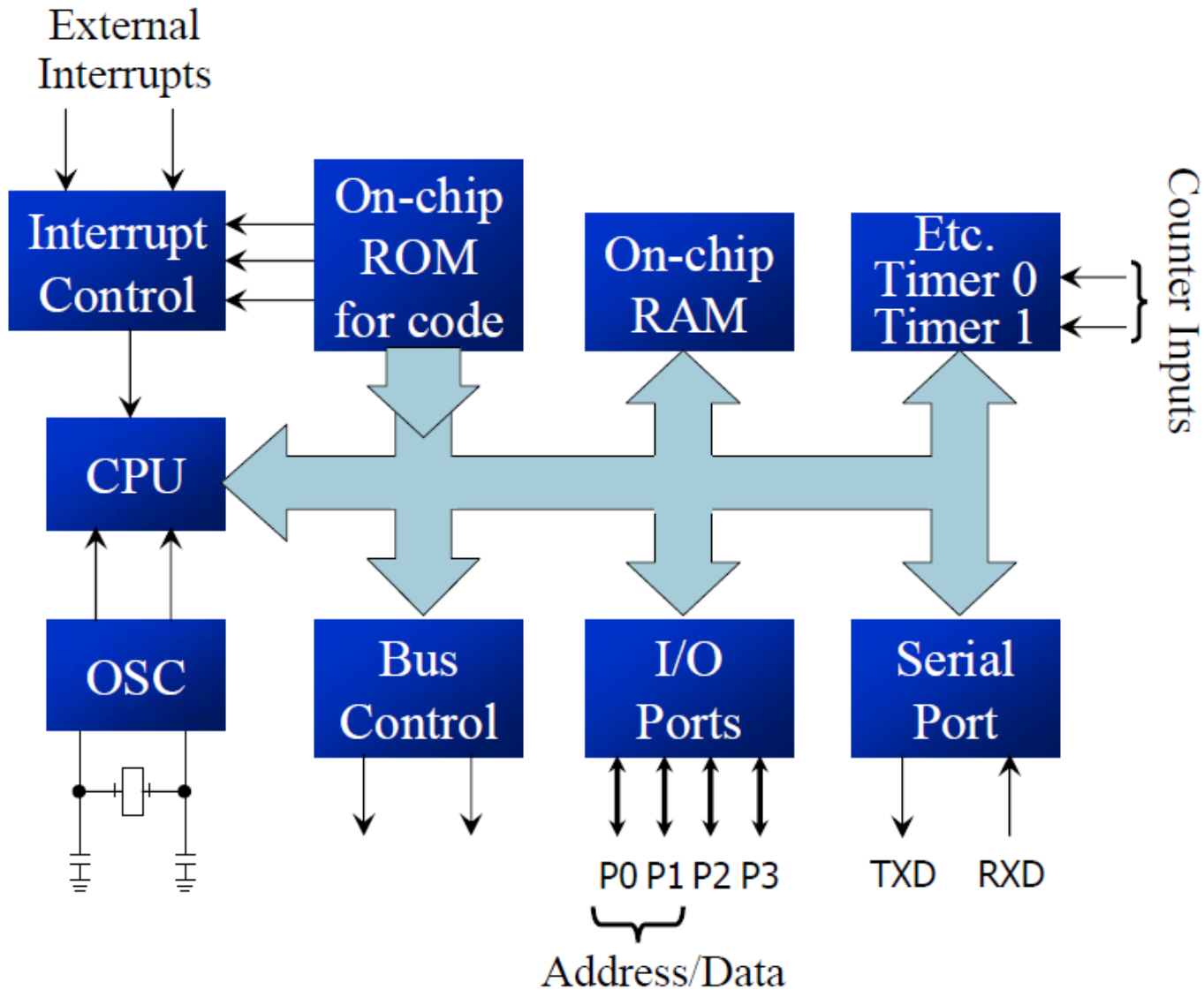
ISC 502
Applications of Microcontroller

Mrs. Kanchan Chavan

- The 8051 family has the largest number of diversified (multiple source) suppliers
  - Intel (original)
  - Atmel
  - Philips/Signetics
  - AMD
  - Infineon (formerly Siemens)
  - Matra
  - Dallas Semiconductor/Maxim

- Intel introduced 8051, referred as MCS-51, in 1981
  - The 8051 is an 8-bit processor
    - The CPU can work on only 8 bits of data at a time
  - The 8051 had
    - 128 bytes of RAM
    - 4K bytes of on-chip ROM
    - Two timers
    - One serial port
    - Four I/O ports, each 8 bits wide
    - 6 interrupt sources
- The 8051 became widely popular after allowing other manufactures to make and market any flavor of the 8051, but remaining code-compatible


External
Interrupts
Interrupt
Control
On-chip
ROM
for code
On-chip
RAM
Etc.
Timer 0
Timer 1
Counter Inputs
CPU
OSC
Bus
Control
I/O
Ports
Serial
Port
P0 P1 P2 P3
TXD
RXD
Address/Data

- The 8051 is a subset of the 8052
- The 8031 is a ROM-less 8051
  - Add external ROM to it
  - You lose two ports, and leave only 2 ports for I/O operations

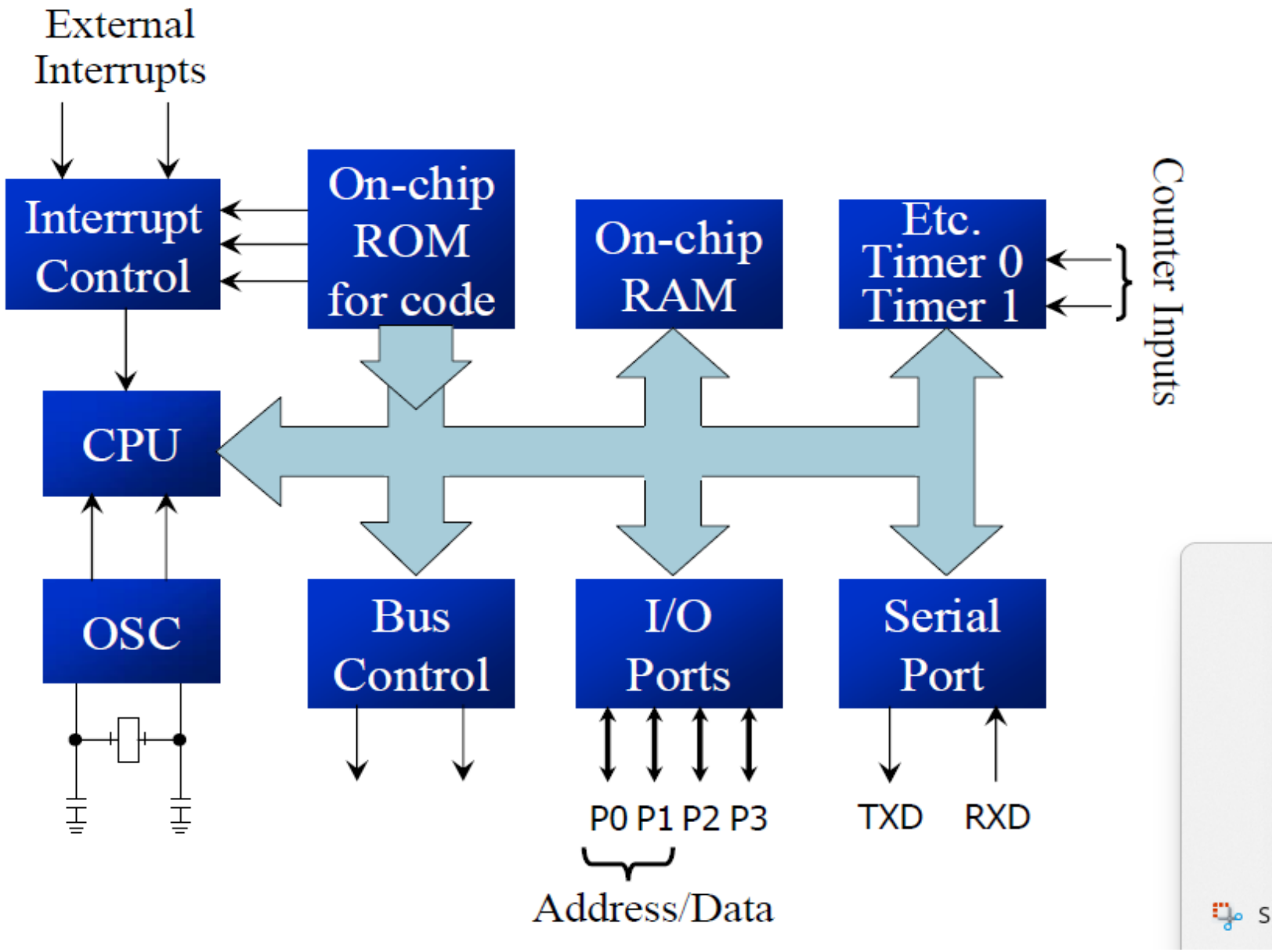
Feature	8051	8052	8031
ROM (on-chip program space in bytes)	4K	8K	0K
RAM (bytes)	128	256	128
Timers	2	3	2
I/O pins	32	32	32
Serial port	1	1	1
Interrupt sources	6	8	6

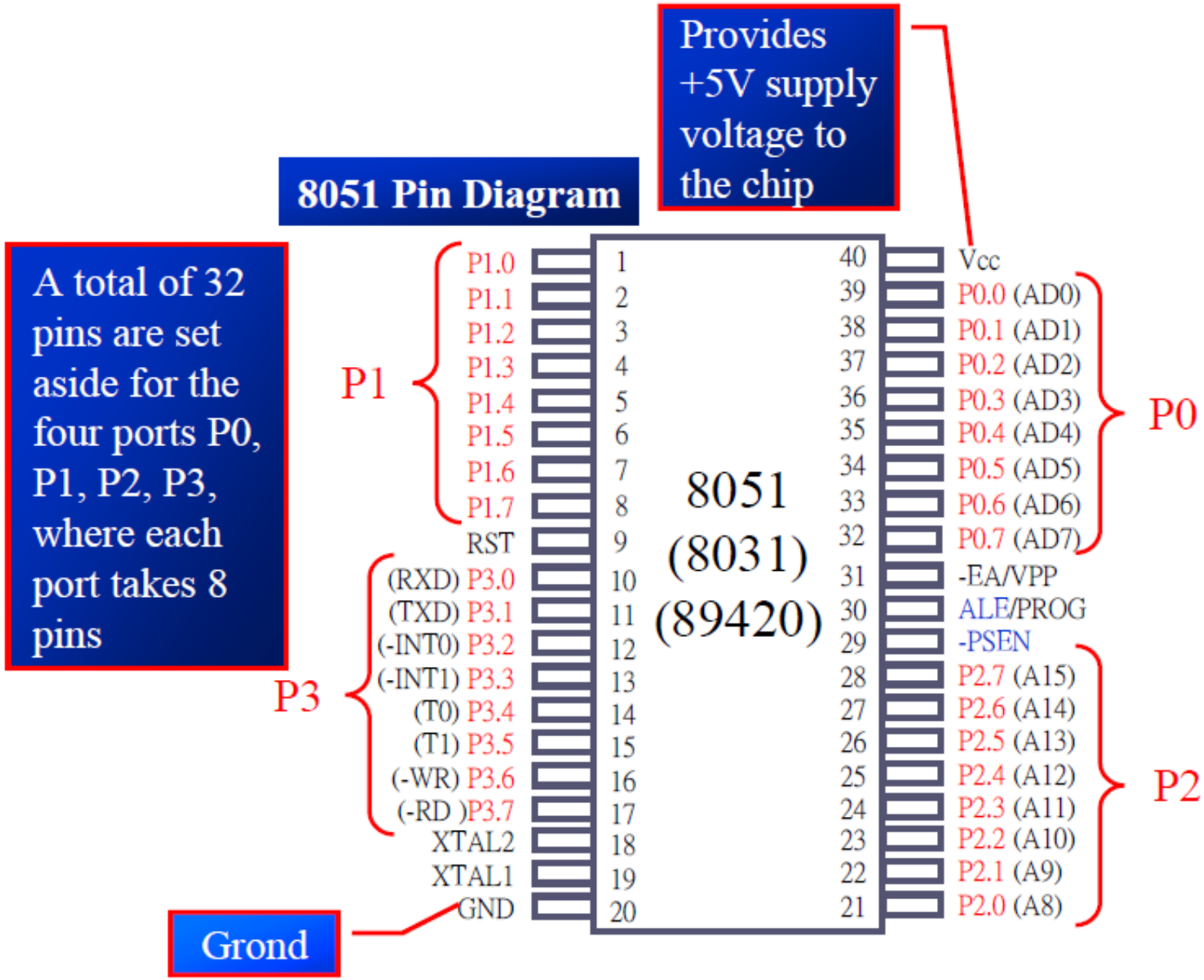
- 8751 microcontroller
  - UV-EPROM
    - PROM burner
    - UV-EPROM eraser takes 20 min to erase
- AT89C51 from *Atmel Corporation*
  - Flash (erase before write)
    - ROM burner that supports flash
    - A separate eraser is not needed
- DS89C4x0 from *Dallas Semiconductor*, now part of *Maxim Corp.*
  - Flash
    - Comes with on-chip loader, loading program to on-chip flash via PC COM port

- DS5000 from *Dallas Semiconductor*
  - NV-RAM (changed one byte at a time), RTC (real-time clock)
    - Also comes with on-chip loader
- OTP (one-time-programmable) version of 8051
- 8051 family from *Philips*
  - ADC, DAC, extended I/O, and both OTP and flash

NVRAM – Non volatile Random Access Memory

- Intel introduced 8051, referred as MCS-51, in 1981
  - The 8051 is an 8-bit processor
    - The CPU can work on only 8 bits of data at a time
  - The 8051 had
    - 128 bytes of RAM
    - 4K bytes of on-chip ROM
    - Two timers
    - One serial port
    - Four I/O ports, each 8 bits wide
    - 6 interrupt sources
- The 8051 became widely popular after allowing other manufactures to make and market any flavor of the 8051, but remaining code-compatible

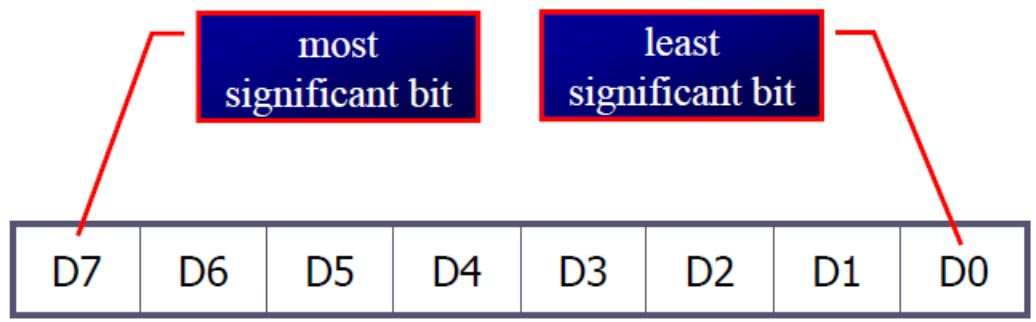

External
Interrupts
Interrupt
Control
On-chip
ROM
for code
On-chip
RAM
Etc.
Timer 0
Timer 1
Counter Inputs
CPU
OSC
Bus
Control
I/O
Ports
Serial
Port
P0 P1 P2 P3
TXD
RXD
Address/Data


8051 Pin Diagram
Provides +5V supply voltage to the chip
A total of 32 pins are set aside for the four ports P0, P1, P2, P3, where each port takes 8 pins
Grond
P1
P3
P0
P2
8051
(8031)
(89420)
1 P1.0
2 P1.1
3 P1.2
4 P1.3
5 P1.4
6 P1.5
7 P1.6
8 P1.7
9 RST
10 (RXD) P3.0
11 (TXD) P3.1
12 (-INT0) P3.2
13 (-INT1) P3.3
14 (T0) P3.4
15 (T1) P3.5
16 (-WR) P3.6
17 (-RD) P3.7
18 XTAL2
19 XTAL1
20 GND
40 Vcc
39 P0.0 (AD0)
38 P0.1 (AD1)
37 P0.2 (AD2)
36 P0.3 (AD3)
35 P0.4 (AD4)
34 P0.5 (AD5)
33 P0.6 (AD6)
32 P0.7 (AD7)
31 -EA/VPP
30 ALE/PROG
29 -PSEN
28 P2.7 (A15)
27 P2.6 (A14)
26 P2.5 (A13)
25 P2.4 (A12)
24 P2.3 (A11)
23 P2.2 (A10)
22 P2.1 (A9)
21 P2.0 (A8)

8051 ASSEMBLY LANGUAGE PROGRAMMING

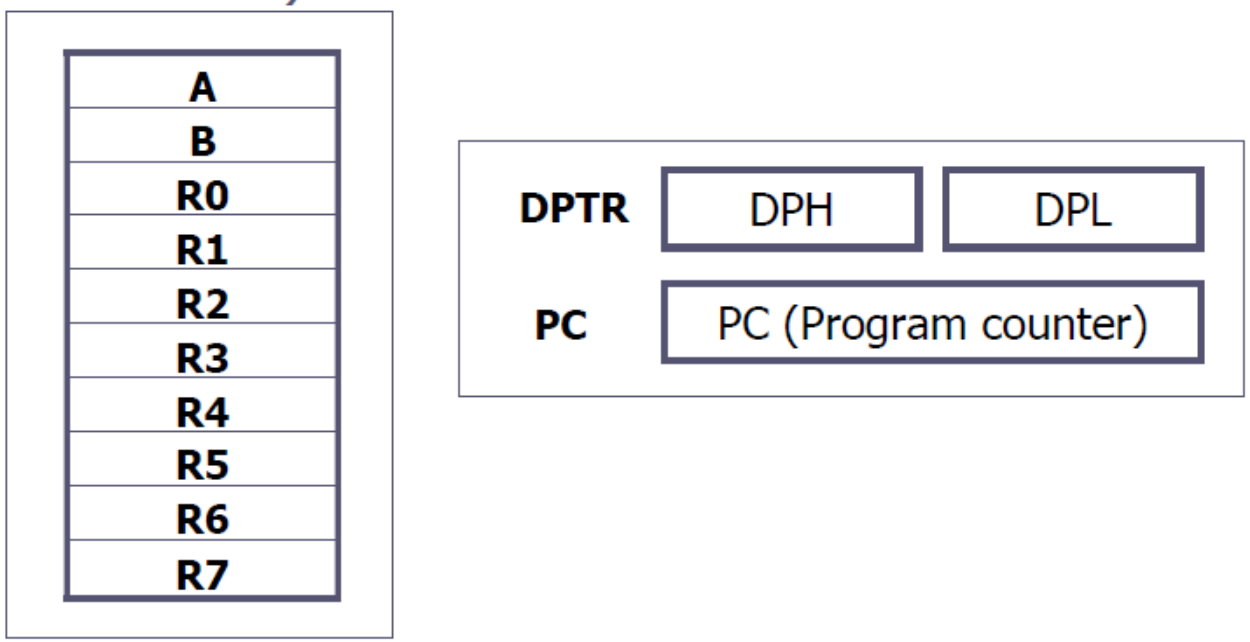
- Register are used to store information temporarily, while the information could be
  - a byte of data to be processed, or
  - an address pointing to the data to be fetched
- The vast majority of 8051 register are 8-bit registers
  - There is only one data type, 8 bits

