Week 2
8051 Assembly Language Programming
Chapter 2

Outline

2.1 Inside the 8051

2.2 Introduction to 8051 Assembly programming

2.3 Assembling and running an 8051 program

2.4 The program counter and ROM space in the 8051

2.5 8051 data types and directives

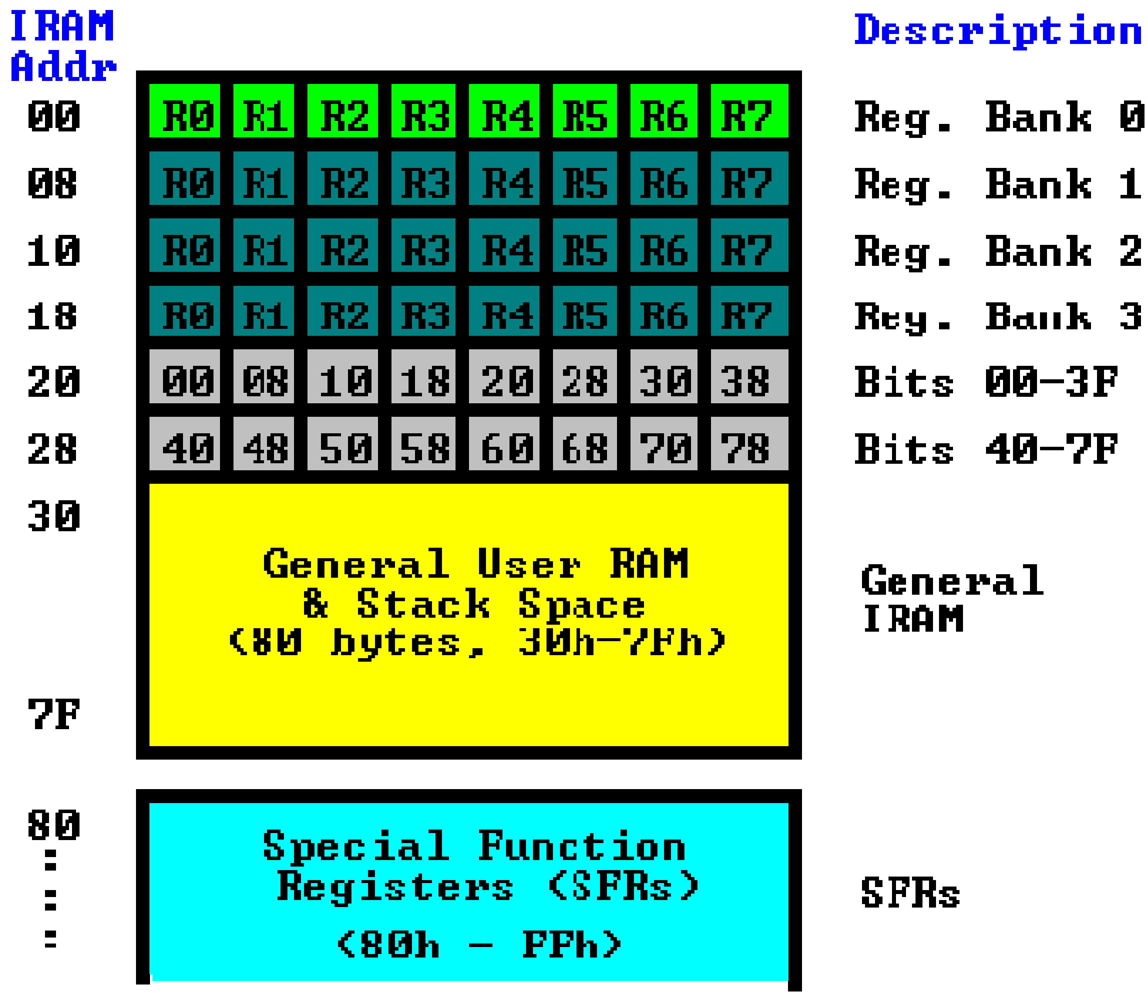
2.6 8051 flag bits and the PSW register

2.7 8051 register banks and stack

Inside the 8051

- **On-chip ROM** to save your program
  - Program is burned in ROM.
  - Program is fixed and is not supposed to change.
- **On-chip RAM** to save some temporary data generated in execution time
  - Data can be changed.
  - Data is lost when the 8051 powers down.
- **Registers** to store information temporarily
  - Some registers are used for internal operations of the 8051.
  - Some registers are located in RAM. Some have their special locations.

On-chip RAM



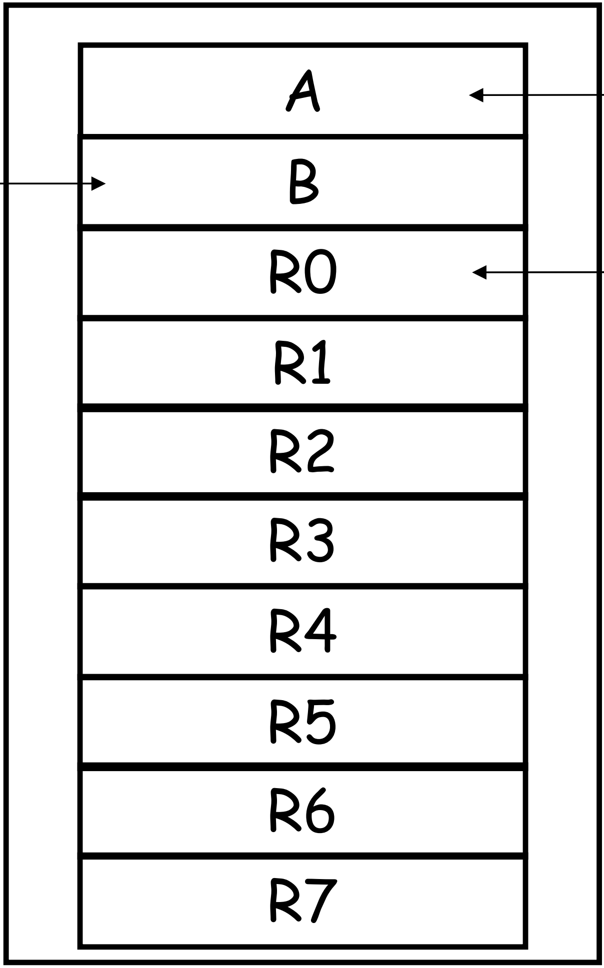
IRAM Addr
Description
00
R0 R1 R2 R3 R4 R5 R6 R7
Reg. Bank 0
08
R0 R1 R2 R3 R4 R5 R6 R7
Reg. Bank 1
10
R0 R1 R2 R3 R4 R5 R6 R7
Reg. Bank 2
18
R0 R1 R2 R3 R4 R5 R6 R7
Reg. Bank 3
20
00 08 10 18 20 28 30 38
Bits 00-3F
28
40 48 50 58 60 68 70 78
Bits 40-7F
30
7F
General User RAM & Stack Space (80 bytes, 30h-7Fh)
General IRAM
80
Special Function Registers (SFRs) (80h - FFh)
SFRs

8051 Registers

- Registers are used to store information temporarily.
- The 8051 has 8-bit registers and 16-bit registers.
  - many 8-bit registers in Figure 2-1 (a)
  - two 16-bit registers in Figure 2-1(b)

