



Chapter 6

The 8051 Microcontroller

Programming in C

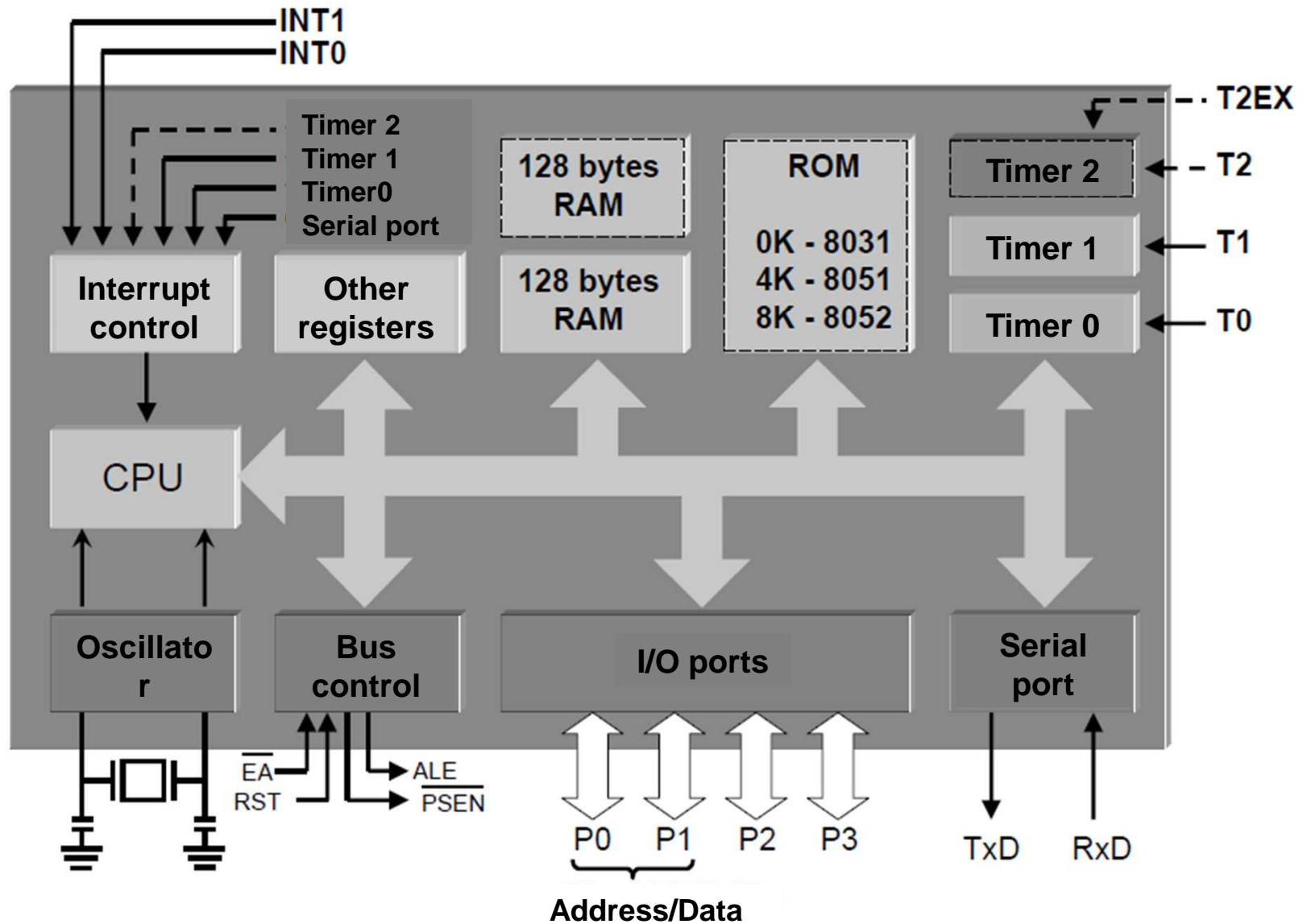


Faculty of Computer Science and Engineering
Department of Computer Engineering

Nguyen Quang Huy

huynguyen@cse.hcmut.edu.vn

Introduction



CPU

I/O

Timer

Serial

Interrupt

Why program 8051 in C?

