reminder). Then restore the original content of r0.

Solution:-after getting answer to restore original content we have to multiply answer with divider and then add reminder in that.

```
MOV A, R0            ; get the content of R0 and R1
MOV B, R1            ; in register A and B
DIV AB               ; divide A by B
MOV R2, A            ; store result in R2
MOV R3, B            ; and reminder in R3
```

```
                MOV B, R1               ; again get content of R1 in B
                MUL AB                  ; multiply it by answer
                ADD A, R3               ; add reminder in new answer
                MOV R0, A               ; finally restore the content of R0
```

Statement 5: - transfer the block of data from 20h to 30h to external location 1020h to 1030h.

Solution: - here we have to transfer 10 data bytes from internal to external RAM. So first, we need one counter. Then we need two pointers one for source second for destination.

```
                MOV R7, #0AH            ; initialize counter by 10d
                MOV R0, #20H            ; get initial source location
                MOV DPTR, #1020H        ; get initial destination location
        NXT:    MOV A, @R0              ; get first content in ACC
                MOVX @DPTR, A           ; move it to external location
                INC R0                  ; increment source location
                INC DPTR                ; increase destination location
                DJNZ R7, NXT            ; decrease R7. if zero then over otherwise move next
```

Statement 6: - find out how many equal bytes between two memory blocks 10h to 20h and 20h to 30h.

Solution: - here we shall compare each byte one by one from both blocks. Increase the count every time when equal bytes are found

```
                MOV R7, #0AH            ; initialize counter by 10d
                MOV R0, #10H            ; get initial location of block1
                MOV R1, #20H            ; get initial location of block2
                MOV R6, #00H            ; equal byte counter. Starts from zero
        NXT:    MOV A, @R0              ; get content of block 1 in acc
                MOV B, A                ; move it to B
                MOV A, @R1              ; get content of block 2 in acc
                CJNE A, B, NOMATCH      ; compare both if equal
                INC R6                  ; increment the counter
NOMATCH:        INC R0                  ; otherwise go for second number
                INC R1
                DJNZ R7, NXT     ; decrease r7. if zero then over otherwise move next
```

Statement 7: - given block of 100h to 200h. Find out how many bytes from this block are greater then the number in r2 and less then number in r3. Store the count in r4.

Solution: - in this program, we shall take each byte one by one from given block. Now here two limits are given higher limit in r3 and lower limit in r2. So we check first higher limit and then lower limit if the byte is in between these limits then count will be incremented.

```
             MOV DPTR, #0100H      ; get initial location
             MOV R7, #0FFH        ; counter
             MOV R4, #00H         ; number counter
             MOV 20H, R2          ; get the upper and lower limits in
             MOV 21H, R3          ; 20h and 21h
       NXT:  MOVX A, @DPTR        ; get the content in acc
             CJNE A, 21H, LOWER   ; check the upper limit first
             SJMP OUT             ; if number is larger
     LOWER:  JNC OUT              ; jump out
             CJNE A, 20H, LIMIT   ; check lower limit
             SJMP OUT             ; if number is lower
     LIMIT:  JC OUT               ; jump out
             INC R4               ; if number within limit increment count
       OUT:  INC DPTR       ; get next location
             DJNZ R7, NXT         ; repeat until block completes
```

8 Bit Addition (Immediate Addressing)

LABEL	MNEMONIC	OPERAND	COMMENTS
	CLR	C	Clear CY Flag
	MOV	A,# data1	Get the data1 in Accumulator
	ADDC	A, # data 2	Add the data1 with data2
	MOV	DPTR, # 4500H	Initialize the memory location
	MOVX	@ DPTR, A	Store the result in memory location
L1	SJMP	L1	Stop the program

8 Bit Subtraction (Immediate Addressing)

LABEL	MNEMONIC	OPERAND	COMMENTS
	CLR	C	Clear CY flag
	MOV	A, # data1	Store data1 in accumulator
	SUBB	A, # data2	Subtract data2 from data1
	MOV	DPTR, # 4500	Initialize memory location
	MOVX	@ DPTR, A	Store the difference in memory location
L1	SJMP	L1	Stop

8 Bit Multiplication

LABEL	MNEMONIC	OPERAND	COMMENTS
	MOV	A ,#data1	Store data1 in accumulator
	MOV	B, #data2	Store data2 in B reg
	MUL	A,B	Multiply both
	MOV	DPTR, # 4500H	Initialize memory location
	MOVX	@ DPTR, A	Store lower order result
	INC	DPTR	Go to next memory location
	MOV	A,B	Store higher order result
	MOV	@ DPTR, A	
STOP	SJMP	STOP	Stop

8 Bit Division

LABEL	MNEMONIC	OPERAND	COMMENTS
	MOV	A, # data1	Store data1 in accumulator
	MOV	B, # data2	Store data2 in B reg
	DIV	A,B	Divide
	MOV	DPTR, # 4500H	Initialize memory location
	MOVX	@ DPTR, A	Store remainder
	INC	DPTR	Go to next memory location
	MOV	A,B	Store quotient
	MOV	@ DPTR, A	
STOP	SJMP	STOP	Stop

To write an ALP to perform logical and bit manipulation operations using 8051 microcontroller.

1. Initialize content of accumulator as FFH
2. Set carry flag (cy = 1).
3. AND bit 7 of accumulator with cy and store PSW format.
4. OR bit 6 of PSW and store the PSW format.
5. Set bit 5 of SCON.
6. Clear bit 1 of SCON.
7. Move SCON.1 to carry register.
8. Stop the execution of program.

PROGRAM TABLE

LABEL	MNEMONICS	OPERAND	COMMENT
	MOV	DPTR,#4500	Initialize memory location
	MOV	A,#FF	Get the data in accumulator
	SETB	C	Set CY bit
	ANL	C, ACC.7	Perform AND with 7th bit of
	MOV	A,DOH	Store the result
	MOVX	@DPTR,A	
	INC	DPTR	Go to next location
	ORL	C, IE.2	OR CY bit with 2nd bit if IE reg
	CLR	PSW.6	Clear 6th bit of PSW
	MOV	A,DOH	Store the result
	MOVX	@DPTR,A	
	INC	DPTR	Go to next location
	SETB	SCON.5	Set 5th of SCON reg
	CLR	SCON.1	Clear 1st bit of SCON reg
	MOV	A,98H	Store the result
	MOVX	@DPTR,A	
	INC	DPTR	Go to next location
	MOV	C,SCON.1	Copy 1st bit of SCON reg to CY flag
	MOV	A,DOH	Store the result
	MOVX	@DPTR,A	
L2	SJMP	L2	Stop