Main Registers

Register B helps Register A for arithmetic/logical operations, ex: MUL, DIV

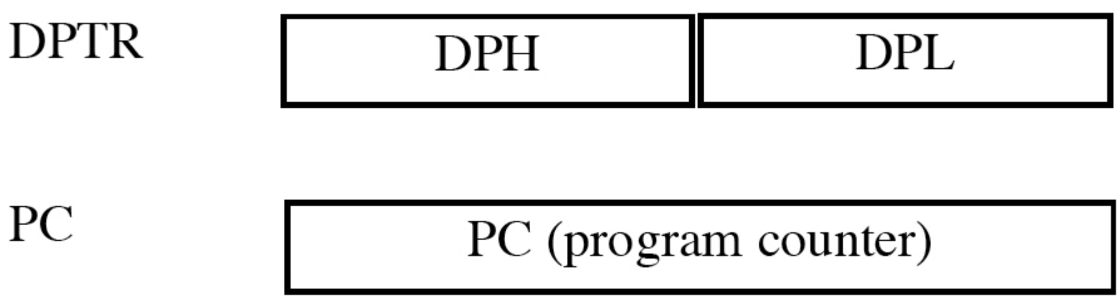


Accumulator for all arithmetic and logic instructions

Registers R0-R7 set of general-purpose registers

16 bit Registers

- DPTR: data pointer - the 16-bit address for the data located in program (ROM) or external RAM
  - DPL low byte of DPTR
  - DPH high byte of DPTR
- PC program counter - the address of the next instruction



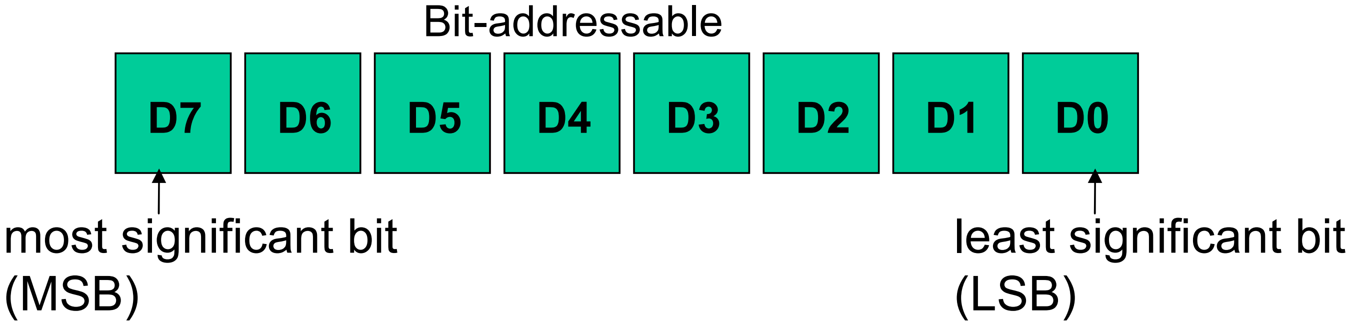
Special Function Registers SFR

80	P0	SP	DPL	DPH				PCON	87
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
90	P1								97
98	SCON	SBUF							9F
A0	P2								A7
A8	IE								AF
B0	P3								B7
B8	IP								B9
C0									C7
C8									CF
D0	PSW								D7
D8									DF
E0	ACC								E7
E8									EF
F0	B								F7
F8									FF

Blue background are I/O port SFRs
Yellow background are control SFRs
Green background are other SFRs

Bit addressable Registers

- The 8051 uses 8-bit data type.
  - Example: integer and character are 8 bits.
  - Bit-addressable (ex: P0) vs. not bit-addressable (ex: DPH)
- Any data larger than 8-bits must be broken into 8-bit chunks before it is processed.



Instruction Set Summary

Table A-1: 8051 Instruction Set Summary

1. Data Transfer : get or store data
   - MOV, PUSH, POP
2. Arithmetic Operations :
   - ADD, SUB, INC, DEC, MUL, DIV
3. Logical Operations :
   - ANL, ORL, XRL, CLR
4. Program Branching : jump, loop, call instruction
   - LCALL, RET, LJMP, JZ, JNZ, NOP

MOV Instruction

- Copy the source operand to the destination operand.

  MOV destination, source

  copy

```
MOV   A,#55H     ;load value 55H into reg. A
                 ;now A=55H (H: hexadecimal)
MOV   R6,#12      ;load 12 decimal into R6
                 ;now R6=12=0CH
MOV   R0,A       ;copy contents of A into R0
                 ;now A=55H, R0=55H
```

  - The pound sign "#" indicates that it is an *immediate* value.
  - You can write your command after the semicolon ";".

MOV - more

- Other examples

```
MOV   R5,#0F9H ;load F9H into R5
               ;now R5=F9H
```

- A 0 is used between the # and F to indicate that F is a hex number and not a letter.

```
MOV   R5,#F9H  ;illegal
```

- The value must be within 0-0FFH (or decimal 0-255).

```
MOV   R5,#425   ;illegal
```

- If no "#" exists, it means to load from a memory location.

```
MOV   A,17H    ;load the value held in memory
               ;location 17H to reg. A
```

MOV - more

- Other examples

```
MOV  A,#'4'    ;load ASCII '4' into A
               ;now A=34H
```

The immediate value can be copied to A, B, R0-R7.

ADD Instruction

- Add the source operand to register A and put the result in A.

ADD A, source

A + source → A

```
MOV   A,#25H   ;load 25H into A
MOV   R2,#34H  ;load 34H into R2
ADD   A,R2     ;add R2 to A=A+R2
               ;now A=59H, R2=34H
```

- Register A must be the destination of any arithmetic operation.

```
ADD   R0,A     ;illegal
```

ADD - more

- Other examples

```
MOV  A,#25H  ;load 25H into A
ADD  A,#34H  ;add 34H to A=A+34H=59H
```

  - The second value is called an *immediate* operand.
  - The format for Assembly language instruction, descriptions of their use, and a listing of legal operand types are provided in Appendix A.1. (to be discussed in Chap 5)

Assembly Language Programming

- Machine language
  - a program that consists of 0s and 1's.
  - CPU can work on machine language directly.
  - Example 7D25
- Low-level language
  - It deals directly with the internal structure of the CPU.
  - Programmers must know all details of the CPU.
  - Example MOV R5,#25H 8051 assembly language
- High-level language
  - Machine independent
  - Example a=37;C++

Assembly Language Programming

- Assembly languages were developed which provided mnemonics for the machine code instructions, plus other features.
  - Mnemonic: the instruction
    - Example: MOV, ADD
  - Provide decimal numbers, named registers, labels, comments
  - programming faster and less prone to error.
- Assembly language programs must be translated into machine code by a program called an assembler.

Example - Program 2-1

directives

instructions

```
        ORG 0H              ;start (origin) at
                            ;location 0
        MOV R5,#25H         ;load 25H into R5
        MOV R7,#34H         ;load 34H into R7
        MOV A,#0            ;load 0 into A
        ADD A,R5            ;add contents of R5 to A
                            ;now A = A + R5
        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                 ;end of asm source file
```

Assembly Language Programs

- An Assembly language program (see Program 2-1) is a series of statements.

  [label:] mnemonic [operands] [;comment]

  - Brackets indicate that a field is optional.
  - Label is the name to refer to a line of program code. A label referring to an instruction must be followed by a common ":".

    ```
    Here:  SJMP HERE
    ```

  - Mnemonic and operand(s) perform the real work of the program.
  - The comment field begins with a semicolon ";".

Mnemonic vs Directives

- Two types of assembly statements
  - Mnemonic tells the CPU what to do
    - Example MOV, ADD
    - These instructions are translated into machine code for the CPU to execute.
  - Pseudo-instruction gives directions to the assembler
    - Example ORG 0H, END
    - Pseudo-instructions are called directives, too.
    - Pseudo-instructions do not generate any machine code and are used only by the assembler.

8051 Directives

- ORG tells the assembler to place the opcode at ROM with a chosen start address.

  ORG start-address

  ```
  ORG    0200H    ;put the following codes
                  ;start at location 200H
  ```

  - ORG indicates the address of next instruction to be run.
- END indicates to the assembler the end of the source code.