- The 8 bits of a register are shown from MSB D7 to the LSB D0
  - With an 8-bit data type, any data larger than 8 bits must be broken into 8-bit chunks before it is processed



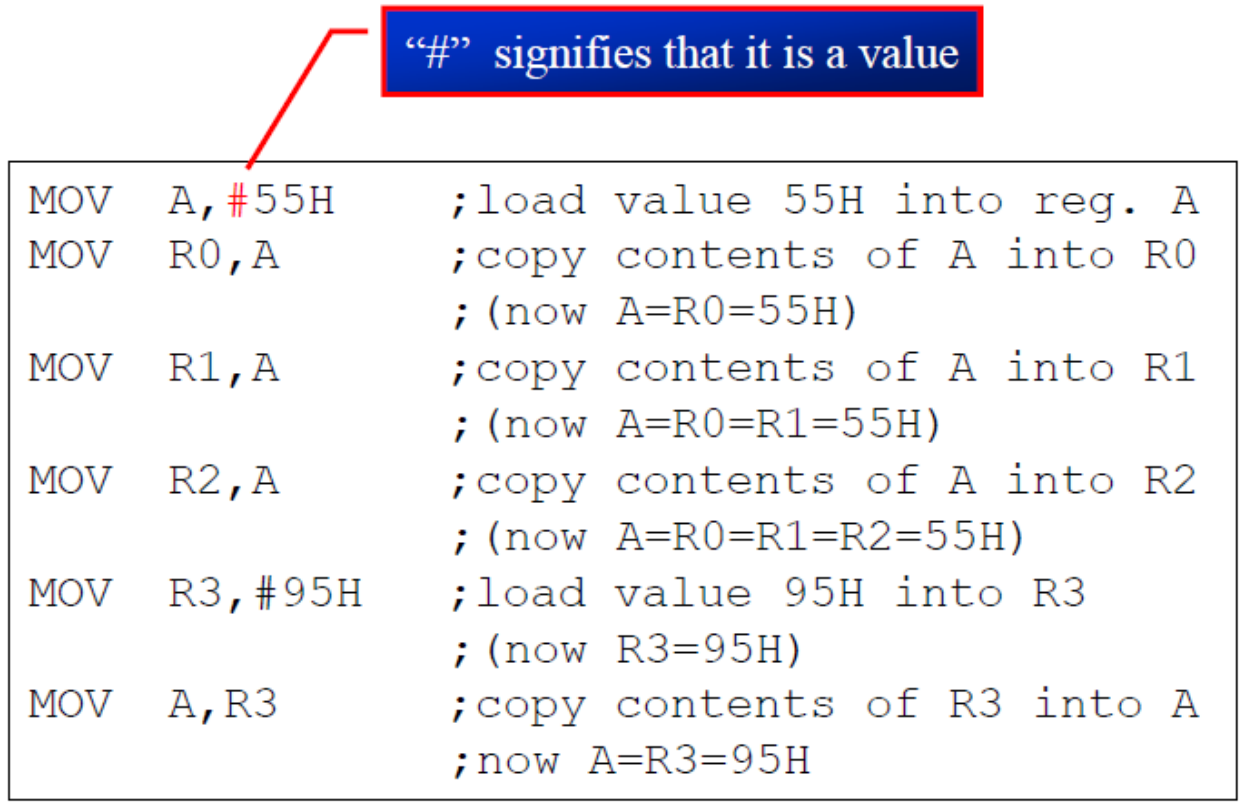
8 bit Registers

- The most widely used registers
  - A (Accumulator)
    - For all arithmetic and logic instructions
  - B, R0, R1, R2, R3, R4, R5, R6, R7
  - DPTR (data pointer), and PC (program counter)



MOV destination, source ;copy source to dest.

- The instruction tells the CPU to move (in reality, COPY) the source operand to the destination operand



- Notes on programming
  - Value (proceeded with #) can be loaded directly to registers A, B, or R0 – R7
    - `MOV A, #23H`
    - `MOV R5, #0F9H`

Add a 0 to indicate that F is a hex number and not a letter

If it's not preceded with #, it means to load from a memory location

  - If values 0 to F moved into an 8-bit register, the rest of the bits are assumed all zeros
    - "`MOV A, #5`", the result will be A=05; i.e., A = 00000101 in binary
  - Moving a value that is too large into a register will cause an error
    - `MOV A, #7F2H` ; ILLEGAL: 7F2H>8 bits (FFH)

INSIDE THE 8051

ADD Instruction

```
ADD A, source     ;ADD the source operand
                  ;to the  accumulator
```

- The ADD instruction tells the CPU to add the source byte to register A and put the result in register A
- Source operand can be either a register or immediate data, but the destination must always be register A
  - "ADD R4, A" and "ADD R2, #12H" are invalid since A must be the destination of any arithmetic operation

```
MOV A, #25H       ;load 25H into A
MOV R2, #34H      ;load 34H into R2
ADD A, R2 ;add R2 to Accumulator
                  ;(A = A + R2)
```

There are always many ways to write the same program, depending on the registers used

```
MOV A, #25H       ;load one operand
          ;into A (A=25H)
ADD A, #34H       ;add the second
                  ;operand 34H to A
```

- In the early days of the computer, programmers coded in *machine language*, consisting of 0s and 1s
  - Tedious, slow and prone to error
- *Assembly languages*, which provided mnemonics for the machine code instructions, plus other features, were developed
  - An Assembly language program consist of a series of lines of Assembly language instructions
- Assembly language is referred to as a *low-level language*
  - It deals directly with the internal structure of the CPU

- Assembly language instruction includes
  - a mnemonic (abbreviation easy to remember)
    - the commands to the CPU, telling it what those to do with those items
  - optionally followed by one or two operands
    - the data items being manipulated
- A given Assembly language program is a series of statements, or lines
  - Assembly language instructions
    - Tell the CPU what to do
  - Directives (or pseudo-instructions)
    - Give directions to the assembler

8051 ASSEMBLY PROGRAMMING

Structure of Assembly Language

- An Assembly language instruction consists of four fields:

```
[label:] Mnemonic [operands] [;comment]
```

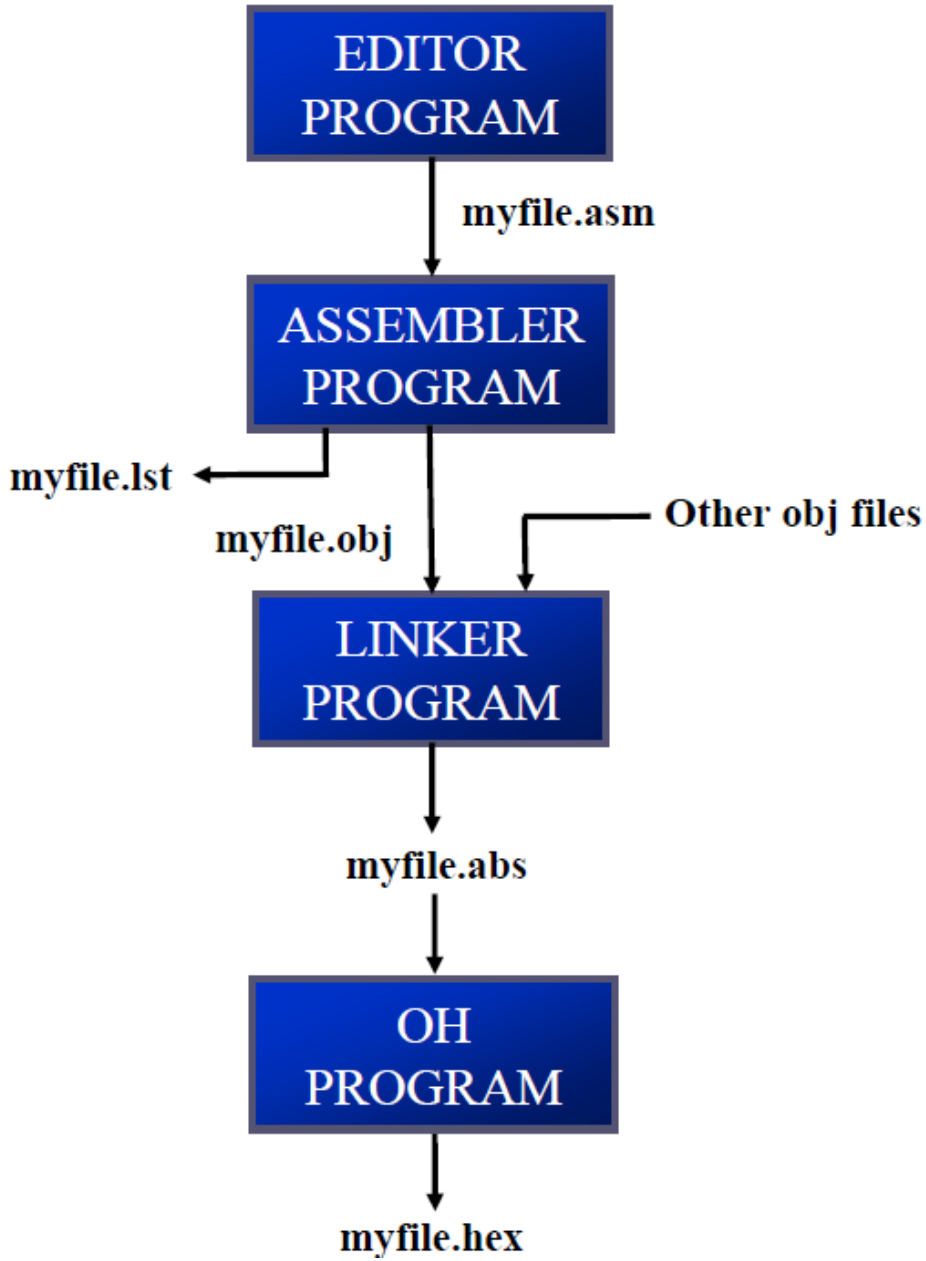
```
   ORG   0H              ;start(origin) at location
   0
   MOV   R5, #25H        ;load 25H into R5
   MOV   R7, #34H        ;load 34H in
   MOV   A, #0           ;load 0 into
   ADD   A, R5           ;add content
                         ;now A = A +
   ADD   A, R7           ;add contents of R7 to A
                         ;now A = A + R7
   ADD   A, #12H         ;add to A value 12H
                         ;now A = A + 12H
HERE: SJMP HERE          ;stay in this loop
   END                   ;en
```

Mnemonics produce opcodes

Directives do not generate any machine code and are used only by the assembler

The label field allows the program to refer to a line of code by name

Comments may be at the end of a line or on a line by themselves
The assembler ignores comments


EDITOR PROGRAM
myfile.asm
ASSEMBLER PROGRAM
myfile.lst
myfile.obj
Other obj files
LINKER PROGRAM
myfile.abs
OH PROGRAM
myfile.hex

- The lst (list) file, which is optional, is very useful to the programmer
  - It lists all the opcodes and addresses as well as errors that the assembler detected
  - The programmer uses the lst file to find the syntax errors or debug

```
1 0000          ORG 0H          ;start (origin) at 0
2 0000  7D25    MOV R5,#25H     ;load 25H into R5
3 0002  7F34    MOV R7,#34H     ;load 34H into R7
4 0004  7400    MOV A,#0        ;load 0 into A
5 0006  2D      ADD A,R5        ;add contents of R5 to A
                                ;now A = A + R5
6 0007  2F      ADD A,R7        ;add contents of R7 to A
                                ;now A = A + R7
7 0008  2412    ADD A,#12H      ;add to A value 12H
                                ;now A = A + 12H
8 000A  80EF HERE: SJMP HERE;stay in this loop
9 000C          END             ;end of asm source file
```

- The program counter points to the address of the next instruction to be executed
  - As the CPU fetches the opcode from the program ROM, the program counter is increasing to point to the next instruction
- The program counter is 16 bits wide
  - This means that it can access program addresses 0000 to FFFFH, a total of 64K bytes of code

- Examine the list file and how the code is placed in ROM

```
1 0000            ORG 0H              ;start (origin) at 0
2 0000  7D25      MOV R5,#25H         ;load 25H into R5
3 0002  7F34      MOV R7,#34H         ;load 34H into R7
4 0004  7400      MOV A,#0            ;load 0 into A
5 0006  2D        ADD A,R5            ;add contents of R5 to A
                                      ;now A = A + R5
6 0007  2F        ADD A,R7            ;add contents of R7 to A
                                      ;now A = A + R7
7 0008  2412      ADD A,#12H          ;add to A value 12H
                                      ;now A = A + 12H
8 000A  80EF      HERE: SJMP HERE     ;stay in this loop
9 000C            END                 ;end of asm source file
```

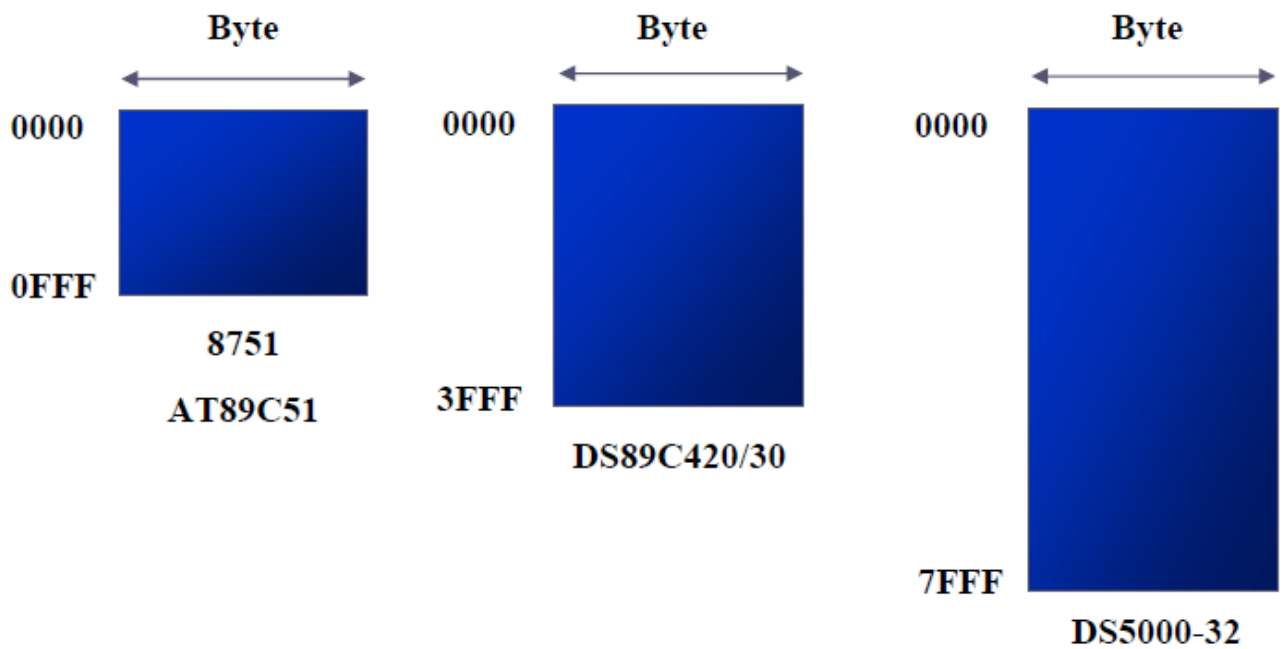
ROM Address	Machine Language	Assembly Language
0000	7D25	MOV R5, #25H
0002	7F34	MOV R7, #34H
0004	7400	MOV A, #0
0006	2D	ADD A, R5
0007	2F	ADD A, R7
0008	2412	ADD A, #12H
000A	80EF	HERE: SJMP HERE

- After the program is burned into ROM, the opcode and operand are placed in ROM memory location starting at 0000

ROM contents

Address	Code
0000	7D
0001	25
0002	7F
0003	34
0004	74
0005	00
0006	2D
0007	2F
0008	24
0009	12
000A	80
000B	FE

- No member of 8051 family can access more than 64K bytes of opcode
  - The program counter is a 16-bit register



AT89C51 – 4 KB
DS89C420 – 16KB
DS5000 – 32 KB

8051 DATA TYPES AND DIRECTIVES

Assembler Directives

- The DB directive is the most widely used data directive in the assembler
  - It is used to define the 8-bit data
  - When DB is used to define data, the numbers can be in decimal, binary, hex, ASCII formats

```
        ORG     500H
DATA1:  DB      28              ;DECIMAL (1C in Hex)
DATA2:  DB      00110101B       ;BINARY (35 in Hex)
DATA3:  DB      39H             ;HEX
        ORG     510H
DATA4:  DB      "2591"
        ORG     518H
DATA6:  DB      "My name is Joe"
                                ;ASCII CHARACTERS
```

The "D" after the decimal number is optional, but using "B" (binary) and "H" (hexadecimal) for the others is required

