8051 Microcontroller

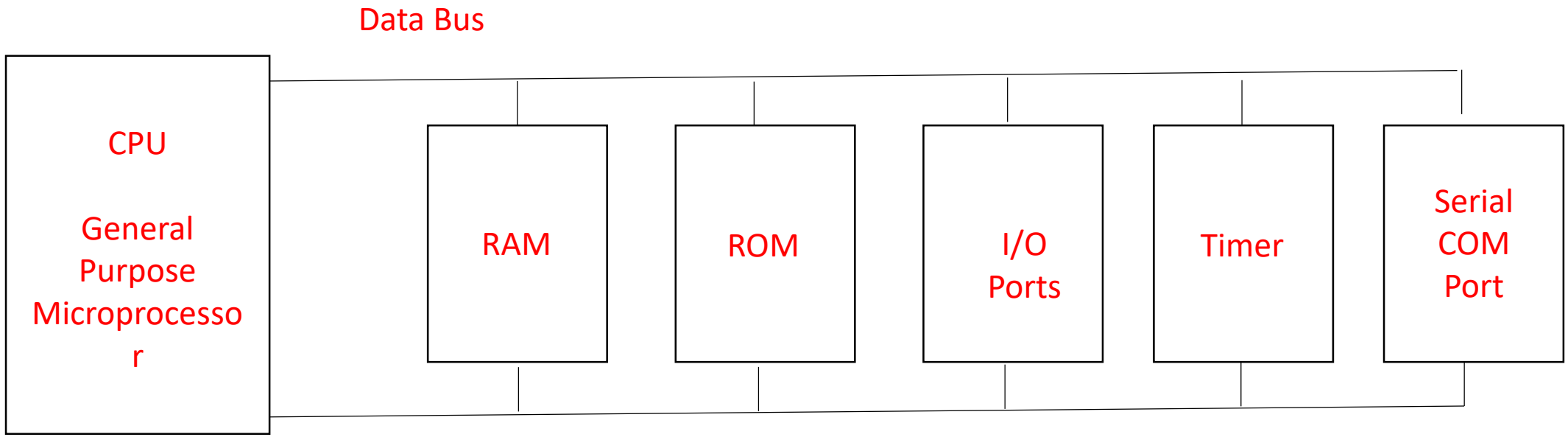
Devendra Rawat

Assistant Professor

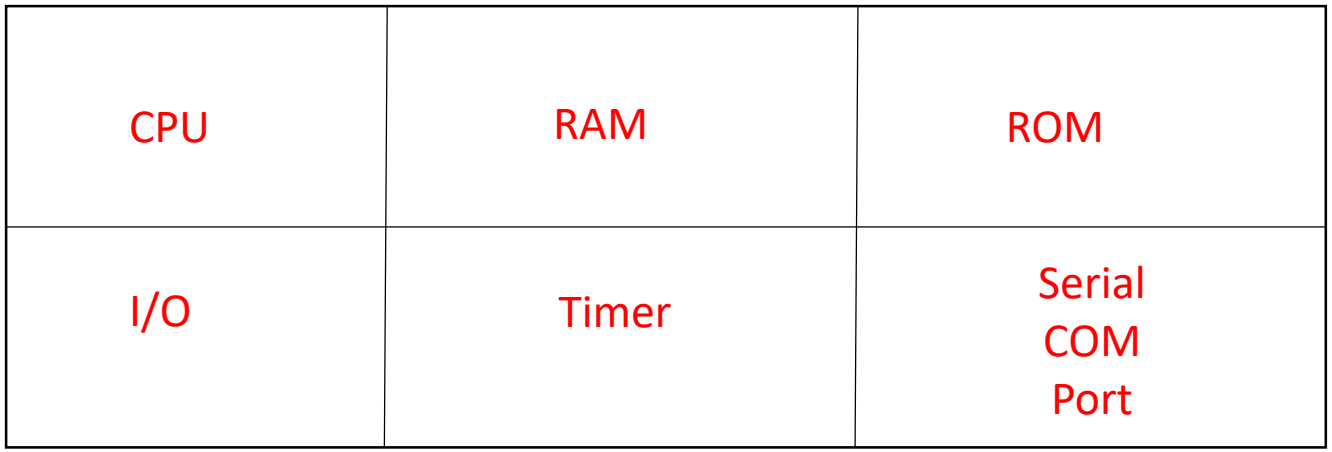
Department of Electrical & Electronics Engineering

UPES

UNIVERSITY WITH A PURPOSE



General Purpose Microprocessor System



Microcontroller

Choosing a microcontroller

- There are number of microcontroller available in the market from different companies like Freescale, Intel, Zilog, Microchip technology.
- Criteria for choosing a microcontroller:

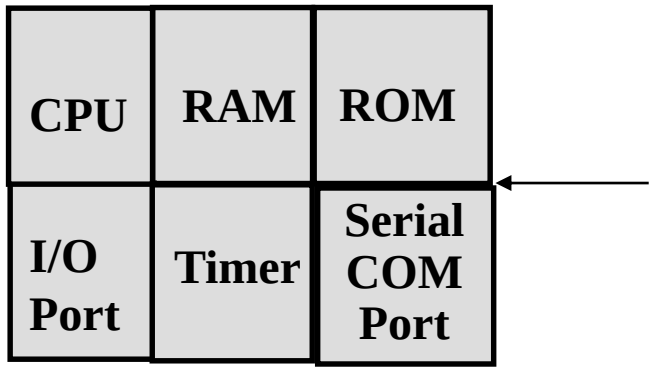
✓ Meeting the computing need of the task at hand efficiently and cost efficiently.

✓ Availability of software development tools such as compilers and assemblers.

✓ Wide availability and reliable sources of the microcontroller.

8051 microcontroller

- In 1981, Intel Corporation introduced 8051 microcontroller
- It is a 8 bit microcontroller
- It has 8 bit data bus and 16 bit address bus
- It is referred to as “system on a chip”
- 4K bytes internal ROM
- 128 bytes internal RAM
- Four 8-bit I/O ports (P0 - P3).
- Two 16-bit timers/counters
- One serial interface



A single chip

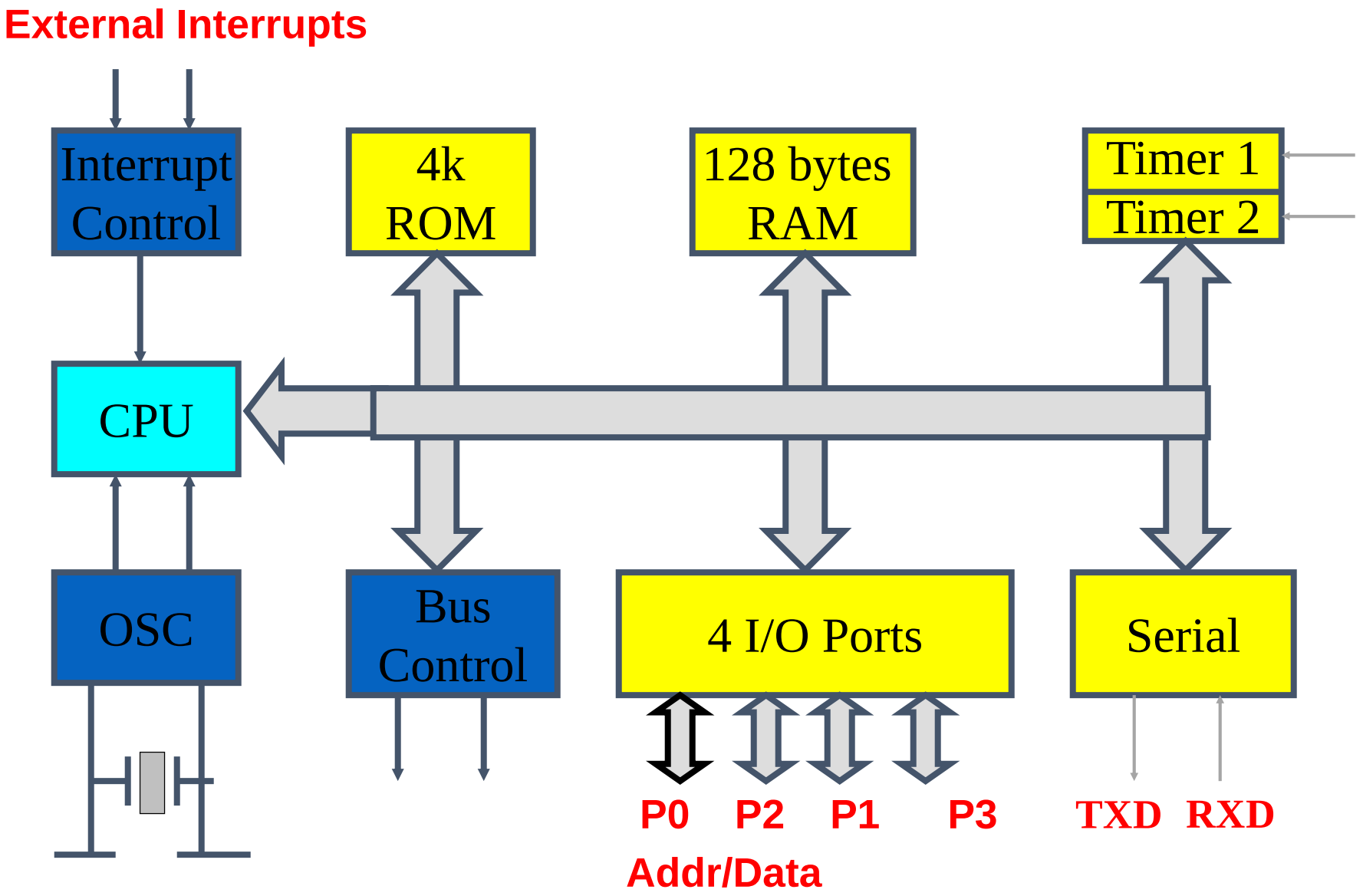
Microcontroller

- 8051 become popular because Intel allowed other manufacturer to make and market other flavors of the 8051 family. This led to many versions of the 8051 with different speed and amount of on-chip ROM. These are compatible with the original 8051 as far as instructions are concerned. Examples are 8052, 8031.
- Many vendors such as Atmel, Philips, and Texas Instruments produce MCS-51 family microcontroller chips.

