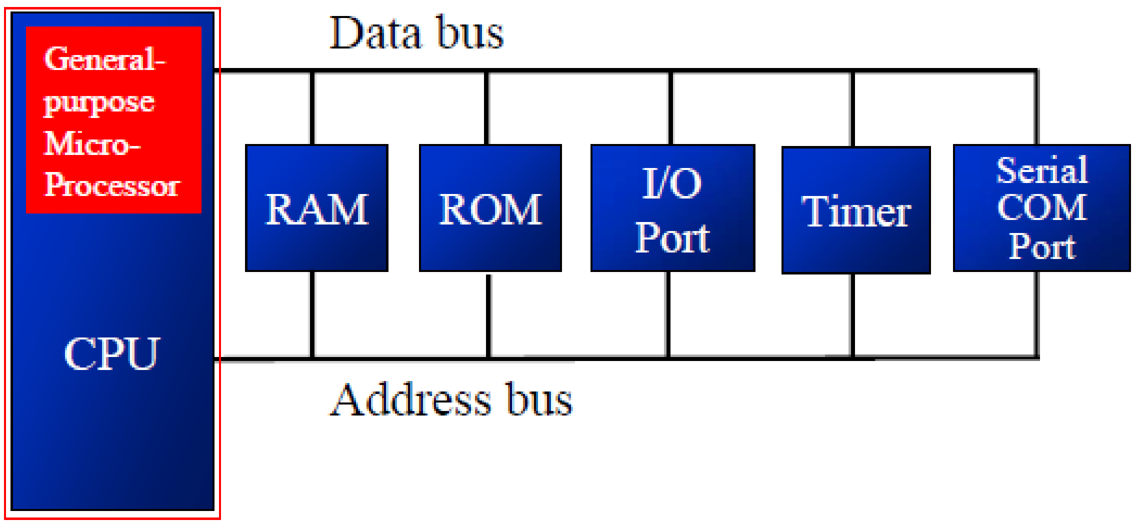
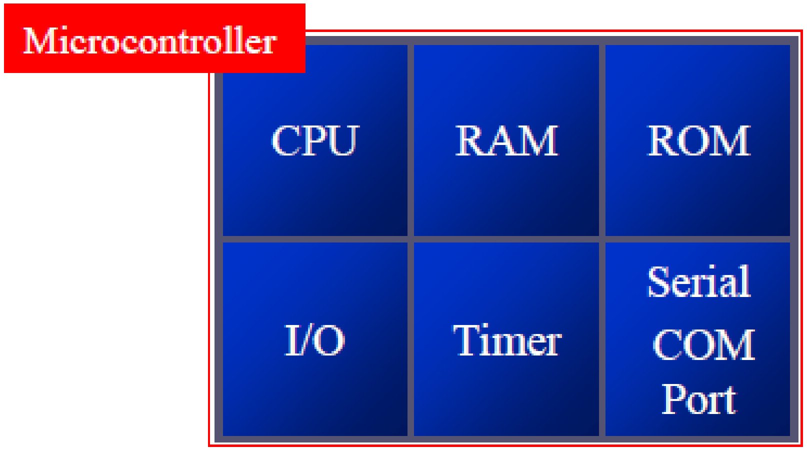
8051 Microcontroller

Microcontroller vs. General Purpose Microprocessor

- General-purpose microprocessors contains
  - No RAM
  - No ROM
  - No I/O ports
- Microcontroller has
  - CPU (microprocessor)
  - RAM
  - ROM
  - I/O ports
  - Timer
  - ADC and other peripherals