- C programming
 - Easier, less time consuming vs. Assembly
 - Easier to modify and update
 - Function libraries
 - Portable to other microcontroller with little modification
- Disadvantage
 - Larger hex file size

Data types

- The size of hex file is the main concern
 - Limited on-chip ROM
- Good understanding of C data types
 - *create smaller hex files*
 - unsigned char
 - char
 - unsigned int
 - int
 - sbit
 - bit and sfr

Unsigned char & char

- 8-bit data types, the character data types
 - The most natural choice
 - 8051 is an 8-bit microcontroller
 - Counter value, ASCII characters
- ***unsigned char***
 - range of 0 – 255 (00 – FFh)
- ***char***
 - Range of –128 to +127
 - Use the MSB D7 to represent – or +
- If no “unsigned” keyword
 - ***char as the default***

Example 1

```
#include <REG51.h>
void main(void)
{
    /* use unsigned char instead of int if possible */
    unsigned char z;
    for (z = 0; z <= 255; z++)
        P1 = z;
}
```

Example 2

```
#include <REG51.h>
void main(void)
{
    char myChar[] = "012345ABCD"; /* character */
    unsigned char z;                /* counter */
    for (z = 0; z <= 10; z++)
        P1 = myChar[z];
}
```


Unsigned int & int

- 16-bit data type
 - Define *16-bit variables* such as memory address
 - Set counter values of *more than 256*
 - Registers & memory accesses of 8051 are 8-bit chunks
 - Misuse of int variables → larger hex file
- **unsigned int**
 - Range of 0 to 65535 (0000 – FFFFh)
- **int**
 - Use the MSB D15 to represent – or +
 - 15 bits for the magnitude
 - Range of –32768 to +32767

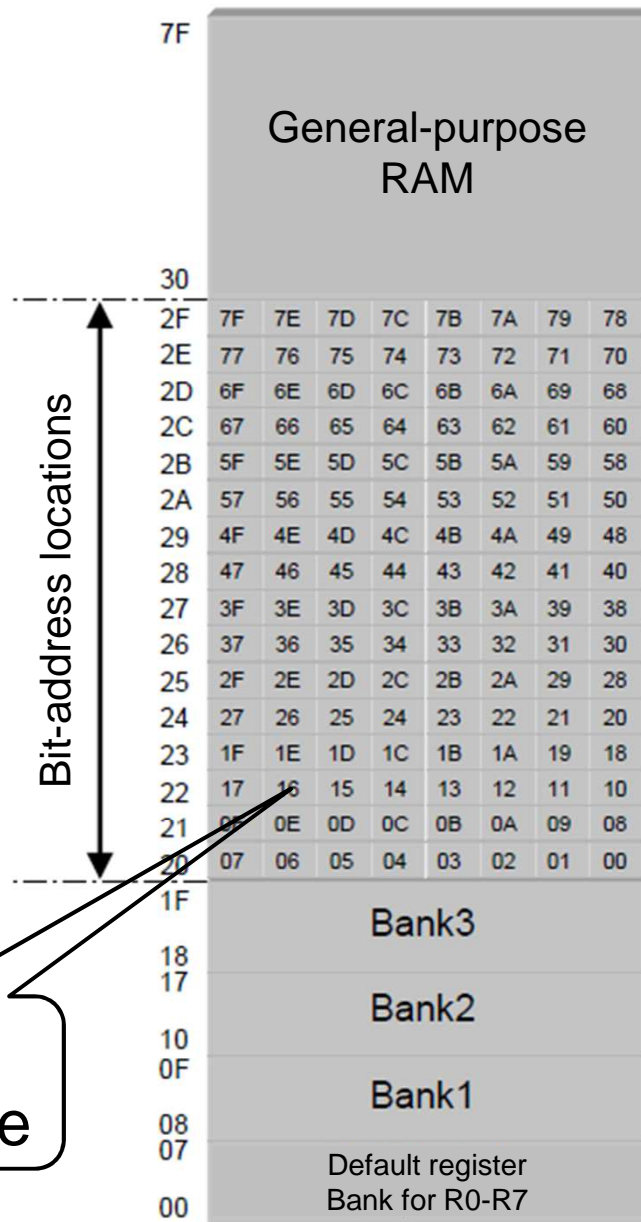
Example 3

```
#include <REG51.h>
void main(void)
{
    unsigned int z;          /* 16-bit variable */
    for (z = 0; z <= 50000; z++)
    {
        P1 = 0x55;           /* hex value */
        P1 = 0xAA;
    }
}
```