The Assembler will convert the numbers into hex

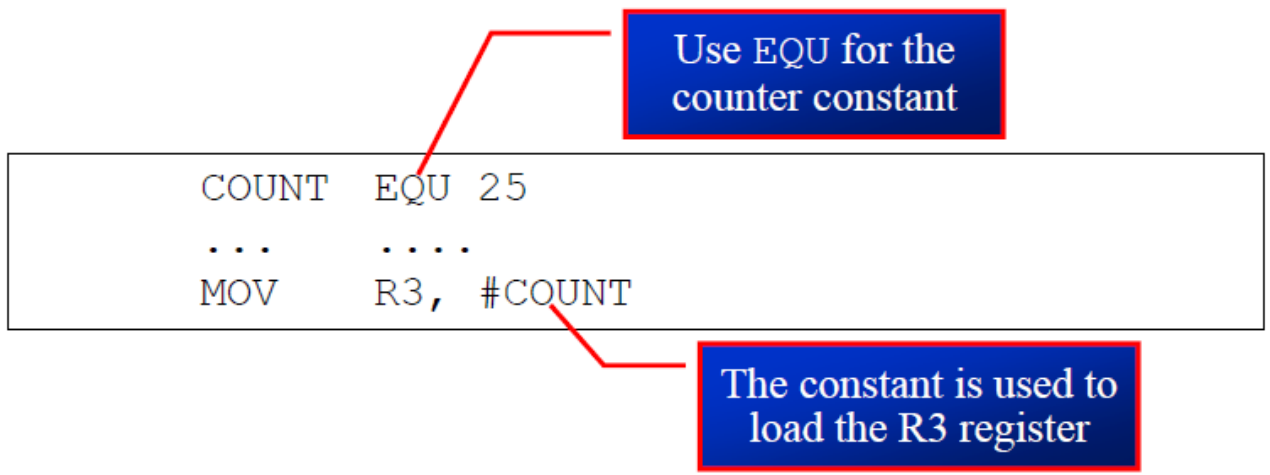
Place ASCII in quotation marks The Assembler will assign ASCII code for the numbers or characters

Define ASCII strings larger than two characters

- ORG (origin)
  - The ORG directive is used to indicate the beginning of the address
  - The number that comes after ORG can be either in hex and decimal
    - If the number is not followed by H, it is decimal and the assembler will convert it to hex
- END
  - This indicates to the assembler the end of the source (asm) file
  - The END directive is the last line of an 8051 program
    - Mean that in the code anything after the END directive is ignored by the assembler

- EQU (equate)
  - This is used to define a constant without occupying a memory location
  - The EQU directive does not set aside storage for a data item but associates a constant value with a data label
    - When the label appears in the program, its constant value will be substituted for the label

- EQU (equate) (cont')
  - Assume that there is a constant used in many different places in the program, and the programmer wants to change its value throughout
    - By the use of EQU, one can change it once and the assembler will change all of its occurrences



FLAG BITS AND PSW REGISTER

Program Status Word (cont')

CY	AC	F0	RS1	RS0	OV	--	P

A carry from D3 to D4

Carry out from the d7 bit

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

The result of signed number operation is too large, causing the high-order bit to overflow into the sign bit

Reflect the number of 1s in register A

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

Instructions that affect flag bits

Instruction	CY	OV	AC
ADD	X	X	X
ADDC	X	X	X
SUBB	X	X	X
MUL	0	X	
DIV	0	X	
DA	X		
RPC	X		
PLC	X		
SETB C	1		
CLR C	0		
CPL C	X		
ANL C, bit	X		
ANL C, /bit	X		
ORL C, bit	X		
ORL C, /bit	X		
MOV C, bit	X		
CJNE	X		

- The flag bits affected by the ADD instruction are CY, P, AC, and OV

Example 2-2

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

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

Solution:

38	00111000
+2F	00101111
67	01100111

CY = 0 since there is no carry beyond the D7 bit

AC = 1 since there is a carry from the D3 to the D4 bi

P = 1 since the accumulator has an odd number of 1s (it has five 1s)

Example 2-3

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

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

Solution:

9C	10011100
+ 64	01100100
100	00000000

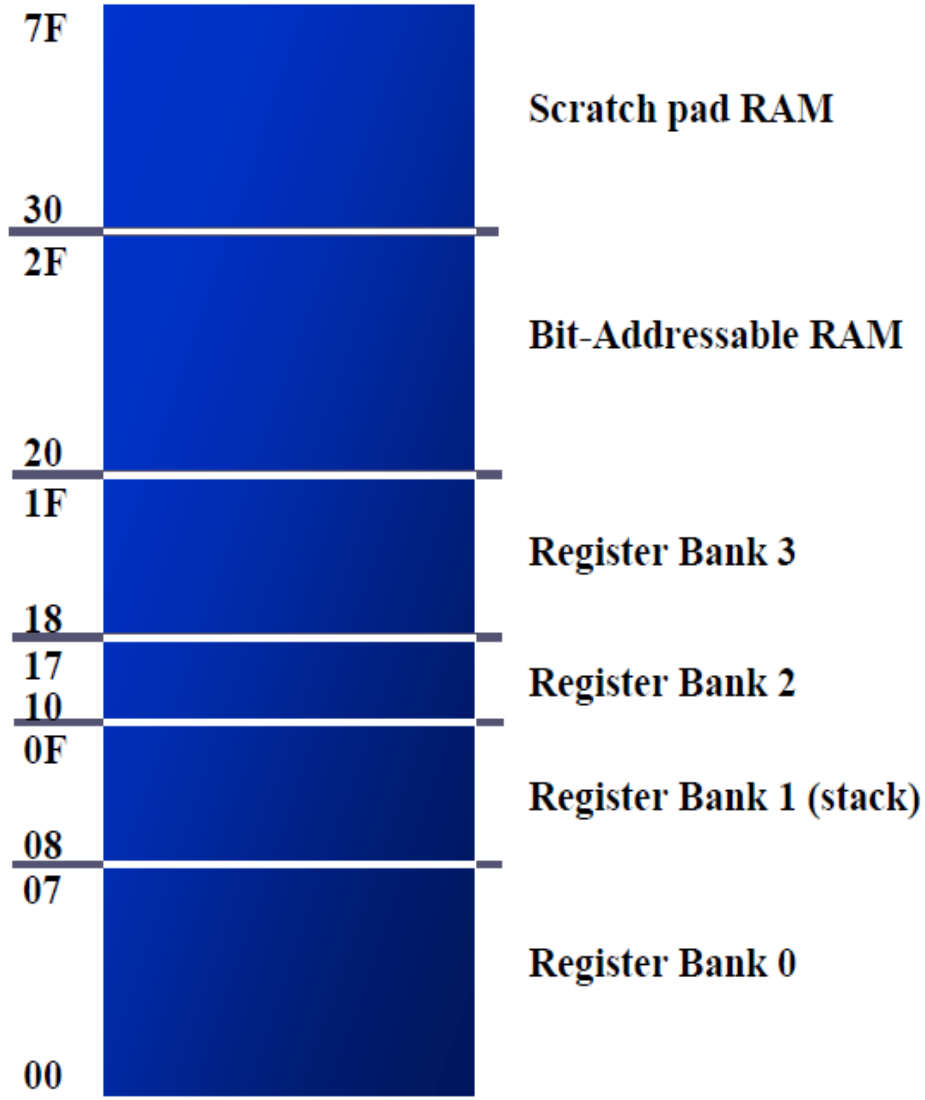
CY = 1 since there is a carry beyond the D7 bit

AC = 1 since there is a carry from the D3 to the D4 bi

P = 0 since the accumulator has an even number of 1s (it has zero 1s)

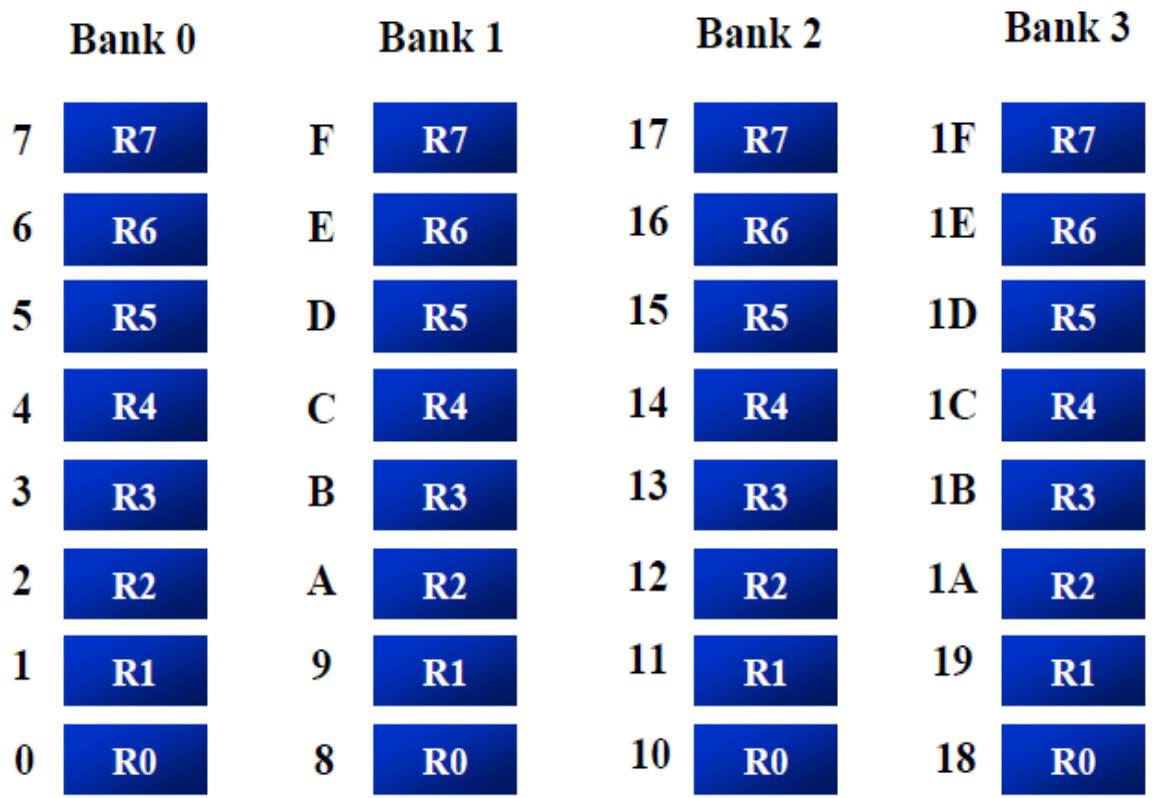
- There are 128 bytes of RAM in the 8051
  - Assigned addresses 00 to 7FH
- The 128 bytes are divided into three different groups as follows:
  1) A total of 32 bytes from locations 00 to 1F hex are set aside for register banks and the stack
  2) A total of 16 bytes from locations 20H to 2FH are set aside for bit-addressable read/write memory
  3) A total of 80 bytes from locations 30H to 7FH are used for read and write storage, called *scratch pad*

RAM Allocation in 8051



- These 32 bytes are divided into 4 banks of registers in which each bank has 8 registers, R0-R7
  - RAM location from 0 to 7 are set aside for bank 0 of R0-R7 where R0 is RAM location 0, R1 is RAM location 1, R2 is RAM location 2, and so on, until memory location 7 which belongs to R7 of bank 0
  - It is much easier to refer to these RAM locations with names such as R0, R1, and so on, than by their memory locations
- Register bank 0 is the default when 8051 is powered up

Register banks and their RAM address



- We can switch to other banks by use of the PSW register
  - Bits D4 and D3 of the PSW are used to select the desired register bank
  - Use the bit-addressable instructions SETB and CLR to access PSW.4 and PSW.3

PSW bank selection

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

Example 2-5

```
MOV R0, #99H      ;load R0 with 99H
MOV R1, #85H      ;load R1 with 85H
```

Example 2-6

```
MOV 00, #99H      ;RAM location 00H has 99H
MOV 01, #85H      ;RAM location 01H has 85H
```

Example 2-7

```
SETB PSW.4        ;select bank 2
MOV R0, #99H      ;RAM location 10H has 99H
MOV R1, #85H      ;RAM location 11H has 85H
```

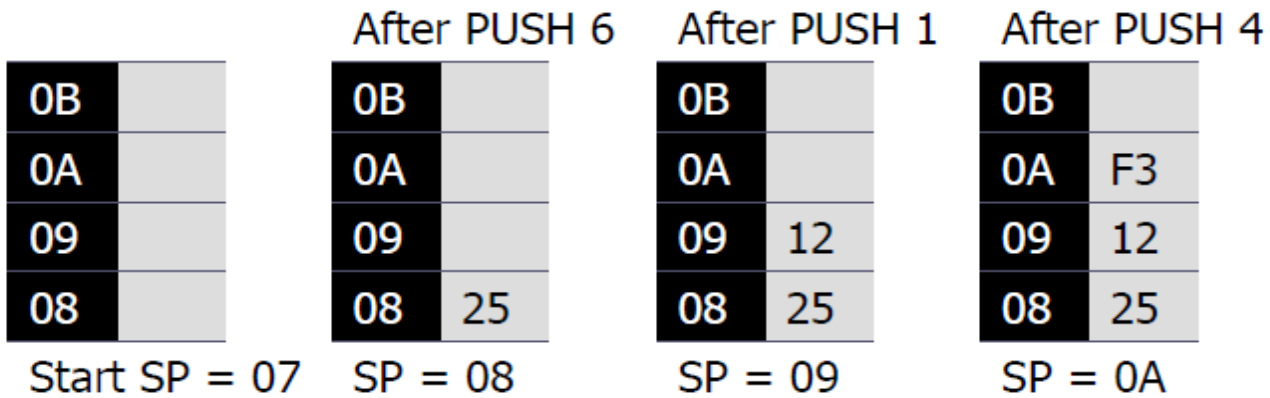
- The stack is a section of RAM used by the CPU to store information temporarily
  - This information could be data or an address
- The register used to access the stack is called the SP (stack pointer) register
  - The stack pointer in the 8051 is only 8 bit wide, which means that it can take value of 00 to FFH
  - When the 8051 is powered up, the SP register contains value 07
    - RAM location 08 is the first location begin used for the stack by the 8051

Example 2-8

Show the stack and stack pointer from the following. Assume the default stack area.

```
MOV R6, #25H
MOV R1, #12H
MOV R4, #0F3H
PUSH  6
PUSH  1
PUSH  4
```

Solution:

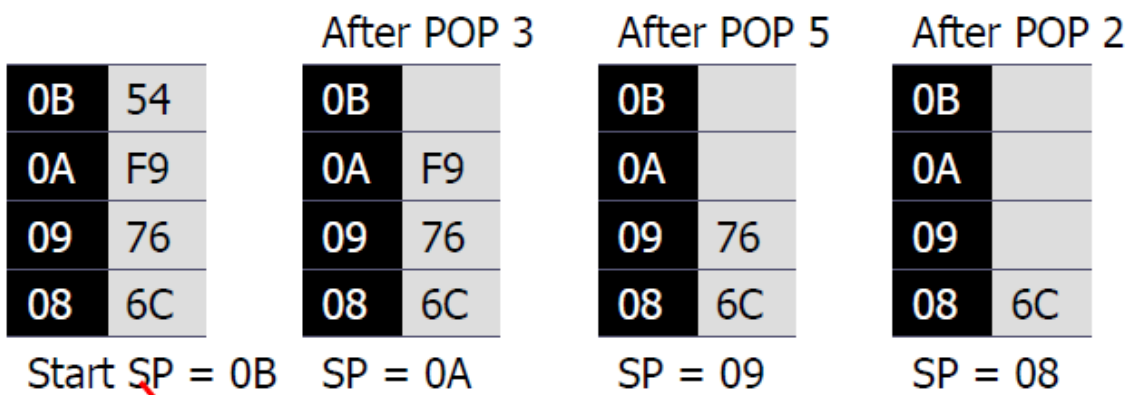


Example 2-9

Examining the stack, show the contents of the register and SP after execution of the following instructions. All value are in hex.

```
POP     3          ; POP stack into R3
POP     5          ; POP stack into R5
POP     2          ; POP stack into R2
```

Solution:



Because locations 20-2FH of RAM are reserved for bit-addressable memory, so we can change the SP to other RAM location by using the instruction "MOV SP, #XX"

- The CPU also uses the stack to save the address of the instruction just below the `CALL` instruction
  - This is how the CPU knows where to resume when it returns from the called subroutine

LOOP AND JUMP INSTRUCTIONS

Looping

A loop can be repeated a maximum of 255 times, if R2 is FFH

- Repeating a sequence of instructions a certain number of times is called a *loop*
  - Loop action is performed by

    `DJNZ reg, Label`
    - The register is decremented
    - If it is not zero, it jumps to the target address referred to by the label
    - Prior to the start of loop the register is loaded with the counter for the number of repetitions
    - Counter can be R0 – R7 or RAM location

```
;This program adds value 3 to the ACC ten times
        MOV  A,#0      ;A=0, clear ACC
        MOV  R2,#10    ;load counter R2=10
AGAIN:  ADD  A,#03     ;add 03 to ACC
        DJNZ R2,AGAIN  ;repeat until R2=0,10 times
        MOV  R5,A      ;save A in R5
```

- If we want to repeat an action more times than 256, we use a loop inside a loop, which is called *nested loop*
  - We use multiple registers to hold the count

Write a program to (a) load the accumulator with the value 55H, and (b) complement the ACC 700 times

```
        MOV   A,#55H    ;A=55H
        MOV   R3,#10    ;R3=10, outer loop count
NEXT:   MOV   R2,#70    ;R2=70, inner loop count
AGAIN:  CPL   A         ;complement A register
        DJNZ  R2,AGAIN  ;repeat it 70 times
        DJNZ  R3,NEXT
```

- Jump only if a certain condition is met

JZ label ;jump if A=0

```
        MOV   A,R0      ;A=R0
        JZ    OVER      ;jump if A = 0
        MOV   A,R1      ;A=R1
        JZ    OVER      ;jump if A = 0
        ...
OVER:
```

Can be used only for register A, not any other register

Determine if R5 contains the value 0. If so, put 55H in it.

```
        MOV   A,R5      ;copy R5 to A
        JNZ   NEXT      ;jump if A is not zero
        MOV   R5,#55H
NEXT:   ...
```

JNC label ;jump if no carry, CY=0

- If CY = 0, the CPU starts to fetch and execute instruction from the address of the label
- If CY = 1, it will not jump but will execute the next instruction below JNC

Find the sum of the values 79H, F5H, E2H. Put the sum in registers R0 (low byte) and R5 (high byte).

```
        MOV   A,#0       ;A=0
        MOV   R5,A       ;clear R5
        ADD   A,#79H     ;A=0+79H=79H
;       JNC   N_1        ;if CY=0, add next number
;       INC   R5         ;if CY=1, increment R5
N_1:    ADD   A,#0F5H    ;A=79+F5=6E and CY=1
        JNC   N_2        ;jump if CY=0
        INC   R5        ;if CY=1,increment R5 (R5=1)
N_2:    ADD   A,#0E2H    ;A=6E+E2=50 and CY=1
        JNC   OVER       ;jump if CY=0
        INC   R5         ;if CY=1, increment 5
OVER:   MOV   R0,A       ;now R0=50H, and R5=02
```