  END

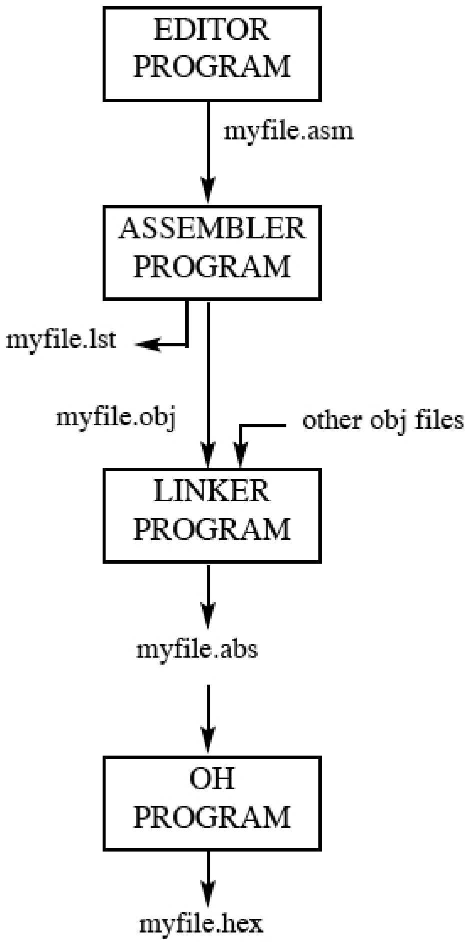
  ```
  END             ;end of asm source file
  ```
- EQU used for alias

  ```
  DATA EQU 25H
  ```
- Some assemblers use .ORG and .END

Steps in Assembly Language Programming

1. Use an editor to type in a program "myfile.asm" (may use other extensions)
2. The assembly source program is fed to an 8051 assembler. "myfile.lst" and "myfile.obj" are generated by the assembler.
3. A link program takes one or more object files to produce an absolute object file "myfile.abs". These abs files are used by 8051 trainers that have a monitor program.
4. The "abs"file is fed into a program called "OH" (object to hex converter) which creates a file "myfile.hex"
5. The "myfile.hex" file is to be burned into ROM by a special burner.

- New Windows-based assemblers combine 2-4 into one step



Program 2-1 - myfile.asm

```
      ORG   0H          ;start at location 0
      MOV   R5,#25H      ;load 25H into R5
      MOV   R7,#34H      ;load 34H into R7
      MOV   A,#0         ;load 0 into A
      ADD   A,R5         ;add contents of R5 to A
                         ;now A = A + R5
      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                ;end of asm source file
```

myfile.lst

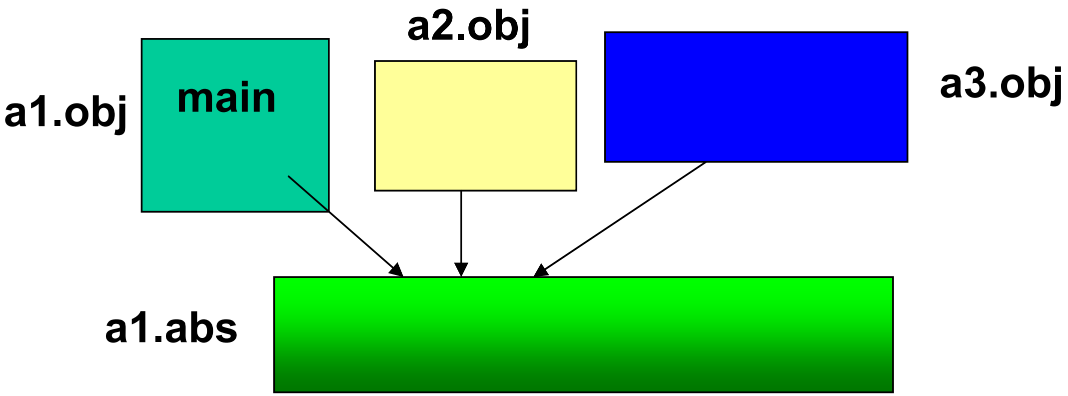
```
1   0000         ORG   0H          ;start at location 0
2   0000 7D25    MOV   R5,#25H     ;load 25H into R5
3   0002 7F34    MOV   R7,#34H     ;load 34H into R7
4   0004 7400    MOV   A,#0        ;load 0 into A
5   0006 2D      ADD   A,R5        ;add contents of R5 to A
6   0007                           ;now A = A + R5
7   0007 2F      ADD   A,R7        ;add contents of R7 to A
8   0008                           ;now A = A + R7
9   0008 2412    ADD   A,#12H      ;add to A value 12H
10  000A                           ;now A = A + 12H
11  000A 80FE HERE:SJMP HERE           ;stay in this loop
12  000A         END               ;end of asm source file
```

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

Linking

- When we write a large program, we may partition the job into several little programs.
- These little programs are assembled separately by different programmers.
- Finally, link them together and produce an absolute program with an absolute addressing.



Intel Hex File

- A record looks like
  :0300300002337A1E
- Breaking this line into it's components we have:
- **Record Length**: 03 (3 bytes of data)
  Address: 0030 (the 3 bytes will be stored at 0030, 0031, and 0032)
  Record Type: 00 (normal data)
  Data: 02, 33, 7A
  Checksum: 1E
- More than one record is possible
- Taking all the data bytes above, we have to calculate the checksum based on the following hexidecimal values:
- 03 + 00 + 30 + 00 + 02 + 33 + 7A = E2
- The two's complement of E2 is 1E which is, as you can, the checksum value.
- For our example

  :0A0000007D257F3474002D
  2F24129B

Program Counter

- The Program Counter PC 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 incremented to point to the next instruction.
- PC is called instruction pointer, too.

PC	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	16-bit register
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000H
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001H
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002H
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	FFFEH
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFFH

Program counter - more

- The PC in the 8051 is 16-bits wide.
  - The 8051 can access program addresses 0000 to FFFFH, a total of 64K bytes of code.
  - The exact range of program addresses depends on the size of on-chip ROM.
- When the 8051 is powered up, the PC has the value of 0000 in it.
  - That is, the address of the first executed opcode at ROM address is 0000H.
- We can examine the list file to loop the action of PC.

Program Counter - more

ORG 0H: put the instruction with the ROM address 0000H

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		

2 byte opcode

next instruction is 0+2=0002

Program is from 0000 to 0009. Total 10 bytes.

000A is the address of the next instruction if exists.