Bit, sbit and sfr

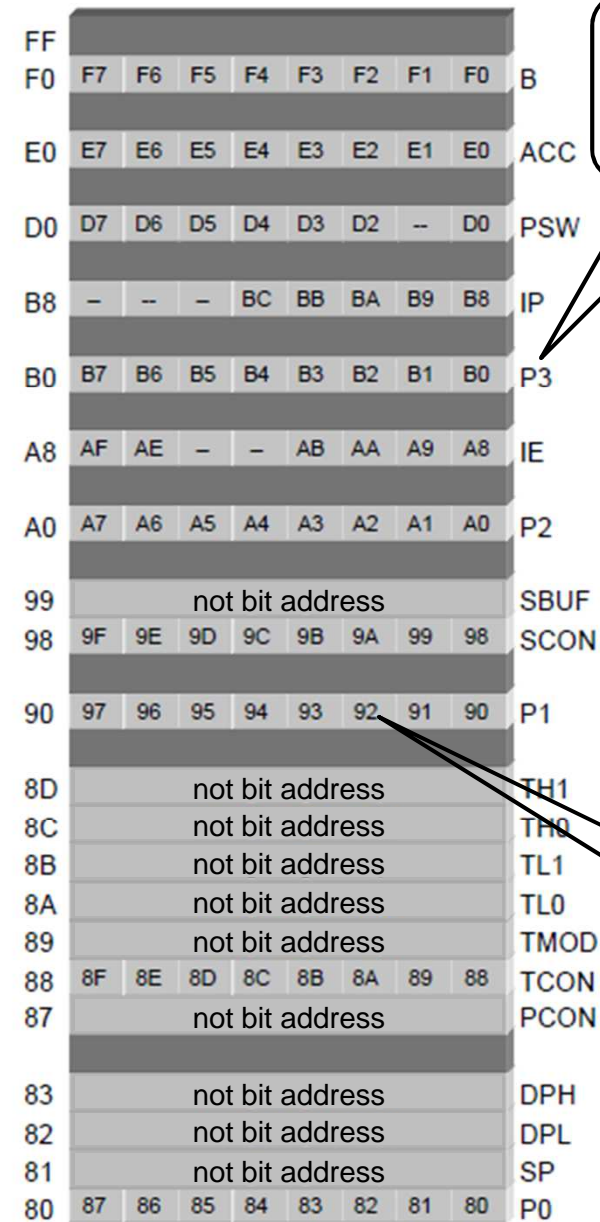
- ***bit***
 - Access to **single-bit** of *RAM bit-addressable* memory spaces from 20 – 2Fh
- ***sbit***
 - Access to **single-bit** of *SFR bit-addressable* memory spaces from 80 – FFh
- ***sfr***
 - Access the **byte-size** SFR register

