Principle and Interface Techniques of Microcontroller

--8051 Microcontroller and Embedded Systems Using Assembly and C

LI, Guang (李光) **Prof. PhD, DIC, MIET**

WANG, You (王酉) **PhD, MIET**

杭州 • 浙江大学 • 2021

Final Examination
（期末考试）

- 闭卷，允许带**一张手写A4纸。不允许打印、复印**，违者后果自负。

- 文具：允许使用计算器，可以带尺子

- 总成绩=**考试成绩×50%**+平时成绩×10%+实验课×40%

感觉系统（眼、耳、鼻、皮肤等）

记忆/存储

逻辑/运算

人脑

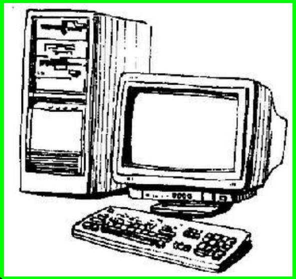
运动系统（骨骼、肌肉）

输入设备（摄像机、麦克风及其他传感器）

存 储 器

数据

指令



运 算 器

控 制 器

电脑（计算机）

输出设备（电机、阀门等动作器）

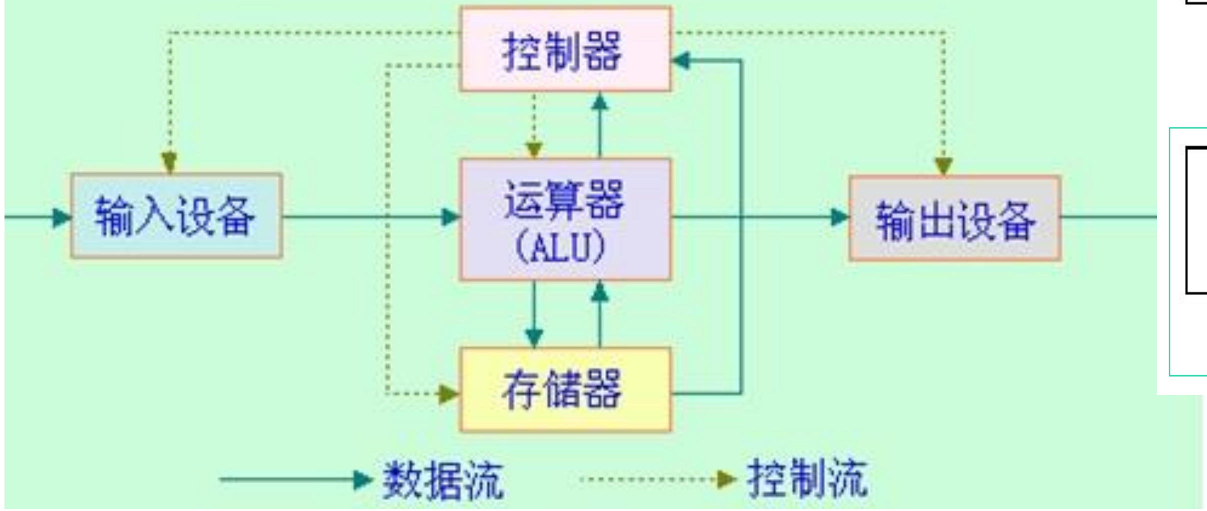
冯·诺依曼

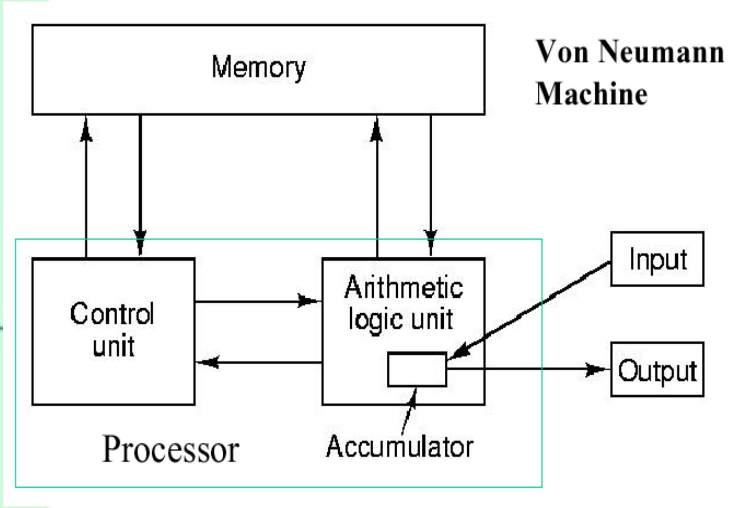
冯·诺依曼（美藉匈牙利科学家）型计算机：

1.计算机完成任务是由事先编号的程序完成的；
2.计算机的程序被事先输入到存储器中，程序运算的结果，也被存放在存储器中。
3.计算机能自动连续地完成程序。
4.程序运行的所需要的信息和结果可以通输入\输出设备完成。
5.计算机由运算器、控制器、存储器、输入设备、输出设备所组成。

迄今为止所有进入实用的电子计算机都是按其1946年提出的结构体系和工作原理设计制造

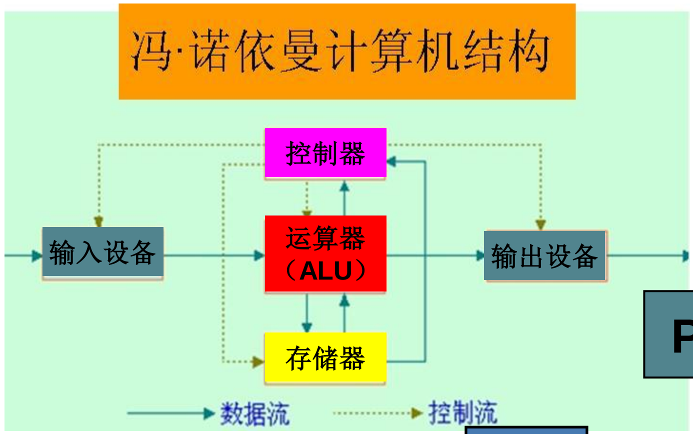
冯·诺依曼计算机结构

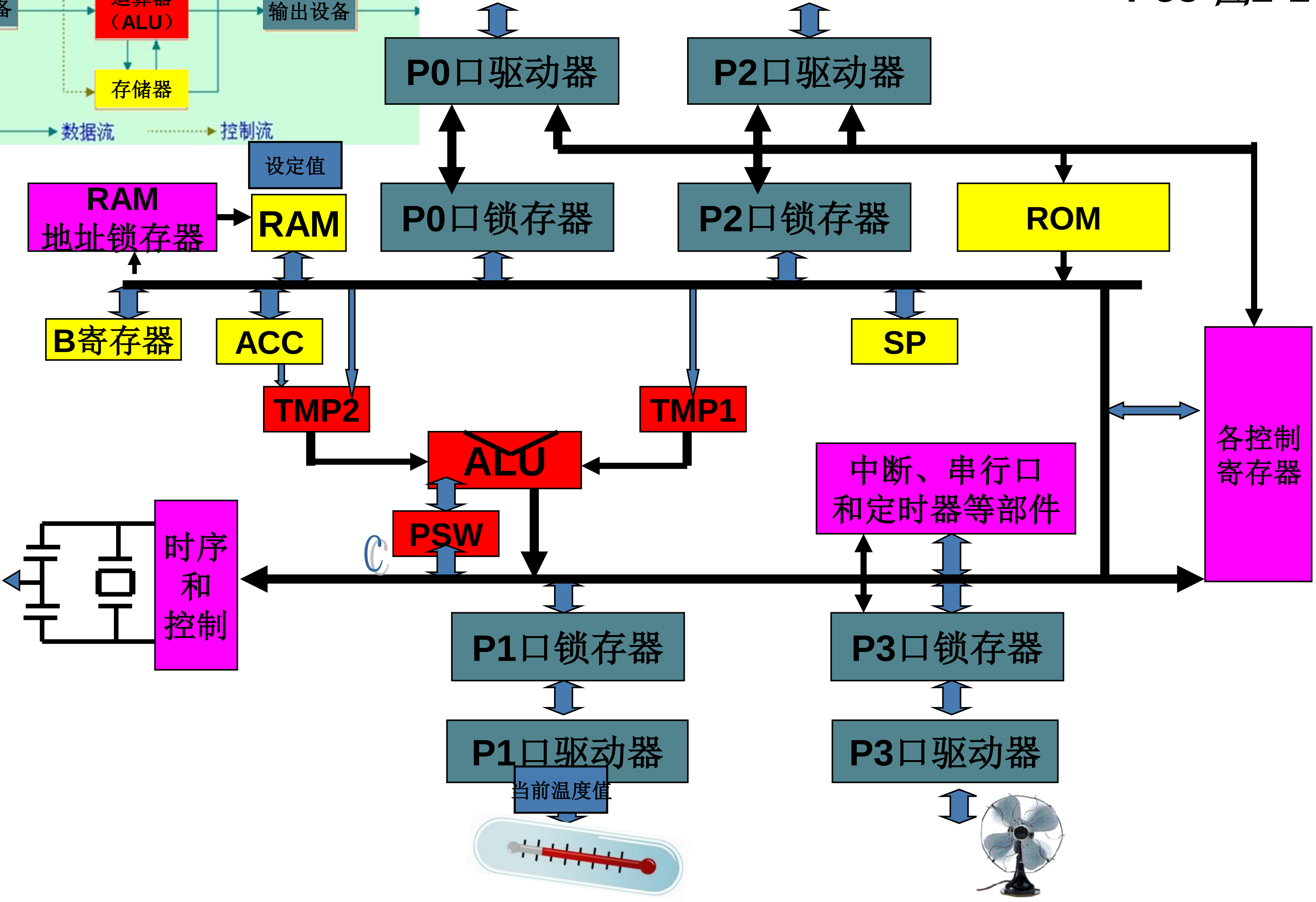




MCS-51硬件结构简图

P53 图2-10





Chapter 2
MCS-51 Assembly Language

Structure of Assembly Language

- An Assembly language instruction consists of four fields:

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

```
     ORG   0H                ;start(origin) at location0
     MOV   R5, #25H          ;
     MOV   R7, #34H          ;
     MOV   A,#0             ;load 0 into A
     ADD   A, R5             ;add contents of R5 to A.
                             ; now A
                             ;add c
                             ;now A = A + R7
                             ;add to A value 12H, now A = A + 12H

HERE: SJMP HERE        ;stay in this loop
     END
```

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

8051 CPU

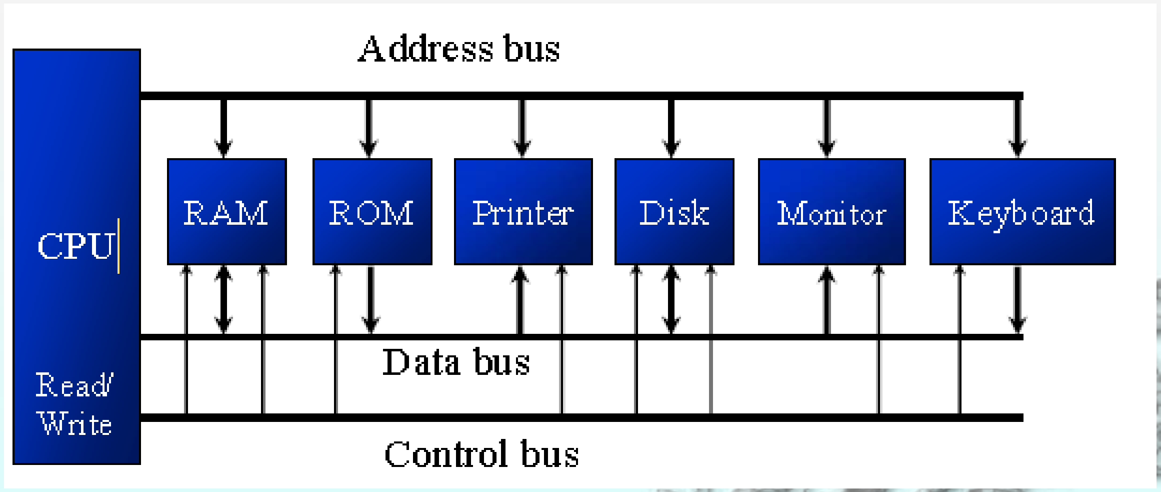
- The CPU is connected to memory and I/O through strips of wire called a bus

  Carries information from place to place:

  Address bus

  Data bus

  Control bus



- ALU: Accomplish arithmetic operation, logic operation, bit manipulation with cooperation of related registers (A, B, PSW).