MOV R5,#0

8051 conditional jump instructions

Instructions	Actions
JZ	Jump if A = 0
JNZ	Jump if A ≠ 0
DJNZ	Decrement and Jump if A ≠ 0
CJNE A,byte	Jump if A ≠ byte
CJNE reg,#data	Jump if byte ≠ #data
JC	Jump if CY = 1
JNC	Jump if CY = 0
JB	Jump if bit = 1
JNB	Jump if bit = 0
JBC	Jump if bit = 1 and clear bit

- All conditional jumps are short jumps
  - The address of the target must within -128 to +127 bytes of the contents of PC

- The unconditional jump is a jump in which control is transferred unconditionally to the target location

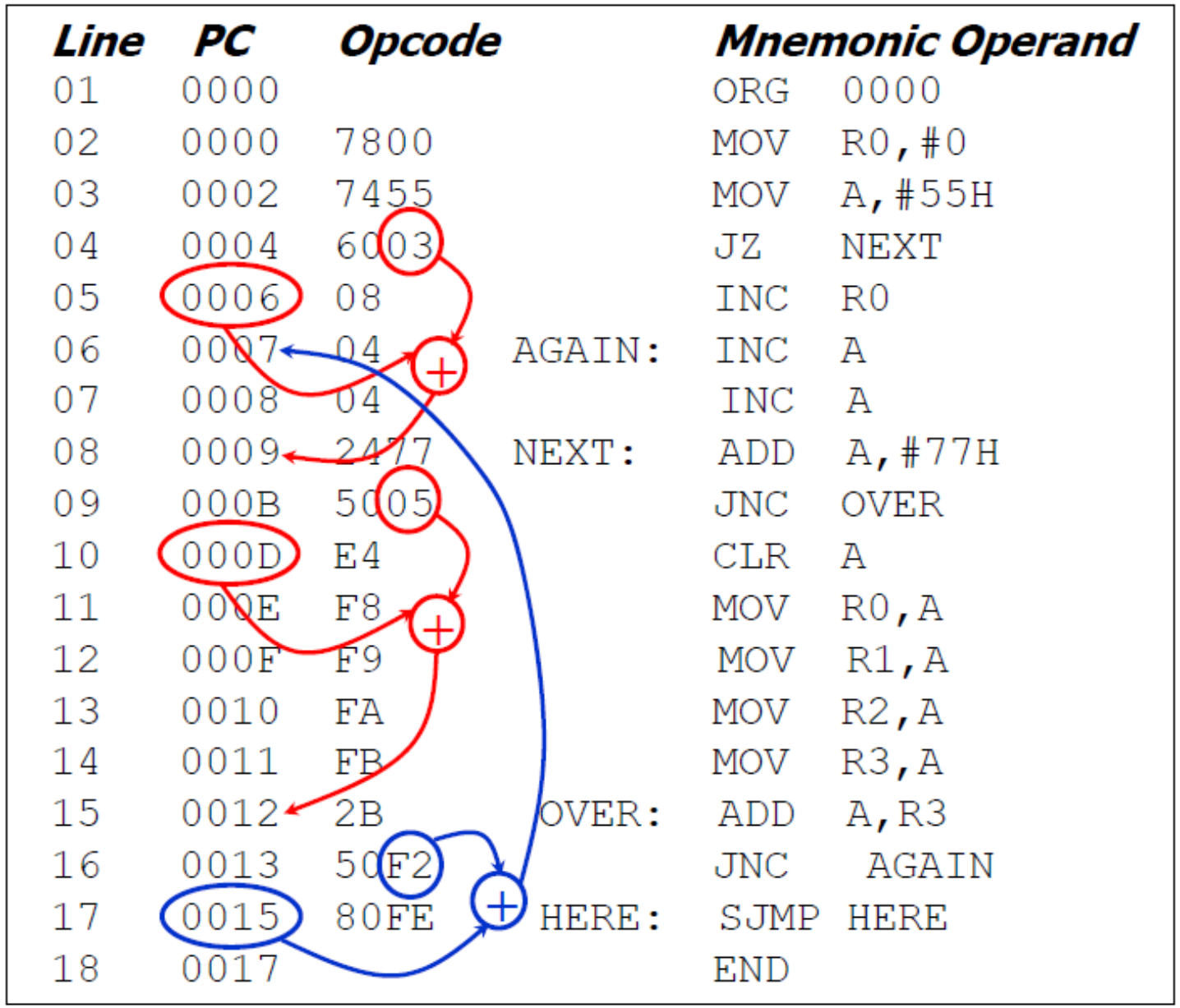
`LJMP` (long jump)

- 3-byte instruction
  - First byte is the opcode
  - Second and third bytes represent the 16-bit target address
    - Any memory location from 0000 to FFFFH

`SJMP` (short jump)

- 2-byte instruction
  - First byte is the opcode
  - Second byte is the relative target address
    - 00 to FFH (forward +127 and backward -128 bytes from the current PC)

- To calculate the target address of a short jump (`SJMP`, `JNC`, `JZ`, `DJNZ`, etc.)
  - The second byte is added to the PC of the instruction immediately below the jump
- If the target address is more than -128 to +127 bytes from the address below the short jump instruction
  - The assembler will generate an error stating the jump is out of range


Line PC Opcode Mnemonic Operand
01 0000 ORG 0000
02 0000 7800 MOV R0,#0
03 0002 7455 MOV A,#55H
04 0004 6003 JZ NEXT
05 0006 08 INC R0
06 0007 04 AGAIN: INC A
07 0008 04 INC A
08 0009 2477 NEXT: ADD A,#77H
09 000B 5005 JNC OVER
10 000D E4 CLR A
11 000E F8 MOV R0,A
12 000F F9 MOV R1,A
13 0010 FA MOV R2,A
14 0011 FB MOV R3,A
15 0012 2B OVER: ADD A,R3
16 0013 50F2 JNC AGAIN
17 0015 80FE HERE: SJMP HERE
18 0017 END

- Call instruction is used to call subroutine
  - Subroutines are often used to perform tasks that need to be performed frequently
  - This makes a program more structured in addition to saving memory space

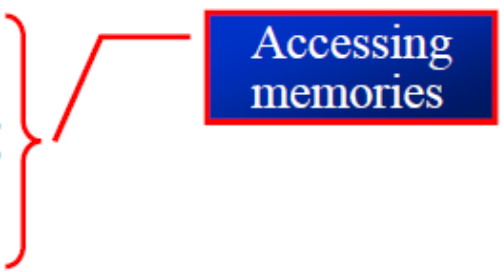
LCALL (long call)

  - 3-byte instruction
    - First byte is the opcode
    - Second and third bytes are used for address of target subroutine
      - Subroutine is located anywhere within 64K byte address space

ACALL (absolute call)

  - 2-byte instruction
    - 11 bits are used for address within 2K-byte range

- The CPU can access data in various ways, which are called *addressing modes*
  - Immediate
  - Register
  - Direct
  - Register indirect
  - Indexed



1. Immediate – Addressing Mode

- The source operand is a constant
  - The immediate data must be preceded by the pound sign, "#"
  - Can load information into any registers, including 16-bit DPTR register
    - DPTR can also be accessed as two 8-bit registers, the high byte DPH and low byte DPL

```
MOV A,#25H       ;load 25H into A
MOV R4,#62       ;load 62 into R4
MOV B,#40H       ;load 40H into B
MOV DPTR,#4521H ;DPTR=4512H
MOV DPL,#21H     ;This is the same
MOV DPH,#45H     ;as above

;illegal!! Value > 65535 (FFFFH)
MOV DPTR,#68975
```

- We can use EQU directive to access immediate data

```
          Count   EQU 30
          ...     ...
          MOV     R4,#COUNT          ;R4=1EH
          MOV     DPTR,#MYDATA       ;DPTR=200H

          ORG     200H
MYDATA:   DB      "America"
```

- We can also use immediate addressing mode to send data to 8051 ports

```
          MOV P1,#55H
```

2. Register – Addressing Mode

- Use registers to hold the data to be manipulated

```
MOV A,R0     ;copy contents of R0 into A
MOV R2,A     ;copy contents of A into R2
ADD A,R5     ;add contents of R5 to A
ADD A,R7     ;add contents of R7 to A
MOV R6,A     ;save accumulator in R6
```

- The source and destination registers must match in size
  - `MOV DPTR,A` will give an error

```
MOV DPTR,#25F5H
MOV R7,DPL
MOV R6,DPH
```

- The movement of data between Rn registers is not allowed
  - `MOV R4,R7` is invalid

3. Direct – Addressing Mode (Memory)

- It is most often used the direct addressing mode to access RAM locations 30 – 7FH
  - The entire 128 bytes of RAM can be accessed
  - The register bank locations are accessed by the register names

Direct addressing mode

```
MOV A,4     ;is same as
MOV A,R4    ;which means copy R4 into A
```

- Contrast this with immediate addressing mode
  - There is no "#" sign in the operand

Register addressing mode

```
MOV R0,40H  ;save content of 40H in R0
MOV 56H,A   ;save content of A in 56H
```

- The SFR (*Special Function Register*) can be accessed by their names or by their addresses

```
MOV 0E0H,#55H    ;is the same as
MOV A,#55h       ;load 55H into A

MOV 0F0H,R0      ;is the same as
MOV B,R0         ;copy R0 into B
```

- The SFR registers have addresses between 80H and FFH
  - Not all the address space of 80 to FF is used by SFR
  - The unused locations 80H to FFH are reserved and must not be used by the 8051 programmer

Special Function Register (SFR) Addresses

Symbol	Name	Address
ACC*	Accumulator	0E0H
B*	B register	0F0H
PSW*	Program status word	0D0H
SP	Stack pointer	81H
DPTR	Data pointer 2 bytes	
DPL	Low byte	82H
DPH	High byte	83H
P0*	Port 0	80H
P1*	Port 1	90H
P2*	Port 2	0A0H
P3*	Port 3	0B0H
IP*	Interrupt priority control	0B8H
IE*	Interrupt enable control	0A8H
…	…	…

Special Function Register (SFR) Addresses

Symbol	Name	Address
TMOD	Timer/counter mode control	89H
TCON*	Timer/counter control	88H
T2CON*	Timer/counter 2 control	0C8H
T2MOD	Timer/counter mode control	OC9H
TH0	Timer/counter 0 high byte	8CH
TL0	Timer/counter 0 low byte	8AH
TH1	Timer/counter 1 high byte	8DH
TL1	Timer/counter 1 low byte	8BH
TH2	Timer/counter 2 high byte	0CDH
TL2	Timer/counter 2 low byte	0CCH
RCAP2H	T/C 2 capture register high byte	0CBH
RCAP2L	T/C 2 capture register low byte	0CAH
SCON*	Serial control	98H
SBUF	Serial data buffer	99H
PCON	Power ontrol	87H

*** Bit addressable**

- Only direct addressing mode is allowed for pushing or popping the stack
  - `PUSH A` is invalid
  - Pushing the accumulator onto the stack must be coded as `PUSH 0E0H`

Example 5-2

Show the code to push R5 and A onto the stack and then pop them back them into R2 and B, where B = A and R2 = R5

Solution:

```
PUSH 05        ;push R5 onto stack
PUSH 0E0H      ;push register A onto stack
POP  0F0H      ;pop top of stack into B
               ;now register B = register A
POP  02        ;pop top of stack into R2
               ;now R2=R6
```

4. Register Indirect – Addressing Mode (Memory)

- A register is used as a pointer to the data
  - Only register R0 and R1 are used for this purpose
  - R2 – R7 cannot be used to hold the address of an operand located in RAM
- When R0 and R1 hold the addresses of RAM locations, they must be preceded by the "@" sign

```
MOV A,@R0   ;move contents of RAM whose
            ;address is held by R0 into A
MOV @R1,B   ;move contents of B into RAM
            ;whose address is held by R1
```

Example 5-3

Write a program to copy the value 55H into RAM memory locations 40H to 41H using
(a) direct addressing mode, (b) register indirect addressing mode without a loop, and (c) with a loop

Example 5-3

Write a program to copy the value 55H into RAM memory locations 40H to 41H using
(a) direct addressing mode, (b) register indirect addressing mode without a loop, and (c) with a loop

Solution:

```
(a)
    MOV A,#55H  ;load A with value 55H
    MOV 40H,A   ;copy A to RAM location 40H
    MOV 41H.A   ;copy A to RAM location 41H
(b)
    MOV A,#55H  ;load A with value 55H
    MOV R0,#40H ;load the pointer. R0=40H
    MOV @R0,A   ;copy A to RAM R0 points to
    INC R0      ;increment pointer. Now R0=41h
    MOV @R0,A   ;copy A to RAM R0 points to
(c)
       MOV A,#55H    ;A=55H
       MOV R0,#40H   ;load pointer.R0=40H,
       MOV R2,#02    ;load counter, R2=3
AGAIN: MOV @R0,A     ;copy 55 to RAM R0 points to
       INC R0        ;increment R0 pointer
       DJNZ R2,AGAIN ;loop until counter = zero
```

- The advantage is that it makes accessing data dynamic rather than static as in direct addressing mode
  - Looping is not possible in direct addressing mode

Example 5-4

Write a program to clear 16 RAM locations starting at RAM address 60H

Solution:

```
        CLR A           ;A=0
        MOV R1,#60H     ;load pointer. R1=60H
        MOV R7,#16      ;load counter, R7=16
AGAIN:  MOV @R1,A       ;clear RAM R1 points to
        INC R1          ;increment R1 pointer
        DJNZ R7,AGAIN   ;loop until counter=zero
```

Example 5-5

Write a program to copy a block of 10 bytes of data from 35H to 60H

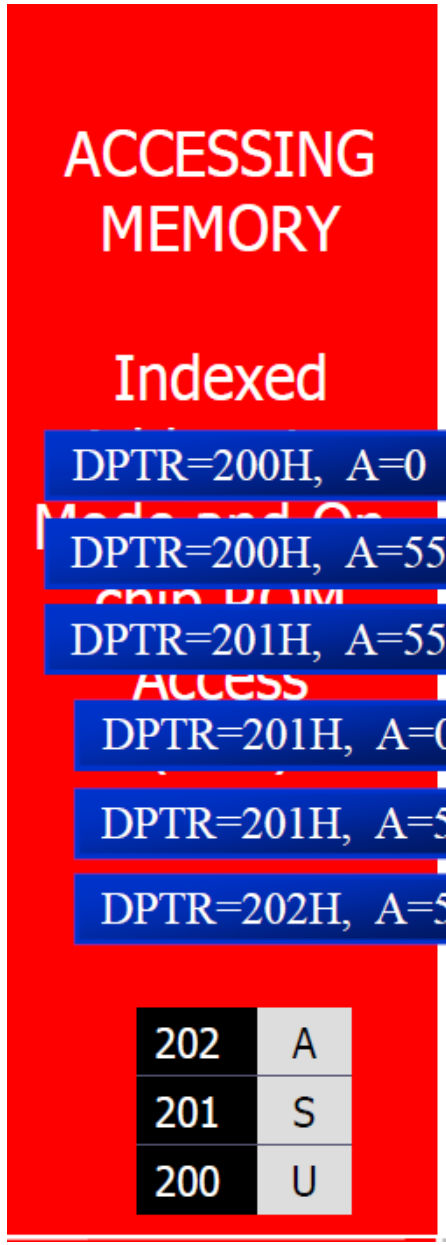
Example 5-5

Write a program to copy a block of 10 bytes of data from 35H to 60H

Solution:

```
      MOV R0,#35H   ;source pointer
      MOV R1,#60H   ;destination pointer
      MOV R3,#10    ;counter
BACK: MOV A,@R0     ;get a byte from source
      MOV @R1,A     ;copy it to destination
      INC R0        ;increment source pointer
      INC R1      ;increment destination pointer
      DJNZ R3,BACK ;keep doing for ten bytes
```

5. Indexed– Addressing Mode (Memory)



Example 5-6

In this program, assume that the word "USA" is burned into ROM locations starting at 200H. And that the program is burned into ROM locations starting at 0. Analyze how the program works and state where "USA" is stored after this program is run.

Solution:

```
        ORG   0000H        ;burn into ROM starting at 0
        MOV   DPTR,#200H   ;DPTR=200H look-up table addr
        CLR   A            ;clear A(A=0)
        MOVC  A,@A+DPTR    ;get the char from code space
        MOV   R0,A         ;save it in R0
        INC   DPTR         ;DPTR=201 point to next char
        CLR   A            ;clear A(A=0)
        MOVC  A,@A+DPTR    ;get the next char
        MOV   R1,A         ;save it in R1
        INC   DPTR         ;DPTR=202 point to next char
        CLR   A            ;clear A(A=0)
        MOVC  A,@A+DPTR    ;get the next char
        MOV   R2,A         ;save it in R2
Here:   SJMP  HERE         ;stay here
;Data is burned into code space starting at 200H

        ORG   200H
MYDATA:DB  "USA"
        END                ;end of program
```

R0=55H

R1=53H

R2=41H

ARITHMETIC & LOGIC INSTRUCTIONS AND PROGRAMS

- When adding two 16-bit data operands, the propagation of a carry from lower byte to higher byte is concerned

```
    1
   3C  E7
+  3B  8D
  ------
   78  74
```

When the first byte is added (E7+8D=74, CY=1).
The carry is propagated to the higher byte, which result in 3C + 3B + 1 =78 (all in hex)

Write a program to add two 16-bit numbers. Place the sum in R7 and R6; R6 should have the lower byte.