Byte address



bit
data type

Byte address



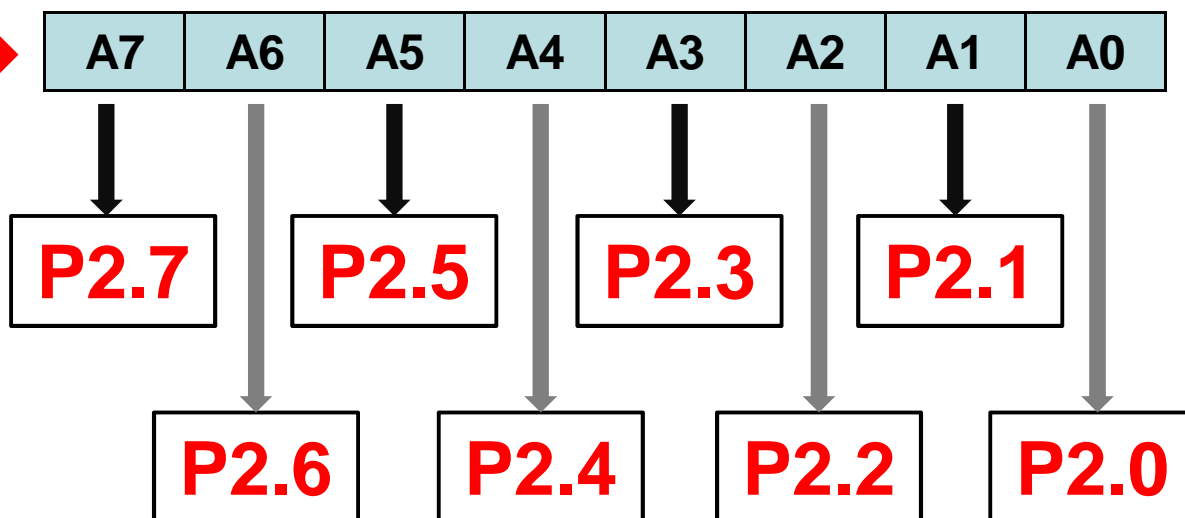
sfr
data type

sbit

Byte address

| | | | | | | | | | |
|----|-----------------|----|----|----|----|----|----|----|------|
| FF | | | | | | | | | |
| F0 | F7 | F6 | F5 | F4 | F3 | F2 | F1 | F0 | B |
| E0 | E7 | E6 | E5 | E4 | E3 | E2 | E1 | E0 | ACC |
| D0 | D7 | D6 | D5 | D4 | D3 | D2 | -- | D0 | PSW |
| B8 | -- | -- | -- | BC | BB | BA | B9 | B8 | IP |
| B0 | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 | P3 |
| A8 | AF | AE | -- | -- | AB | AA | A9 | A8 | IE |
| A0 | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | P2 |
| 99 | not bit address | | | | | | | | SBUF |
| 98 | 9F | 9E | 9D | 9C | 9B | 9A | 99 | 98 | SCON |
| 90 | 97 | 96 | 95 | 94 | 93 | 92 | 91 | 90 | P1 |
| 8D | not bit address | | | | | | | | TH1 |
| 8C | not bit address | | | | | | | | TH0 |
| 8B | not bit address | | | | | | | | TL1 |
| 8A | not bit address | | | | | | | | TL0 |
| 89 | not bit address | | | | | | | | TMOD |
| 88 | 8F | 8E | 8D | 8C | 8B | 8A | 89 | 88 | TCON |
| 87 | not bit address | | | | | | | | PCON |
| 83 | not bit address | | | | | | | | DPH |
| 82 | not bit address | | | | | | | | DPL |
| 81 | not bit address | | | | | | | | SP |
| 80 | 87 | 86 | 85 | 84 | 83 | 82 | 81 | 80 | P0 |

P2 (RAM address)



Example 4

```
#include <REG51.h>

/* declare bit-addressable memory */
    bit myBit;          /* RAM space 20 - 2Fh */
    bit flag = 0x00;    /* Bit at addr 0x20 */

/* access a single bit of P1 (P1.0) */
    sbit myPin = 0x90; /* SFR area */
or  sbit myPin = P1^0; /* Px^y
                        /* x: port 0,1,2,3 */
                        /* y: pin (bit) 0-7 */

/* access SFR RAM space (byte) */
    sfr P0 = 0x80;      /* Port 0 address */
```

REG51.h

```
/* BYTE Register (SFR) */
```

```
sfr P0      = 0x80;
```

```
sfr P1      = 0x90;
```

```
sfr P2      = 0xA0;
```

```
sfr P3      = 0xB0;
```

```
...
```

```
/* BIT Register: PSW, TCON, IE... */
```

```
sbit CY      = 0xD7;
```

```
sbit AC      = 0xD6;
```

```
...
```

```
sbit TF1     = 0x8F;
```

```
sbit TR1     = 0x8E;
```

```
...
```

Summarize

| Data Type | Size in Bits | Data Range/Usage |
|---------------|--------------|------------------------------|
| unsigned char | 8-bit | 0 – 255 |
| (signed) char | 8-bit | -128 to +127 |
| unsigned int | 16-bit | 0 to 65535 |
| (signed) int | 16-bit | -32768 to +32767 |
| sbit | 1-bit | SFR bit-addressable only |
| bit | 1-bit | RAM bit-addressable only |
| sfr | 8-bit | RAM addresses 80h – FFh only |

- Other data types
 - enum, (unsigned) short int, (unsigned) long int, float, double, sfr16

I/O Programming

```
#include <REG51.h>
```

```
sbit myPin = P2^1;
```

```
void main(void)
```

```
{
```

```
    while (1)
```

```
    {
```

```
        myPin = ~myPin;    /* toggle P2.1 */
```