Table 1. Comparison of 8051 Family Members

FEATURE	8051	8052	8031
ROM	4K	8K	0K
RAM	128	256	128
TIMERS	2	3	2
I/O PINS	32	32	32
SERIAL PORT	1	1	1
INTERRUPT SOURCES	6	8	6

Block diagram of 8051 microcontroller

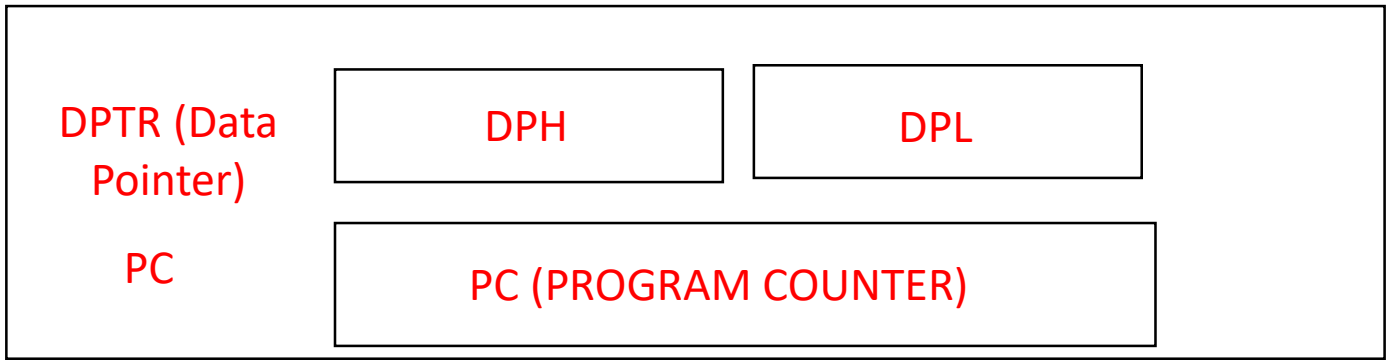


- The Intel 8051 is a very popular general purpose microcontroller widely used for small scale embedded systems.
- The 8051 is an 8-bit microcontroller with 8 bit data bus and 16-bit address bus.
- The 16 bit address bus can address a 64K(2^{16}) byte code memory space and a separate 64K byte of data memory space.
- Besides internal RAM, the 8051 has various *Special Function Registers* (SFR) such as the Accumulator, the B register, and many other control registers.

8-Bit Registers of 8051

A
B
R0
R1
R2
R3
R4
R5
R6
R7

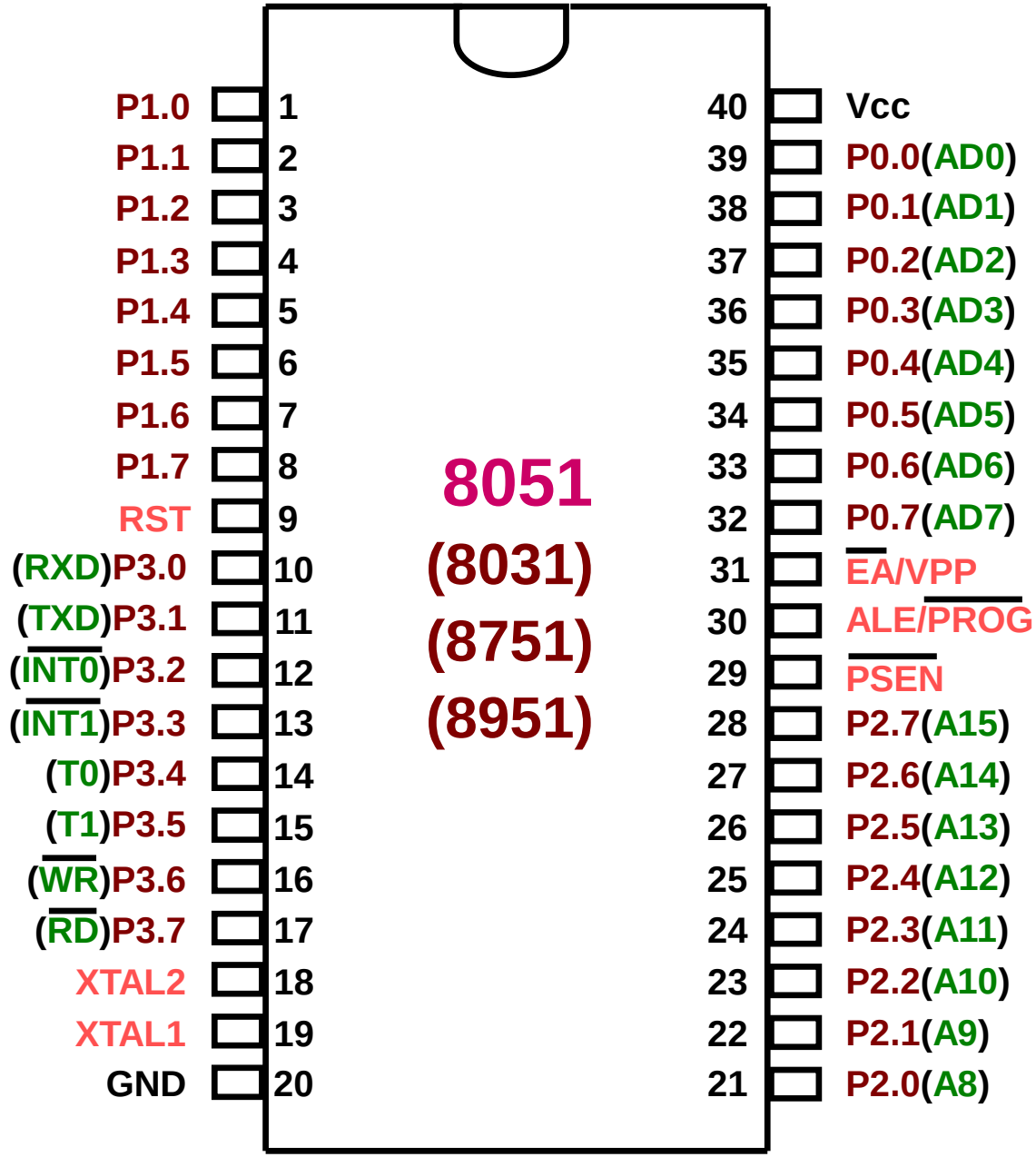
General Purpose Registers



16-Bit Registers of 8051

- The ALU performs one 8-bit operation at a time.
- Two 16 bit /Counter timers.
- 3 internal interrupts (one serial), 2 external interrupts.
- 4 8-bit I/O ports (3 of them are dual purposed). One of them used for serial port.
- 8051 chips come with UART for serial communication and ADC for analog to digital conversion.

8051 pin diagram



PORTS 0,1,2,3

- **One of the most useful features of the 8051 is that it contains four I/O ports (P0 - P3)**

- **Port 0 （pins 32-39） ：P0（P0.0～P0.7）**

  - ✓8-bit R/W - General Purpose I/O.
  - ✓In 8051 as no external ROM memory is connected, the pins of 8051 are connected to 10K-ohm external pull-up resistor. As with external pull-up resistor it can be used as a simple I/O ports.
  - ✓In 8031, where we are connecting external ROM memory, Port 0 provides both data and address.
  - ✓Port 0 is also designated as AD0-AD7.

- **Port 1 （pins 1-8） ：P1（P1.0～P1.7）**

  - ✓8-bit R/W - General Purpose I/O
  - ✓ It have internal pull-up resistors

- **Port 2 （pins 21-28） ：P2（P2.0～P2.7）**

  ✓8-bit R/W - General Purpose I/O.

  ✓ With external memory connections, Port 2 is used along with Port 0 to provide 16-bit address for the external memory.

  ✓ While P0 provides the lower 8 bits via A0-A7, it is the job of P2 to provide bits A8-A15 of the address and at this time it cannot be used for I/O.

  ✓ It have internal pull-up resistors

- **Port 3 （pins 10-17） ：P3（P3.0～P3.7）**

  ✓ It can be used for 8-bit R/W - General Purpose I/O.

  ✓ Port 3 is configured as an input port upon reset, this is not the way it is most commonly used.

  ✓ Port 3 has the additional functions of providing some extremely important signals such as interrupt.

  ✓It have internal pull-up resistors

Port 3 Alternate Functions

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

IMPORTANT PINS

- **PSEN** (out): Program Store Enable:

✓ It is an output pin.

✓ In 8031-based system in which an external ROM holds the program code, this pin is connected to the OE pin of the ROM.

- **ALE** (out): Address Latch Enable, to latch address outputs at Port0 and Port2

- **EA** (in): External Access Enable, active low to access external program memory locations 0 to 4K

- **RXD**,**TXD**: UART pins for serial I/O on Port 3

- **XTAL1** & **XTAL2**: Crystal inputs for internal oscillator.

- 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）：
  - These 2 pins provide external clock.
  - Way 1：using a quartz crystal oscillator
  - Way 2：using a TTL oscillator

- /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.

Inside the 8051

Registers

- ✓ **Registers are used to store information temporarily.**
- ✓ **The information can be byte of data to be processed, or an address pointing to the data to be fetched.**
- ✓ **The most widely used registers of 8051 are general purpose registers and special function registers.**
- ✓ **The most widely used 8-bit registers are A, B, R0, R1, R2, R3, R4, R5, R6, R7.**
- ✓ **The most widely used 16-bit registers are DPTR (data pointer) and PC.**

MOVE Instruction
MOV destination, source

```
MOV A, #55H
MOV R0, A
MOV R1, A
MOV R2, A
MOV R3, #0F5H
MOV A, R3
```

ADD Instruction
ADD A, source

```
MOV A, #25H
MOV R2, #34H
ADD A, R2
```

MOV A, 55H is also a valid instruction, means moves the value held in memory address 55H into accumulator

Flag bits and psw register

- PSW register is an 8 bit register and is also referred to as flag register.
- Only 6 bits of 8 bits are used by 8051.
- Two unused bits are user-definable flags.
- Four of the flags are conditional flags, meaning that they indicate some conditions after the instruction is executed.
- These four flags are CY (carry), AC (auxiliary carry), P(parity), OV (overflow).

PSW: PROGRAM STATUS WORD. BIT ADDRESSABLE.

CY	AC	F0	RS1	RS0	OV	—	P

CY	PSW.7	Carry Flag.
AC	PSW.6	Auxiliary Carry Flag.
F0	PSW.5	Flag 0 available to the user for general purpose.
RS1	PSW.4	Register Bank selector bit 1 (SEE NOTE 1).
RS0	PSW.3	Register Bank selector bit 0 (SEE NOTE 1).
OV	PSW.2	Overflow Flag.
—	PSW.1	User definable flag.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of '1' bits in the accumulator.

NOTE:
1. The value presented by RS0 and RS1 selects the corresponding register bank.