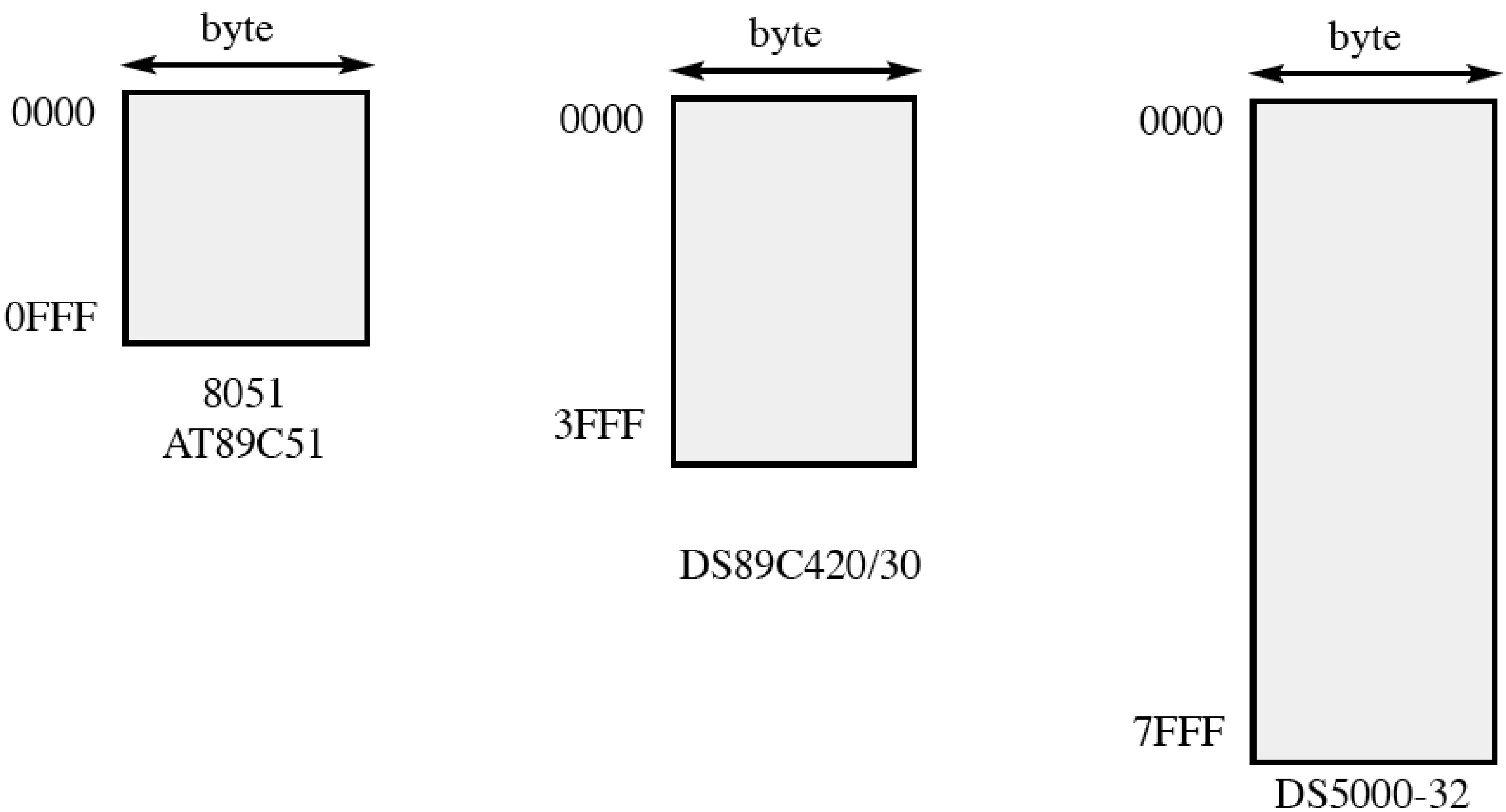
Operation with 8051

1. The 8051 is powered up. The PC is set to the value 0000H.
2. The CPU fetches the instruction with address 0000H and gets the machine code 7D. The CPU decodes it and asks the operand 25. The PC is set to the value 0002H.
3. The CPU fetches the instruction with address 0002H and gets the machine code 7F. The CPU decodes it and asks the operand 34. The PC is set to the value 0004H.
4. Do the same work until the 8051 is powered down.

8051 ROM Map

- The 8051 can access 64K bytes of ROM since the PC is 16-bit register.
  - 10000H bytes = 2^{16} bytes = 64K bytes
  - 0000 to FFFF address range
- However, the exact program size depends on the selected chip.
  - 8751, AT8951 have only 4K bytes.
  - AT89C52 has 8K bytes
  - Dallas Semiconductor's DS5000-32 has 32K bytes on-chip ROM.

Fig. 2-3 On-chip ROM address range



Data Types and Directives

- The 8051 microcontroller has only one data type.
  - 8-bit data
  - 00 to FFH data range (0 to 255 in decimal)
  - Programmers must take care of the meaning of data type by themselves.
  - Assembly Language defines some data representations and pseudo instructions.
    - DB, EQU
    - Assembler can translate these data representations to be processed by the 8051.

Define Byte (DB) Directive

- Define byte in the program.

  data-name DB data-value

  data-value → data-name

address	ROM
0500H	1C
0501H	35
0502H	39
0503H	

```
        ORG   500H
DATA1:  DB    28           ;decimal (1C in hex)
DATA2:  DB    00110101B;binary   (35 in hex)
DATA3:  DB    39H          ;hexadecimal
```

  - `data-name` is the label referring to the ROM address containing the content `data-value.`
  - It is an address, `DATA1=0500H`, `DATA2=0501H`.

DB - more

- Define ASCII number or characters

```
        ORG   510H
DATA1:  DB    "2591"
        ORG   518H
DATA2:  DB    "My name is Joe"
```

  - Assembler translates the ASCII numbers or characters to binary number.
  - ASCII Codes in Appendix F (p401)
  - The label is the address of first content at ROM. You can think of them as a table.

address	ROM
0510H	32
0511H	35
0512H	39
0513H	31
	⋮
0518H	4D
0519H	79
051AH	20
051BH	6E
051CH	61
051DH	6D
051EH	65
	⋮

DB - more

- The `data-label` is a 16-bit value.
- Usually, we use the register DPTR to point the first content of data.

```
        ORG   0H
        MOV   DPTR,#DATA1
        ...   ....
DATA1:  DB    "2591"
```

  - Reference to Chapter 5, Example 5-7.
  - Labels (`DATA1`) will be replaced by its value (`0510H`) by assembler.
  - Labels and DB are not transferred to opcodes.

address	ROM
0510H	32
0511H	35
0512H	39
0513H	31
	⋮

DATA1=0510H

DPTR=0510H

EQU

- Define a constant *without* occupying a memory location.

data-name EQU data-value

data-value $\rightarrow$ data-name

```
COUNT   EQU   25        ;25 in decimal =19H
...     ....
MOV     R3,#COUNT       ;R3=19H now
```

  - EQU associates a constant value with a data label.
  - When the label appears in the program, its constant value (`25`) will be substituted for name (`COUNT`) by assembler.
  - Data-names and EQU are not translated to opcodes.

Flags

- When the CPU performs arithmetic operations, sometimes an exception may occur.
  - Example overflow
- How does the CPU tell programmers that an exception occurs?
- Answer is the flags.
  - A flag is a bit to denote an exception occurred.
  - Carry flag CY
  - Auxiliary carry flag AC
  - Parity check P
  - Overflow OV

Carry Flag CY

- If there is an carry out from the D7 bit during an operation, CY is set; otherwise CY is cleared.
  - The CY is used to detect errors in arithmetic operations.
  - FFH+80H=17FH $\Rightarrow$ Carry out overflow
  - It is large than the data range of 8-bit register.

$$
\begin{array}{rr}
 & 1111\ 1111 \\
+ & 1000\ 0000 \\
\hline
1 & 0111\ 1111
\end{array}
$$

Overflow
CY=1

Auxiliary Carry Flag AC

- If there is a carry from D3 to D4 during an operation, AC is set; otherwise AC is cleared.
  - The AC flag is used by instructions that perform BCD (binary coded decimal) arithmetic. (See Chapter 8)
  - 88h+08h = 96h ⇒ Auxiliary carry out overflow

$$
\begin{array}{r}
1000\ 1000 \\
+\quad 0000\ 1000 \\
\hline
1001\ 0000
\end{array}
$$

Overflow AC=1

Add 6 and get the correct result

Overflow Flag OV

- OV is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit. (See Chapter 6)
  - 2's complement method is used in the signed numbers.
  - The range of 8-bit number is –128 to 127 in decimal.
- In 8-bit signed number operations, OV is set to 1 if either of the following two conditions occurs:
  1. There is a carry from D6 to D7 but no carry out of D7.
  2. There is a carry from D7 out but no carry from D6 to D7.

Example

- In the following example, there is a carry from D6 to D7 but no carry out of D7. Thus, the overflow bit is set.
  - 40H + 40H = 80 H $\Rightarrow$ Overflow the range 80H to 7FH
  - CY=0, OV=1, AC=0

$$
\begin{array}{rr}
 & 0100\ 0000 \\
+ & 0100\ 0000 \\
\hline
 & 1000\ 0000
\end{array}
$$

the result = 80H
= -128 in decimal
wrong!