Solution:

```
        CLR   C          ;make CY=0
        MOV   A, #0E7H   ;load the low byte now A=E7H
        ADD   A, #8DH    ;add the low byte
        MOV   R6, A      ;save the low byte sum in R6
        MOV   A, #3CH    ;load the high byte
        ADDC  A, #3BH    ;add with the carry
        MOV   R7, A      ;save the high byte sum
```

```
SUBB A,source ;A = A - source - CY
```

- To make SUB out of SUBB, we have to make CY=0 prior to the execution of the instruction
  - Notice that we use the CY flag for the borrow

ARITHMETIC INSTRUCTIONS

Subtraction of Unsigned Numbers (cont')

- SUBB when CY = 0
  1. Take the 2's complement of the subtrahend (source operand)
  2. Add it to the minuend (A)
  3. Invert the carry

```
        CLR    C
        MOV    A,#4C   ;load A with value 4CH
        SUBB   A,#6EH  ;subtract 6E from A
        JNC    NEXT    ;if CY=0 jump to NEXT
        CPL    A  ;if CY=1, take 1's complement
        INC    A  ;and increment to get 2's comp
NEXT:   MOV    R1,A    ;save A in R1
```

Solution:

```
    4C      0100 1100        0100 1100
-   6E      0110 1110        1001 0010
   -22                      01101 1110
```

① 2's complement

② +

CY =1

③ Invert carry

CY=0, the result is positive; CY=1, the result is negative and the destination has the 2's complement of the result

2.5.2 Binary Subtraction

The rules of binary subtraction are given in Table 2.5.

Table 2.5 ***Rules of Binary Subtraction***

Minuend	Subtrahend	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Except in the second row above, the *borrow* = 0. When the *borrow* = 1, as in the second row, this is to be subtracted from the next higher binary bit as it is done in decimal subtraction.

- The 8051 supports byte by byte multiplication only
  - The byte are assumed to be unsigned data

```
MUL AB  ;AxB, 16-bit result in B, A
```

```
        MOV     A,#25H      ;load 25H to reg.  A
        MOV     B,#65H      ;load 65H to reg.  B
        MUL     AB          ;25H * 65H = E99 where
                            ;B = OEH and A = 99H
```

Unsigned Multiplication Summary (MUL AB)

Multiplication	Operand1	Operand2	Result
Byte x byte	A	B	B = high byte A = low byte

- The 8051 supports byte over byte division only
  - The byte are assumed to be unsigned data

```
DIV   AB    ;divide A by B, A/B
```

Unsigned Division Summary (DIV AB)

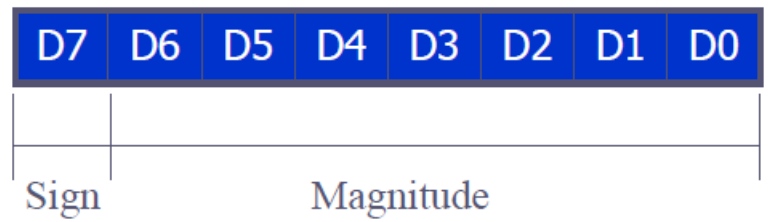
Division	Numerator	Denominator	Quotient	Remainder
Byte / byte	A	B	A	B

CY is always 0
If B ≠ 0, OV = 0
If B = 0, OV = 1 indicates error

Application – convert decimal to hex

SIGNED ARITHMETIC INSTRUCTIONS

- D7 (MSB) is the sign and D0 to D6 are the magnitude of the number
  - If D7=0, the operand is positive, and if D7=1, it is negative



- Positive numbers are 0 to +127
- Negative number representation (2's complement)
  1. Write the magnitude of the number in 8-bit binary (no sign)
  2. Invert each bit
  3. Add 1 to it

Show how the 8051 would represent -34H

Solution:

```
1.   0011 0100     34H given in binary
2.   1100 1011     invert each bit
3.   1100 1100     add 1 (which is CC in hex)
```

Signed number representation of -34 in 2's complement is CCH

Decimal	Binary	Hex
-128	1000 0000	80
-127	1000 0001	81
-126	1000 0010	82
…	… …	…
-2	1111 1110	FE
-1	1111 1111	FF
0	0000 0000	00
+1	0000 0001	01
+2	0000 0010	02
…	… …	…
+127	0111 1111	7F

- If the result of an operation on signed numbers is too large for the register
  - An overflow has occurred and the programmer must be noticed

Examine the following code and analyze the result.

```
MOV    A,#+96        ;A=0110 0000 (A=60H)
MOV    R1,#+70       ;R1=0100 0110(R1=46H)
ADD    A,R1          ;A=1010 0110
                     ;A=A6H=-90,INVALID
```

Solution:

```
   +96    0110 0000
+  +70    0100 0110
+  166    1010 0110 and OV =1
```

According to the CPU, the result is -90, which is wrong. The CPU sets OV=1 to indicate the overflow

```
MOV  A,#-2     ;A=1111 1110(A=FEH)
MOV  R1,#-5    ;R1=1111 1011(R1=FBH)
ADD  A,R1      ;A=1111 1001(A=F9H=-7,
               ;Correct, OV=0)
      -2            1111 1110
   +  -5            1111 1011
      -7            1111 1001 and OV=0
```

OV = 0
The result -7 is correct

```
MOV  A,#+7     ;A=0000 0111(A=07H)
MOV  R1,#+18   ;R1=0001 0010(R1=12H)
ADD  A,R1      ;A=0001 1001(A=19H=+25,
               ;Correct,OV=0)
       7            0000 0111
   +  18            0001 0010
      25            0001 1001 and OV=0
```

OV = 0
The result +25 is correct

- In 8-bit signed number operations, OV is set to 1 if either occurs:
  1. There is a carry from D6 to D7, but no carry out of D7 (CY=0)
  2. There is a carry from D7 out (CY=1), but no carry from D6 to D7

```
MOV  A,#-128   ;A=1000 0000(A=80H)
MOV  R4,#-2    ;R4=1111 1110(R4=FEH)
ADD  A,R4      ;A=0111 1110(A=7EH=+126,INVALID)
     -128           1000 0000
   +   -2           1111 1110
     -130           0111 1110 and OV=1
```

OV = 1
The result +126 is wrong

- In unsigned number addition, we must monitor the status of CY (carry)
  - Use JNC or JC instructions
- In signed number addition, the OV (overflow) flag must be monitored by the programmer
  - `JB PSW.2` or `JNB PSW.2`

LOGIC AND COMPARE INSTRUCTIONS

```
ANL destination,source
            ;dest = dest AND source
```

- This instruction will perform a logic AND on the two operands and place the result in the destination
  - The destination is normally the accumulator
  - The source operand can be a register, in memory, or immediate

Show the results of the following.

```
      MOV   A,#35H   ;A = 35H
      ANL   A,#0FH   ;A = A AND 0FH
35H     0 0 1 1 0 1 0 1
0FH     0 0 0 0 1 1 1 1
05H     0 0 0 0 0 1 0 1
```

ANL is often used to mask (set to 0) certain bits of an operand

```
ORL destination,source
          ;dest = dest OR source
```

- The destination and source operands are ORed and the result is placed in the destination
  - The destination is normally the accumulator
  - The source operand can be a register, in memory, or immediate

Show the results of the following.

```
     MOV   A,#04H   ;A = 04
     ORL   A,#68H   ;A = 6C
```

```
04H    0 0 0 0 0 1 0 0
68H    0 1 1 0 1 0 0 0
6CH    0 1 1 0 1 1 0 0
```

ORL instruction can be used to set certain bits of an operand to 1

```
XRL destination,source
          ;dest = dest XOR source
```

- This instruction will perform XOR operation on the two operands and place the result in the destination
  - The destination is normally the accumulator
  - The source operand can be a register, in memory, or immediate

Show the results of the following.

```
     MOV   A,#54H
     XRL   A,#78H
54H    0 1 0 1 0 1 0 0
78H    0 1 1 1 1 0 0 0
2CH    0 0 1 0 1 1 0 0
```

XRL instruction can be used to toggle certain bits of an operand

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

The XRL instruction can be used to clear the contents of a register by XORing it with itself. Show how `XRL A,A` clears A, assuming that AH = 45H.

```
45H     0 1 0 0 0 1 0 1
45H     0 1 0 0 0 1 0 1
-----------------------
00H     0 0 0 0 0 0 0 0
```

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

Solution:

```
        MOV P2,#00     ;clear P2
        MOV P1,#0FFH   ;make P1 an input port
        MOV R3,#45H    ;R3=45H
        MOV A,P1       ;read P1
        XRL A,R3
        JNZ EXIT       ;jump if A is not 0
        MOV P2,#99H
EXIT:   ...
```

XRL can be used to see if two registers have the same value

If both registers have the same value, 00 is placed in A. JNZ

```
CPL A ;complements the register A
```

- This is called 1's complement

```
        MOV A, #55H
        CPL A           ;now A=AAH
                        ;0101 0101(55H)
                        ;becomes 1010 1010(AAH)
```

- To get the 2's complement, all we have to do is to to add 1 to the 1's complement

CJNE destination,source,rel. addr.

- The actions of comparing and jumping are combined into a single instruction called CJNE (compare and jump if not equal)
  - The CJNE instruction compares two operands, and jumps if they are not equal
  - The destination operand can be in the accumulator or in one of the Rn registers
  - The source operand can be in a register, in memory, or immediate
    - The operands themselves remain unchanged
  - It changes the CY flag to indicate if the destination operand is larger or smaller

```
        CJNE R5,#80,NOT_EQUAL ;check R5 for 80
        ...                   ;R5 = 80
NOT_EQUAL:
        JNC  NEXT             ;jump if R5 > 80
        ...                   ;R5 < 80
NEXT:   ...
```

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

- Notice in the CJNE instruction that any Rn register can be compared with an immediate value
  - There is no need for register A to be involved

- The compare instruction is really a subtraction, except that the operands remain unchanged
  - Flags are changed according to the execution of the SUBB instruction

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

Solution:

```
        MOV  P1,#0FFH    ;make P1 an input port
        MOV  A,P1        ;read P1 port
        CJNE A,#75,OVER  ;jump if A is not 75
        SJMP EXIT        ;A=75, exit
OVER:   JNC  NEXT        ;if CY=0 then A>75
        MOV  R1,A        ;CY=1, A<75, save in R1
        SJMP EXIT        ; and exit
NEXT:   MOV  R2,A        ;A>75, save it in R2
EXIT:   ...
```

- Write a program to monitor P1 continuously for a value 63H. It should stop monitoring only if P1=63H.

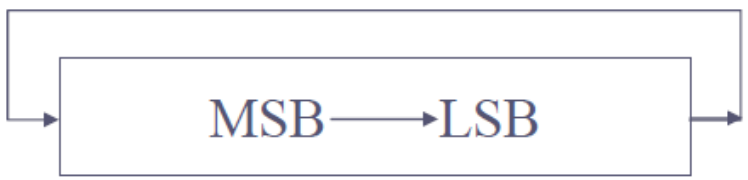
- Assume internal RAM memory locations 40H – 44H contains the daily temperature for 5 days as given below. Search to see if any value equals 65. If value 65 does exist in the table, give its location to R4, otherwise make R4=0.
  40H = (76), 41H = (79), 42H = (69), 43H = (65), 44H = (62)

ROTATE INSTRUCTION AND DATA SERIALIZATION

Rotating Right and Left

```
RR A        ;rotate right A
```

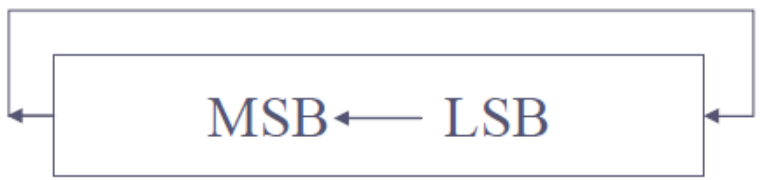
- In rotate right
  - The 8 bits of the accumulator are rotated right one bit, and
  - Bit D0 exits from the LSB and enters into MSB, D7




```
        MOV A,#36H    ;A = 0011 0110
        RR  A         ;A = 0001 1011
        RR  A         ;A = 1000 1101
        RR  A         ;A = 1100 0110
        RR  A         ;A = 0110 0011
```

```
RL A          ;rotate left A
```

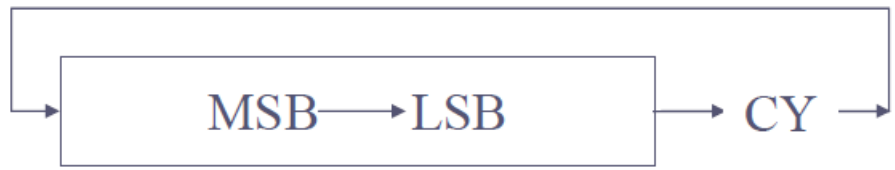
- In rotate left
  - The 8 bits of the accumulator are rotated left one bit, and
  - Bit D7 exits from the MSB and enters into LSB, D0




```
MOV A,#72H    ;A = 0111 0010
RL  A         ;A = 1110 0100
RL  A         ;A = 1100 1001
```

```
RRC A    ;rotate right through carry
```

- In RRC A
  - Bits are rotated from left to right
  - They exit the LSB to the carry flag, and the carry flag enters the MSB




```
        CLR C           ;make CY = 0
        MOV A,#26H      ;A = 0010 0110
        RRC A           ;A = 0001 0011     CY = 0
        RRC A           ;A = 0000 1001     CY = 1
        RRC A           ;A = 1000 0100     CY = 1
```

```
RLC A    ;rotate left through carry
```

- In RLC A
  - Bits are shifted from right to left
  - They exit the MSB and enter the carry flag, and the carry flag enters the LSB

Write a program that finds the number of 1s in a given byte.

```
        MOV     R1,#0
        MOV     R7,#8       ;count=08
        MOV     A,#97H
AGAIN:  RLC     A
        JNC     NEXT        ;check for CY
        INC     R1          ;if CY=1 add to count
NEXT:   DJNZ    R7,AGAIN
```

- There are several instructions by which the 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 b,C	Copy carry status to bit location (CY = b)
MOV C,b	Copy bit location status to carry (b = CY)
JNC target	Jump to target if CY = 0
JC target	Jump to target if CY = 1
ANL C,bit	AND CY with bit and save 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. Show how to turn on the outside light and turn off the inside one.

Solution:

```
        SETB    C           ;CY = 1
        ORL     C,P2.2      ;CY = P2.2 ORed w/ CY
        MOV     P2.2,C      ;turn it on if not on
        CLR     C           ;CY = 0
        ANL     C,P2.5      ;CY = P2.5 ANDed w/ CY
        MOV     P2.5,C     ;turn it off if not off
```

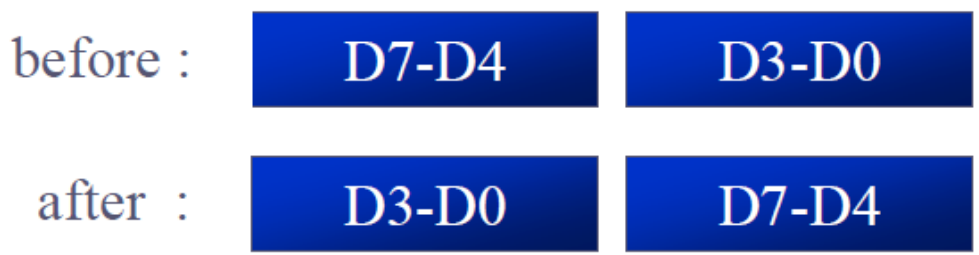
Write a program that finds the number of 1s in a given byte.

Solution:

```
        MOV    R1,#0      ;R1 keeps number of 1s
        MOV    R7,#8   ;counter, rotate 8 times
        MOV    A,#97H ;find number of 1s in 97H
AGAIN:  RLC    A          ;rotate it thru CY
        JNC    NEXT       ;check CY
        INC    R1         ;if CY=1, inc count
NEXT:   DJNZ   R7,AGAIN   ;go thru 8 times
```

`SWAP A`

- It swaps the lower nibble and the higher nibble
  - In other words, the lower 4 bits are put into the higher 4 bits and the higher 4 bits are put into the lower 4 bits
- SWAP works only on the accumulator (A)



BCD Number System

- The binary representation of the digits 0 to 9 is called BCD (Binary Coded Decimal)
  - Unpacked BCD
    - In unpacked BCD, the lower 4 bits of the number represent the BCD number, and the rest of the bits are 0
    - Ex. 00001001 and 00000101 are unpacked BCD for 9 and 5
  - Packed BCD
    - In packed BCD, a single byte has two BCD number in it, one in the lower 4 bits, and one in the upper 4 bits
    - Ex. 0101 1001 is packed BCD for 59H

Digit	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

- Adding two BCD numbers must give a BCD result

```
MOV     A, #17H
ADD     A, #28H
```

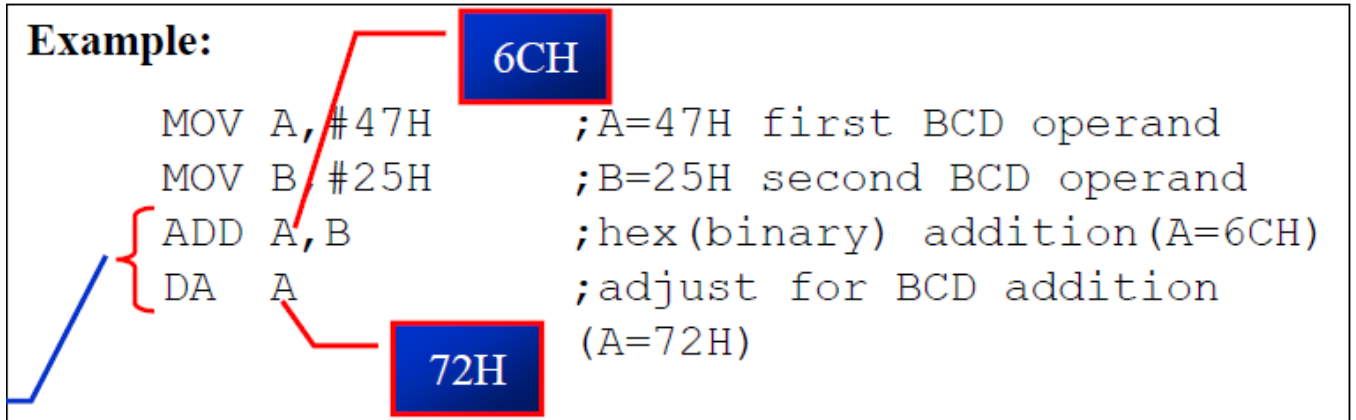
Adding these two numbers gives 0011 1111B (3FH), Which is not BCD!

The result above should have been 17 + 28 = 45 (0100 0101).
To correct this problem, the programmer must add 6 (0110) to the low digit: 3F + 06 = 45H.

DA A ;decimal adjust for addition

- The DA instruction is provided to correct the aforementioned problem associated with BCD addition
  - The DA instruction will add 6 to the lower nibble or higher nibble if need

Example:



The "DA" instruction works only on A. In other word, while the source can be an operand of any addressing mode, the destination must be in register A in order for DA to work.