RS1	RS0	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

Carry Flag (CY):

- This flag is set whenever there is a carry out from the D7 bit.
- This flag can also be 1 or 0 directly by an instruction “SETB C” (set bit carry) and “CLR C” (clear carry)

Auxiliary Carry Flag (AC):

- This flag is set whenever there is a carry from D3 to D4 during an arithmetic operation.

Parity Flag (P):

- It reflects the number of 1s in the accumulator.
- Odd number of 1s, P=1
- Even number of 1s, P=0

Overflow Flag (OV):

- This flag is only used to detect errors in signed arithmetic operations.
- The 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.

Show the status of the CY, AC and P flag after the addition of 9CH and 64H.

```
MOV A, #9CH
ADD A, #64H
```

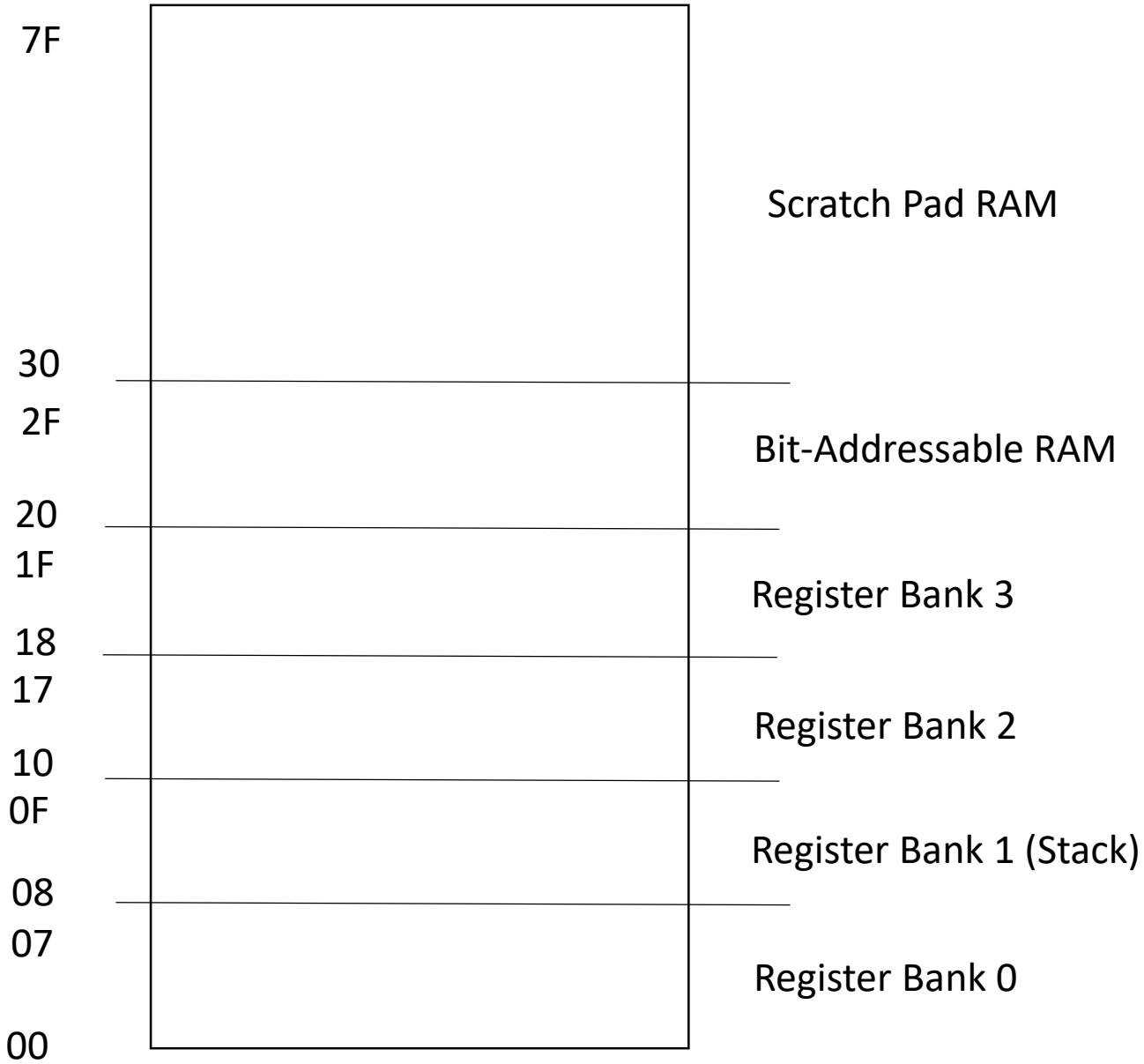
```
  9C      10011100
+ 64      01100100
100        00000000
```

CY = 1
AC = 1
P = 0

8051 REGISTER BANKS AND STACK

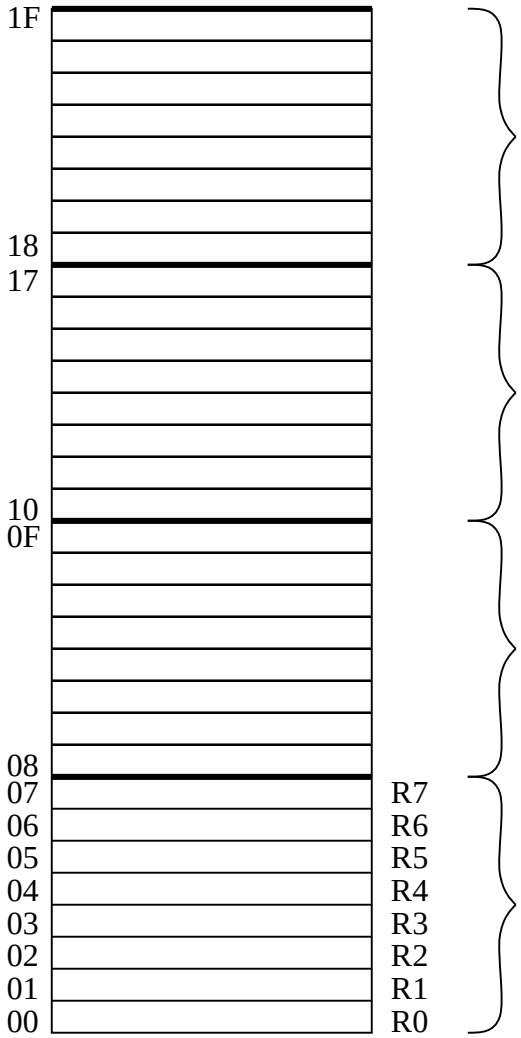
RAM MEMORY SPACE ALLOCATION IN 8051

1. A total of 32 bytes from location 00 to 1F hex are set aside for register banks and the stack.
2. A total of 16 bytes from location 20H to 2FH are set aside for bit-addressable read/write memory.
3. A total of 80 bytes from location 30H to 7FH are used for read and write storage or what is normally called as scratch pad. These 80 locations of RAM are widely used for the purpose of storing data and parameters by 8051 programmers. It can be used to store data brought into the CPU via I/O ports.

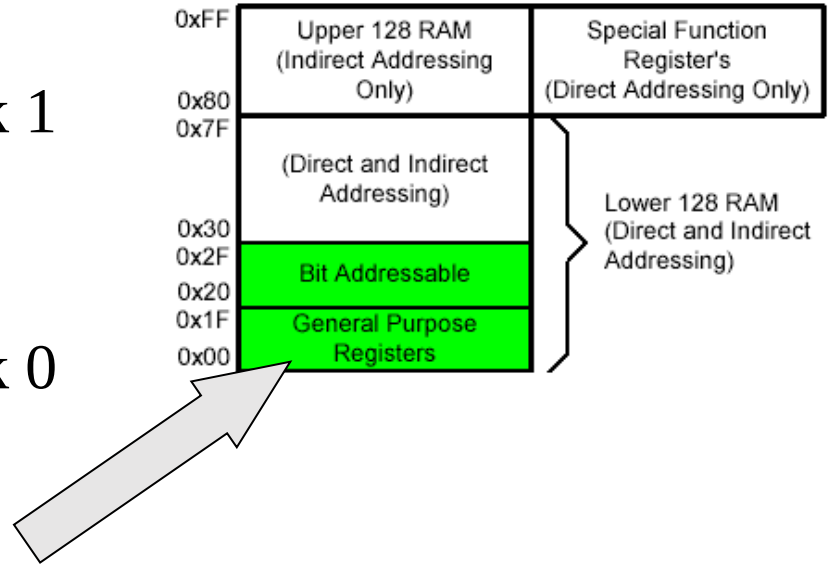


RAM Allocation in 8051

Registers



Four Register Banks
Each bank has R0-R7
Selectable by psw.2,3



Default Register Bank

- When 8051 is powered up, register bank 0 is accessed with the names R0, R1, R2, R3, R4, R5, R6, R7.

Switch Register Bank

- Using PSW register we can switch to other register banks.
- D3 and D4 bits of register PSW referred to as PSW.4 and PSW.3 are accessed by bit-addressable instructions SETB and CLR.