Parity Flag P

- The parity flag reflects the number of 1s in the accumulator register only.
- If *A* contains an odd number of 1s, then P=1. If *A* has an even number of 1s, then P=0.
- Example
  - *A* = 0011 0011 $\Rightarrow$ # of 1s = 4 $\Rightarrow$ P = 0
  - *A* = 1011 0011 $\Rightarrow$ # of 1s = 5 $\Rightarrow$ P = 1

Examples

- Example 1

  ```
  MOV   A,#FFH
  ADD   A,#03H
  ```

  - A=FFH+03H=02H
  - CY=1, AC=1, OV=1

	1111 1111
+	0000 0011
1	0000 0010

CY=1, AC=1, OV=0

- Example 2

  ```
  MOV   A,#41H
  ADD   A,#4EH
  ```

  - A=41H+4EH=8FH
  - CY=0, AC=0, OV=1

	0100 0001
+	0100 1110
	1000 1111

CY=0, OV=1, AC=0

Where are the Flags?

80	P0	SP	DPL	DPH				PCON	87
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
90	P1								97
98	SCON	SBUF							9F
A0	P2								A7
A8	IE								AF
B0	P3								B7
B8	IP								B9
C0									C7
C8									CF
D0	PSW								D7
D8									DF
E0	ACC								E7
E8									EF
F0	B								F7
F8									FF

Blue background are I/O port SFRs
Yellow background are control SFRs
Green background are other SFRs

PSW

CY	AC	F0	RS1	RS0	OV	--	P

CY	PSW.7	Carry flag.
AC	PSW.6	Auxiliary carry flag.
F0	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 instuction cycle to indicate an odd/even number of 1 bits in the accumulator.

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

How Instructions Affect Flags?

- Table 2-1 shows how the instructions affect flag bits.
  - ADD affects CY, OV and AC.
  - "MUL AB" affects OV and CY=0 (CY always equals 0).
    - Multiply register A with register B. The result is placed in A and B where A has the lower byte and B has the higher byte.
    - If the product is greater than FFH, OV=1; otherwise, OV=0.

```
MOV   A,#5H   ;load 5H into A
MOV   B,#7H   ;load 7H into B
MUL   AB      ;B=0, A=35=23H, CY=0, OV=0
```

  - SETB C affects CY=1 (CY always equals 0) only.
    - SETB C is to set CY=PSW.7=1.

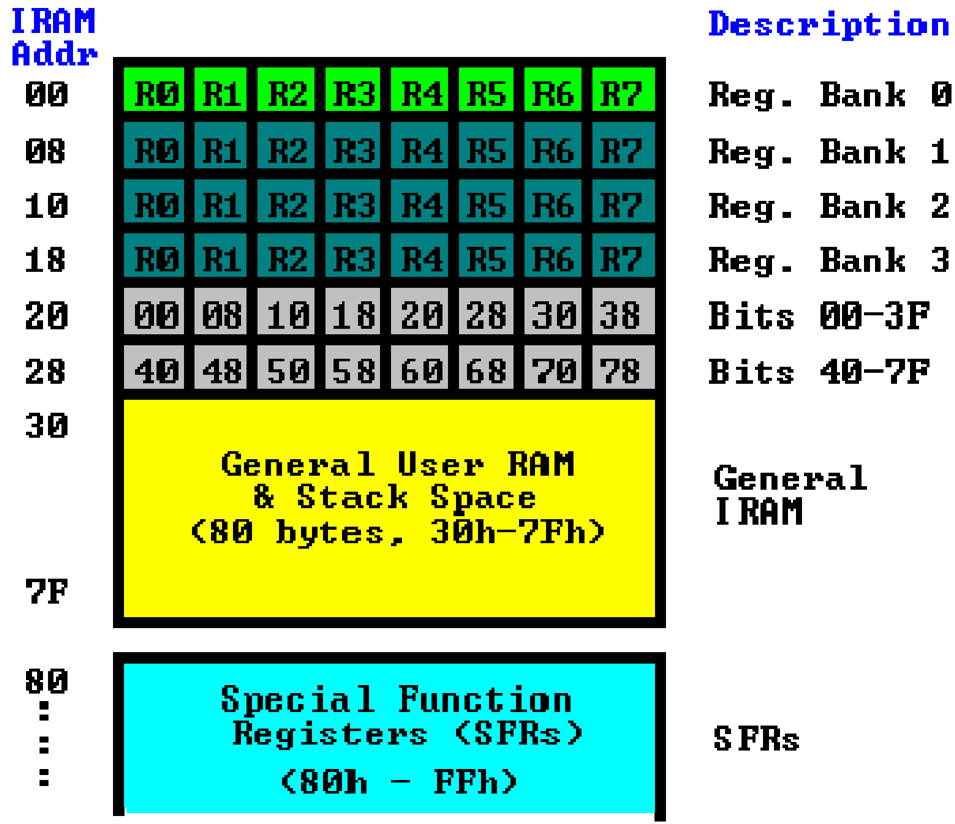
Example 2-2

- Show the status of the CY, AC, and P flags after the addition of 38H and 2FH in the following instructions.
- MOV A,#38H
- ADD A,#2FH ;after the addition A=67H,
- CY=0
- AC=1
- P=1

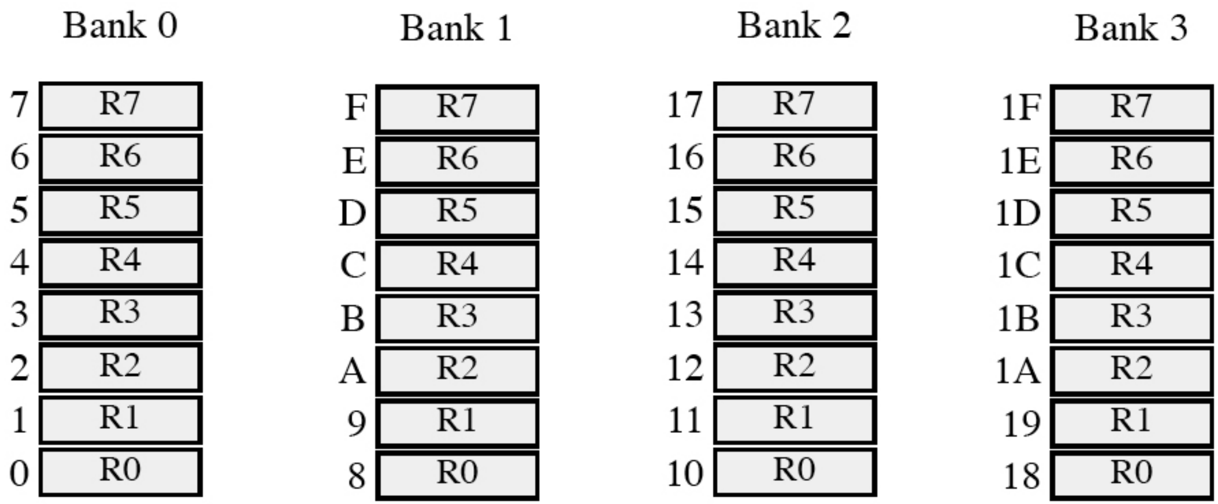
38	00111000
+ 2F	00101111
67	01100111

RAM in the 8051

- 128 bytes of RAM in the 8051
- These 128 bytes are divided into three different groups:
  - 32 bytes for register banks and the stack
    - 00 to 1FH RAM
  - 16 bytes for bit-addressable read/write memory
    - 20H to 2FH RAM
    - Each bit has a bit address (00-7FH)
  - 80 bytes for scratch pad
    - 30H to 7FH RAM



Register Banks in the 8051



- The 8051 uses 8 registers as general registers.
  - They are named as R0,R1,...,R7.
  - They form a register bank.
  - Only 8 register names (R0-R7). Are they enough?