Assume that 5 BCD data items are stored in RAM locations starting at 40H, as shown below. Write a program to find the sum of all the numbers. The result must be in BCD.

40=(71)
41=(11)
42=(65)
43=(59)
44=(37)

Assume that 5 BCD data items are stored in RAM locations starting at 40H, as shown below. Write a program to find the sum of all the numbers. The result must be in BCD.

40=(71)
41=(11)
42=(65)
43=(59)
44=(37)

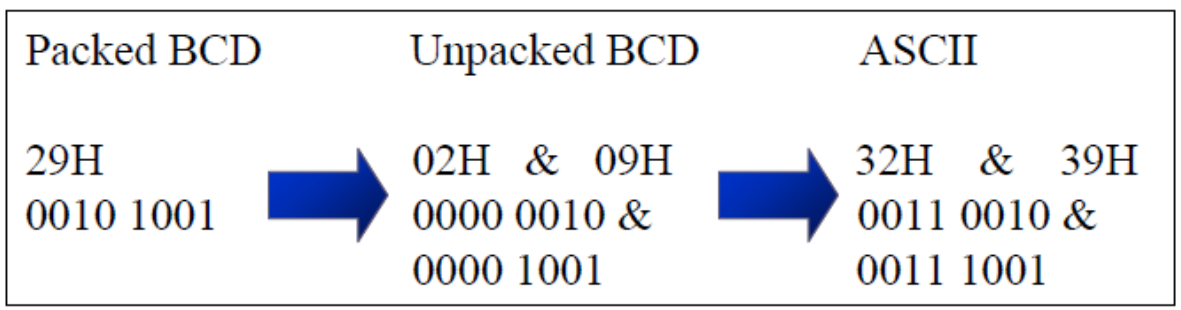
Solution:

```
        MOV     R0,#40H     ;Load pointer
        MOV     R2,#5       ;Load counter
        CLR     A           ;A=0
        MOV     R7,A        ;Clear R7
AGAIN:  ADD     A,@R0       ;add the byte pointer
                            ;to by R0
        DA      A           ;adjust for BCD
        JNC     NEXT        ;if CY=0 don't
                            ;accumulate carry
        INC     R7          ;keep track of carries
NEXT:   INC     R0          ;increment pointer
        DJNZ    R2,AGAIN    ;repeat until R2 is 0
```

ASCII code and BCD for digits 0 - 9

Key	ASCII (hex)	Binary	BCD (unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

- The DS5000T microcontrollers have a real-time clock (RTC)
  - The RTC provides the time of day (hour, minute, second) and the date (year, month, day) continuously, regardless of whether the power is on or off
- However this data is provided in packed BCD
  - To be displayed on an LCD or printed by the printer, it must be in ACSII format



Assume that register A has packed BCD, write a program to convert packed BCD to two ASCII numbers and place them in R2 and R6.

```
MOV    A,#29H   ;A=29H, packed BCD
MOV    R2,A     ;keep a copy of BCD data
ANL    A,#0FH   ;mask the upper nibble (A=09)
ORL    A,#30H   ;make it an ASCII, A=39H('9')
MOV    R6,A     ;save it
MOV    A,R2     ;A=29H, get the original
data
ANL    A,#0F0H ;mask the lower nibble
RR     A        ;rotate right
RR     A        ;rotate right
RR     A        ;rotate right
RR     A        ;rotate right
ORL    A,#30H   ;A=32H, ASCII char. '2'
MOV    R2,A     ;save ASCII char in R2
```

SWAP A

BINARY	DECIMAL	HEXADECIMAL	SYMBOL
010 0000	32	20	SPACE
010 0001	33	21	!
010 0010	34	22	“
010 0011	35	23	#
010 0100	36	24	$
010 0101	37	25	%
010 0110	38	26	&
010 0111	39	27	‘
010 1000	40	28	(
010 1001	41	29	)
010 1010	42	2A	*
010 1011	43	2B	+
010 1100	44	2C	,
010 1101	45	2D	-
010 1110	46	2E	.
010 1111	47	2F	/
011 0000	48	30	0
011 0001	49	31	1
011 0010	50	32	2
011 0011	51	33	3
011 0100	52	34	4
011 0101	53	35	5
011 0110	54	36	6
011 0111	55	37	7
011 1000	56	38	8

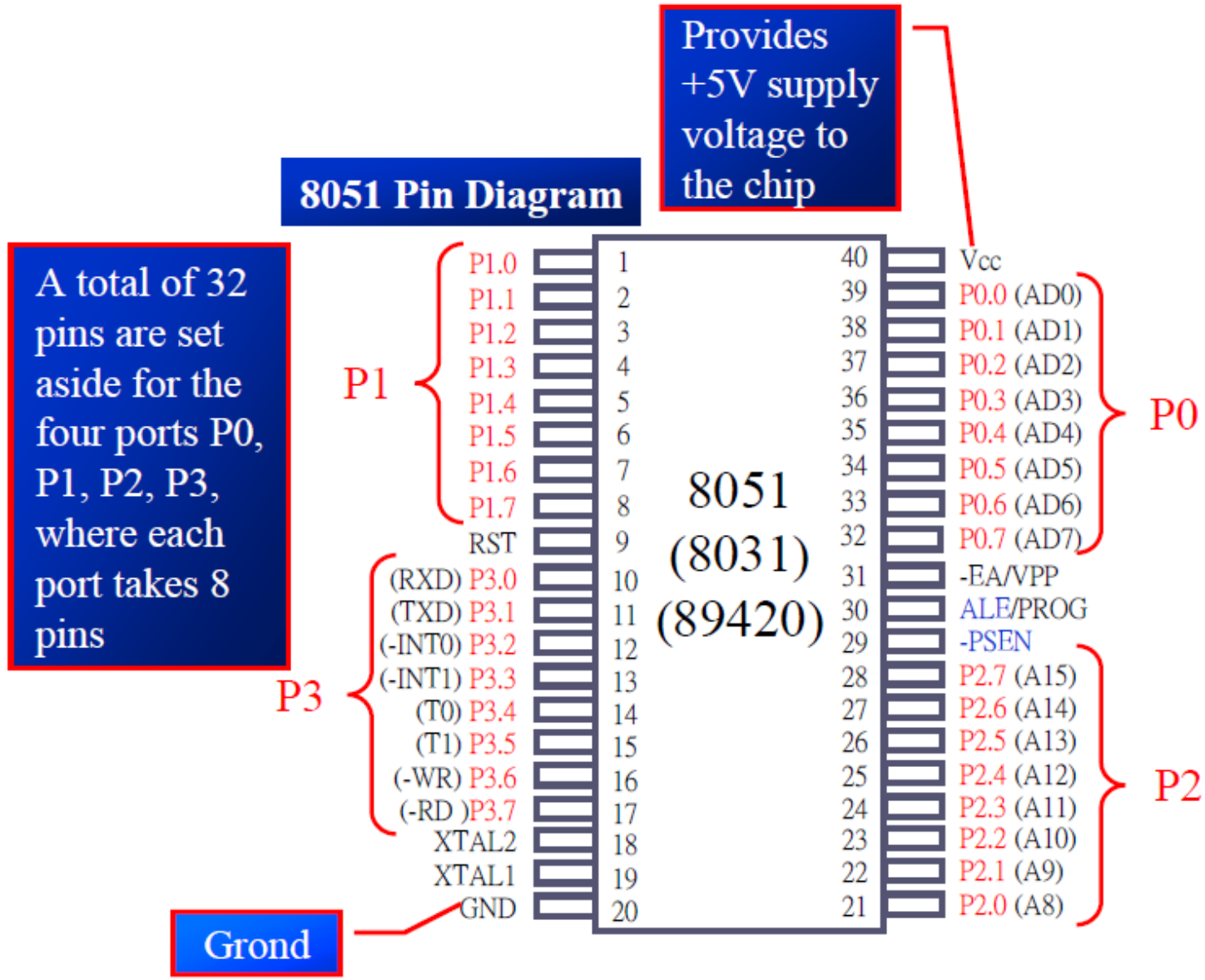
BINARY	DECIMAL	HEXADECIMAL	SYMBOL
101 0000	80	50	P
101 0001	81	51	Q
101 0010	82	52	R
101 0011	83	53	S
101 0100	84	54	T
101 0101	85	55	U
101 0110	86	56	V
101 0111	87	57	W
101 1000	88	58	X
101 1001	89	59	Y
101 1010	90	5A	Z
101 1011	91	5B	[
101 1100	92	5C	\
101 1101	93	5D	]
101 1110	94	5E	^
101 1111	95	5F	_
110 0000	96	60	`
110 0001	97	61	a
110 0010	98	62	b
110 0011	99	63	c
110 0100	100	64	d
110 0101	101	65	e
110 0110	102	66	f
110 0111	103	67	g
110 1000	104	68	h

Key	ASCII	Unpacked BCD	Packed BCD
4	34	00000100	
7	37	00000111	01000111 or 47H

```
MOV     A, #'4'    ;A=34H, hex for ASCII char 4
ANL     A, #0FH    ;mask upper nibble (A=04)
SWAP    A          ;A=40H
MOV     B, A
MOV     A, #'7'    ;R1=37H, hex for ASCII char 7
ANL     A, #0FH    ;mask upper nibble (R1=07)
ORL     A, B       ;A=47H, packed BCD
```

- Many ADC (analog-to-digital converter) chips provide output data in binary (hex)
  - To display the data on an LCD or PC screen, we need to convert it to ASCII
    - Convert 8-bit binary (hex) data to decimal digits, 000 – 255
    - Convert the decimal digits to ASCII digits, 30H – 39H

I/O PORT PROGRAMMING


8051 Pin Diagram
Provides +5V supply voltage to the chip
A total of 32 pins are set aside for the four ports P0, P1, P2, P3, where each port takes 8 pins
Grond
P1
P3
P0
P2
8051
(8031)
(89420)
1 P1.0
2 P1.1
3 P1.2
4 P1.3
5 P1.4
6 P1.5
7 P1.6
8 P1.7
9 RST
10 (RXD) P3.0
11 (TXD) P3.1
12 (-INT0) P3.2
13 (-INT1) P3.3
14 (T0) P3.4
15 (T1) P3.5
16 (-WR) P3.6
17 (-RD) P3.7
18 XTAL2
19 XTAL1
20 GND
40 Vcc
39 P0.0 (AD0)
38 P0.1 (AD1)
37 P0.2 (AD2)
36 P0.3 (AD3)
35 P0.4 (AD4)
34 P0.5 (AD5)
33 P0.6 (AD6)
32 P0.7 (AD7)
31 -EA/VPP
30 ALE/PROG
29 -PSEN
28 P2.7 (A15)
27 P2.6 (A14)
26 P2.5 (A13)
25 P2.4 (A12)
24 P2.3 (A11)
23 P2.2 (A10)
22 P2.1 (A9)
21 P2.0 (A8)

- The four 8-bit I/O ports P0, P1, P2 and P3 each uses 8 pins
- All the ports upon RESET are configured as input, ready to be used as input ports
  - When the first 0 is written to a port, it becomes an output
  - To reconfigure it as an input, a 1 must be sent to the port
    - To use any of these ports as an input port, it must be programmed

The following code will continuously send out to port 0 the alternating value 55H and AAH

```
BACK:   MOV     A,#55H
        MOV     P0,A
        ACALL   DELAY
        MOV     A,#0AAH
        MOV     P0,A
        ACALL   DELAY
        SJMP    BACK
```

55 – 0101 0101

AA – 1010 1010

- Call instruction is used to call subroutine
  - Subroutines are often used to perform tasks that need to be performed frequently
  - This makes a program more structured in addition to saving memory space

LCALL (long call)

  - 3-byte instruction
    - First byte is the opcode
    - Second and third bytes are used for address of target subroutine
      - Subroutine is located anywhere within 64K byte address space

ACALL (absolute call)

  - 2-byte instruction
    - 11 bits are used for address within 2K-byte range

- Port 0 is also designated as AD0-AD7, allowing it to be used for both address and data
  - When connecting an 8051/31 to an external memory, port 0 provides both address and data

- To make port 2 an input port, it must be programmed as such by writing 1 to all its bits
- In many 8051-based system, P2 is used as simple I/O
- In 8031-based systems, port 2 must be used along with P0 to provide the 16-bit address for the external memory
  - Port 2 is also designated as A8 – A15, indicating its dual function
  - Port 0 provides the lower 8 bits via A0 – A7

- Port 3 has the additional function of providing some extremely important signals

P3 Bit	Function	Pin
P3.0	RxD	10
P3.1	TxD	11
P3.2	$\overline{\text{INT0}}$	12
P3.3	$\overline{\text{INT1}}$	13
P3.4	T0	14
P3.5	T1	15
P3.6	$\overline{\text{WR}}$	16
P3.7	$\overline{\text{RD}}$	17

Serial communications

External interrupts

Timers

Read/Write signals of external memories

In systems based on 8751, 89C51 or DS89C4x0, pins 3.6 and 3.7 are used for I/O while the rest of the pins in port 3 are normally used in the alternate function role

The entire 8 bits of Port 1 are accessed

```
BACK:   MOV     A,#55H
        MOV     P1,A
        ACALL   DELAY
        MOV     A,#0AAH
        MOV     P1,A
        ACALL   DELAY
        SJMP    BACK
```

Rewrite the code in a more efficient manner by accessing the port directly without going through the accumulator

```
BACK:   MOV     P1,#55H
        ACALL   DELAY
        MOV     P1,#0AAH
        ACALL   DELAY
        SJMP    BACK
```

Another way of doing the same thing

```
        MOV     A,#55H
BACK:   MOV     P1,A
        ACALL   DELAY
        CPL     A
        SJMP    BACK
```

- Sometimes we need to access only 1 or 2 bits of the port

```
BACK:  CPL    P1.2          ;complement P1.2
       ACALL  DELAY
       SJMP   BACK

;another variation of the above program
AGAIN: SETB   P1.2           ;set only P1.2
       ACALL  DELAY
       CLR    P1.2          ;clear only P1.2
       ACALL  DELAY
       SJMP   AGAIN
```

P0	P1	P2	P3	Port Bit
P0.0	P1.0	P2.0	P3.0	D0
P0.1	P1.1	P2.1	P3.1	D1
P0.2	P1.2	P2.2	P3.2	D2
P0.3	P1.3	P2.3	P3.3	D3
P0.4	P1.4	P2.4	P3.4	D4
P0.5	P1.5	P2.5	P3.5	D5
P0.6	P1.6	P2.6	P3.6	D6
P0.7	P1.7	P2.7	P3.7	D7

- Instructions that are used for signal-bit operations are as following

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)

- The JNB and JB instructions are widely used single-bit operations
  - They allow you to monitor a bit and make a decision depending on whether it's 0 or 1
  - These two instructions can be used for any bits of I/O ports 0, 1, 2, and 3
    - Port 3 is typically not used for any I/O, either single-bit or byte-wise

Instructions for Reading an Input Port

Mnemonic	Examples	Description
MOV A,PX	MOV A,P2	Bring into A the data at P2 pins
JNB PX.Y, ..	JNB P2.1,TARGET	Jump if pin P2.1 is low
JB PX.Y, ..	JB P1.3,TARGET	Jump if pin P1.3 is high
MOV C,PX.Y	MOV C,P2.4	Copy status of pin P2.4 to CY

Example 4-2

Write the following programs.
Create a square wave of 50% duty cycle on bit 0 of port 1.

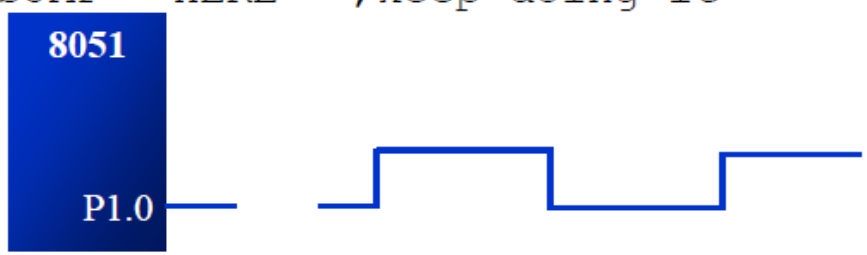
Solution:

The 50% duty cycle means that the "on" and "off" state (or the high and low portion of the pulse) have the same length. Therefore, we toggle P1.0 with a time delay in between each state.

```
HERE:  SETB   P1.0   ;set to high bit 0 of port 1
       LCALL  DELAY  ;call the delay subroutine
       CLR    P1.0   ;P1.0=0
       LCALL  DELAY
       SJMP   HERE   ;keep doing it
```

Another way to write the above program is:

```
HERE:  CPL    P1.0   ;set to high bit 0 of port 1
       LCALL  DELAY  ;call the delay subroutine
       SJMP   HERE   ;keep doing it
```



Example 4-3

Write a program to perform the following:
(a) Keep monitoring the P1.2 bit until it becomes high
(b) When P1.2 becomes high, write value 45H to port 0
(c) Send a high-to-low (H-to-L) pulse to P2.3

Solution:

```
        SETB P1.2        ;make P1.2 an input
        MOV  A,#45H      ;A=45H
AGAIN:  JNB  P1.2,AGAIN  ; get out when P1.2=1
        MOV  P0,A        ;issue A to P0
        SETB P2.3        ;make P2.3 high
        CLR  P2.3        ;make P2.3 low for H-to-L
```

Example 4-4

Assume that bit P2.3 is an input and represents the condition of an oven. If it goes high, it means that the oven is hot. Monitor the bit continuously. Whenever it goes high, send a high-to-low pulse to port P1.5 to turn on a buzzer.

Example 4-4

Assume that bit P2.3 is an input and represents the condition of an oven. If it goes high, it means that the oven is hot. Monitor the bit continuously. Whenever it goes high, send a high-to-low pulse to port P1.5 to turn on a buzzer.

Solution:

```
HERE:  JNB  P2.3,HERE ;keep monitoring for high
       SETB P1.5      ;set bit P1.5=1
       CLR  P1.5      ;make high-to-low
       SJMP HERE      ;keep repeating
```

Example 4-5

A switch is connected to pin P1.7. Write a program to check the status of SW and perform the following:
(a) If SW=0, send letter 'N' to P2
(b) If SW=1, send letter 'Y' to P2

Solution:

```
        SETB P1.7          ;make P1.7 an input
AGAIN:  JB   P1.2,OVER     ;jump if P1.7=1
        MOV  P2,#'N'       ;SW=0, issue 'N' to P2
        SJMP AGAIN         ;keep monitoring
OVER:   MOV  P2,#'Y'       ;SW=1, issue 'Y' to P2
        SJMP AGAIN         ;keep monitoring
```

Example 4-6

A switch is connected to pin P1.7. Write a program to check the status of SW and perform the following:
(a) If SW=0, send letter 'N' to P2
(b) If SW=1, send letter 'Y' to P2
Use the carry flag to check the switch status.