A (ACC): For all arithmetic and logic instructions

B: For multiplication and division

PSW, also referred to as the flag register, is an 8 bit register. Only 6 bits are used

D7	D6	D5	D4	D3	D2	D1	D0
Cy	AC	F0	RS1	RS0	OV	_	P

PSW

D7	D6	D5	D4	D3	D2	D1	D0
Cy	AC	F0	RS1	RS0	OV	_	P

Cy (Carry): Carry flag

AC (Auxiliary Carry): Auxiliary carry flag

F0 (Flag): Available to the user for general purpose

RS1、 RS0： Register Bank selector

OV (Overflow) : Overflow flag

P (Parity): Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of 1 bits in the accumulator.

Stack

- The stack is a section of RAM information temporarily

  This information could be da

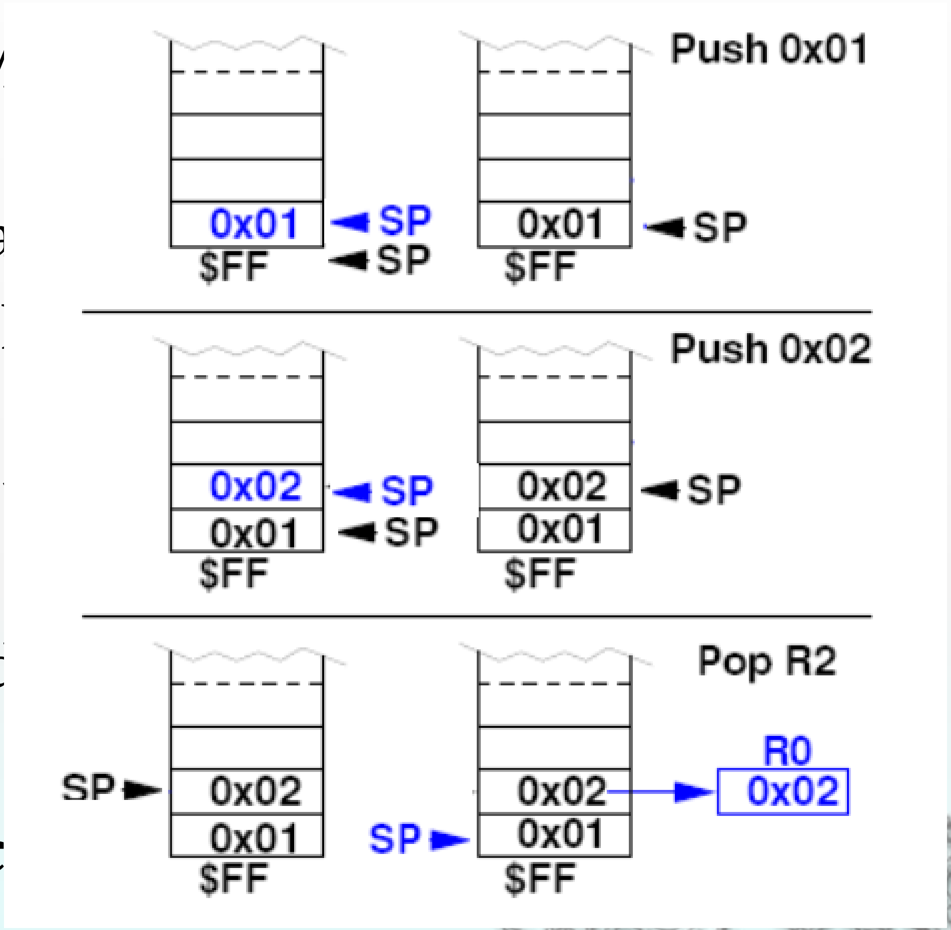
- The register used to access t pointer) register

  The stack pointer in the 805 means that it can take value

    When the 8051 is powered value 07

 RAM location 08 is the first loc

    stack by the 80



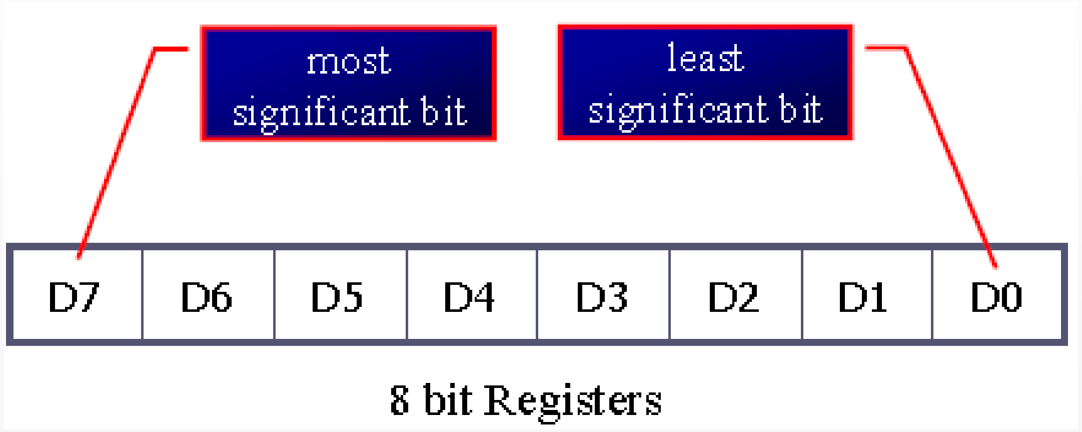
```
PUSH byte   ;increment stack pointer,
            ;move byte on stack
POP byte    ;move from stack to byte,
            ;decrement stack pointer
```

PC

- PC (program counter) points to the address of the next instruction to be executed

  As the CPU fetches the opcode from the program ROM, PC is increasing to point to the next instruction
- PC is 16 bits wide which means that it can access program addresses 0000 to FFFFH, a total of 64K bytes of code

8051 Registers



- **Register are used to store information temporarily, while the information could be:**

  a byte of data to be processed, or

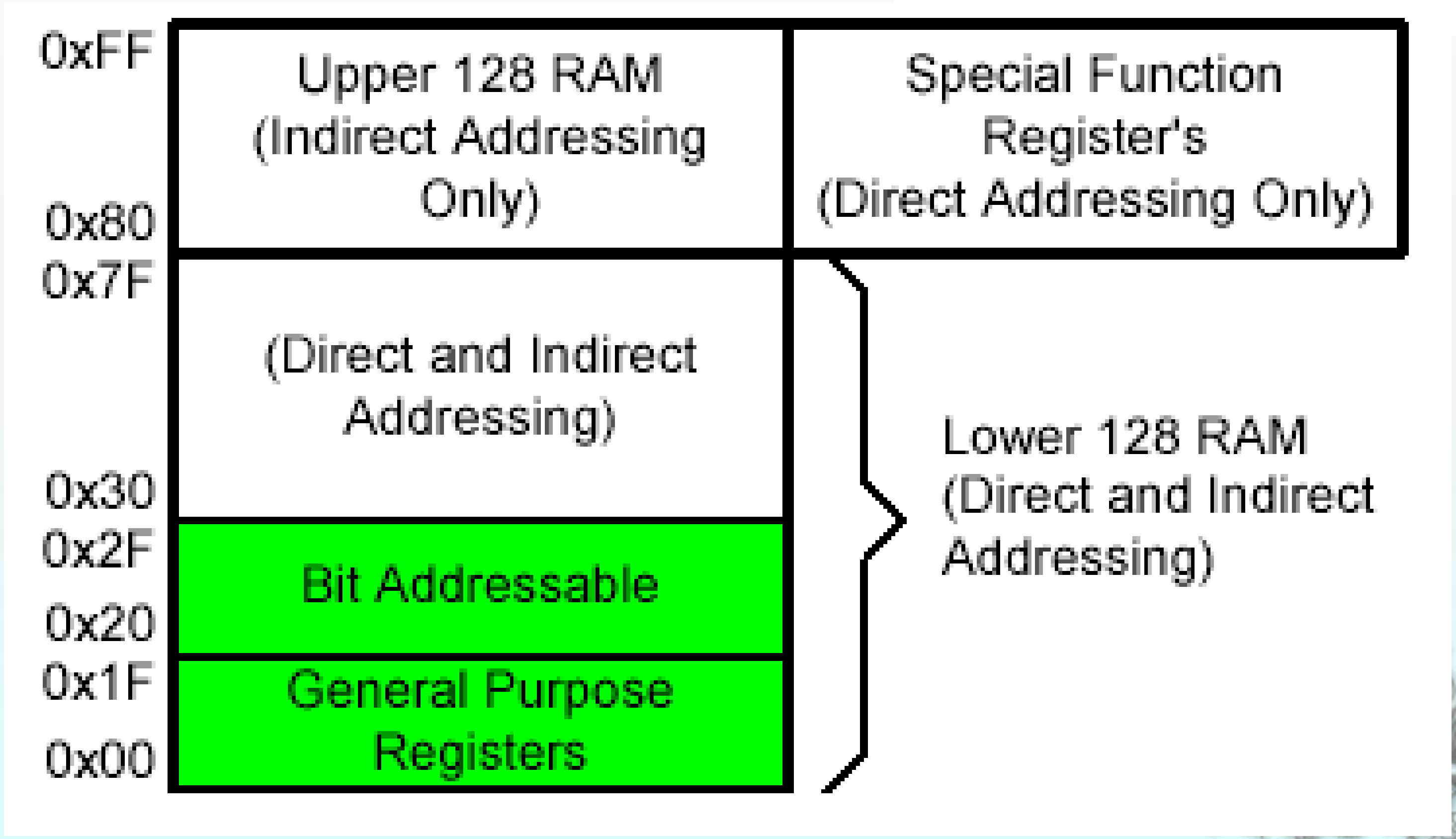
  an address pointing to the data to be fetched

- **The vast majority of 8051 register are 8-bit registers**

  There is only one data type, 8 bits, any data larger than 8 bits must be broken into 8-bit chunks before it is processed