Microcontroller vs. General Purpose Microprocessor





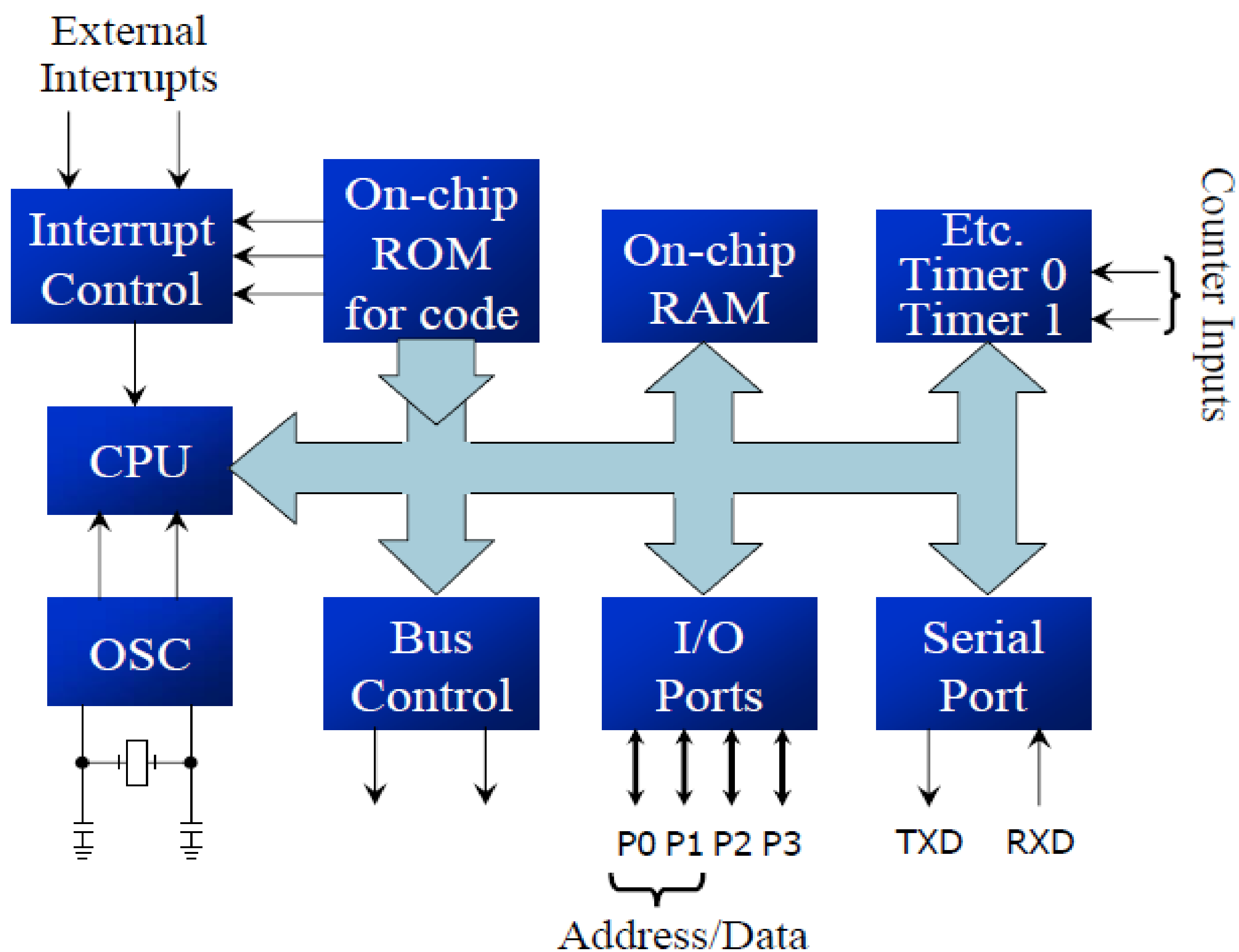
Microcontroller vs. General Purpose Microprocessor

- General-purpose microprocessors
  - Must add RAM, ROM, I/O ports, and timers externally to make them functional
  - Make the system bulkier and much more expensive
  - Have the advantage of versatility on the amount of RAM, ROM, and I/O ports
- Microcontroller
  - The fixed amount of on-chip ROM, RAM, and number of I/O ports makes them ideal for many applications in which cost and space are critical
  - In many applications, the space it takes, the power it consumes, and the price per unit are much more critical considerations than the computing power

8051 Microcontroller

- The 8051 is an 8-bit processor
  - The CPU can work on only 8 bits of data at a time
- The 8051 has
  - 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

8051 Microcontroller Block Diagram

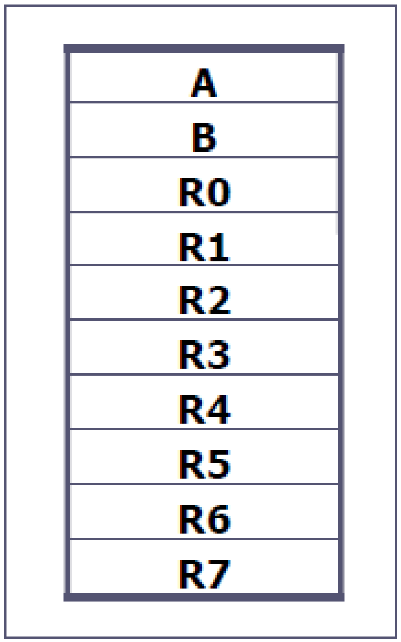


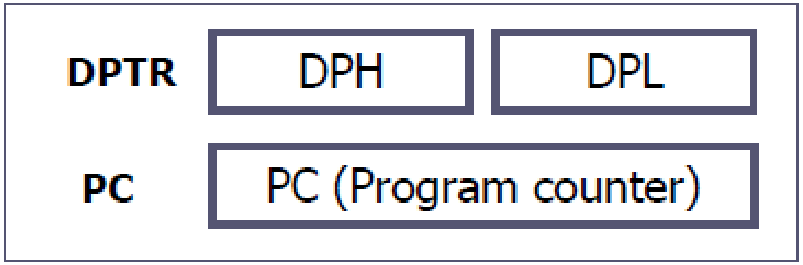
8051 Registers

- 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

8051 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 Instruction

`MOV destination, source` ;copy source to dest.