- The 8051 provides 4 banks

  Bank 0   Bank 1
  00-07H   08H-0FH
  Bank 2   Bank 3
  10H-17H   18H-1FH
  All are called R0-R7.

How Banks are Chosen

- RS1 and RS0 decide the bank used by R0-R7.
  - RS1 and RS0 are bits 4 and 3 of PSW register, respectively.
- Default register bank
  - When the 8051 is powered up, RS1=RS0=0. That is, the RAM locations 00-07H are accessed with R0-R7.
  - If we don't change the values of RS1 and RS0, we use the default register bank: Bank 0.

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

More

- Bits D4 and D3 of the PSW are used to select the desired register bank.
  - D4 is referred to as PSW.4 RS1
  - D3 is referred to as PSW.3 RS0
- Use SETB and CLR

```
SETB   PSW.4   ;set RS1=1
CLR    PSW.3   ;clear RS0=0
```

  - Choose Bank 2 Addresses: 10H-17H for R0-R7

Bit-Addressable Memory

- The bit-addressable RAM locations are 20H to 2FH.
- Only 16 bytes of RAM are bit-addressable.
  - 16 * 8 bits = 128 bits (in decimal) = 80H bits (in hex)
  - They are addressed as 00 to 7FH
  - Note that the bit addresses 80H to F7H belong to SFR.
  - See Figure A-2

Byte address

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							

Bit-addressable locations

Example 2-5

State the contents of RAM locations after the following program:

```
MOV R0,#99H      ;load R0 with value 99H
MOV R1,#85H      ;load R1 with value 85H
MOV R2,#3FH      ;load R2 with value 3FH
MOV R7,#63H      ;load R7 with value 63H
MOV R5,#12H      ;load R5 with value 12H
```

Solution:

After the execution of above program we have the following:

RAM location 0 has value 99H   RAM location 1 has value 85H

RAM location 2 has value 3FH   RAM location 7 has value 63H

RAM location 5 has value 12H

Example 2-6

Repeat Example 2-5 using RAM addresses instead of register names.
Solution:

This is called *direct addressing mode* and uses the RAM address location for the destination address. See Chapter5 for a more detailed discussion of addressing modes.

with #: it's a value.

```
MOV 00,#99H     ;load R0 with value 99H
MOV 01,#85H     ;load R1 with value 85H
MOV 02,#3FH     ;load R2 with value 3FH
MOV 07,#63H     ;load R7 with value 63H
MOV 05,#12H     ;load R5 with value 12H
```

no #: it's an address.

Example 2-7

State the contents of the RAM locations after the following program:

```
SETB PSW.4      ;select bank 2
MOV R0,#99H     ;load R0 with value 99H
MOV R1,#85H     ;load R1 with value 85H
MOV R2,#3FH     ;load R2 with value 3FH
MOV R7,#63H     ;load R7 with value 63H
MOV R5,#12H     ;load R5 with value 12H
```

Solution:

By default, PSW.3=0 & PSW.4=0

"SETB PSW.4" sets RS1=1 and RS0=0 $\Rightarrow$ Register bank 2.

Register bank 2 uses RAM locations 10H – 17H.

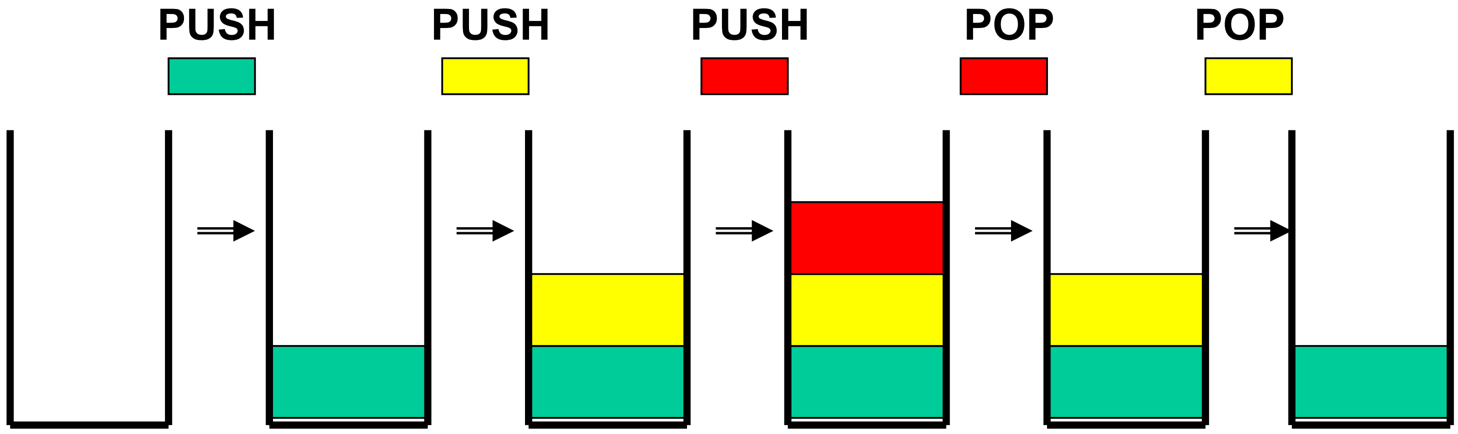
RAM location 10H has value 99H   RAM location 11H has value 85H

RAM location 12H has value 3FH   RAM location 17H has value 63H

RAM location 15H has value 12H

Stack

- Stack: a section of RAM to store data items
- Two operations on the stack
  - PUSH puts an item onto the *top* of the stack
  - POP removes an item from the *top* of the stack

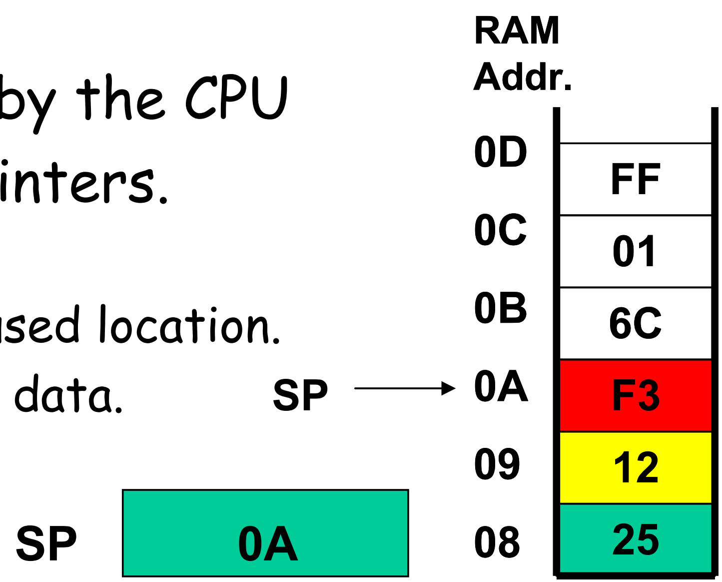


Role of stack

- To save the temporary data
- To save the return address
  - CPU wants to execute a subroutine.
  - The CPU uses the stack to save the address of the instruction just below the CALL instruction It is the return address.
  - That is how the CPU knows where to resume when the CPU returns from the called subroutine.
  - See Chapter 3.

Stack - more

- The stack is a section of RAM used by the CPU to store information temporarily.
- The stack is in the 8051 RAM location 08H to 1FH.
- How the stack is accessed by the CPU
- The answer is SP: Stack Pointers.
  - SP is an 8-bit register.
  - SP always points to the last used location.
  - SP stores the address of top data.