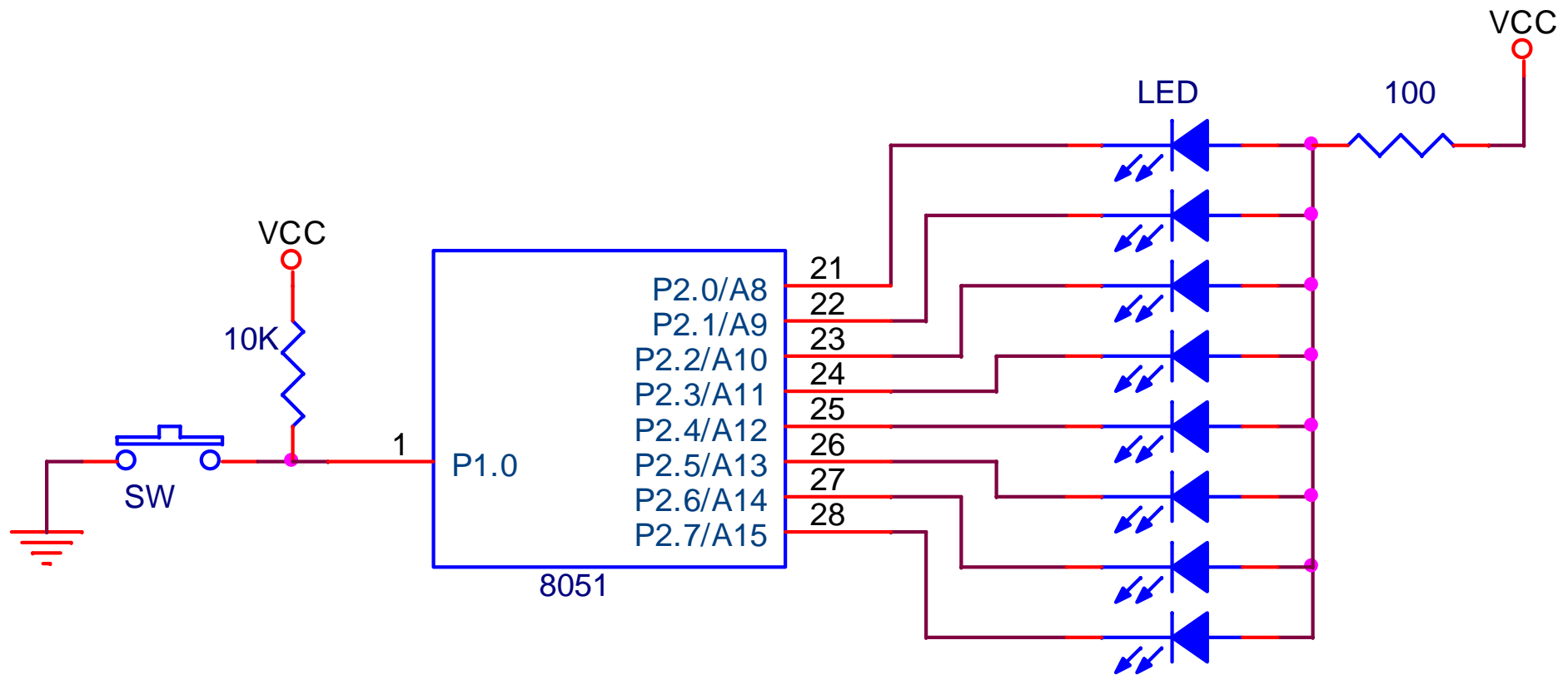
```
        P1++;              /* increase P1 */
```

```
    }
```

```
}
```

I/O Programming

- If SW is pushed, all LEDs are ON. Otherwise all LEDs are OFF.



I/O Programming

```
#include <REG51.h>

#define LED P2; /* P0-P3 are byte-accessable */
sbit myPin = P1^0;
bit myBit; /* declare bit-addressable memory */

void main(void) {
    while (1) {
        myBit = myPin; /* Read P1.0 status */
        if (myBit == 0)
            P2 = 0x00; /* clear P2 - LED ON */
        else
            P2 = 0xFF; /* LED OFF */
    }
}
```

Logical and Bit-wise operators

- Logical operators
 - AND (&), OR (||) and NOT (!)
- Bit-wise operators
 - AND (&), OR (|), EX-OR (^), Inverter (~), Shift Right (>>) and Shift Left (<<)

| | | AND | OR | EX-OR | Inverter |
|---|---|-------|-------|-------|----------|
| A | B | A & B | A B | A ^ B | ~B |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | |
| 1 | 1 | 1 | 1 | 0 | |