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

"#" signifies that it is a value

```
MOV   A,#55H      ;load value 55H into reg. A
MOV   R0,A        ;copy contents of A into R0
                  ;(now A=R0=55H)
MOV   R1,A        ;copy contents of A into R1
                  ;(now A=R0=R1=55H)
MOV   R2,A        ;copy contents of A into R2
                  ;(now A=R0=R1=R2=55H)
MOV   R3,#95H     ;load value 95H into R3
                  ;(now R3=95H)
MOV   A,R3        ;copy contents of R3 into A
                  ;now A=R3=95H
```

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)

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)
```

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

8051 ASSEMBLY PROGRAMMING

- 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
```

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

Mnemonics produce opcodes

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

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

PROGRAM COUNTER AND ROM SPACE

Program Counter

- 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

Program Counter (contd.)

- All 8051 members start at memory address 0000 when they're powered up
  - Program Counter has the value of 0000
  - The first opcode is burned into ROM address 0000H, since this is where the 8051 looks for the first instruction when it is booted
  - We achieve this by the ORG statement in the source program

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

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

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

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

```
        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 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

8051 DATA TYPES AND DIRECTIVES

Assembler Directives (cont')

- 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

8051 DATA TYPES AND DIRECTIVES

Assembler directives (cont')

- `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

FLAG BITS AND PSW REGISTER

Program Status Word

- The program status word (PSW) register, also referred to as the *flag register*, is an 8 bit register
  - Only 6 bits are used
    - These four are CY (*carry*), AC (*auxiliary carry*), P (*parity*), and OV (*overflow*)
      - They are called *conditional flags*, meaning that they indicate some conditions that resulted after an instruction was executed
    - The PSW3 and PSW4 are designed as RS0 and RS1, and are used to change the bank
  - The two unused bits are user-definable

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.

Reflect the number of 1s in register A

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

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

FLAG BITS AND PSW REGISTER

ADD Instruction And PSW (cont')

- 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)

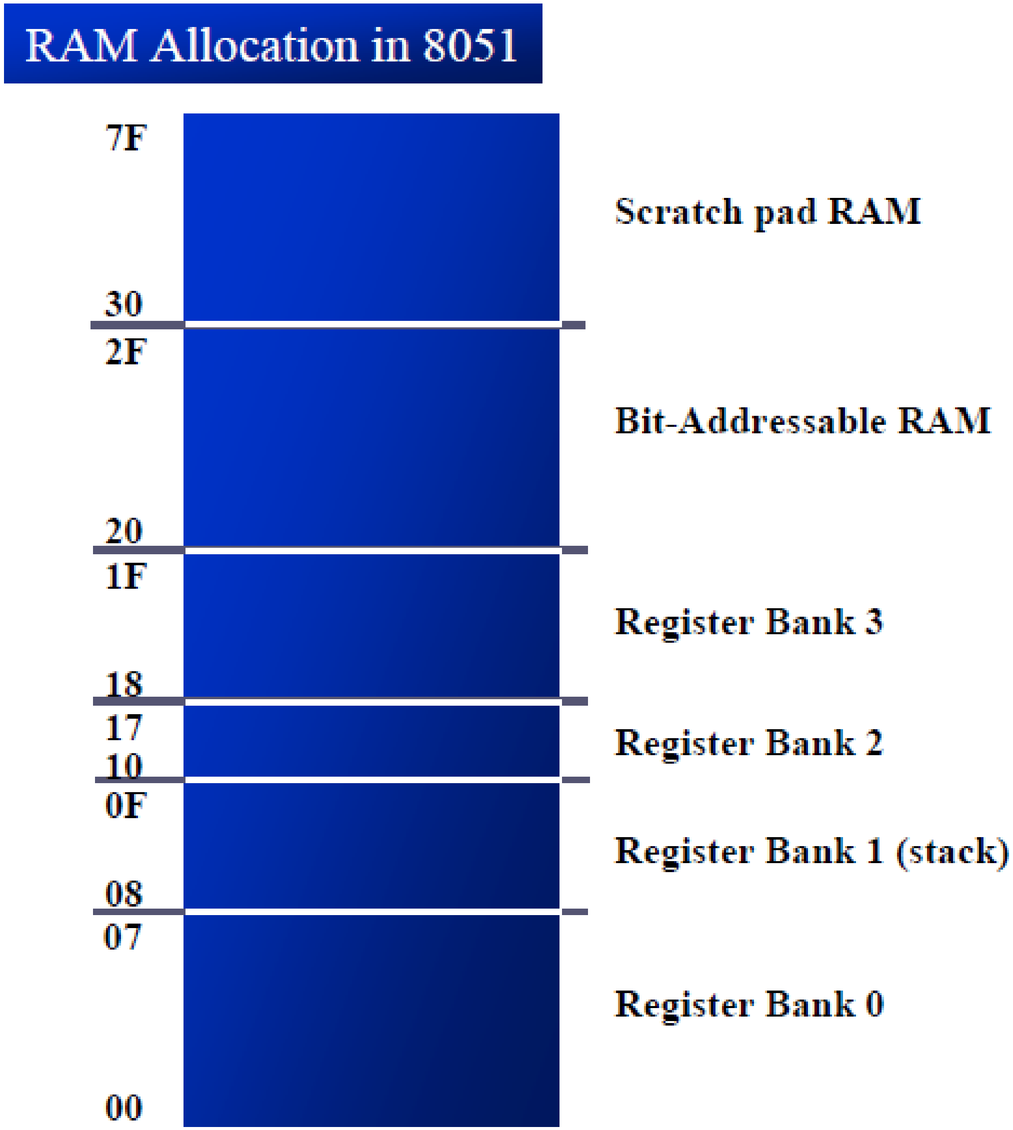
REGISTER BANKS AND STACK

RAM Memory Space Allocation

- 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*

8051 REGISTER BANKS AND STACK

RAM Memory Space Allocation (cont’)



8051 REGISTER BANKS AND STACK

Register Banks (cont')

Register banks and their RAM address

Bank 0		Bank 1		Bank 2		Bank 3	
7	R7	F	R7	17	R7	1F	R7
6	R6	E	R6	16	R6	1E	R6
5	R5	D	R5	15	R5	1D	R5
4	R4	C	R4	14	R4	1C	R4
3	R3	B	R3	13	R3	1B	R3
2	R2	A	R2	12	R2	1A	R2
1	R1	9	R1	11	R1	19	R1
0	R0	8	R0	10	R0	18	R0

8051 REGISTER BANKS AND STACK

Register Banks (cont')

- 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

8051 REGISTER BANKS AND STACK

Register Banks (cont')

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
```