Where is SP

80	P0	SP	DPL	DPH				PCON	87
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
90	P1								97
98	SCON	SBUF							9F
A0	P2								A7
A8	IE								AF
B0	P3								B7
B8	IP								B9
C0									C7
C8									CF
D0	PSW								D7
D8									DF
E0	ACC								E7
E8									EF
F0	B								F7
F8									FF

Blue background are I/O port SFRs
Yellow background are control SFRs
Green background are other SFRs

SP Stack Pointer

- SP register is used to access the stack.
  - When the 8051 is powered up i.e., no data in the stack, the SP register contains value 07H.
  - The stack size is 08H-1FH (24 bytes).
  - The locations 20-2HF of RAM are reserved for bit-addressable memory and must not be used by the stack.
  - If in a given program we need more than 24 bytes of stack, we can change the SP to point to RAM location 30H - 7FH.

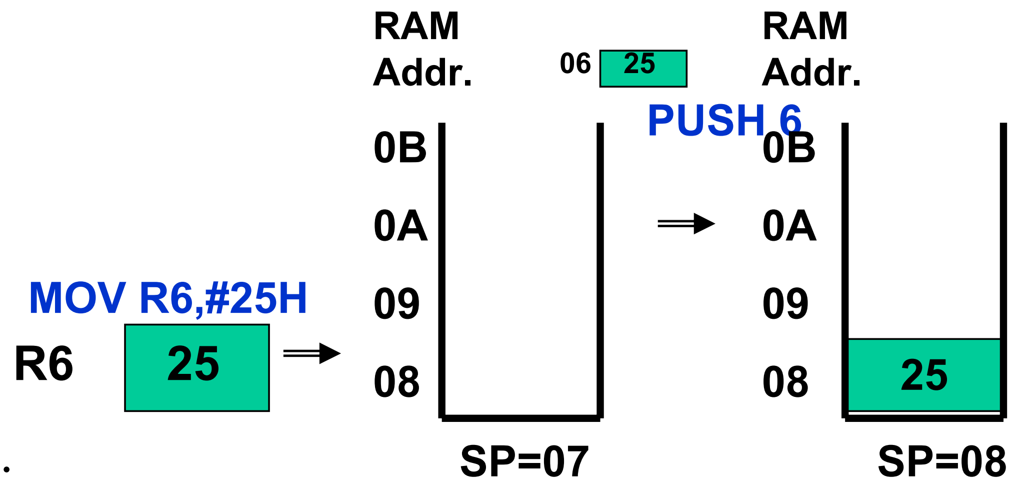
```
MOV    SP,#5FH   ;make RAM location 60H is
                 ;the first stack location
```

PUSH

- Put the content of the RAM-address into the top of the stack.

  PUSH RAM-address

```
MOV  R6,#25H
PUSH 6
```



  - Register Bank 0 is used.
    - R6 has the address 06H.
  - First SP is incremented by 1 automatically and data saved
  - The storing of a CPU register in the stack is called a PUSH.

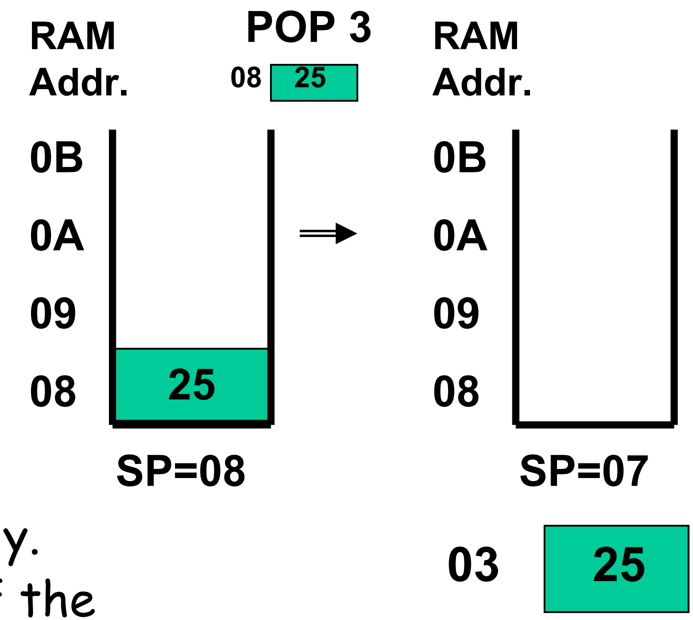
POP

- Remove the top value from the stack to the assigned RAM-address.

  POP RAM-address

  ```
  POP 3
  ```

  - Register Bank 0 is used.
    - R3 has the address 03H.
  - First read and then SP is decremented by 1 automatically.
  - The loading of the contents of the stack back into a CPU register is called a POP.

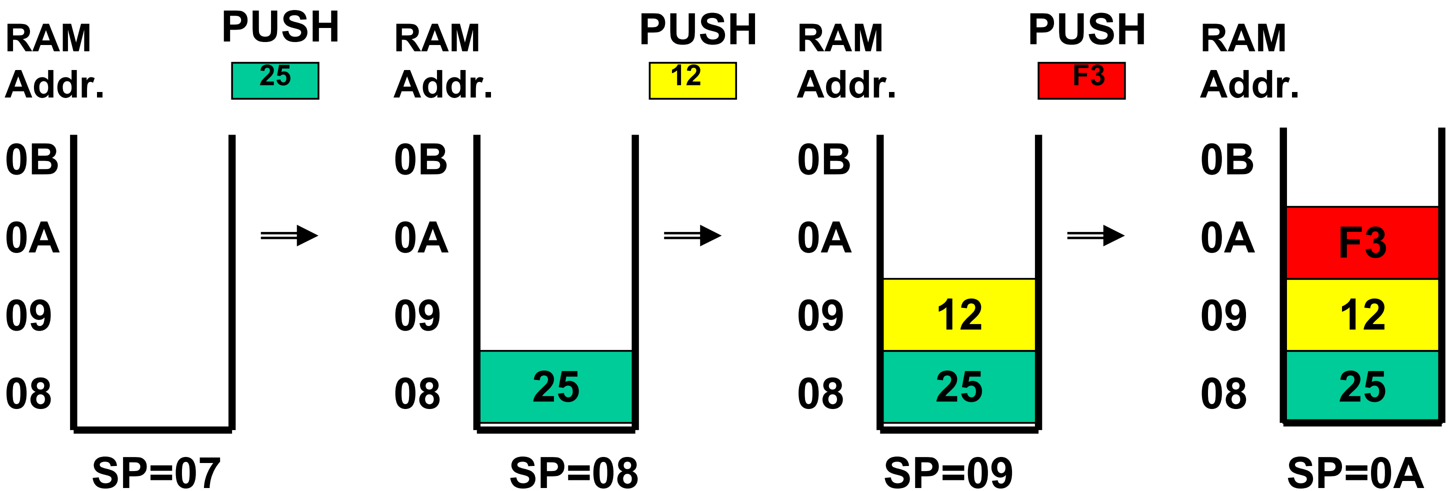


Example

Show the stack and stack pointer for the following

Assume the default stack area.

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



Example

Examining the stack, show the contents of the registers and SP after execution of the following instructions. All values HEX.