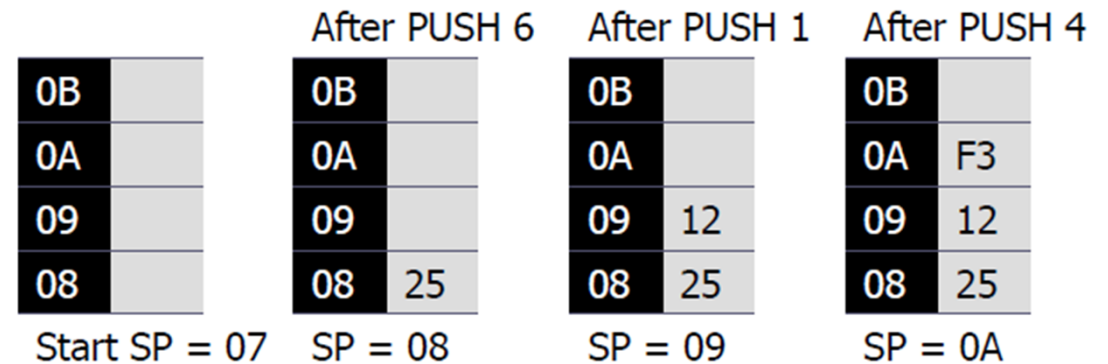
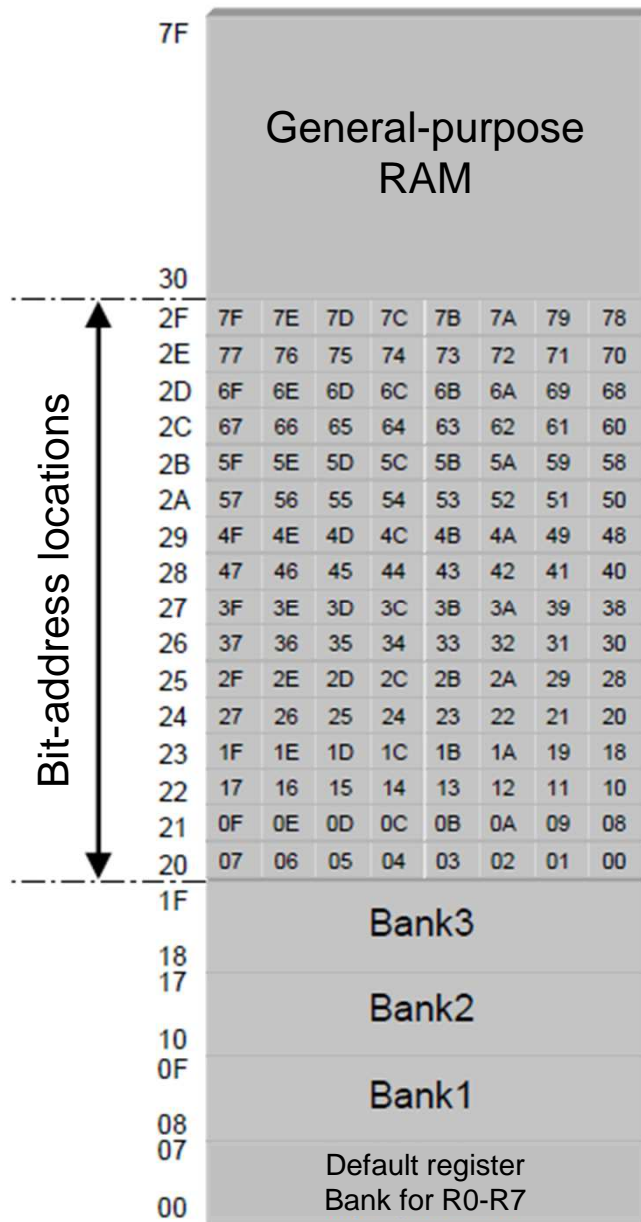
Example

```
#include <reg51.h>
void main(void)
{
    P0 = 0x35 & 0x0F;    // ANDing P0 = 0x05
    P1 = 0x04 | 0x68;    // Oring
    P2 = 0x54 ^ 0x78;    // XORing
    P0 = ~0x55;          // Inversing
    P1 = 0x9A >> 3;      // Shifting right 3
    P2 = 0x77 >> 4;      // Shifting right 4
    P0 = 0x6 << 4;       // Shifting left 4
                        // P0 = 0x60
}
```

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 **08H** is the first location begin used for the stack by the 8051

Byte address



Stack Pointer
SP = 07h (default)

Reference

- “*The 8051 Microcontroller and Embedded Systems Using Assembly and C – 2nd*” - Muhammad Ali Mazidi, Janice Gillispie Mazidi, Rolin D.McKinlay
- “*The 8051 Microcontroller - 2nd*” - I. Scott Mackenzie, Prentice-Hall 1995