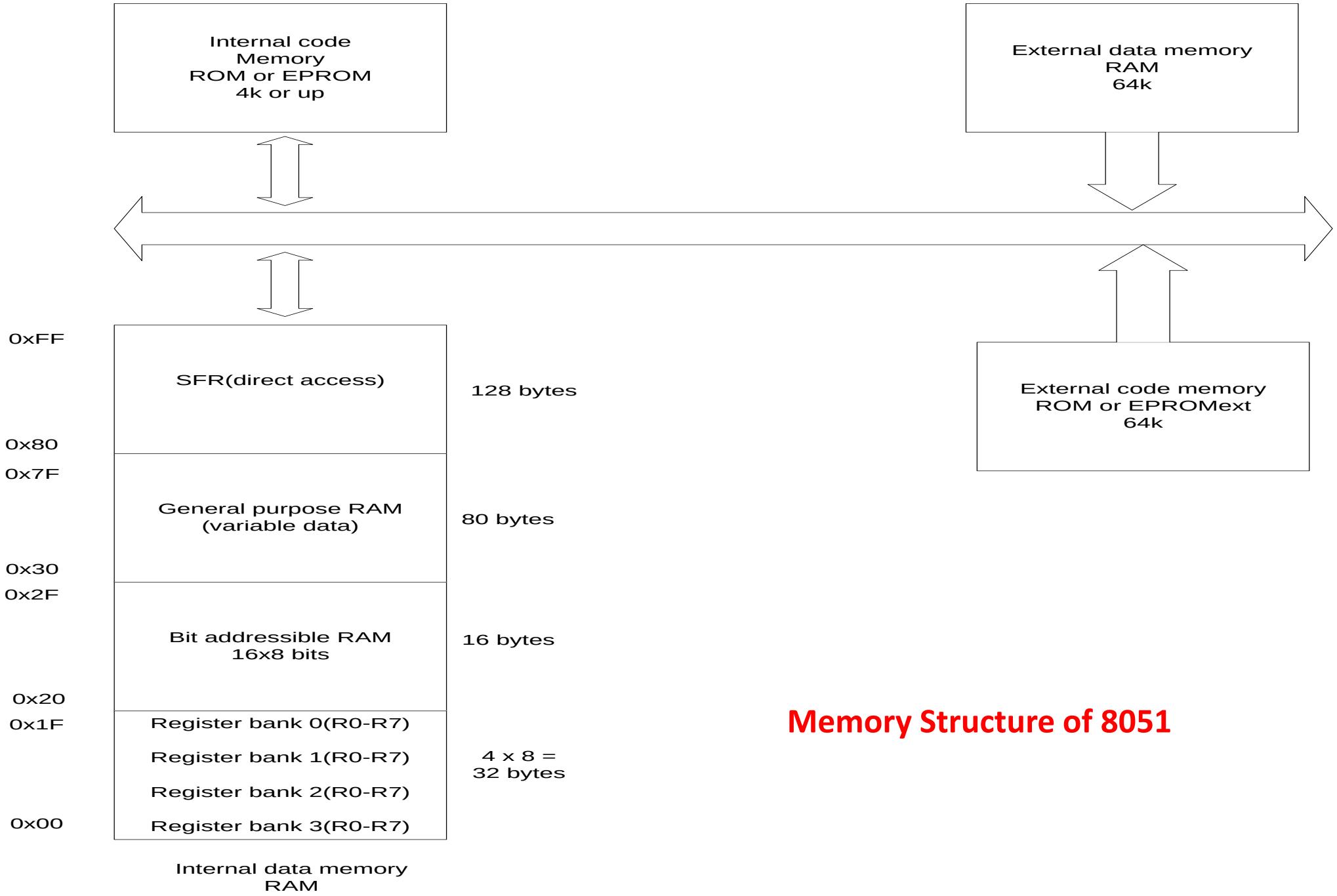
	RS1 (PSW.4)	RS0 (PSW.3)
Bank 0	0	0
Bank 1	0	1
Bank 2	1	0
Bank 3	1	1

Stack in 8051:

- Stack is a section of RAM used by the CPU to store information temporarily.
- Information can be data or address.
- Stack Pointer (SP) is used to access the stack.
- SP is only 8 bits wide and take the value from 00 to FFH.
- When the 8051 is powered up, the SP register contains value 07.
- RAM location 08 is the first location used for the stack by 8051.
- PUSH and POP instruction are used to save and retrieve content from the stack.
- In PUSH instruction SP is incremented by one while in POP instruction stack pointer is decremented by one.

Upper limit of Stack:

- Location 08 to 1F of RAM can be used for the stack.
- If more than 24 bytes are required for storage then we can change the SP to RAM location 30-7FH. This is done with the instruction “MOV SP, #xx”



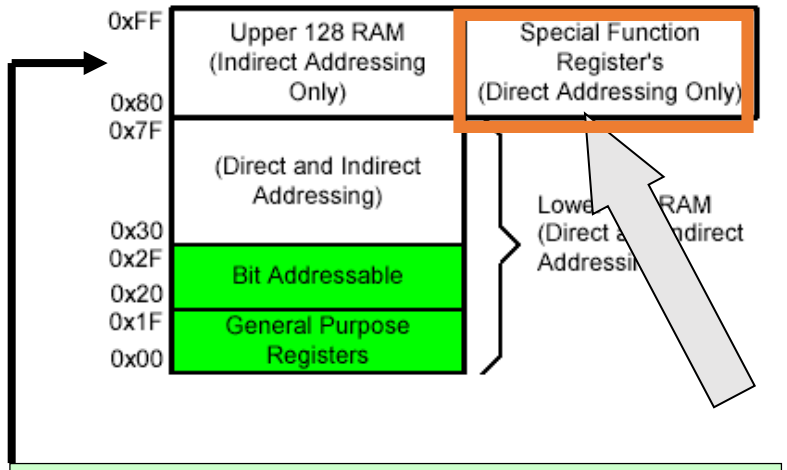
Memory Structure of 8051

SPECIAL FUNCTION REGISTERS

- Register A, B, PSW, DPTR are some of special function registers.
- SFR can be accessed by their names or by their addresses. For example- A has address E0H and B has address F0H.
- SFR have addresses between 80H and FFH.
- Not all the address space of 80H and FF is used by SFR. The unused locations 80H to FFH are reserved and must not be used by the 8051 programmer.

SPECIAL FUNCTION REGISTERS

- DATA registers
- CONTROL registers
  - Timers
  - Serial ports
  - Interrupt system
  - Analog to Digital converter
  - Digital to Analog converter
  - Etc.



Addresses 80h – FFh

Direct Addressing used to access SPRs

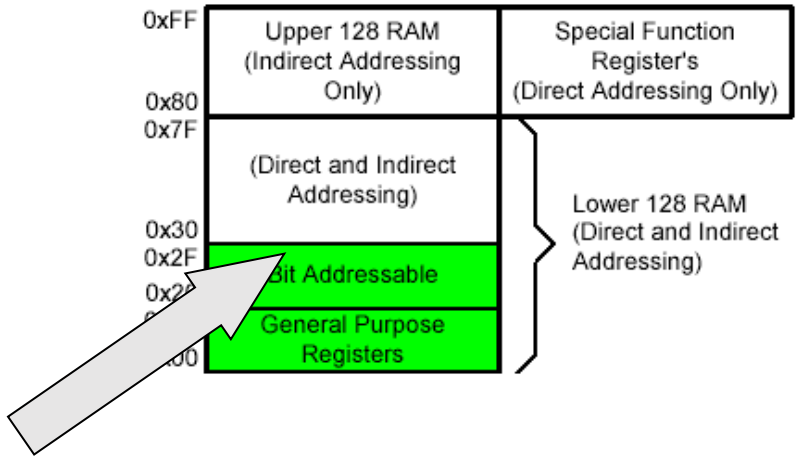
Table 1. 80C51 Special Function Registers

SYMBOL	DESCRIPTION	DIRECT ADDRESS	BIT ADDRESS, SYMBOL, OR ALTERNATIVE PORT FUNCTION MSB							LSB	RESET VALUE
ACC*	Accumulator	E0H	E7	E6	E5	E4	E3	E2	E1	E0	00H
B*	B register	F0H	F7	F6	F5	F4	F3	F2	F1	F0	00H
DPTR	Data pointer (2 bytes)										
DPH	Data pointer high	83H									00H
DPL	Data pointer low	82H									00H
			AF	AE	AD	AC	AB	AA	A9	A8	
IE*	Interrupt enable	A8H	EA	–	–	ES	ET1	EX1	ET0	EX0	0x000000B
			BF	BE	BD	BC	BB	BA	B9	B8	
IP*	Interrupt priority	B8H	–	–	–	PS	PT1	PX1	PT0	PX0	xx000000B
			87	86	85	84	83	82	81	80	
P0*	Port 0	80H	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	FFH
			97	96	95	94	93	92	91	90	
P1*	Port 1	90H	–	–	–	–	–	–	T2EX	T2	FFH
			A7	A6	A5	A4	A3	A2	A1	A0	
P2*	Port 2	A0H	A15	A14	A13	A12	A11	A10	A9	A8	FFH
			B7	B6	B5	B4	B3	B2	B1	B0	
P3*	Port 3	B0H	$\overline{RD}$	$\overline{WR}$	T1	T0	$\overline{INT1}$	$\overline{INT0}$	TxD	Rxd	FFH
PCON[1]	Power control	87H	SMOD	–	–	–	GF1	GF0	PD	IDL	0xxxxxxxB
			D7	D6	D5	D4	D3	D2	D1	D0	
PSW*	Program status word	D0H	CY	AC	F0	RS1	RS0	OV	–	P	00H
SBUF	Serial data buffer	99H									xxxxxxxxB
			9F	9E	9D	9C	9B	9A	99	98	
SCON*	Serial controller	98H	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	00H
SP	Stack pointer	81H									07H
			8F	8E	8D	8C	8B	8A	89	88	
TCON*	Timer control	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
TH0	Timer high 0	8CH									00H
TH1	Timer high 1	8DH									00H
TL0	Timer low 0	8AH									00H
TL1	Timer low 1	8BH									00H
TMOD	Timer mode	89H	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0	00H

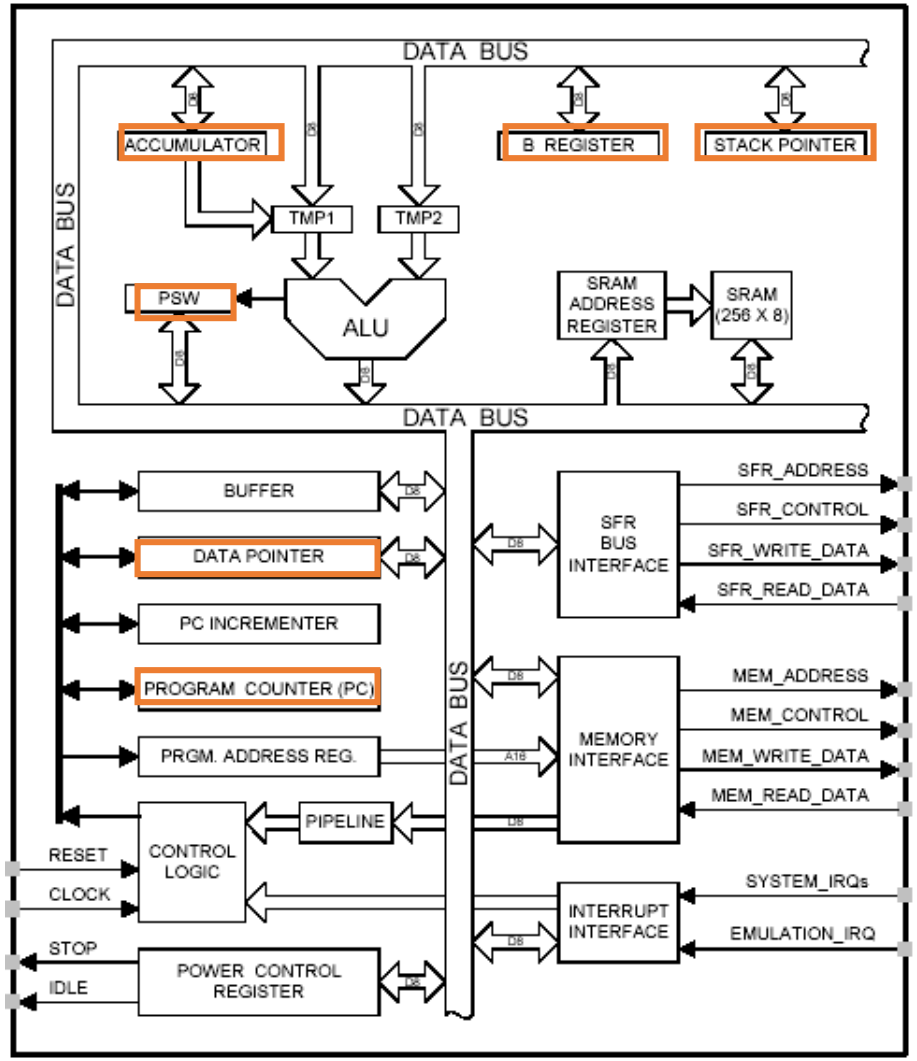
Bit Addressable Memory

2F	7F							78
2E								
2D								
2C								
2B								
2A								
29								
28								
27								
26								
25								
24						1A		
23								10
22	0F							08
21	07	06	05	04	03	02	01	00
20								