RAM Memory Space Allocation

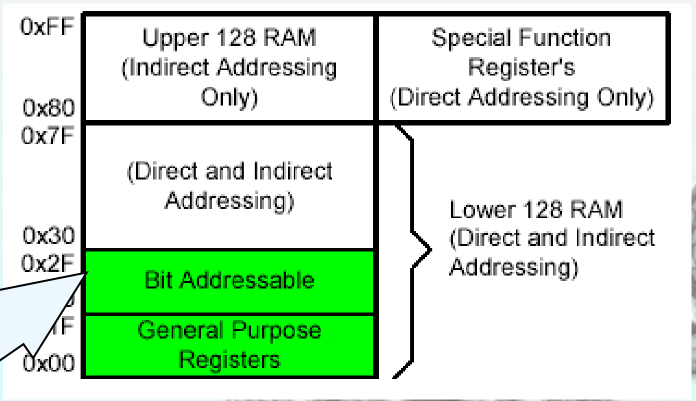
Internal RAM


0xFF
0x80
0x7F
0x30
0x2F
0x20
0x1F
0x00
Upper 128 RAM (Indirect Addressing Only)
Special Function Register's (Direct Addressing Only)
(Direct and Indirect Addressing)
Bit Addressable
General Purpose Registers
Lower 128 RAM (Direct and Indirect Addressing)

Bit Addressable Memory

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

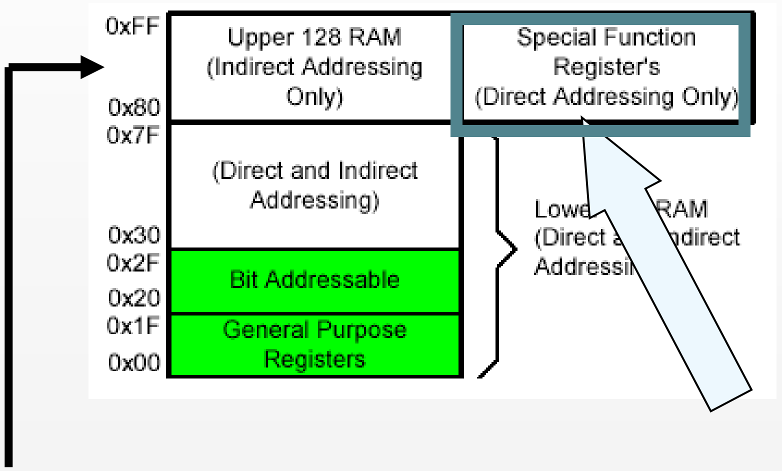
20h – 2Fh (16 locations X 8-bits = 128 bits)


0xFF
0x80
0x7F
0x30
0x2F
F
0x00
Upper 128 RAM (Indirect Addressing Only)
Special Function Register's (Direct Addressing Only)
(Direct and Indirect Addressing)
Bit Addressable
General Purpose Registers
Lower 128 RAM (Direct and Indirect Addressing)

Special Function Registers

DATA registers

CONTROL registers


0xFF
0x80
0x7F
0x30
0x2F
0x20
0x1F
0x00
Upper 128 RAM (Indirect Addressing Only)
Special Function Register's (Direct Addressing Only)
(Direct and Indirect Addressing)
Bit Addressable
General Purpose Registers
Lowe RAM (Direct a ndirect Addressi

Addresses 80h – FFh

Direct Addressing used to access SPRs

Data Transfer Instructions

--- MOV Instructions

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

6 basic types:

```
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
```

Looping

- 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

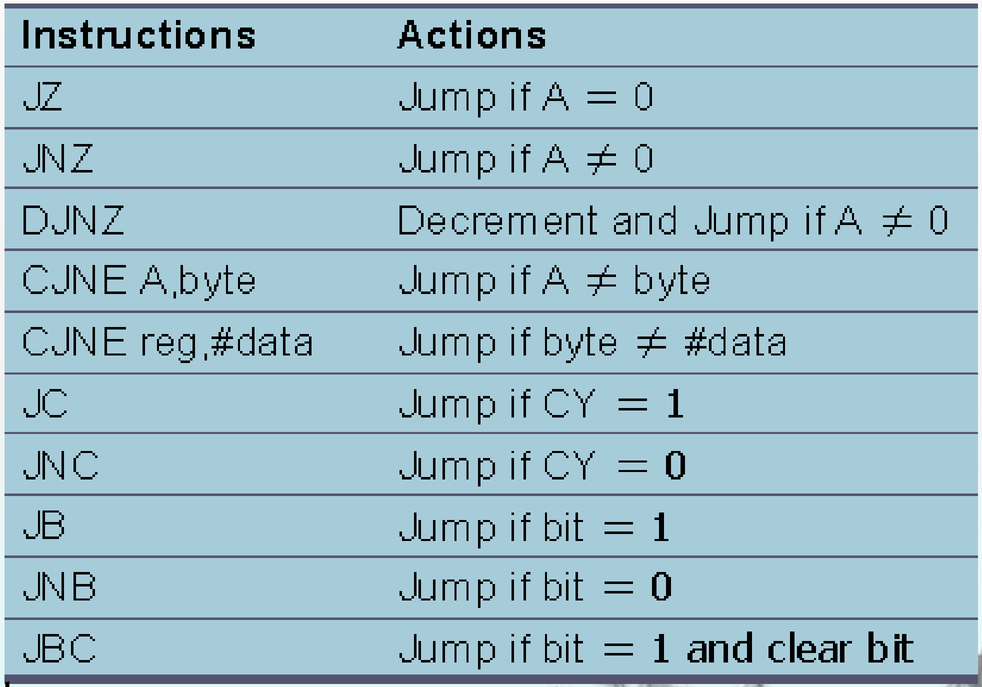
Counter can be R0 – R7 or RAM location

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

```
;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
```

Conditional Jumps

- 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

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)

CALL INSTRUCTIONS

- 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

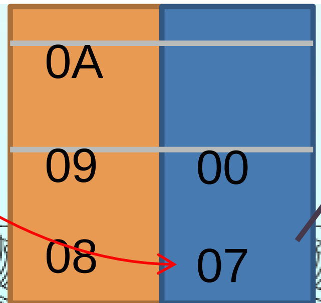
ACALL (absolute call)

2-byte instruction

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

CALL Instruction and Stack

```
0000                         ORG  0
0000  7455     BACK:         MOV  A, #55H        ;load A with 55H
0002  F590                   MOV  P1, A          ;send 55H to p1
0004  120300                 LCALL  DELAY        ;time delay
0007  74AA                   MOV  A, #0AAH       ;load A with AAH
0009  F590                   MOV  P1, A          ;send AAH to p1
000B  120300                 LCALL  DELAY
000E  80F0                   SJMP  BACK          ;keep doing this
0010
0010   ;-------this is the delay subroutine------
0300                         ORG  300H
0300           DELAY:
0300  7DFF                   MOV R5, #0FFH       ;R5=255
0302  DDFE     AGAIN:        DJNZ R5, AGAIN      ;stay here
0304  22                     RET                 ;return to caller
0305                         END                 ;end of asm file
```



Stack frame after the first LCALL

SP=09

Low byte goes first and high byte is last.

Time Delay for Various 8051 Chips

- 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

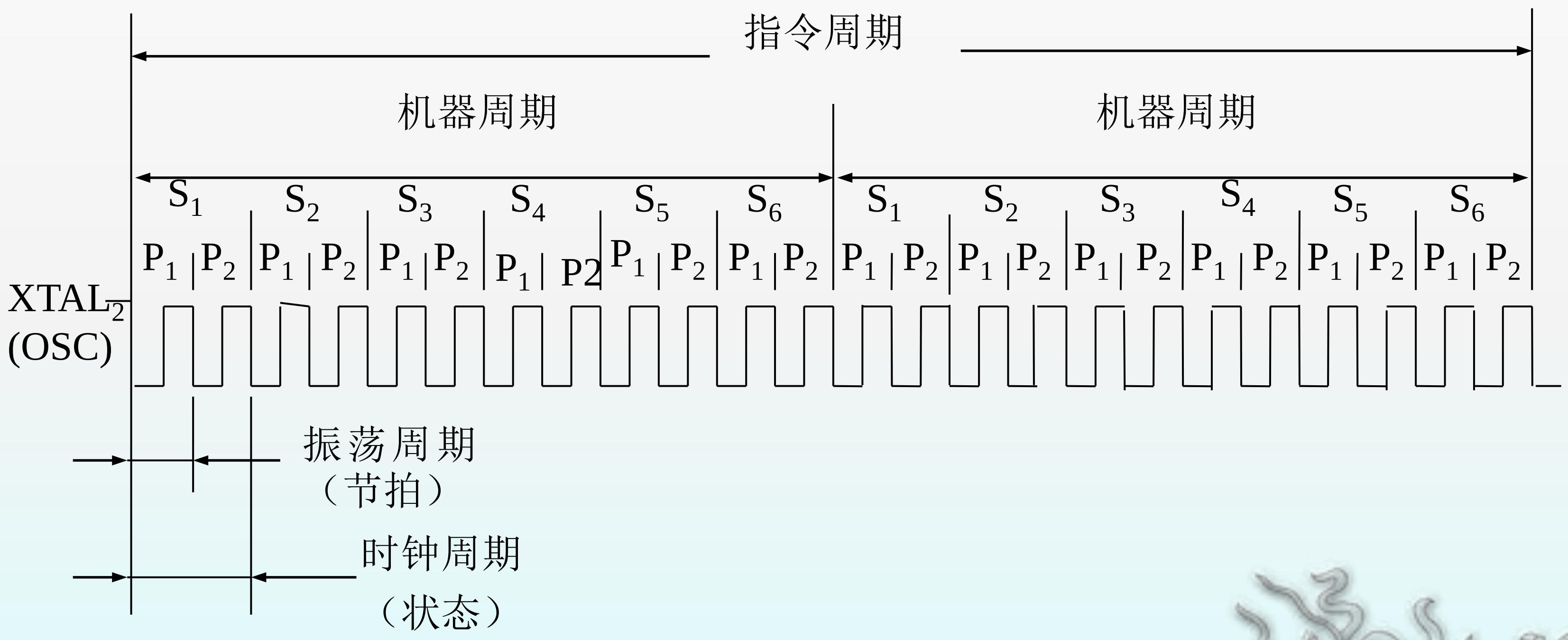
Find the period of the machine cycle for 11.0592 MHz crystal frequency

Solution:

11.0592/12 = 921.6 kHz;

machine cycle is 1/921.6 kHz = 1.085μs

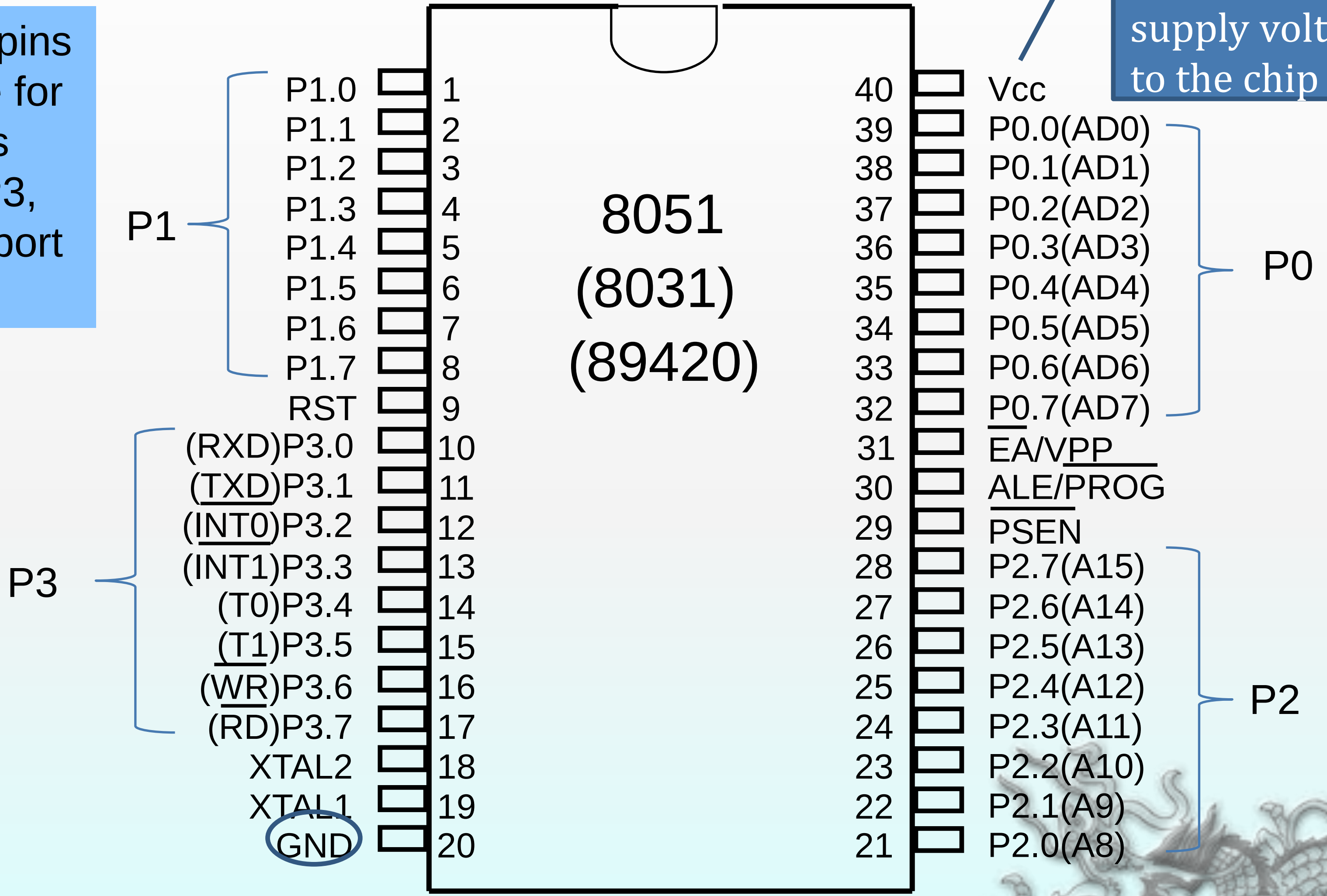
Timing


指令周期
机器周期
机器周期
S1 S2 S3 S4 S5 S6 S1 S2 S3 S4 S5 S6
P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2
XTAL2
(OSC)
振荡周期
（节拍）
时钟周期
（状态）

I/O Ports Configuration

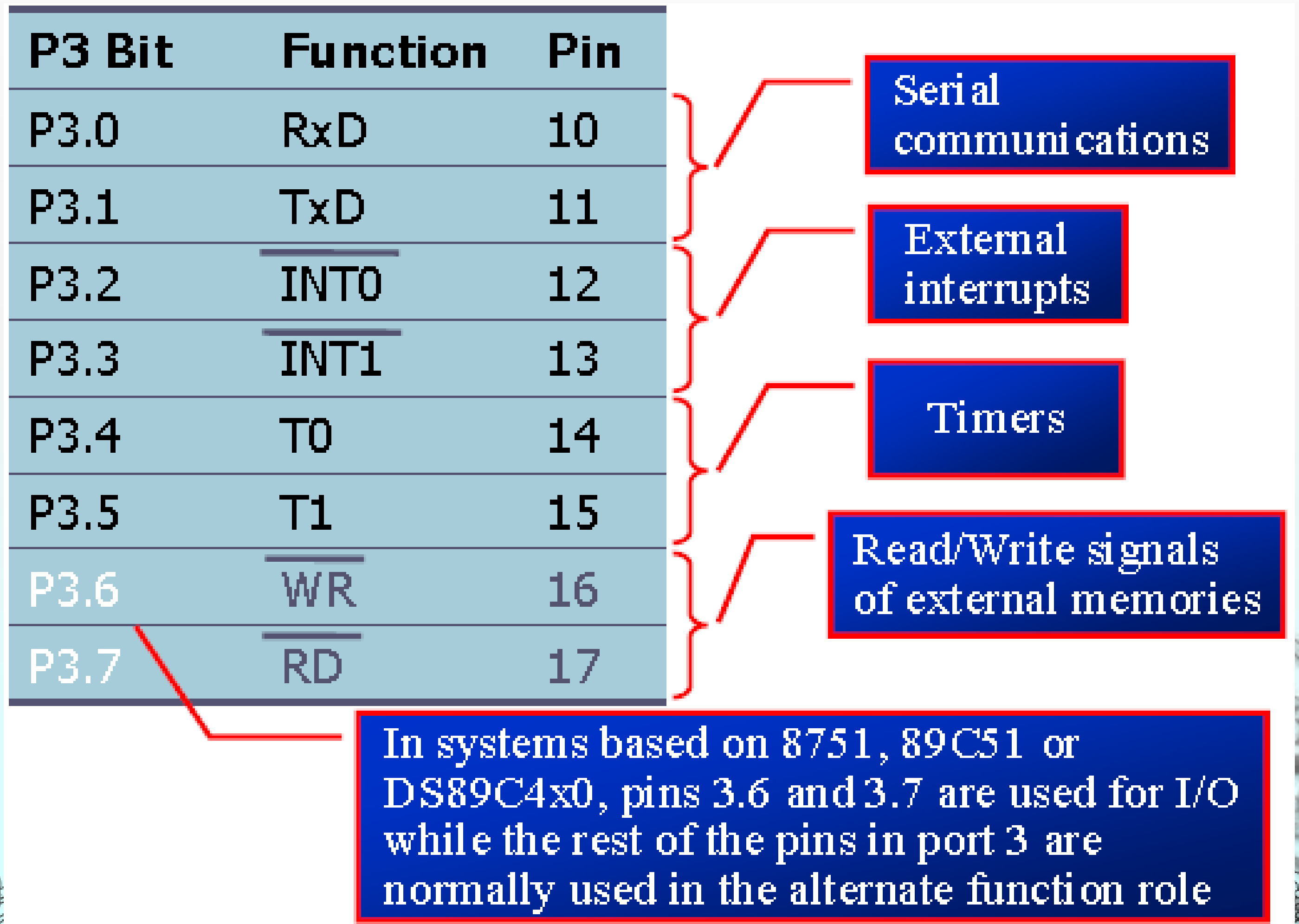
A total of 32 pins are set aside for the four ports P0,P1, P2, P3, where each port takes 8 pins

Provides +5V supply voltage to the chip



P3

- 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

I/O Ports and Bit Addressability

- 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)

Addition of Unsigned Numbers

```
ADD  A, source     ;A = A + source
```

- The instruction ADD is used to add two operands
  - Destination operand is always in register A
  - Source operand can be a register, immediate data, or in memory
  - Memory-to-memory arithmetic operations are never allowed in 8051 Assembly language

Show how the flag register is affected by the following instruction.

```
        MOV A,#0F5H       ;A=F5 hex
        ADD A,#0BH        ;A=F5+0B=00
```

Solution:

```
        F5H          1111  0101
+       0BH        + 0000  1011
      ------       ------------
       100H          0000  0000
```

CY =1, since there is a carry out from D7
P（PSW.0）=0, because the number of 1s is zero (an even number), PF is set to 0.
AC =1, since there is a carry from D3 to D4

ADDC and Addition of 16-Bit Numbers

- 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
```

Subtraction of Unsigned Numbers

- In many microprocessor there are two different instructions for subtraction:
  SUB and SUBB (subtract with borrow)
  - In the 8051 we have only SUBB
  - The 8051 uses adder circuitry to perform the subtraction

```
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

Subtraction of Unsigned Numbers

- SUBB when CY = 1
  This instruction is used for multi-byte numbers and will take care of the borrow of the lower operand

A = 62H – 96H – 0 = CCH
CY = 1

```
CLR    C
MOV    A,#62H          ;A=62H
SUBB   A,#96H          ;62H-96H=CCH with CY=1
MOV    R7,A            ;save the result
MOV    A,#27H          ;A=27H
SUBB   A,#12H          ;27H-12H-1=14H
MOV    R6,A            ;save the result
```

A = 27H - 12H - 1 = 14H
CY = 0

Solution:

We have 2762H - 1296H = 14CCH.

Unsigned Multiplication

- 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

Unsigned Division

- The 8051 supports byte over byte division only
  The byte are assumed to be unsigned data
  DIV AB ;divide A by B, A/B

```
MOV    A, #95          ;load 95 to reg. A
MOV    B, #10          ;load 10 to reg. B
DIV    AB              ;A = 09(quotient) and
                       ;B = 05(remainder)
```

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

Chapter 7

8051 Programming in C

DATA TYPES

- ➢ A good understanding of C data types for 8051 can help programmers to create smaller hex files
- ✓ Unsigned char
- ✓ Signed char
- ✓ Unsigned int
- ✓ Signed int
- ✓ Sbit (single bit)
- ✓ Bit and sfr

Write an 8051 C program to send values 00 – FF to port P1.

Solution:

```
#include <reg51.h>
void main(void)
 {
   unsigned char z;
   for (z=0;z<=255;z++)
   P1=z;
 }
```

1. Pay careful attention to the size of the data
2. Try to use unsigned *char* instead of *int if possible*

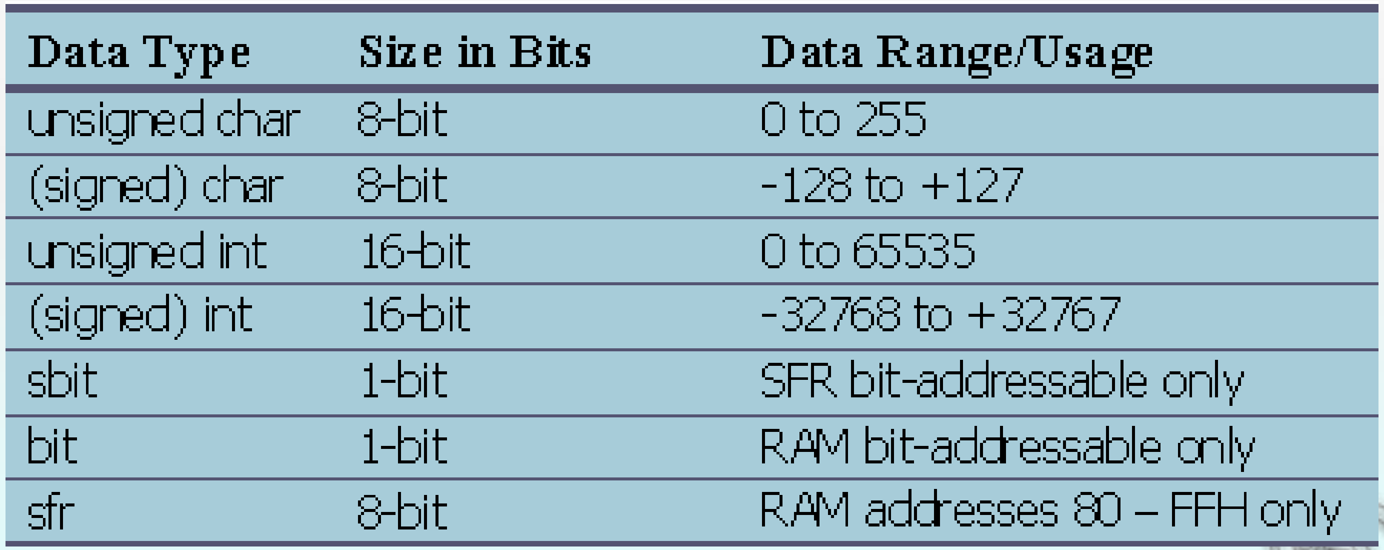
Write an 8051 C program to send hex values for ASCII characters of 0, 1, 2, 3, 4, 5, A, B, C, and D to port P1.