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

Solution:

0B	54
0A	F9
09	76
08	6C

Start SP=0B

After POP 3

0B	
0A	F9
09	76
08	6C

SP=0A

After POP 5

0B	
0A	
09	76
08	6C

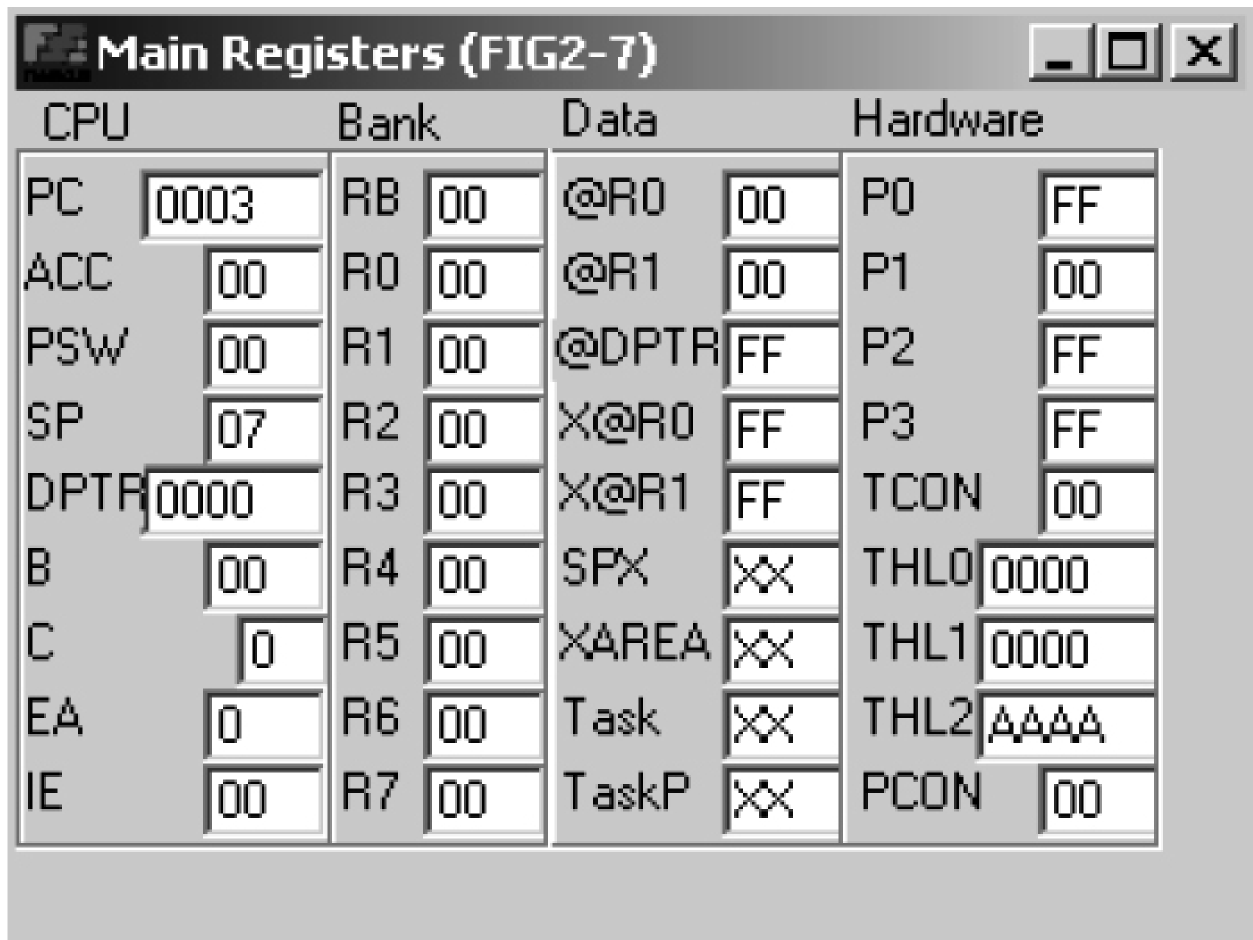
SP=09

After POP 2

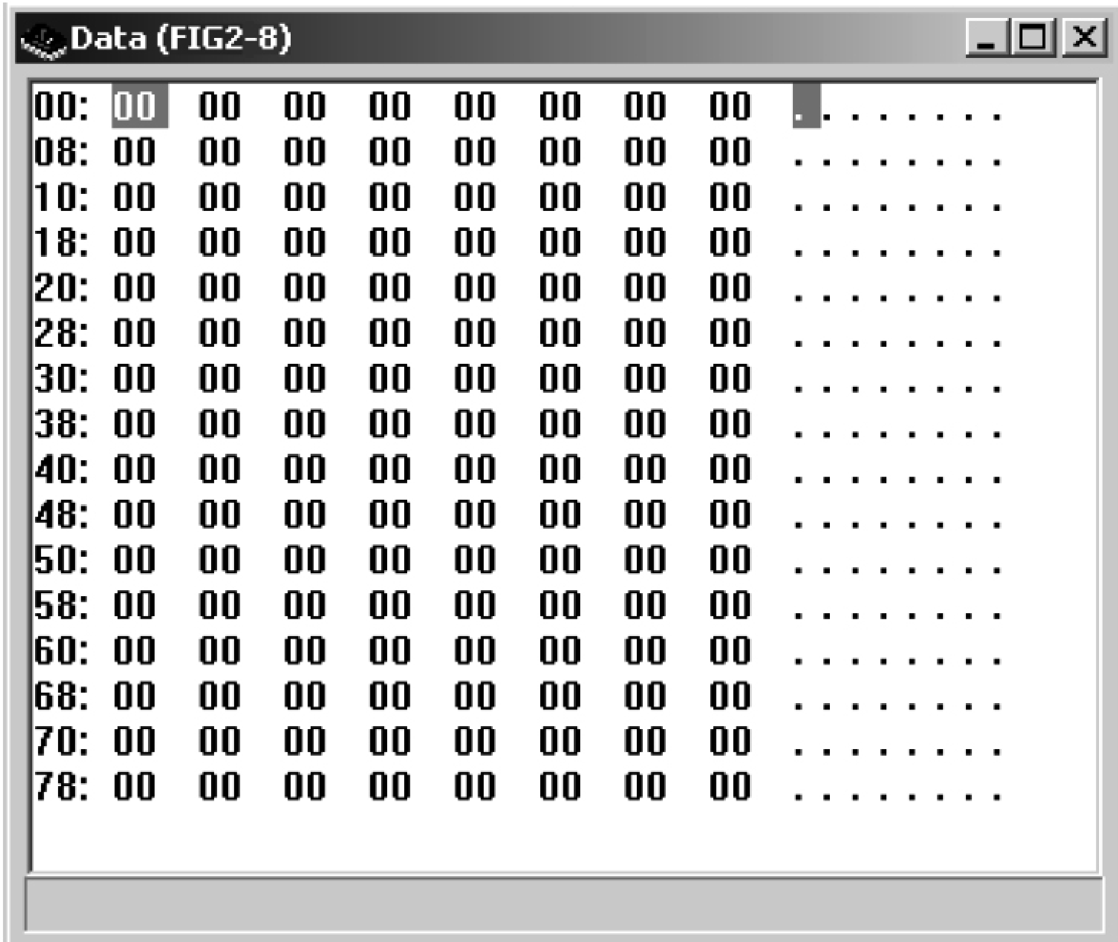
0B	
0A	
09	
08	6C

SP=08

Proview 32 - In-class example



More


Data (FIG2-8)
00: 00 00 00 00 00 00 00 00 ....
08: 00 00 00 00 00 00 00 00 ....
10: 00 00 00 00 00 00 00 00 ....
18: 00 00 00 00 00 00 00 00 ....
20: 00 00 00 00 00 00 00 00 ....
28: 00 00 00 00 00 00 00 00 ....
30: 00 00 00 00 00 00 00 00 ....
38: 00 00 00 00 00 00 00 00 ....
40: 00 00 00 00 00 00 00 00 ....
48: 00 00 00 00 00 00 00 00 ....
50: 00 00 00 00 00 00 00 00 ....
58: 00 00 00 00 00 00 00 00 ....
60: 00 00 00 00 00 00 00 00 ....
68: 00 00 00 00 00 00 00 00 ....
70: 00 00 00 00 00 00 00 00 ....
78: 00 00 00 00 00 00 00 00 ....