20H – 2FH (16 locations
X 8-bits = 128 bits)



8051 CPU Registers



- A (Accumulator)
- B
- PSW (Program Status Word)
- SP (Stack Pointer)
- PC (Program Counter)
- DPTR (Data Pointer)

Used in assembler instructions

8051 Assembly Language

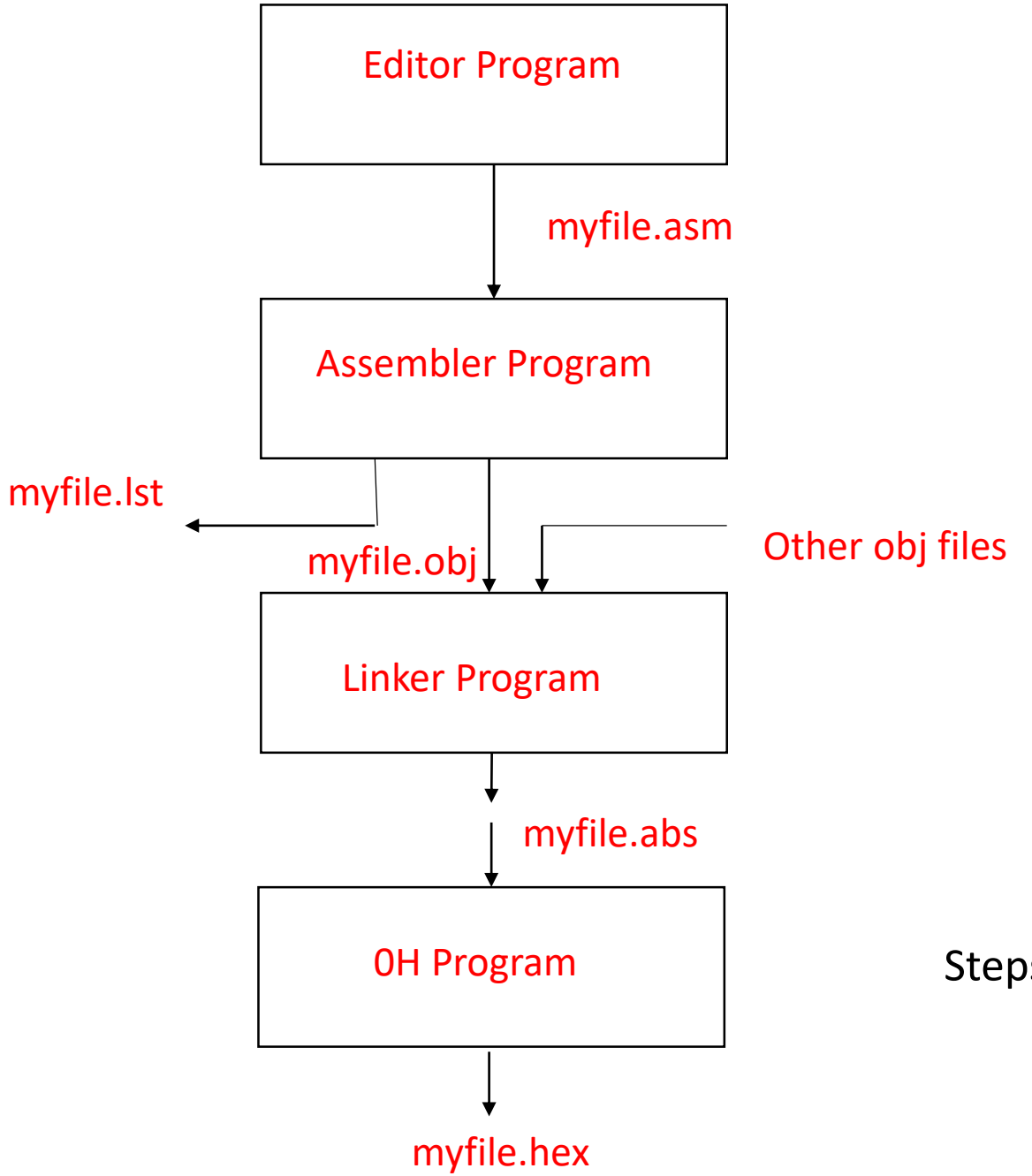
- CPU can only work on binary data and it can do so at a very high speed.
- A program consist of 0s and 1s is called machine language.
- For humans it is very difficult to remember code in machine language.
- Assembly language were developed that provide mnemonics for the instruction and make programming faster and less prone to error.
- An assembler is a program converts the assembly language program into machine language.
- Assembly language program is referred to as a low language because it directly deals with the internal structure of the CPU.
- To program in Assembly language, the programmer must know all the registers and their size.

- These days programmers are using different high level programming languages such as BASIC, C, C++, Java.
- These languages are called as high level language because the programmer does not have to be concerned with the internal details of the CPU.
- Compilers is a program that converts high-level language into machine language.

Structure of assembly language

- Assembly language programming consists of series of instructions, which consist of a mnemonic, optionally followed by one or two operands.
- Assembly language program consists of instructions such as MOV, ADD or the statements called directives.
- Instructions tells CPU what to do while directives gives directions to assembler.
- MOV, ADD instruction are command to CPU
- ORG and END are directives to the assembler.
- ORG- Start of the Program
- END- End of the source code
- [label:] mnemonic [operands] [; comment]

```
        ORG   0000H        ; Start origin at location 0
        MOV R5, #25H       ; Load 25H into R5
        MOV R7, #34H       ; Load 34H into R7
        MOV A, #0          ; Load 0 into A
        ADD A, R5          ; A=A+R5
        ADD A,R7           ; A=A+R7
        ADD A, #12H        ; A=A+12H
HERE: SJMP HERE              ; Stay in this loop
        END                ; end of the asm source file
```



Steps to create a program

- asm file- It is also called the source file. This file is created with an editor such as DOS EDIT or Windows Notepad. The 8051 assembler converts the asm file into machine language and provides the obj (object) file. In addition to creating the object file, the assembler also produces the lst file (list file).
- lst file: lst file lists all the opcodes and addresses as well as the errors that the assembler detects. The programmer uses the list file to find the errors. It is only after fixing all the errors in the list file that the object file is ready to be input to the linker program.

* When 8051 is powered up, the PC has the value 0000. Thus first opcode must be burned into memory location 0000H of ROM. This is achieved by the ORG instruction in the source program.

8051 Addressing Modes

- CPU can access data in various ways. The data could be a register, or in memory, or be powered as an immediate value. The various ways of accessing data are called addressing modes. 8051 provides five different addressing modes:

✓ Immediate

✓ Register

✓ Direct

✓ Register Indirect

✓ Indexed

Immediate Addressing Modes

Immediate Mode – specify data by its value

```
MOV A, #0           ;put 0 in the accumulator
                    ;A = 00000000

MOV R4, #11H        ;put 11hex in the R4 register
                    ;R4 = 00010001

MOV B, #11          ;put 11 decimal in b register
                    ;B = 00001011

MOV DPTR,#7521H     ;put 7521 hex in DPTR
                    ;DPTR = 0111010100100001
```

MOV DPTR,#7521h

```
MOV DPL,#21H
MOV DPH, #75
```

This addressing mode can also be used to send data to 8051 ports.

MOV P1, # 55H

Register Addressing MODE

Register Addressing – either source or destination is one of CPU register

```
MOV R0,A
MOV A,R7
ADD A,R4
ADD A,R7
MOV DPTR,#25F5H
MOV R5,DPL
MOV R6,DPH
```

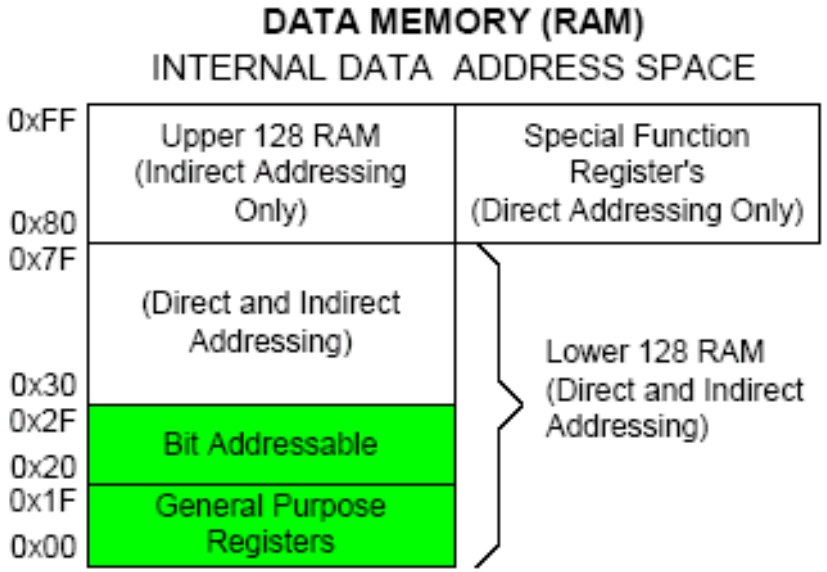
Note: that `MOV R4,R7` is incorrect

Direct Addressing Mode

Direct Mode – specify data by its 8-bit address

Usually for 30h-7Fh of RAM

```
Mov A, 70H          ; Save contents of RAM location 70H in
  A
Mov R0,40H            ; Save contents of RAM location
  40H in R0
Mov 56H,A             ; Save contents of A at 56H
Mov 0D0H,A                                  into PSW
```



Direct Mode – play with R0-R7 by direct address

```
MOV A,4   ≡   MOV A,R4

MOV A,7   ≡   MOV A,R7

MOV 7,2   ≡   MOV R7,R6
```

Invalid

```
MOV R2,#5    ;Put 5 in R2
MOV R2,5     ;Put content of RAM at 5 in R2
```

Register Indirect addressing mode

Register Indirect – the address of the source or destination is specified in registers. Registers is used as a pointer of the data.