Solution:

```
        SETB P1.7       ;make P1.7 an input
AGAIN:  MOV  C,P1.2     ;read SW status into CF
        JC   OVER       ;jump if SW=1
        MOV  P2,#'N'    ;SW=0, issue 'N' to P2
        SJMP AGAIN      ;keep monitoring
OVER:   MOV  P2,#'Y'    ;SW=1, issue 'Y' to P2
        SJMP AGAIN      ;keep monitoring
```

Example 4-7

A switch is connected to pin P1.0 and an LED to pin P2.7. Write a program to get the status of the switch and send it to the LED

Solution:

```
        SETB P1.7        ;make P1.7 an input
AGAIN:  MOV  C,P1.0      ;read SW status into CF
        MOV  P2.7,C      ;send SW status to LED
        SJMP AGAIN       ;keep repeating
```

The instruction 'MOV P2.7,P1.0' is wrong , since such an instruction does not exist

However 'MOV P2,P1' is a valid instruction

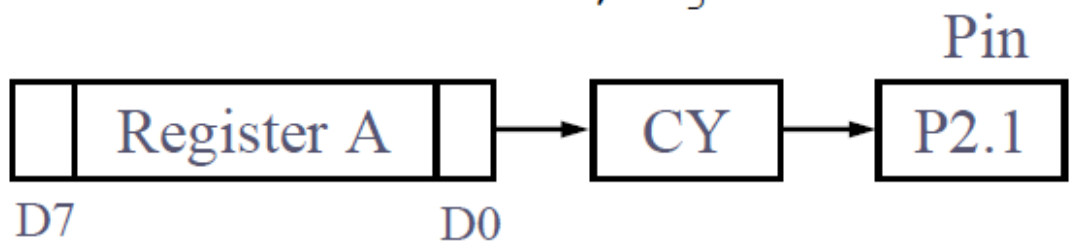
- Serializing data is a way of sending a byte of data one bit at a time through a single pin of microcontroller

- To transfer data one bit at a time and control the sequence of data and spaces in between them

Write a program to transfer value 41H serially (one bit at a time) via pin P2.1. Put two highs at the start and end of the data. Send the byte LSB first.

Solution:

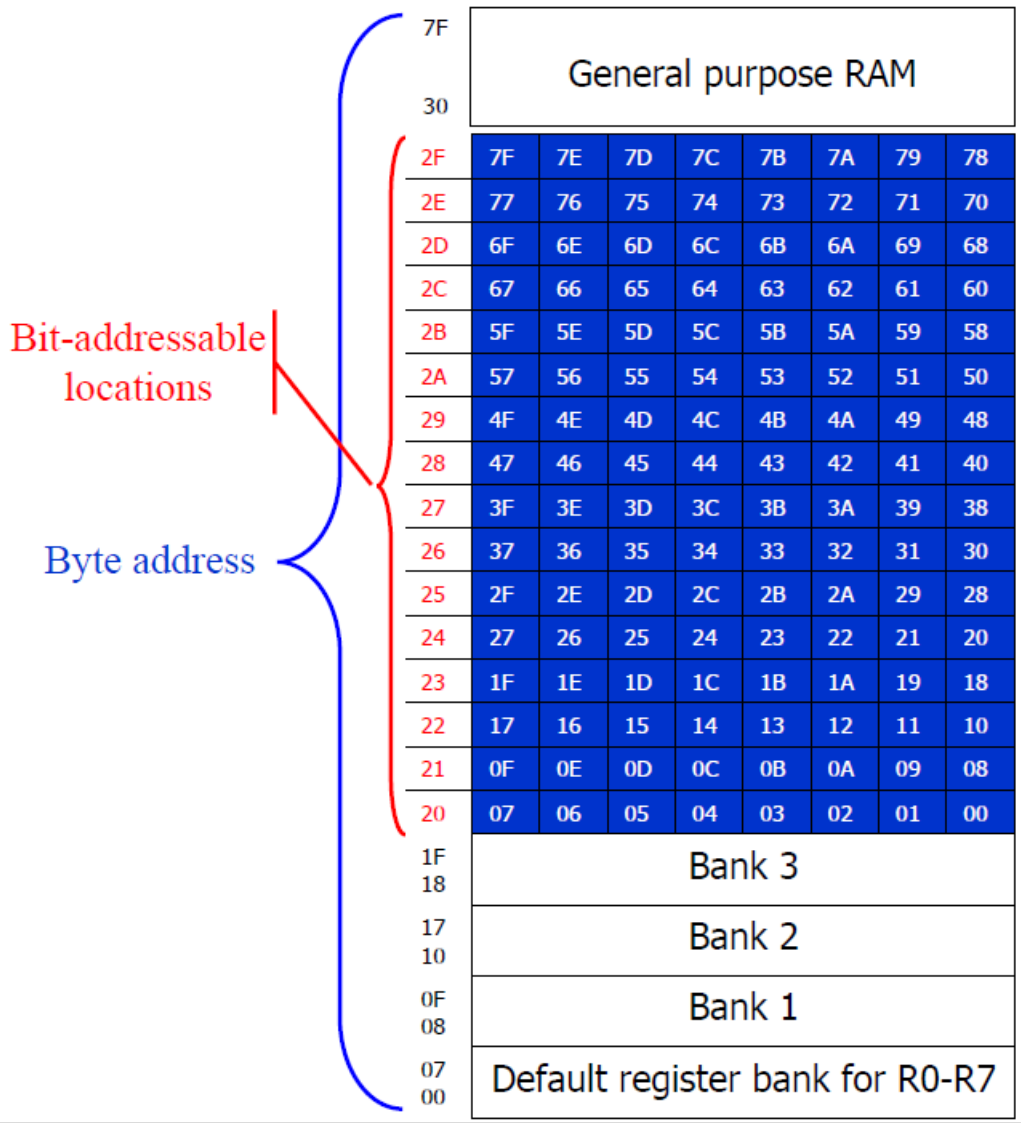
```
        MOV     A,#41H
        SETB    P2.1        ;high
        SETB    P2.1        ;high
        MOV     R5,#8
AGAIN:  RRC     A
        MOV     P2.1,C      ;send CY to P2.1
        DJNZ    R5,HERE
        SETB    P2.1        ;high
        SETB    P2.1        ;high
```



BIT ADDRESSES

Bit-Addressable RAM

- The bit-addressable RAM location are 20H to 2FH
  - These 16 bytes provide 128 bits of RAM bit-addressability, since 16 × 8 = 128
    - 0 to 127 (in decimal) or 00 to 7FH
  - The first byte of internal RAM location 20H has bit address 0 to 7H
  - The last byte of 2FH has bit address 78H to 7FH
- Internal RAM locations 20-2FH are both byte-addressable and bit-addressable
  - Bit address 00-7FH belong to RAM byte addresses 20-2FH
  - Bit address 80-F7H belong to SFR P0, P1, …


Bit-addressable locations
Byte address
7F
30
General purpose RAM
2F 7F 7E 7D 7C 7B 7A 79 78
2E 77 76 75 74 73 72 71 70
2D 6F 6E 6D 6C 6B 6A 69 68
2C 67 66 65 64 63 62 61 60
2B 5F 5E 5D 5C 5B 5A 59 58
2A 57 56 55 54 53 52 51 50
29 4F 4E 4D 4C 4B 4A 49 48
28 47 46 45 44 43 42 41 40
27 3F 3E 3D 3C 3B 3A 39 38
26 37 36 35 34 33 32 31 30
25 2F 2E 2D 2C 2B 2A 29 28
24 27 26 25 24 23 22 21 20
23 1F 1E 1D 1C 1B 1A 19 18
22 17 16 15 14 13 12 11 10
21 0F 0E 0D 0C 0B 0A 09 08
20 07 06 05 04 03 02 01 00
1F
18
Bank 3
17
10
Bank 2
0F
08
Bank 1
07
00
Default register bank for R0-R7

Example 5-11

Find out to which by each of the following bits belongs. Give the address of the RAM byte in hex

(a) `SETB 42H`, (b) `CLR 67H`, (c) `CLR 0FH`

(d) `SETB 28H`, (e) `CLR 12`, (f) `SETB 05`

Solution:

(a) D2 of RAM location 28H

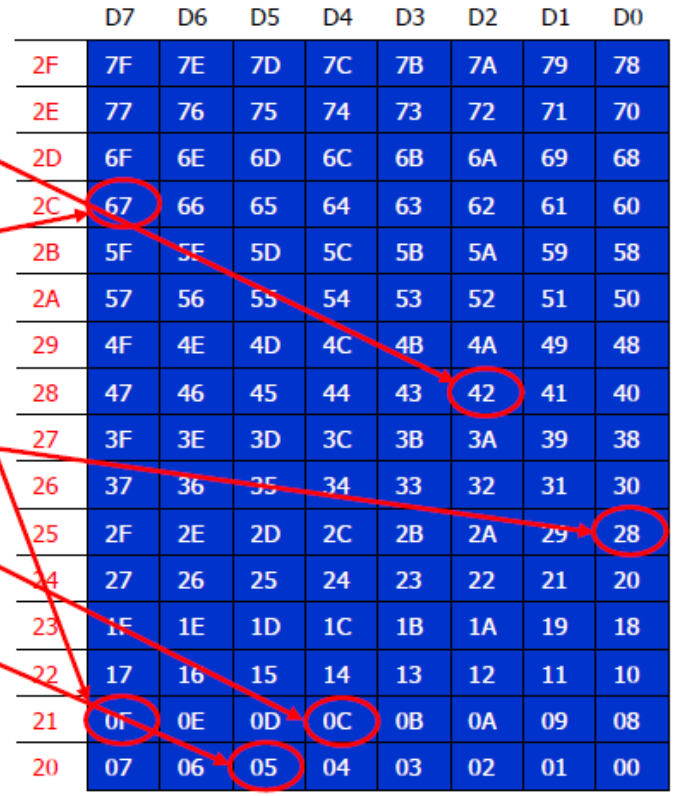
(b) D7 of RAM location 2CH

(c) D7 of RAM location 21H

(d) D0 of RAM location 25H

(e) D4 of RAM location 21H

(f) D5 of RAM location 20H



	D7	D6	D5	D4	D3	D2	D1	D0
2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00

- To avoid confusion regarding the addresses 00 – 7FH
  - The 128 bytes of RAM have the byte addresses of 00 – 7FH can be accessed in byte size using various addressing modes
    - Direct and register-indirect
  - The 16 bytes of RAM locations 20 – 2FH have bit address of 00 – 7FH
    - We can use only the single-bit instructions and these instructions use only direct addressing mode

Single-Bit Instructions

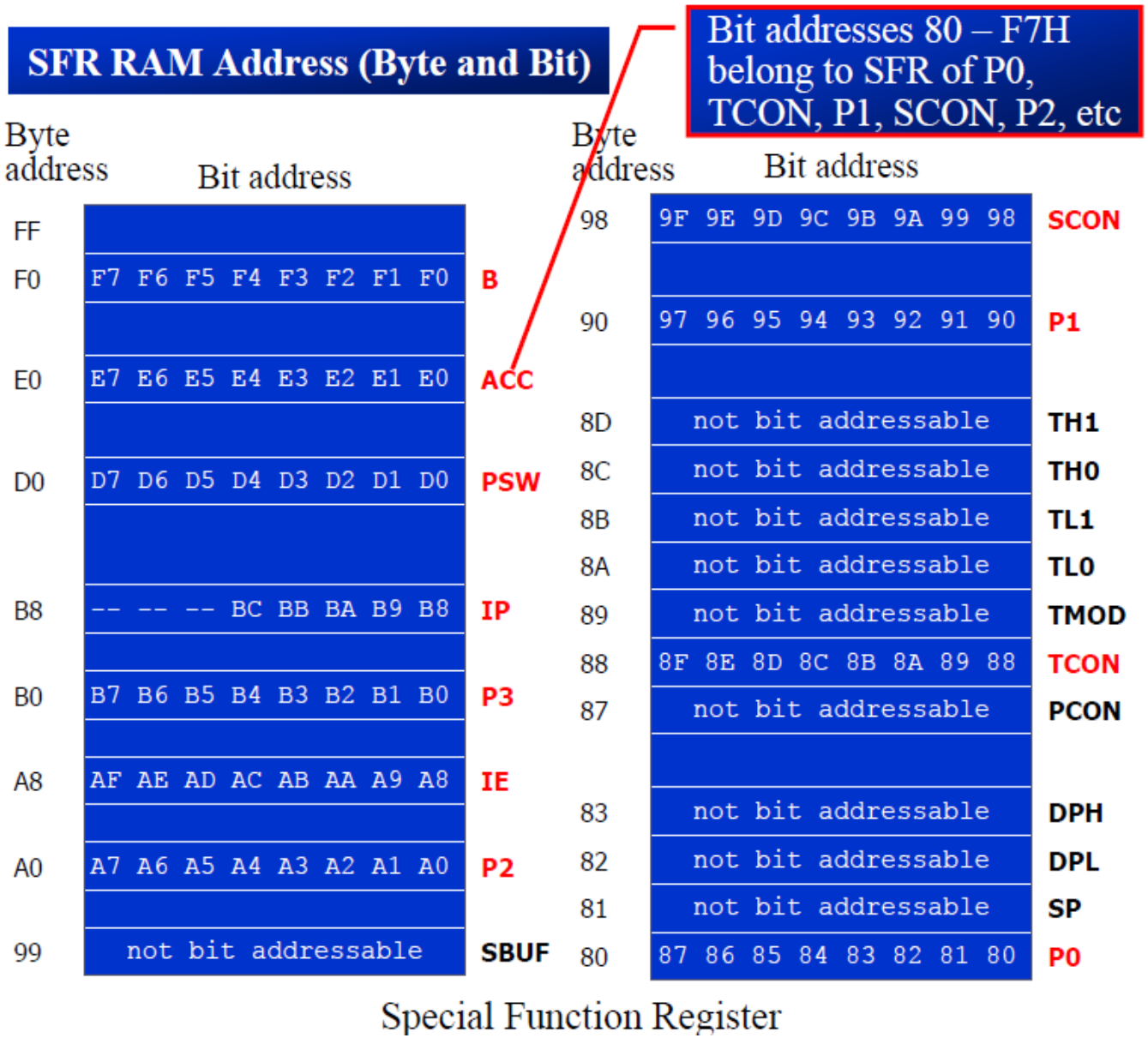
Instruction	Function
SETB bit	Set the bit (bit = 1)
CLR bit	Clear the bit (bit = 0)
CPL bit	Complement the bit (bit = NOT bit)
JB bit, target	Jump to target if bit = 1 (jump if bit)
JNB bit, target	Jump to target if bit = 0 (jump if no bit)
JBC bit, target	Jump to target if bit = 1, clear bit (jump if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

- While all of the SFR registers are byte-addressable, some of them are also bit-addressable
  - The P0 – P3 are bit addressable
- We can access either the entire 8 bits or any single bit of I/O ports P0, P1, P2, and P3 without altering the rest
- When accessing a port in a single-bit manner, we use the syntax `SETB X.Y`
  - X is the port number P0, P1, P2, or P3
  - Y is the desired bit number from 0 to 7 for data bits D0 to D7
  - ex. `SETB P1.5` sets bit 5 of port 1 high

- Notice that when code such as `SETB P1.0` is assembled, it becomes `SETB 90H`
  - The bit address for I/O ports
    - P0 are 80H to 87H
    - P1 are 90H to 97H
    - P2 are A0H to A7H
    - P3 are B0H to B7H

Single-Bit Addressability of Ports

P0	P1	P2	P3	Port Bit
P0.0 (80)	P1.0 (90)	P2.0 (A0)	P3.0 (B0)	D0
P0.1	P1.1	P2.1	P3.1	D1
P0.2	P1.2	P2.2	P3.2	D2
P0.3	P1.3	P2.3	P3.3	D3
P0.4	P1.4	P2.4	P3.4	D4
P0.5	P1.5	P2.5	P3.5	D5
P0.6	P1.6	P2.6	P3.6	D6
P0.7 (87)	P1.7 (97)	P2.7 (A7)	P3.7 (B7)	D7



Special Function Register

Only registers A, B, PSW, IP, IE, ACC, SCON, and TCON are bit-addressable

- While all I/O ports are bit-addressable

Example 5-17

The status of bits P1.2 and P1.3 of I/O port P1 must be saved before they are changed. Write a program to save the status of P1.2 in bit location 06 and the status of P1.3 in bit location 07

Solution:

```
        CLR     06          ;clear bit addr. 06
        CLR     07          ;clear bit addr. 07
        JNB     P1.2,OVER   ;check P1.2, if 0 then jump
        SETB    06          ;if P1.2=1,set bit 06 to 1
OVER:   JNB     P1.3,NEXT   ;check P1.3, if 0 then jump
        SETB    07          ;if P1.3=1,set bit 07 to 1
NEXT:   ...
```

- The `BIT` directive is a widely used directive to assign the bit-addressable I/O and RAM locations
  - Allow a program to assign the I/O or RAM bit at the beginning of the program, making it easier to modify them

Example 5-22

A switch is connected to pin P1.7 and an LED to pin P2.0. Write a program to get the status of the switch and send it to the LED.