Solution:

```
#include <reg51.h>
void main(void)
 {
   unsigned char mynum[]="012345ABCD";
   unsigned char z;
   for (z=0;z<=10;z++)
   P1=mynum[z];
 }
```

Bit and sfr

- The bit data type allows access to single bits of bit-addressable memory spaces 20 – 2FH
- To access the byte-size SFR registers, we use the sfr data type



Data Type	Size in Bits	Data Range/Usage
unsigned char	8-bit	0 to 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 80 – FFH only

TIME DELAY

- There are two way s to create a time delay in 8051 C
  - ✓Using the 8051 timer (Chap. 9)
  - ✓Using a simple for loop

    be mindful of three factors that can affect the accuracy of the delay
    - The 8051 design
      - The number of machine cycle
      - The number of clock periods per machine cycle
    - The crystal frequency connected to the X1 – X2 input pins
    - Compiler choice
      - C compiler converts the C statements and functions to Assembly language instructions
      - Different compilers produce different code

Write an 8051 C program to toggle bits of P1 ports continuously with a 250 ms.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
  {
    while (1) //repeat forever
    {
     P1=0x55;
     MSDelay(250);
     P1=0xAA;
     MSDelay(250);
    }
  }
void MSDelay(unsigned int itime)
  {
    unsigned int i,j;
    for (i=0;i<itime;i++)
              for (j=0;j<1275;j++);
  }
```

Bit-addressable I/O

Write an 8051 C program to toggle only bit P2.4 continuously without
disturbing the rest of the bits of P2.

Solution:

```
//Toggling an individual bit
#include <reg51.h>
sbit mybit=P2^4;
void main(void)
  {
    while (1)
     {
       mybit=1; //turn on P2.4
       mybit=0; //turn off P2.4
     }
  }
```

Ports P0 – P3 are bit-addressable and we use *sbit data type to access* a single bit of P0 - P3

Use the Px^y format, where x is the port 0, 1, 2, or 3 and y is the bit 0 – 7 of that port

Using bit Data Type for Bit-addressable RAM

Write an 8051 C program to get the status of bit P1.0, save it, and send it to P2.7 continuously.

Solution:

```
#include <reg51.h>
sbit inbit=P1^0;
sbit outbit=P2^7;
bit  membit; //use bit to declare
             //bit- addressable memory
void main(void)
 {
   while (1)
   {
     membit=inbit; //get a bit from P1.0
     outbit=membit; //send it to P2.7
   }
}
```

We use bit data type to access data in a bit-addressable section of the data RAM space 20 – 2FH

Logic Operations

Bit-wise Operators in C

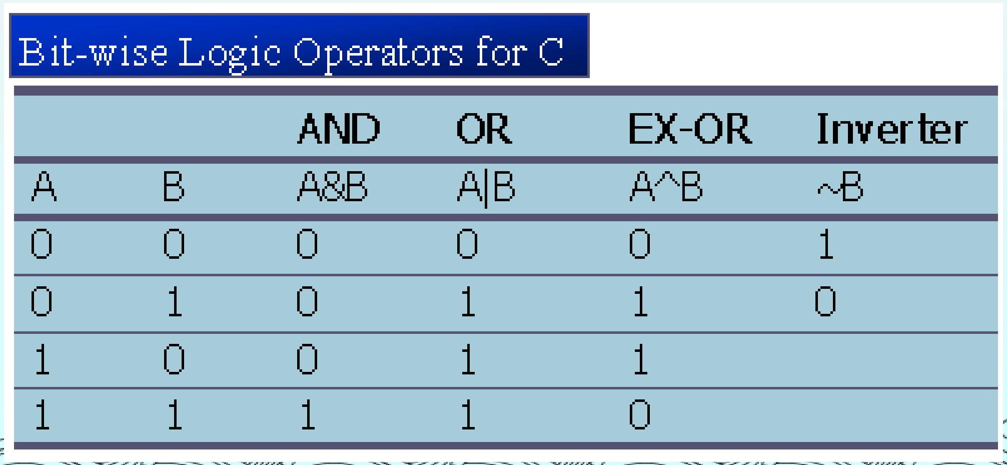
- Logical operators

  AND (&&), OR (||), and NOT (!)

- Bit-wise operators

✓ AND (&), OR (|), EX-OR (^), Inverter (~), Shift Right (>>), and Shift Left (<<)

These operators are widely used in software engineering for embedded systems and control



Bit-wise Logic Operators for C

		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	

Data Conversion

Packed BCD to ASCII Conversion

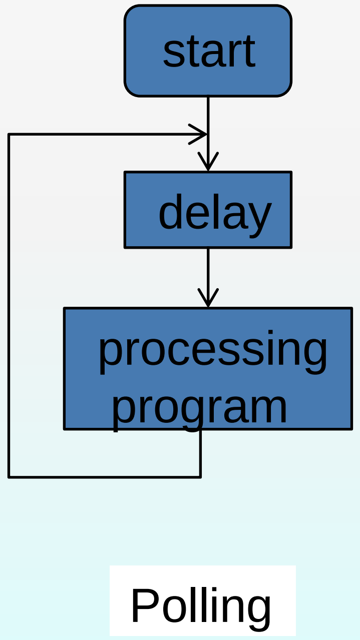
Write an 8051 C program to convert packed BCD 0x29 to ASCII and
display the bytes on P1 and P2.

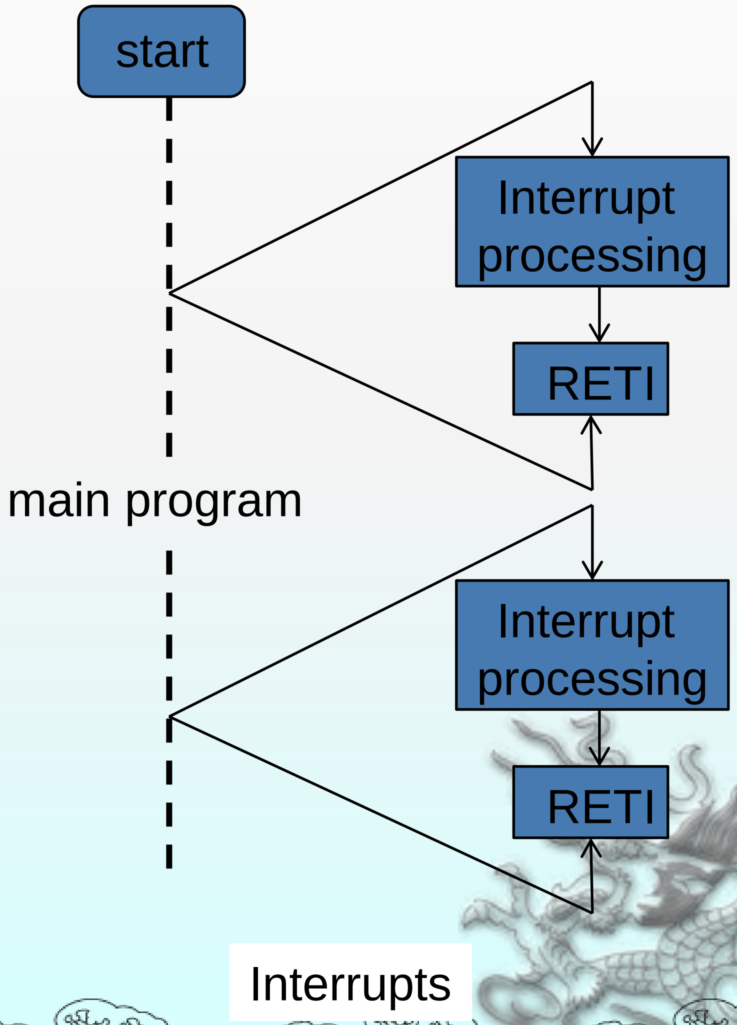
Solution:

```
#include <reg51.h>
void main(void)
  {
    unsigned char x,y,z;
    unsigned char mybyte=0x29;
    x=mybyte&0x0F;
    P1=x|0x30;
    y=mybyte&0xF0;
    y=y>>4;
    P2=y|0x30;
  }
```

Chapter 9
Interrupts Programming

Interrupts vs. Polling





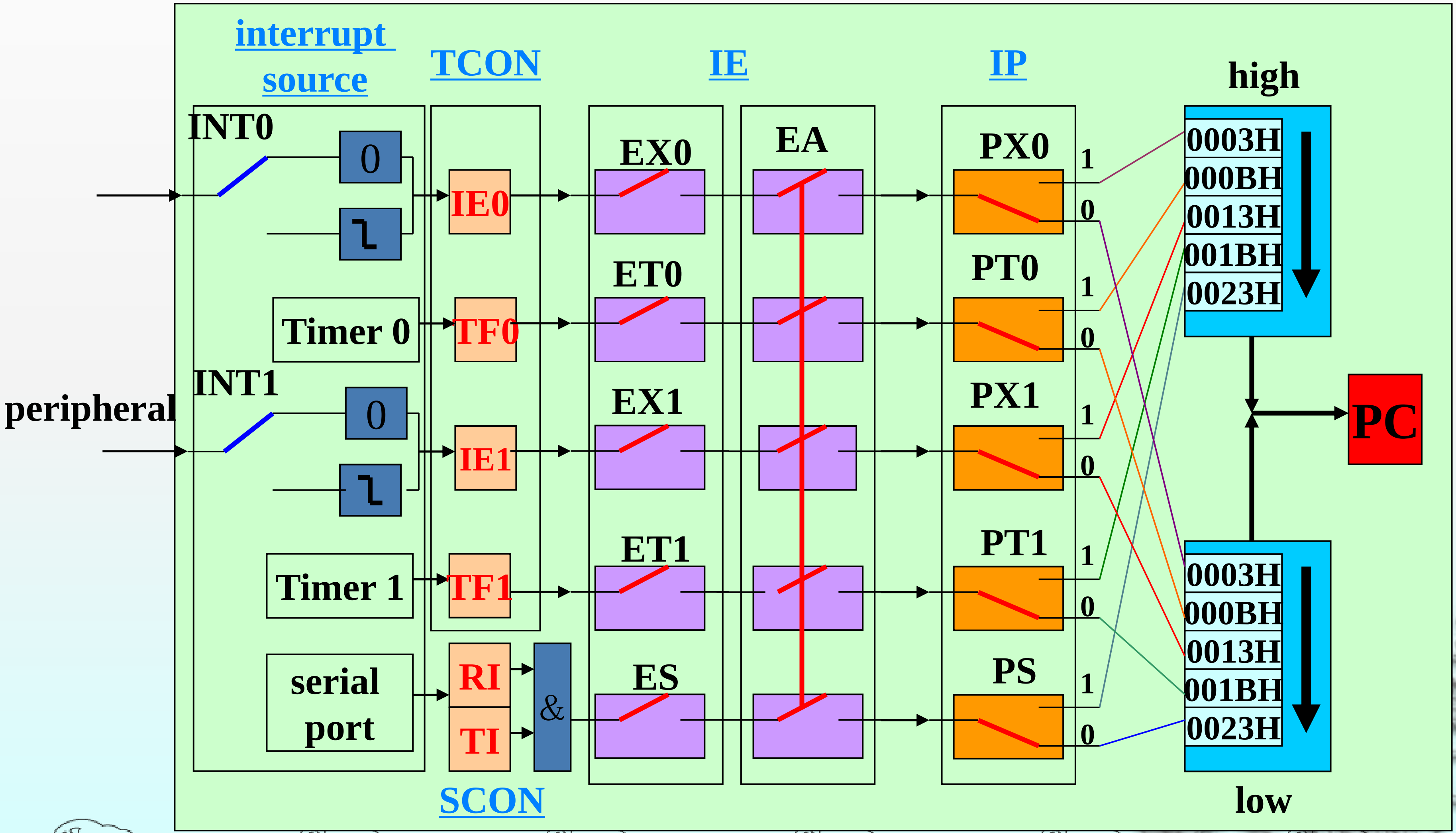
Steps in Executing an Interrupt

- Upon activation of an interrupt, the microcontroller goes through the following steps
  - 1. It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack
  - 2. It also saves the current status of all the interrupts internally (i.e: not on the stack)
  - 3. It jumps to a fixed location in memory, called the interrupt vector table, that holds the address of the ISR

Steps in Executing an Interrupt

- 4. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it
  - ✓ It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine which is RETI (return from interrupt)
- 5. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted
  - ✓ First, it gets the program counter (PC) address from the stack by popping the top two bytes of the stack into the PC
  - ✓ Then it starts to execute from that address

Interrupt system structure chart



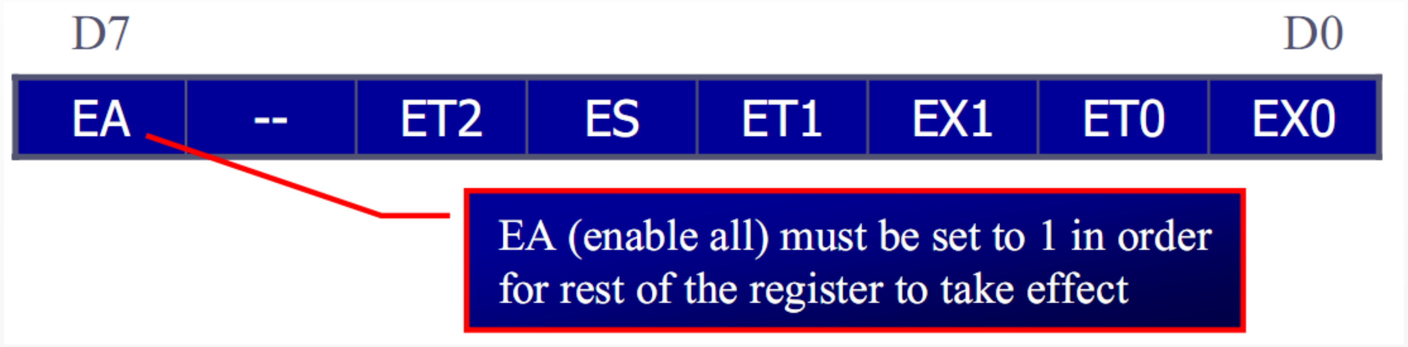
Interrupt vector table

Interrupt	ROM Location (hex)	Pin
Reset	0000	9
External HW (INT0)	0003	P3.2 (12)
Timer 0 (TF0)	000B	
External HW (INT1)	0013	P3.3 (13)
Timer 1 (TF1)	001B	
Serial COM (RI and TI)	0023	

```
        ORG  0       ;wake-up ROM reset location
        LJMP MAIN    ;by-pass int. vector table
;-----  the wake-up program
        ORG  30H
MAIN:
        ....
        END
```

Only three bytes of ROM space assigned to the reset pin. We put the LJMP as the first instruction and redirect the processor away from the interrupt vector table.

IE (Interrupt Enable) Register


D7
D0
EA
--
ET2
ES
ET1
EX1
ET0
EX0
EA (enable all) must be set to 1 in order for rest of the register to take effect

EA	IE.7	Disables all interrupts
--	IE.6	Not implemented, reserved for future use
ET2	IE.5	Enables or disables timer 2 overflow or capture interrupt (8952)
ES	IE.4	Enables or disables the serial port interrupt
ET1	IE.3	Enables or disables timer 1 overflow interrupt
EX1	IE.2	Enables or disables external interrupt 1
ET0	IE.1	Enables or disables timer 0 overflow interrupt
EX0	IE.0	Enables or disables external interrupt 0

TCON (Timer/Counter) Register (Bit-addressable)

D7							D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	TCON.7	Timer 1 overflow flag. Set by hardware when timer/counter1 overflows. Cleared by hardware as the processor vectors to the interrupt service routine
TR1	TCON.6	Timer 1 run control bit. Set/cleared by software to turn timer/counter 1 on/off
TF0	TCON.5	Timer 0 overflow flag. Set by hardware when timer/counter 0 overflows. Cleared by hardware as the processor vectors to the interrupt service routine
TR0	TCON.4	Timer 0 run control bit. Set/cleared by software to turn timer/counter 0 on/off

IE1	TCON.3	External interrupt 1 edge flag. Set by CPU when the external interrupt edge (H-to-L transition) is detected. Cleared by CPU when the interrupt is processed
IT1	TCON.2	Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt
IE0	TCON.1	External interrupt 0 edge flag. Set by CPU when the external interrupt edge (H-to-L transition) is detected. Cleared by CPU when the interrupt is processed
IT0	TCON.0	Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt

§ 9-3 Interrupt Priority

- When the 8051 is powered up, the priorities are assigned according to the following
  - In reality, the priority scheme is nothing but an internal polling sequence in which the 8051 polls the interrupts in the sequence listed and responds accordingly

Interrupt Priority Upon Reset

Highest To Lowest Priority	
External Interrupt 0	(INT0)
Timer Interrupt 0	(TF0)
External Interrupt 1	(INT1)
Timer Interrupt 1	(TF1)
Serial Communication	(RI + TI)

§ 9-3 Interrupt Priority

- We can alter the sequence of interrupt priority by assigning a higher priority to any one of the interrupts by programming a register called IP (interrupt priority)
  - To give a higher priority to any of the interrupts, we make the corresponding bit in the IP register high
  - When two or more interrupt bits in the IP register are set to high
    - While these interrupts have a higher priority than others, they are serviced according to the sequence of Table 11-13

Interrupt Priority Register (Bit-addressable)

D7							D0
--	--	PT2	PS	PT1	PX1	PT0	PX0

--	IP.6	Reserved
--	IP.7	Reserved
PT2	IP.5	Timer 2 interrupt priority bit (8052 only)
PS	IP.4	Serial port interrupt priority bit
PT1	IP.3	Timer 1 interrupt priority bit
PX1	IP.2	External interrupt 1 priority bit
PT0	IP.1	Timer 0 interrupt priority bit
PX0	IP.0	External interrupt 0 priority bit

Priority bit=1 assigns high priority
Priority bit=0 assigns low priority

两个按键分别控制LED灯的开关, P3.2接口的按键按下时开灯, P3.3接口的按键按下时关灯。

```
void  extern0( )   interrupt 0{}
void  timer0( )   interrupt 1 {}
void  extern1( )   interrupt 2{}
void  timer1( )   interrupt 3 {}
void  serial0( )   interrupt 4 {}
```

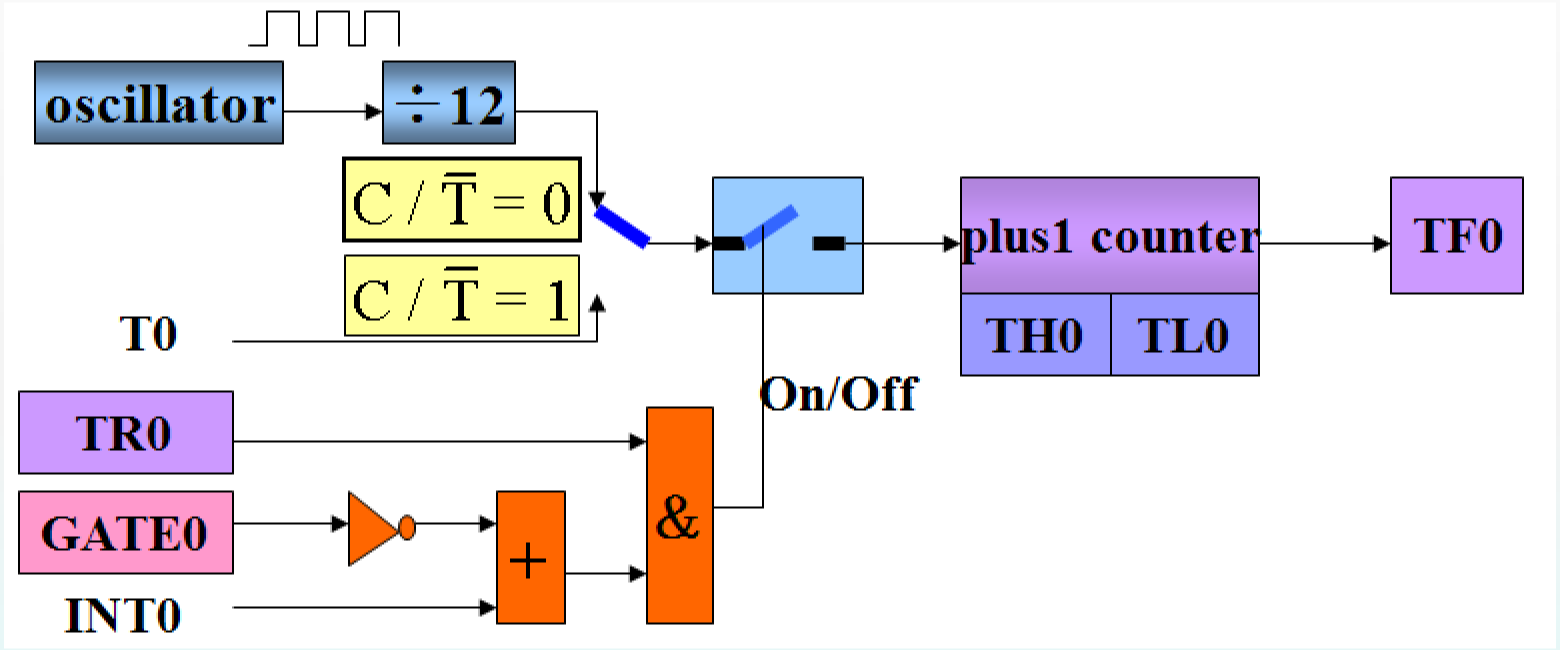
```
#include <reg51.h>
sbit   LED = P1 ^ 0;
void INT_init (void){
            EA = 1;
            EX1 = 1;
            EX0 = 1;
            IT1 = 1;     //1:   falling edge-triggered
            IT0 = 1;
}
void INT_1 (void) interrupt 2  //using 2
{