Only registers R0 or R1 for 8-bit address:

```
MOV A, @R0          ; move contents of RAM location whose
  address                                    is held by
  R0 into A

MOV @R1, B          ; move contents of B into RAM location
  whose                                      address is
  held by R1
```

In cases where we have to access external RAM or on-chip ROM, where we need 16-bit pointer, DPTR register is used.

Register indexed Addressing Modes

Register Indexed Mode

- It is used to access data elements of look-up table entries located in the program ROM space of 8051.
- The instruction used is **`MOVC A, @A + DPTR.`**
- The 16-bit register DPTR and register A are used to form the address of the data element stored in on-chip ROM
- Base address can be DPTR or PC

```
MOV DPTR, #4000H
MOV A, #5
MOVC A, @A + DPTR  ;A ← M[4005]
```

In the indexed addressing mode, the source memory can only be accessed from program memory only. The destination operand is always the register A. These are some examples of Indexed addressing mode

```
MOVCA, @A+PC;
MOVCA, @A+DPTR;
```

The C in MOVC instruction refers to code byte. For the first instruction, let us consider A holds 30H. And the PC value is1125H. The contents of program memory location 1155H (30H + 1125H) are moved to register A

Implied Addressing Mode

In the implied addressing mode, there will be a single operand. These types of instruction can work on specific registers only.
These types of instructions are also known as register specific instruction. Here are some examples of Implied Addressing Mode.
RLA;
SWAPA;
These are 1- byte instruction.
The first one is used to rotate the A register content to the Left.
The second one is used to swap the nibbles in A.

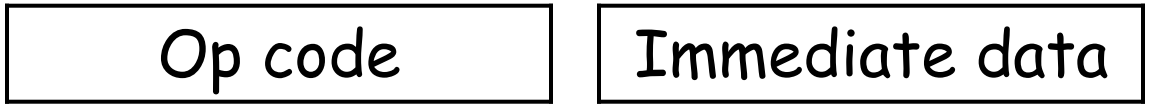
Question: Assume that ROM space starting at 250H contains “America”, write a program to transfer the bytes into RAM locations starting at 40H.

```
        ORG 0000
        MOV DPTR, #0250H      ; load ROM Pointer
        MOV R0, #40H          ; load RAM Pointer
        MOV R2, #7            ; load counter
BACK:   CLR A                 ; A= 0
        MOVC A, @A+DPTR       ; move data from code space
        MOV @R0, A            ; save it in RAM
        INC DPTR              ; increment ROM pointer
        INC R0                ; increment RAM pointer
        DJNZ R2, BACK         ; loop until counter = 0
HERE:   SJMP HERE
        ORG 250H
MYDATA: DB “AMERICA”
        END
```

DJNZ: Decrement and Jump if register ≠ 0
SJMP: Short Jump

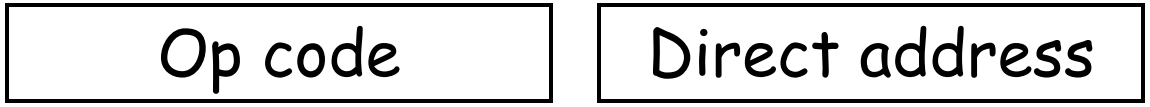
8051 Instruction Format

- Immediate Addressing



ADD A,#3DH ;machine code=**243DH**

- Direct Addressing



MOV R3,058H ;machine code= **AB58H**

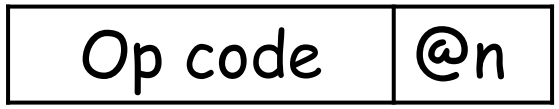
- **In 8051 family only direct addressing mode is allowed for PUSH and POP instructions.**
- **PUSH A is invalid while PUSH 0E0H is valid.**

- Register Addressing


```
070D E8     MOV A,R0        ;E8 = 1110 1000
070E E9     MOV A,R1        ;E9 = 1110 1001
070F EA     MOV A,R2        ;EA = 1110 1010
0710 ED     MOV A,R5        ;ED = 1110 1101
0711 EF     MOV A,R7        ;Ef = 1110 1111
0712 2F     ADD A,R7
0713 F8     MOV R0,A
0714 F9     MOV R1,A
0715 FA     MOV R2,A
0716 FD     MOV R5,A
0717 FD     MOV R5,A
```

Source and Destination registers must match in size.

- Register Indirect addressing



MOV A, @Ri ; i = 0 or 1

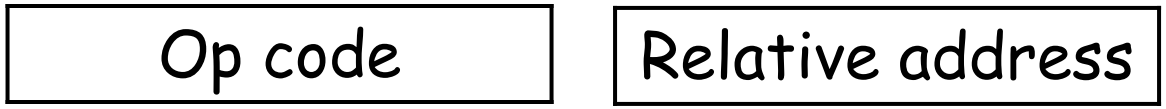
```
070D E7     MOV  A,@R1
0711 F2     MOV @R0,A
0712 E3     MOV A,@R1
```

- Register Indexed addressing

MOVC A, @A+DPTR

```
MOVC  A,@A+DPTR
MOVC R0,@A+DPTR
MOVC R1,@A+DPTR
```

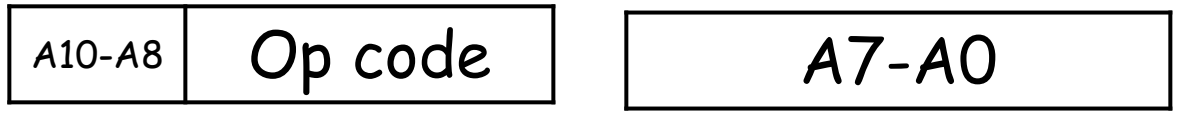
- Relative Addressing



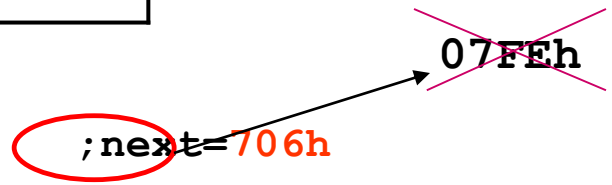
here: SJMP here ;machine code=**80FE** **(FE=-2)**

Range = (-128 ~ 127)

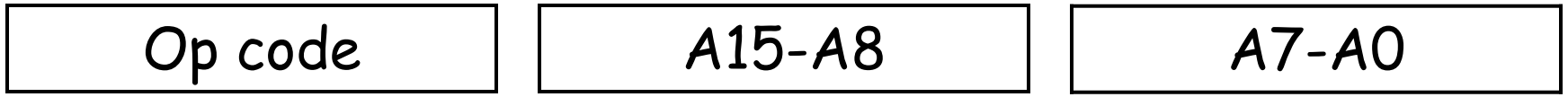
- Absolute addressing (limited in 2k current mem block)




```
0700                    1           org 0700h
0700 E106               2           AJMP next    ;next=706h
0702 00                 3           NOP
0703 00                 4           NOP
0704 00                 5           NOP
0705 00                 6           NOP
                        7     next:
                        8           END
```



- Long Distance Address



Range = (0000H ~ FFFFH)

```
0700                    1           org 0700H
0700 020707             2           AJMP next    ;next=0707H
0703 00                 3           NOP
0704 00                 4           NOP
0705 00                 5           NOP
0706 00                 6           NOP
                        7     next:
                        8           END
```

Accumulator Register

- A register can be accessed by direct and register addressing mode

- The two instruction has same function with different code

```
0703 E500          MOV A,00H
0708 8500E0        MOV 0E0H,00H
```

- Also this two instruction

```
070B E9            MOV A,R0
070E 89E0          MOV 0E0H,R0
```

SFRs Address

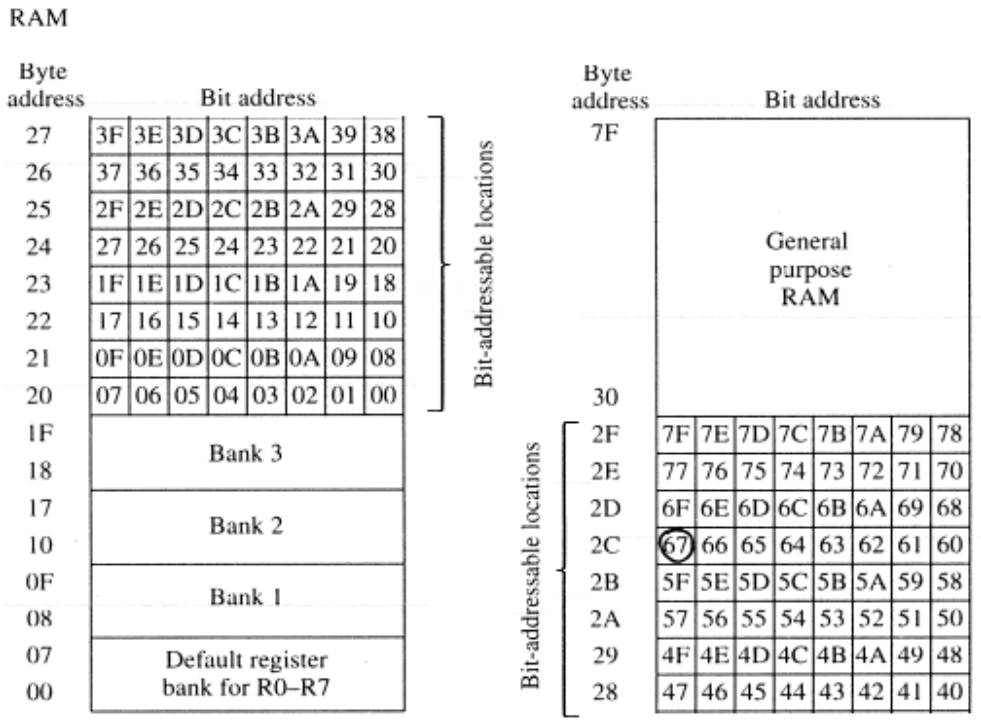
- SFR can be accessed by their names or by their addresses.
- MOV 0E0H, # 55H MOV A, # 55H
- MOV 0F0H, #25H MOV B, #25H
- MOV P1, A MOV 90H, A

Transfer the content of the registers A, R0, R1 of bank 0 to the registers B, R0, R1 of bank 1 using stack operation.

```
MOV SP, # 3FH
PUSH 00
PUSH 01
PUSH 0E0H
SETB PSW.3
POP 0F0H
POP 09
POP 08
```

Bit ADDRESSES FOR I/O AND RAM

- Internal RAM locations 20-2FH are both byte-addressable and bit addressable.
- These 16 bytes provides 128 bits of RAM bit-addressability since: 16 X 8 = 128.
- They are addressed as 00 to 7FH.