8051 REGISTER BANKS AND STACK

Stack

- 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

8051 REGISTER BANKS AND STACK

Stack (cont')

- The storing of a CPU register in the stack is called a `PUSH`
  - SP is pointing to the last used location of the stack
  - As we push data onto the stack, the SP is incremented by one
    - This is different from many microprocessors
- Loading the contents of the stack back into a CPU register is called a `POP`
  - With every pop, the top byte of the stack is copied to the register specified by the instruction and the stack pointer is decremented once

8051 REGISTER BANKS AND STACK

Pushing onto Stack

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:

			After PUSH 6		After PUSH 1		After PUSH 4	
0B			0B		0B		0B	
0A			0A		0A		0A	F3
09			09		09	12	09	12
08			08	25	08	25	08	25
Start SP = 07			SP = 08		SP = 09		SP = 0A	

8051 REGISTER BANKS AND STACK

Popping From Stack

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:

		After POP 3	After POP 5	After POP 2
0B	54			
0A	F9	F9		
09	76	76	76	
08	6C	6C	6C	6C
	Start SP = 0B	SP = 0A	SP = 09	SP = 08

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
```

LOOP AND JUMP INSTRUCTIONS

Nested Loop

- 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
```

LOOP AND JUMP INSTRUCTIONS

Conditional Jumps

- 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:   ...
```

LOOP AND JUMP INSTRUCTIONS

Conditional Jumps (cont')

- (cont')

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 R5,#0

```
        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
```

LOOP AND JUMP INSTRUCTIONS

Conditional Jumps (cont')

8051 conditional jump instructions

Instructions	Actions
JZ	Jump if A = 0
JNZ	Jump if A $\neq$ 0
DJNZ	Decrement and Jump if A $\neq$ 0
CJNE A,byte	Jump if A $\neq$ byte
CJNE reg,#data	Jump if byte $\neq$ #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

LOOP AND JUMP INSTRUCTIONS

Unconditional Jumps

- 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)

THANK YOU