            LED = 1;   // turn off the light
}

void INT_0 (void) interrupt 0  //  using 0
{

            LED = 0;   //turn the light on
}
void main(void){
            INT_init();  //extern interrupt initialization
            while(1){
                        //other program
            }
```

Chapter 10
Timer Programming

§ 10-1 Programming Timer

- The 8051 has two timers/counters, they can be used either as
  - Timers to generate a time delay or as
  - Event counters to count events happening outside the microcontroller
- Both Timer 0 and Timer 1 are 16 bits wide
  - Since 8051 has an 8-bit architecture, each 16-bits timer is accessed as two separate registers of low byte and high byte

- Structure of Timer0

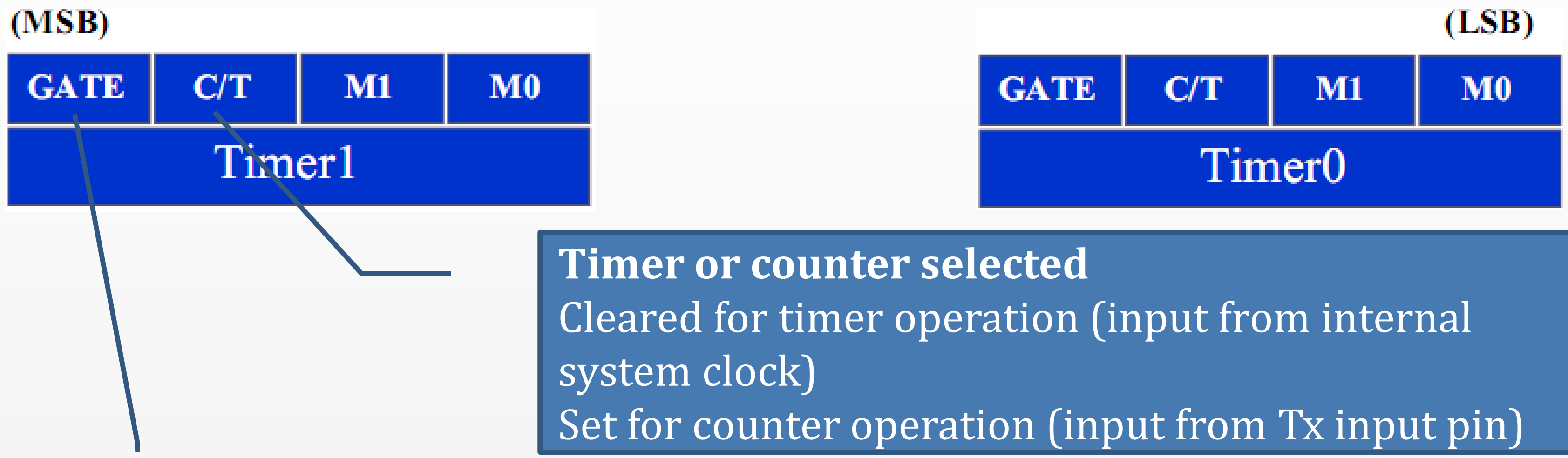


TMOD Register

- Both timers 0 and 1 use the same register, called TMOD (timer mode), to set the various timer operation modes TMOD is a 8-bit register
  - The lower 4 bits are for Timer 0
  - The upper 4 bits are for Timer 1
  - In each case,
    - ✓ The lower 2 bits are used to set the timer mode
    - ✓ The upper 2 bits to specify the operation

(MSB)							(LSB)
GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer1				Timer0			

TMOD Register



Timer or counter selected
Cleared for timer operation (input from internal system clock)
Set for counter operation (input from Tx input pin)

Gating control when set. Timer/counter is enable only while the INTx pin is high and the TRx control pin is set
When cleared, the timer is enabled whenever the TRx control bit is set

M1	M0	Mode	Operating Mode
0	0	0	**13-bit timer mode** 8-bit timer/counter THx with TLx as 5-bit prescaler
0	1	1	**16-bit timer mode** 16-bit timer/counter THx and TLx are cascaded; there is no prescaler
1	0	2	**8-bit auto reload** 8-bit auto reload timer/counter; THx holds a value which is to be reloaded TLx each time it overfolws
1	1	3	**Split timer mode**

Steps to Mode 1 Program

To generate a time delay

1. Load the TMOD value register indicating which timer (timer 0 or timer 1) is to be used and which timer mode (0 or 1) is selected
2. Load registers TL and TH with initial count value
3. Start the timer
4. Keep monitoring the timer flag (TF) with the JNB TFx, target instruction to see if it is raised
   - Get out of the loop when TF becomes high
5. Stop the timer
6. Clear the TF flag for the next round
7. Go back to Step 2 to load TH and TL again

Finding the Loaded Timer Values

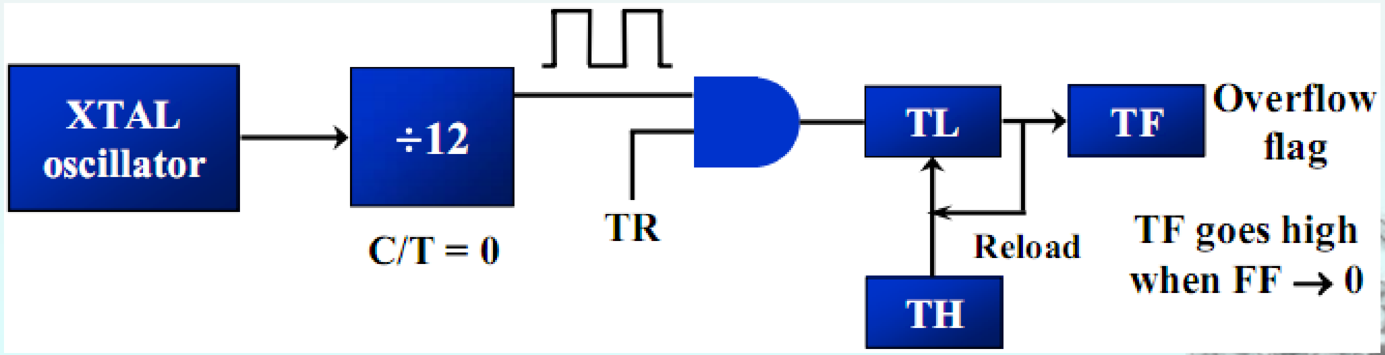
- To calculate the values to be loaded into the TL and TH registers, look at the following example
  - Assume XTAL = 11.0592 MHz, we can use the following steps for finding the TH, TL registers' values

1. Divide the desired time delay by 1.085 us
2. Perform 65536 – n, where n is the decimal value we got in Step1
3. Convert the result of Step2 to hex, where yyxx is the initial hex value to be loaded into the timer's register
4. Set TL = xx and TH = yy

Mode 2 Programming

4. When the TL register rolls from FFH to 0 and TF is set to 1, TL is reloaded automatically with the original value kept by the TH register

- To repeat the process, we must simply clear TF and let it go without any need by the programmer to reload the original value
- This makes mode 2 an auto-reload, in contrast with mode 1 in which the programmer has to reload TH and TL



§ 10-2 Counter Programming

- Timers can also be used as counters counting events happening outside the 8051
  - When it is used as a counter, it is a pulse outside of the 8051 that increments the TH, TL registers
  - TMOD and TH, TL registers are the same as for the timer discussed previously
- Programming the timer in the last section also applies to programming it as a counter
  - Except the source of the frequency

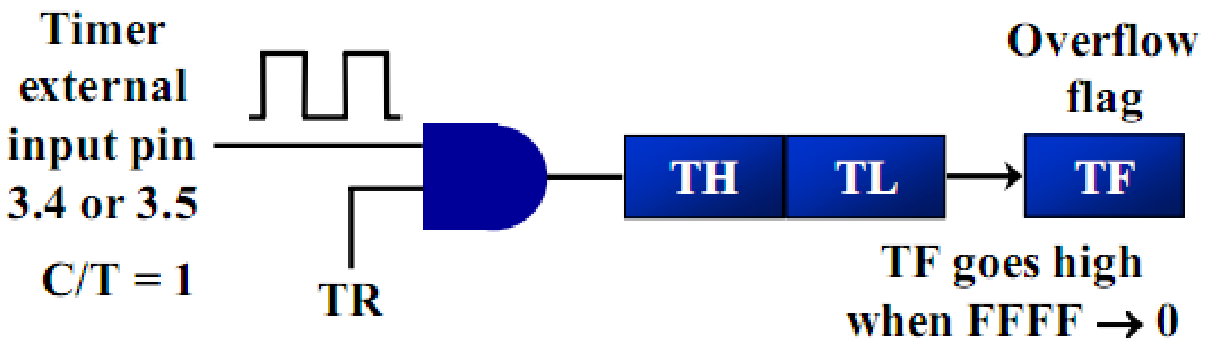
C/T Bit in TMOD Register

- The C/T bit in the TMOD registers decides the source of the clock for the timer
  - When C/T = 1, the timer is used as a counter and gets its pulses from outside the 8051
    - The counter counts up as pulses are fed from pins 14 and 15, these pins are called T0 (timer 0 input) and T1 (timer 1 input)

Port 3 pins used for Timers 0 and 1

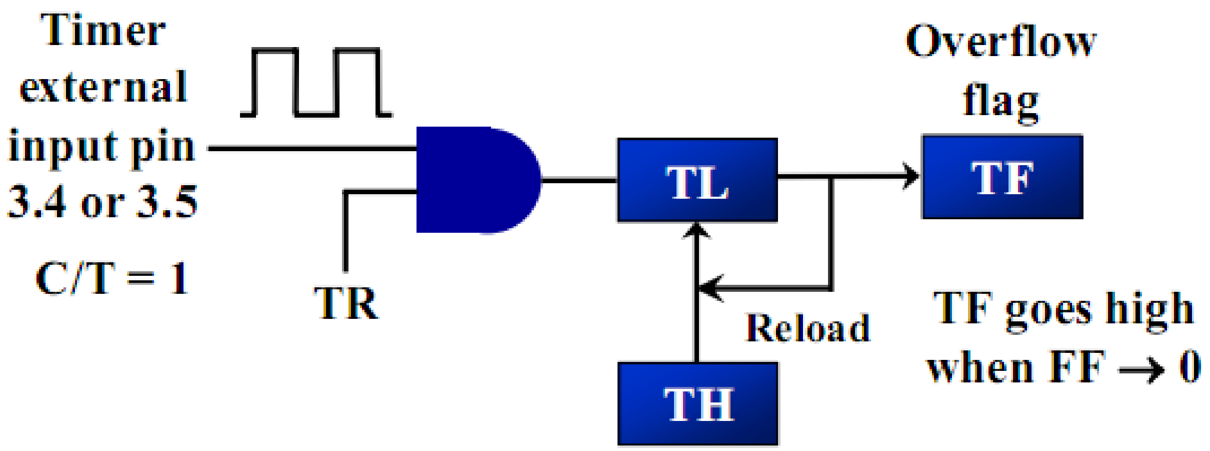
Pin	Port Pin	Function	Description
14	P3.4	T0	Timer/counter 0 external input
15	P3.5	T1	Timer/counter 1 external input

Timer with external input (Mode 1)



Timer external input pin 3.4 or 3.5
C/T = 1
TR
TH
TL
Overflow flag
TF
TF goes high when FFFF → 0

Timer with external input (Mode 2)



TCON Register

- TCON (timer control) register is an 8-bit register

TCON: Timer/Counter Control Register

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

The upper four bits are used to store the TF and TR bits of both timer 0 and 1

The lower 4 bits are set aside for controlling the interrupt bits

§ 10-3 Programming Timers in C

Accessing Timer Registers

Example 10-17

Write an 8051 C program to toggle all the bits of port P1 continuously with some delay in between. Use Timer 0, 16-bit mode to generate the delay.

Solution:

```
#include <reg51.h>
void T0Delay(void);
void main(void){
    while (1) {
        P1=0x55;
        T0Delay();
        P1=0xAA;
        T0Delay();
    }
 }
```