Single bit Instructions

Instruction	Function
SETB bit	Set the bit (bit=1)
CLR bit	Clear the bit (bit = 0)
CPL bit	Complement the bit (bit = NOT bit)
JB bit, target	Jump to target if bit = 1 (Jump if bit)
JNB bit, target	Jump to target if bit = 0 (Jump if no bit)
JBC bit, target	Jump to target if bit = 1, clear bit (Jump if bit, then clear)

- Single bit instruction use only one addressing mode that is direct addressing

I/O port bit addresses

- 8051 family has four 8-bits I/O ports: P0, P1, P2, P3.
- Either the entire 8 bits or any single bit can be accessed.
- Syntax SETB X.Y is used. X=Port number and Y= Desired bit.

  Example: SETB P1.5 sets high bit 5 of port 1.
- Ports are also addressable by their addresses.

P0	Addr	P1	Addr	P2	Addr	P3	Addr	Port's Bit
P0.0	80	P1.0	90	P2.0	A0	P3.0	B0	D0
P0.1	81	P1.1	91	P2.1	A1	P3.1	B1	D1
P0.2	82	P1.2	92	P2.2	A2	P3.2	B2	D2
P0.3	83	P1.3	93	P2.3	A3	P3.3	B3	D3
P0.4	84	P1.4	94	P2.4	A4	P3.4	B4	D4
P0.5	85	P1.5	95	P2.5	A5	P3.5	B5	D5
P0.6	86	P1.6	96	P2.6	A6	P3.6	B6	D6
P0.7	87	P1.7	97	P2.7	A7	P3.7	B7	D7

Bit Addresses for all Ports

Data Transfer Instructions

- MOV dest, source dest ← source
- Stack instructions

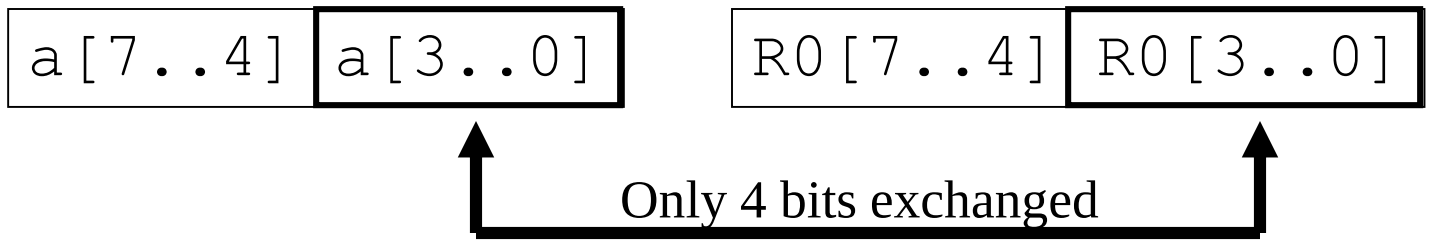
```
PUSH byte       ;increment stack pointer,
                ;move byte on stack
POP byte        ;move from stack to byte,
                ;decrement stack pointer
```

Exchange Instructions

- Exchange instructions

two way data transfer

```
XCH A, 30H          ; a ←→ M[30]
XCH A, R0           ; a ←→ R0
XCH A, @R0          ; a ←→ M[R0]
XCHD A, R0          ; exchange “digit”
```



Data Processing Instructions

- Arithmetic Instructions
- Logic Instructions

Arithmetic Instructions

- Add
- Subtract
- Increment
- Decrement
- Multiply
- Divide
- Decimal adjust

Arithmetic Instructions

Mnemonic	Description
`ADD A, byte`	add A to byte, put result in A
`ADDC A, byte`	add with carry
`SUBB A, byte`	subtract with borrow
`INC A`	increment A
`INC byte`	increment byte in memory
`INC DPTR`	increment data pointer
`DEC A`	decrement accumulator
`DEC byte`	decrement byte
`MUL AB`	multiply accumulator by b register
`DIV AB`	divide accumulator by b register
`DA A`	decimal adjust the accumulator

ADD Instructions

```
ADD A, byte          ; A ← A + byte
ADDC A, byte                ; A ← A + byte + C
```

These instructions affect 3 bits in PSW:

C = 1 if result of add is greater than FF

AC = 1 if there is a carry out of bit 3

OV = 1 if there is a carry out of bit 7, but not from bit 6, or visa versa.

Program Status Word (PSW)

Bit	7	6	5	4	3	2	1	0
Flag	CY	AC	F0	RS1	RS0	OV	F1	P
Name	Carry Flag	Auxiliary Carry Flag	User Flag 0	Register Bank Select 1	Register Bank Select 0	Overflow flag	User Flag 1	Parity Bit

ADD Examples

```
MOV A, #3FH
ADD A, #D3H
```

```
  0011 1111
  1101 0011
-------------
  0001 0010
```

```
C = 1
AC = 1
OV = 0
```

- What is the value of the C, AC, OV flags after the second instruction is executed?

Addition Example

Two numbers are stored in registers R0 and R1. Verify if their sum is greater than FFH.

```
        MOV A, R0
        ADD A, R1
        JC MESSAGE
        SJMP NEXT
MESSAGE: MOV A, #'Y'
        MOV P1, A
NEXT:   NOP
        END
```

The 16-bit ADD example

Add two numbers, the numbers are FC45H and 02ECH

```
CLR C
MOV A, #45H
ADD A, #0ECH
MOV R0, A
MOV A, #02H
ADDC A, #0FCH
MOV R1,A
```

R0 = 31H

R1 = FFH

Subtraction

SUBB A, byte	subtract with borrow

Example:

```
SUBB A, #4FH      ;A ← A – 4F – C
```

Notice that
There is no subtraction WITHOUT borrow.
Therefore, if a subtraction without borrow is desired,
it is necessary to clear the C flag.

Example:

```
ClR  c
SUBB A, #04FH     ;A ← A – 4F
```

Increment and Decrement

INC A	increment A
INC byte	increment byte in memory
INC DPTR	increment data pointer
DEC A	decrement accumulator
DEC byte	decrement byte

- The increment and decrement instructions do NOT affect the C flag.
- Notice we can only INCREMENT the data pointer, not decrement.

Example: Increment 16-bit Word

- Assume 16-bit word in R3:R2

```
MOV A, R2
ADD A, #1          ; use add rather than increment to affect C
MOV R2, A
MOV A, R3
ADDC A, #0         ; add C to most significant byte
MOV R3, A
```

Multiplication

- Multiplication and Division works only with A and B registers.
- When multiplying two 8-bit numbers, the size of the maximum product is 16-bits

FF x FF = FE01

(255 x 255 = 65025)

MUL AB; AB ← A * B, Places the 16-bit result in B and A

Note : B gets the High byte
A gets the Low byte

Example:

```
MOV A, # 25H
MOV B, # 65H
MUL AB
```

Division

- Integer Division

```
DIV AB      ; divide A by B

A ← Quotient(A/B)
B ← Remainder(A/B)
```

OV - used to indicate a divide by zero condition.
C – set to zero

MOV A, #95H
MOV B, #10H
DIV AB

Decimal Adjust

DA A ; decimal adjust Accumulator

- Used to facilitate BCD addition.
- Adds “6” to either high or low nibble after an addition
- To create a valid BCD number.
- DA works only after an ADD instruction and not work after the INC instruction

EXAMPLE:

```
MOV A, #23H
MOV B, #29H
ADD A, B          ; A ← 23H + 29H = 4CH (wanted 52)
DA A              ; A ← A + 6 = 52
```

Logic Instructions

- Bitwise logic operations
  - (AND, OR, XOR, NOT)
- Clear
- Rotate
- Swap

Logic instructions do NOT affect the flags in PSW

Bitwise Logic

ANL → AND

ORL → OR

XRL → XOR

CPL → Complement

Examples:

```
  00001111
ANL 10101100
  00001100

  00001111
ORL 10101100
  10101111

  00001111
XRL 10101100
  10100011

CPL 10101100
  01010011
```

AND OPERATION

ANL destination, source; destination = destination AND source

- The destination is normally the accumulator,
- Source operand can be a register, memory, immediate
- ANL instruction is often used to mask certain bits of an operand

Representation:

- ANL A, Rn
- ANL A, direct
- ANL A, @ Ri
- ANL A, #data
- ANL direct, #data

```
MOV A, #35H
ANL A, #0FH
```

OR OPERATION

ORL destination, source; destination = destination OR source

- The destination is normally the accumulator,
- Source operand can be a register, memory, immediate
- ORL instruction is often used to set certain bits of an operand to 1

Representation:

- ORL A, Rn
- ORL A, direct
- ORL A, @ Ri
- ORL A, #data
- ORL direct, #data

```
MOV A, #04H
ORL A, #30H
```

XOR OPERATION

XRL destination, source; destination = destination XOR source

- The destination is normally the accumulator,
- Source operand can be a register, memory, immediate
- XRL instruction is often used to see if two registers have the same value

```
MOV A, #54H
XRL A, #78H
```

- Representation:
- XRL A, Rn
- XRL A, direct
- XRL A, @ Ri
- XRL A, #data
- XRL direct, #data

COMPLEMENT OPERATION

CPL A Complement the content of register A

- This is also called 1's complement
- CPL instruction cannot be used to complement R0-R7
- CPL instruction works with A and P0-P3 ports

- Representation:
- CPL A
- CPL bit

Find the 2's complement of the value 85H

```
MOV A, #85H
CPL A
ADD A, #1H
```

COMPARE OPERATION

CJNE destination, source, relative address; Compare and Jump if not equal

- In 8051, the actions of comparing and Jumping are combined into a single instruction called CJNE
- It changes the Carry flag to indicate if the destination operand is larger or smaller.
- After the operation operands remain unchanged
- The destination operand can be in the accumulator or in one of Rn registers
- Source operand can be in a register, memory or immediate.
- Any Rn register can be compared with immediate value, no need for register A

Representation

- CJNE A, direct, relative address
- CJNE A, #data, relative address
- CJNE Rn, #data, relative address
- CJNE @Ri, #data, relative address

Compare	Carry Flag
Destination ≥ source	CY = 0
Destination < source	CY = 1

Other Logic Instructions

CLR - clear

RL - rotate left

RLC - rotate left through Carry

RR - rotate right

RRC - rotate right through Carry

SWAP - swap accumulator nibbles

CLear operation (Set all bits to 0)