Solution:

```
LED     BIT     P1.7    ;assign bit
SW      BIT     P2.0    ;assign bit
HERE:   MOV     C,SW    ;get the bit from the port
        MOV     LED,C   ;send the bit to the port
        SJMP    HERE    ;repeat forever
```

- There are several instructions by which the 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 b,C	Copy carry status to bit location (CY = b)
MOV C,b	Copy bit location status to carry (b = CY)
JNC target	Jump to target if CY = 0
JC target	Jump to target if CY = 1
ANL C,bit	AND CY with bit and save 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

- Use the EQU to assign addresses
  - Defined by names, like P1.7 or P2
  - Defined by addresses, like 97H or 0A0H

Example 5-24

A switch is connected to pin P1.7. Write a program to check the status of the switch and make the following decision.
(a) If SW = 0, send "0" to P2
(b) If SW = 1, send "1" to P2

Solution:

```
SW       EQU  P1.7
MYDATA   EQU  P2
HERE:    MOV      C,SW
         JC       OVER
         MOV      MYDATA,#'0'
         SJMP     HERE
OVER:    MOV      MYDATA,#'1'
         SJMP     HERE
         END
```

```
SW       EQU  97H
MYDATA   EQU  0A0H
```

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

Solution:

```
        SETB    C           ;CY = 1
        ORL     C,P2.2      ;CY = P2.2 ORed w/ CY
        MOV     P2.2,C      ;turn it on if not on
        CLR     C           ;CY = 0
        ANL     C,P2.5      ;CY = P2.5 ANDed w/ CY
        MOV     P2.5,C     ;turn it off if not off
```

Example 5-18

While a program to save the status of bit P1.7 on RAM address bit 05.

Solution:

```
        MOV     C,P1.7
        MOV     05,C
```

Example 5-15

Write a program to see if the RAM location 37H contains an even value. If so, send it to P2. If not, make it even and then send it to P2.

Solution:

```
        MOV     A,37H       ;load RAM 37H into ACC
        JNB     ACC.0,YES   ;if D0 of ACC 0? If so jump
        INC     A           ;it's odd, make it even
YES:    MOV     P2,A        ;send it to P2
```

Example 5-3

Write a program to copy the value 55H into RAM memory locations 40H to 45H using
(a) direct addressing mode,
(b) register indirect addressing mode without a loop, and
(c) with a loop.

(a)

```
MOV A,#55H      ;load A with value 55H
MOV 40H,A       ;copy A to RAM location 40H
MOV 41H,A       ;copy A to RAM location 41H
MOV 42H,A       ;copy A to RAM location 42H
MOV 43H,A       ;copy A to RAM location 43H
MOV 44H,A       ;copy A to RAM location 44H
```

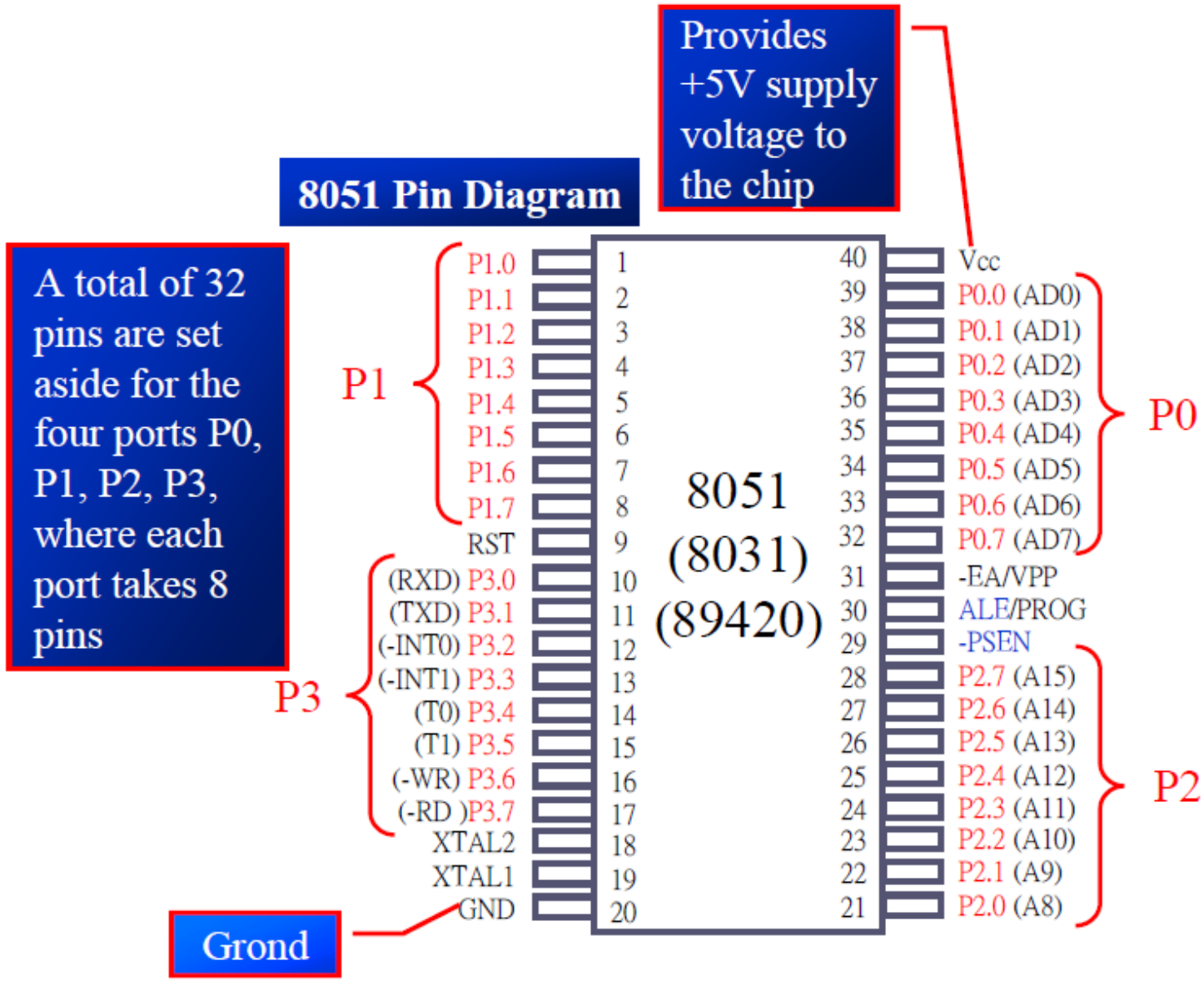
(b)

```
MOV A,#55H      ;load A with value 55H
MOV R0,#40H     ;load the pointer. R0=40H
MOV @R0,A       ;copy A to RAM location R0 points to
INC R0          ;increment pointer. Now R0=41H
MOV @R0,A       ;copy A to RAM location  R0 points to
INC R0          ;increment pointer. Now R0=42H
MOV @R0,A       ;copy A to RAM location  R0 points to
INC R0          ;increment pointer. Now R0=43H
MOV @R0,A       ;copy A to RAM location R0 points to
INC R0          ;increment pointer. Now R0=44H
MOV @R0,A
```

(c)

```
        MOV  A,#55   ;A=55H
        MOV  R0,#40H ;load pointer. R0=40H, RAM address
        MOV  R2,#05  ;load counter, R2=5
AGAIN:  MOV  @R0,A   ;copy 55H to RAM location R0 points to
        INC  R0      ;increment R0 pointer
        DJNZ R2,AGAIN     ;loop until counter = zero
```

Delay Calculation


8051 Pin Diagram
Provides +5V supply voltage to the chip
A total of 32 pins are set aside for the four ports P0, P1, P2, P3, where each port takes 8 pins
Grond
P1
P3
P0
P2
8051
(8031)
(89420)
P1.0 1
P1.1 2
P1.2 3
P1.3 4
P1.4 5
P1.5 6
P1.6 7
P1.7 8
RST 9
(RXD) P3.0 10
(TXD) P3.1 11
(-INT0) P3.2 12
(-INT1) P3.3 13
(T0) P3.4 14
(T1) P3.5 15
(-WR) P3.6 16
(-RD) P3.7 17
XTAL2 18
XTAL1 19
GND 20
40 Vcc
39 P0.0 (AD0)
38 P0.1 (AD1)
37 P0.2 (AD2)
36 P0.3 (AD3)
35 P0.4 (AD4)
34 P0.5 (AD5)
33 P0.6 (AD6)
32 P0.7 (AD7)
31 -EA/VPP
30 ALE/PROG
29 -PSEN
28 P2.7 (A15)
27 P2.6 (A14)
26 P2.5 (A13)
25 P2.4 (A12)
24 P2.3 (A11)
23 P2.2 (A10)
22 P2.1 (A9)
21 P2.0 (A8)

- CPU executing an instruction takes a certain number of clock cycles
  - These are referred as to as *machine cycles*
- The length of machine cycle depends on the frequency of the crystal oscillator connected to 8051
- In original 8051, one machine cycle lasts 12 oscillator periods

Find the period of the machine cycle for 11.0592 MHz crystal frequency

Solution:

11.0592/12 = 921.6 kHz;

machine cycle is 1/921.6 kHz = 1.085 μs

For 8051 system of 11.0592 MHz, find how long it takes to execute each instruction.

(a) `MOV R3,#55` (b) `DEC R3` (c) `DJNZ R2 target`

(d) `LJMP` (e) `SJMP` (f) `NOP` (g) `MUL AB`

Solution:

	Machine cycles	*Time to execute*
(a)	1	$1 \times 1.085\,\mu s = 1.085\,\mu s$
(b)	1	$1 \times 1.085\,\mu s = 1.085\,\mu s$
(c)	2	$2 \times 1.085\,\mu s = 2.17\,\mu s$
(d)	2	$2 \times 1.085\,\mu s = 2.17\,\mu s$
(e)	2	$2 \times 1.085\,\mu s = 2.17\,\mu s$
(f)	1	$1 \times 1.085\,\mu s = 1.085\,\mu s$
(g)	4	$4 \times 1.085\,\mu s = 4.34\,\mu s$

Find the size of the delay in following program, if the crystal frequency is 11.0592MHz.

```
        MOV   A,#55H
AGAIN:  MOV   P1,A
        ACALL DELAY
        CPL   A
        SJMP  AGAIN
;---time delay-------
DELAY:  MOV   R3,#200
HERE:   DJNZ  R3,HERE
        RET
```

A simple way to short jump to itself in order to keep the microcontroller busy

```
HERE:  SJMP  HERE
```

We can use the following:

```
SJMP $
```

Solution:

		Machine cycle
DELAY:	MOV R3,#200	1
HERE:	DJNZ R3,HERE	2
	RET	2

Therefore, [(200x2)+1+2]x1.085μs = 436.255μs.

Find the size of the delay in following program, if the crystal frequency is 11.0592MHz.

```
                          Machine Cycle
DELAY: MOV  R3,#250           1
HERE:  NOP                    1
       NOP                    1
       NOP                    1
       NOP                    1
       DJNZ R3,HERE           2
       RET                    2
```

Solution:

The time delay inside HERE loop is

$[250(1+1+1+1+2)]\times 1.085\mu s = 1627.5\mu s.$

Adding the two instructions outside loop we have $1627.5\mu s + 3 \times 1.085\mu s = 1630.755\mu s$

Find the size of the delay in following program, if the crystal frequency is 11.0592MHz.

```
                          Machine Cycle
DELAY: MOV   R2,#200          1
AGAIN: MOV   R3,#250          1
HERE:  NOP                    1
       NOP                    1
       DJNZ  R3,HERE          2
       DJNZ  R2,AGAIN         2
       RET                    2
```

Notice in nested loop, as in all other time delay loops, the time is approximate since we have ignored the first and last instructions in the subroutine.

Solution:

For HERE loop, we have (4x250)x1.085μs=1085μs. For AGAIN loop repeats HERE loop 200 times, so we have 200x1085μs=217000μs. But "MOV R3,#250" and "DJNZ R2,AGAIN" at the start and end of the AGAIN loop add (3x200x1.805)=651μs. As a result we have 217000+651=217651μs.

Clocks per machine cycle for various 8051 versions

Chip/Maker	Clocks per Machine Cycle
AT89C51 Atmel	12
P89C54X2 Philips	6
DS5000 Dallas Semi	4
DS89C420/30/40/50 Dallas Semi	1

Find the period of the machine cycle (MC) for various versions of 8051, if XTAL=11.0592 MHz.
(a) AT89C51 (b) P89C54X2 (c) DS5000 (d) DS89C4x0

Solution:

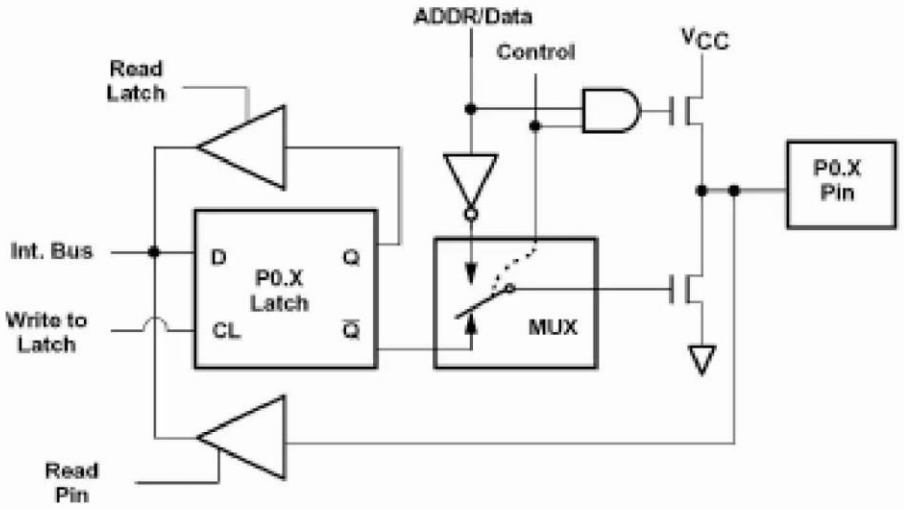
(a) 11.0592MHz/12 = 921.6kHz;
MC is 1/921.6kHz = 1.085μs = 1085ns
(b) 11.0592MHz/6 = 1.8432MHz;
MC is 1/1.8432MHz = 0.5425μs = 542ns
(c) 11.0592MHz/4 = 2.7648MHz ;
MC is 1/2.7648MHz = 0.36μs = 360ns
(d) 11.0592MHz/1 = 11.0592MHz;
MC is 1/11.0592MHz = 0.0904μs = 90ns

Write a program to toggle all the bits of P1 every 200 ms. Assume that the crystal frequency is 11.0592 MHz, and that the system is using the AT89C51.

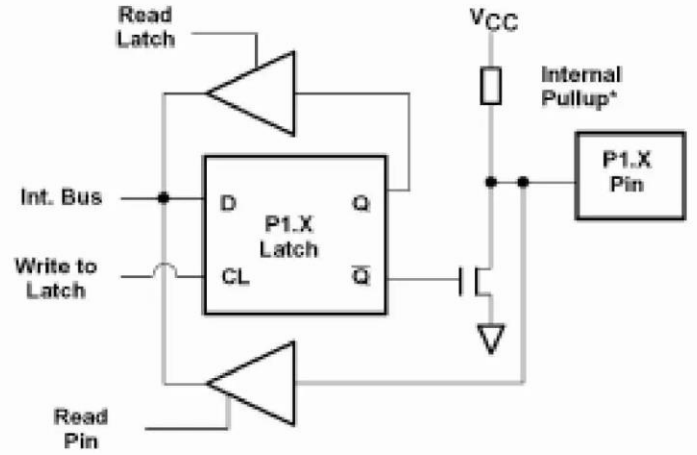
```
            MOV   A,#55H
AGAIN:      MOV   P1,A
            ACALL DELAY
            CPL   A
            SJMP  AGAIN

;----Time delay
DELAY:      MOV   R5,#2
HERE1:      MOV   R4,#180
HERE2:      MOV   R3,#255
HERE3:      DJNZ  R3,HERE3
            DJNZ  R4,HERE2
            DJNZ  R5,HERE1
            RET
```

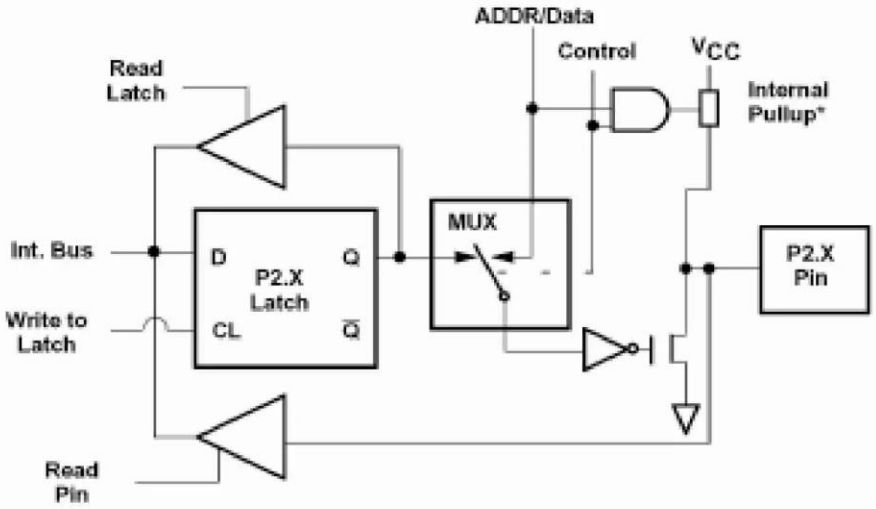
$2 \times 180 \times 255 \times 2$ MC $\times$ 1.085 µs = 199,206 µs



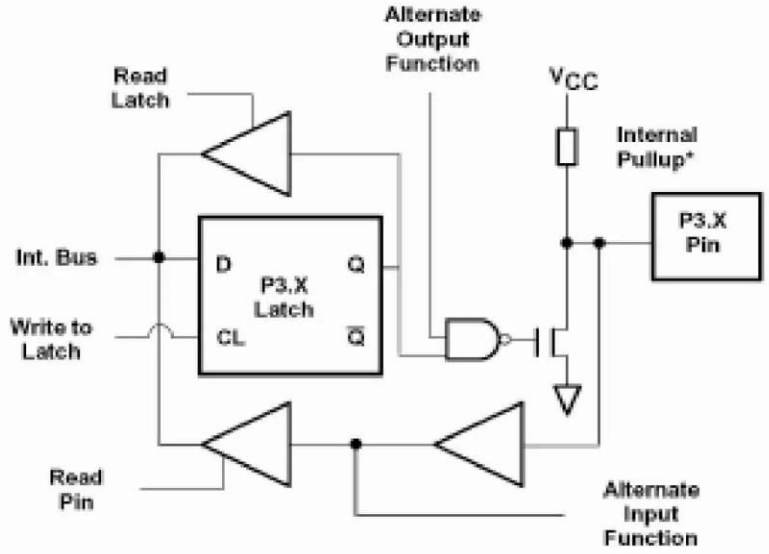
a. Port 0 Bit



b. Port 1 Bit



c. Port 2 Bit



d. Port 3 Bit