CLR A (Register addressing mode)

CLR byte (Direct addressing mode)

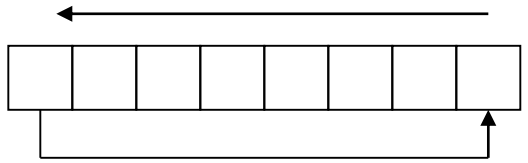
CLR Ri (Register addressing mode)

CLR @Ri (Register Indirect addressing mode)

Rotate operation

- Rotate instructions operate only on Accumulator

- **RL A**

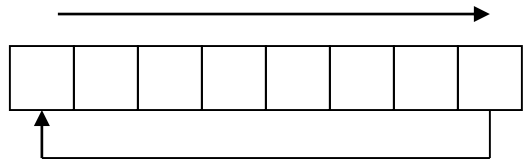


In rotate left, 8 bits of the accumulator are rotated left one bit, and bit D7 exists from MSB and enters D0 LSB.

```
Mov A,#0F0 ; A← 11110000
RL A       ; A← 11100001
```

- **RR A**

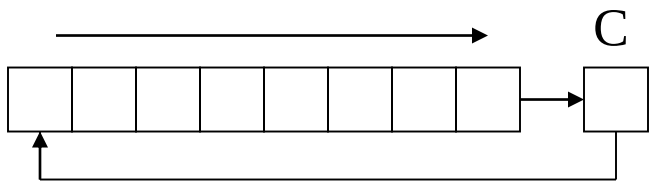
In rotate right, 8 bits of the accumulator are rotated right one bit, and bit D0 exists from LSB and enters D7 MSB.



```
Mov A,#0F0      ; A← 11110000
RR A            ; A← 01111000
```

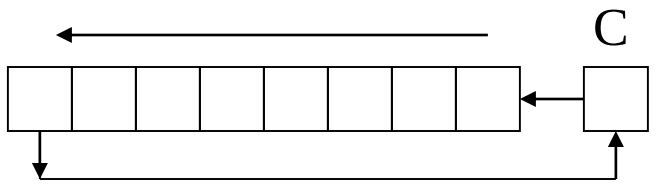
Rotate through Carry

- **RRC A**




```
MOV A, #0A9H ; A ← A9
ADD A, #14H        ; A ← BD (10111101), C←0
RRC A              ; A ←  01011110, C←1
```

- **RLC A**




```
MOV A, #3CH        ; A ← 3CH(00111100)
SETB C       ; C ← 1
RLC A              ; A ←  01111001, C←0
```

Swap operation

SWAP A

- It works only on the accumulator (A)
- It swaps the lower nibble and the higher nibble.
- The lower 4 bits are put into the higher 4 bits, and the higher 4 bits are put into the lower 4 bits.

```
MOV A, #72H          ; A ← 72H
SWAP A               ; A ← 27H
```

Bit Logic Operations

- Some logic operations can be used with single bit operands

```
ANL C, bit
ORL C, bit
CLR C
CLR bit
CPL C
CPL bit
SETB C
SETB bit
```

- “bit” can be any of the bit-addressable RAM locations or SFRs.

Program Flow Control INSTRUCTIONS

- Unconditional Jumps (“go to”) Instructions
- Conditional Jumps Instructions
- Call and Return Instructions

Unconditional Jumps

Unconditional jump is a jump in which control is transferred unconditionally to the target location

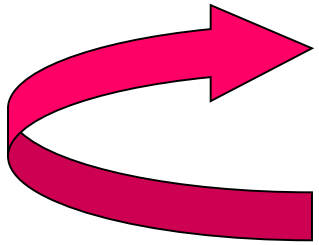
- **LJMP <address 16>** ; Long jump

It is a three byte instruction which allows a jump to any memory location from 0000 to FFFFH.

- **SJMP <rel addr>** ; Short jump

It is a two byte instruction, second byte is the relative address of the target location ranges from 00-FFH (-128 to 127), so jump can be up to 127 locations forward, or 128 locations back.

Infinite Loops



```
Start: MOV C, p3.7
       MOV p1.6, C
       SJMP Start
```

Microcontroller application programs are almost always infinite loops!

Conditional Jump

- All conditional jump instructions are short jump, means the target address must be within -128 to 127 bytes of the contents of the program counter (PC).
- These instructions cause a jump to occur only if a condition is true. Otherwise, program execution continues with the next instruction.

```
loop: MOV A, P1
      JZ  loop      ; if A=0, goto loop,else goto next instruction
      MOV B, A
```

- There is no zero flag (z)
- Content of A checked for zero on time

Conditional jumps

MNEMONIC	DESCRIPTION
JZ <rel addr>	Jump if A = 0
JNZ <rel addr>	Jump if A != 0
JC <rel addr>	Jump if C = 1
JNC <rel addr>	Jump if C != 1
JB <bit>, <rel addr>	Jump if bit = 1
JNB <bit>,<rel addr>	Jump if bit != 1
JBC <bit>, <rel addr>	Jump if bit =1, &clear bit
CJNE A, direct, <rel addr>	Compare A and memory, jump if not equal

Conditional Jumps

Mnemonic	Description
CJNE A, #data <rel addr>	Compare A and data, jump if not equal
CJNE Rn, #data <rel addr>	Compare Rn and data, jump if not equal
CJNE @Rn, #data <rel addr>	Compare Rn and memory, jump if not equal
DJNZ Rn, <rel addr>	Decrement Rn and then jump if not zero
DJNZ direct, <rel addr>	Decrement memory and then jump if not zero

Iterative Loops (examples)

```
        MOV A,#50H
        MOV B,#00H
        CJNE A,#50H,NEXT
        MOV  B,#01H
NEXT:   NOP
        END
```

```
        MOV A,#0AAH
        MOV B,#10H
Back1:  MOV R6,#50
Back2:  CPL A
        DJNZ R6,BACK2
        DJNZ B,BACK1
        END
```

```
        MOV A,#25H
        MOV R0,#10H
        MOV R2,#5
Again:  MOV @Ro,A
        INC R0
        DJNZ R2,AGAIN
        END
```

```
        MOV A,#0H
        MOV R4,#12H
Back:   ADD A,#05
        DJNZ R4,BACK
        MOV  R5,A
        END
```

Call INSTRUCTION

- CALL is similar to a JUMP, but
  - CALL pushes PC on stack before branching

```
ACALL <address 11>          ; stack ← PC
                            ; PC ← address 11 bit

LCALL <address 16>          ; stack ← PC
                            ; PC ← address 16 bit
```

Return INSTRUCTION

- RETURN is also similar to a jump, but
  - Return instruction pops PC from stack to get address to jump to

```
RET                    ; PC ← stack
```

Single bit operations with carry

- Carry bit is altered by arithmetic and logic instructions, in 8051 there are also several instructions by which CY flag can be manipulated directly.

Instruction	Function
SETB C	Make CY = 1
CLR C	Clear carry bit CY = 0
CPL C	Complement carry bit
MOV bit, C	Copy carry status to bit location
MOV C, bit	Copy bit location status to carry
JNC target	Jump to target if CY = 0
JC target	Jump to target if CY =1
ANL C, bit	AND CY with bit and it on CY
ANL C, /bit	AND CY with inverted bit and save it on CY
ORL C, bit	OR CY with bit and save it on CY
ORL C, /bit	OR CY with inverted bit and save it on CY

Assume that bit P2.2 is used to control an outdoor light and bit P2.5 a light inside a building. WAP to turn on the outside light and turn off the inside light.

```
SETB C
ORL C, P2.2
MOV P2.2, C
CLR C
ANL C, P2.5
MOV P2.5, C
END
```

Assembly language programs

Two numbers are stored in registers R0 and R1. Verify if their sum is greater than FFH.

```
        MOV A, R0
        ADD A, R1
        JC MESSAGE
        SJMP NEXT
MESSAGE: MOV A, #'Y'
        MOV P1, A
NEXT:    NOP
        END
```

Assume that RAM locations 40-44H have values 7DH, EBH, C5H, 5BH, 30H. WAP to find the sum of all values. At the end of the program, register A should contain the low byte and R7 the high byte.

```
        MOV R0, #40H
        MOV R2, #5H
        CLR A
        MOV R7, A
AGAIN: ADD A, @R0
        JNC NEXT
        INC R7
NEXT:   INC R0
        DJNZ R2, AGAIN
        END
```

- WAP to subtract two 16 bit numbers 2762H and 1296H. Save the result in R6 and R7 registers.

```
CLR C
MOV A, #62H
SUBB A, #96H
MOV R7, A
MOV A, #27H
SUBB A, #12H
MOV R6, A
```

- In a semester, a student has to take six courses. The marks of the student out of 25 are stored in RAM locations 47H onwards. Find the average marks, and output it on port 1.

```
        MOV R1,#06H
        MOV B, #06H
        MOV R0, #47H
        MOV A, #0H
BACK:   ADD A, @R0
        INC R0
        DJNZ R1, BACK
        DIV AB
        MOV P1, A
        END
```

WAP to read and test P1 to see whether it has the value 45H. If it does, send 99H to P2; otherwise, it stay cleared.

```
        MOV P2, #00H
        MOV P1, #0FFH       ; make P1an input port
        MOV R3, #45H
        MOV A, P1
        XRL A, R3
        JNZ EXIT
        MOV P2, #99H
EXIT: END
```

Ten hex numbers are stored in RAM location 50H onwards. Write a program to find the biggest number in the set. The biggest number should finally be saved in 60H.

```
        MOV R0, #50H
        MOV R1, #10H
        MOV B, #0H
BACK:   MOV A, @R0
        CJNE A, B, LOOP
LOOP:   JC LOOP1
        MOV B, A
        INC R0
        DJNZ R1, BACK
        SJMP NEXT
LOOP1:  INC R0
        DJNZ R1, BACK
NEXT:   MOV A, B
        MOV 60H, A
        END
```

Assume that P1 is an input port connected to a temperature sensor. Write a program to read the temperature and test it for the value 75. According to the test results, place the temperature value into the registers indicated by the following.

IF T = 75 then A = 75

IF T < 75 then R1 = T

IF T > 75 then R2 = T

```
        MOV P1, #0FFH
        MOV A, P1
        CJNE A, #75H,OVER
        SJMP EXIT
OVER: JNC NEXT
        MOV R1, A
        SJMP EXIT
NEXT:  MOV R2, A
EXIT:    END
```

- WAP to take data continuously from PORT 0 and send it to PORT 1.

```
        MOV A, #0FFH
        MOV P0, A
BACK:   MOV A, P0
        MOV P1, A
        SJMP BACK
```