```
void T0Delay(){
    TMOD=0x01;
    TL0=0x00;
    TH0=0x35;
    TR0=1;
    while (TF0==0);
    TR0=0;
    TF0=0;
}
```

FFFFH – 3500H = CAFFH
= 51967 + 1 = 51968
$51968 \times 1.085\mu s = 56.384ms$
is the approximate delay

Example 10-22

万年历

Solution:

```
#include <reg51.h>
void Time0Isr(void) interrupt 1
{
   TH0=0x3c;
   TL0=0xb0;
   sec_50ms++;
   if (sec_50ms==_____)
            sec++;
   if (sec==____)
   {
            min++;
            sec=0;
   }
   if (min==____)
   {
            hour++;
            min=0;
   }
```

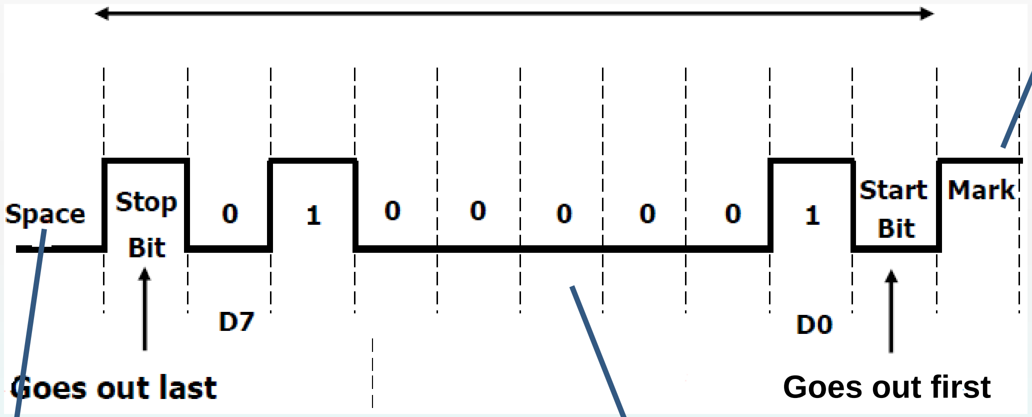
```
   If (hour==_____)
   {
            day++;
            hour=0;
    }
}
```

Chapter 11
Serial Communication

Start and Stop Bits

- The start bit is always a 0 (low) and the stop bit(s) is 1 (high)

ASCII character "A" (8-bit binary 0100 0001)



When there is no transfer, the signal is 1 (high), which is referred to as *mark*

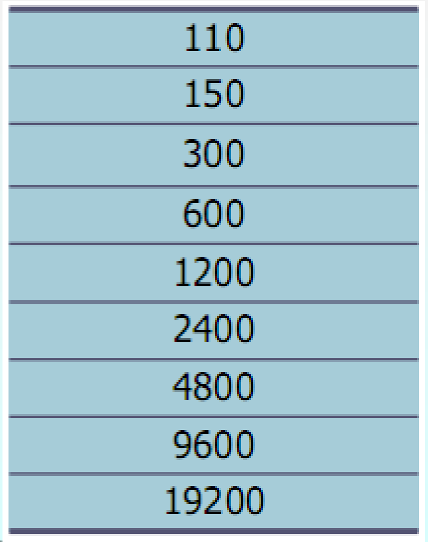
The 0 (low) is referred to as *space*

The transmission begins with a start bit followed by D0, the LSB, then the rest of the bits until MSB (D7), and finally, the one stop bit indicating the end of the character

§ 11-3 Serial Communication Programming

- To allow data transfer between the PC and an 8051 system without any error, we must make sure that the baud rate of 8051 system matches the baud rate of the PC's COM port
- Hyperterminal function supports baud rates much higher than listed below

PC Baud Rates



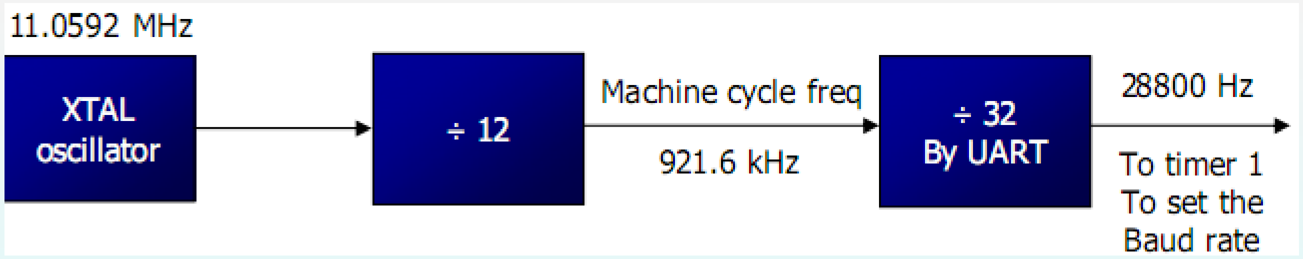
110
150
300
600
1200
2400
4800
9600
19200

Baud rates supported by 486/Pentium IBM PC BIOS

With XTAL = 11.0592 MHz, find the TH1 value needed to have the following baud rates. (a) 9600 (b) 2400 (c) 1200

Solution:

The machine cycle frequency of 8051 = 11.0592 / 12 = 921.6 kHz, and 921.6 kHz / 32 = 28,800 Hz is frequency by UART to timer 1 to set baud rate.
(a) 28,800 / 3 = 9600 where -3 = FD (hex) is loaded into TH1
(b) 28,800 / 12 = 2400 where -12 = F4 (hex) is loaded into TH1
(c) 28,800 / 24 = 1200 where -24 = E8 (hex) is loaded into TH1
Notice that dividing 1/12 of the crystal frequency by 32 is the default value upon activation of the 8051 RESET pin.



Baud Rate	TH1 (Decimal)	TH1 (Hex)
9600	-3	FD
4800	-6	FA
2400	-12	F4
1200	-24	E8

TF is set to 1 every 12 ticks, so it functions as a frequency divider

SBUF Register

- SBUF is an 8-bit register used solely for serial communication
  - For a byte data to be transferred via the TxD line, it must be placed in the SBUF register
    - ✓ The moment a byte is written into SBUF, it is framed with the start and stop bits and transferred serially via the TxD line
  - SBUF holds the byte of data when it is received by 8051 RxD line
    - ✓ When the bits are received serially via RxD, the 8051 deframes it by eliminating the stop and start bits, making a byte out of the data received, and then placing it in SBUF

```
MOV SBUF,#'D'   ;load SBUF=44h, ASCII for 'D'
MOV SBUF,A      ;copy accumulator into SBUF
MOV A,SBUF      ;copy SBUF into accumulator
```

SCON Register

- SCON is an 8-bit register used to program the start bit, stop bit, and data bits of data framing, among other things

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial port mode specifier
SM1	SCON.6	Serial port mode specifier
SM2	SCON.5	Used for multiprocessor communication
REN	SCON.4	Set/cleared by software to enable/disable reception
TB8	SCON.3	Not widely used
RB8	SCON.2	Not widely used
TI	SCON.1	Transmit interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW
RI	SCON.0	Receive interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW

Note: Make SM2, TB8, and RB8 =0

SCON Register

- SM0, SM1
  - They determine the framing of data by specifying the number of bits per character, and the start and stop bits

SM0	SM1	
0	0	Serial Mode 0
0	**1**	Serial Mode 1, 8-bit data, 1 stop bit, 1 start bit
1	0	Serial Mode 2
1	1	Serial Mode 3

Only mode 1 and 3 is of interest to us

- SM2
  - This enables the multiprocessing capability of the 8051

- REN (receive enable)
  - It is a bit-adressable register
    - When it is high, it allows 8051 to receive data on RxD pin
    - If low, the receiver is disable
- TI (transmit interrupt)
  - When 8051 finishes the transfer of 8-bit character
    - It raises TI flag to indicate that it is ready to transfer another byte
    - TI bit is raised at the beginning of the stop bit
- RI (receive interrupt)
  - When 8051 receives data serially via RxD, it gets rid of the start and stop bits and places the byte in SBUF register
    - It raises the RI flag bit to indicate that a byte has been received and should be picked up before it is lost
    - RI is raised halfway through the stop bit

Programming Serial Data Transmitting

- In programming the 8051 to transfer character bytes serially
  1. TMOD register is loaded with the value 20H, indicating the use of timer1 in mode 2 (8-bit auto-reload) to set baud rate
  2. The TH1 is loaded with one of the values to set baud rate for serial data transfer
  3. The SCON register is loaded with the value 50H, indicating serial mode1, where an 8-bit data is framed with start and stop bits
  4. TR1 is set to 1 to start timer 1
  5. TI is cleared by CLR TI instruction
  6. The character byte to be transferred serially is written into SBUF register
  7. The TI flag bit is monitored with the use of instruction JNB TI , xx to see if the character has been transferred completely
  8. To transfer the next byte, go to step 5

Programming Serial Data Receiving

- In programming the 8051 to receive character bytes serially

  1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode2 (8-bit auto-reload) to set baud rate
  2. TH1 is loaded to set baud rate
  3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits
  4. TR1 is set to 1 to start timer 1
  5. RI is cleared by CLR RI instruction
  6. The RI flag bit is monitored with the use of instruction JNB RI,xx to see if an entire character has been received yet
  7. When RI is raised, SBUF has the byte, its contents are moved into a safe place
  8. To receive the next character, go to step 5

Doubling Baud Rate

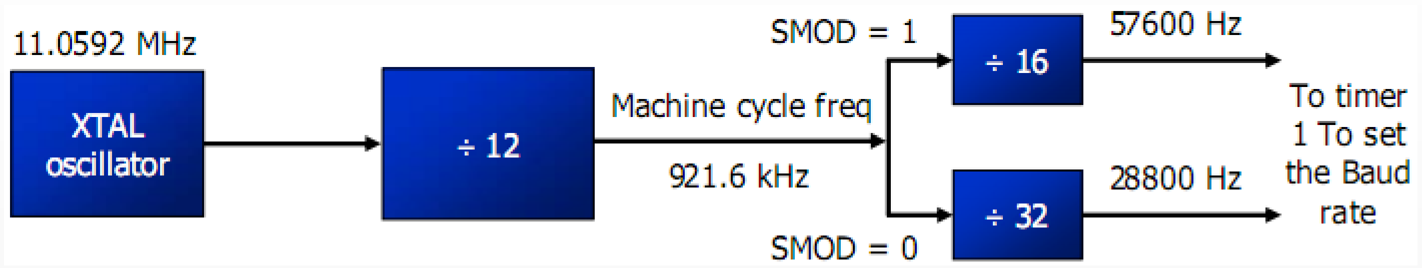
- There are two ways to increase the baud rate of data transfer
  - To use a higher frequency crystal
  - To change a bit in the PCON register
- PCON register is an 8-bit register
  - When 8051 is powered up, SMOD is zero
  - We can set it to high by software and thereby double the baud rate

The system crystal is fixed

SMOD	--	--	--	GF1	GF0	PD	IDL

It is not a bit-addressable register

```
MOV  A , PCON       ;place a copy of PCON in ACC
SETB ACC.7          ;make D7=1
MOV  PCON , A       ;changing any other bits
```



Baud Rate comparison for SMOD=0 and SMOD=1

TH1	(Decimal)	(Hex)	SMOD=0	SMOD=1
	-3	FD	9600	19200
	-6	FA	4800	9600
	-12	F4	2400	4800
	-24	E8	1200	2400

§ 11-5 Serial Port Programming in C

Transmitting and Receiving Data

Example 11-7

Write a C program for 8051 to transfer the letter “A” serially at 4800 baud continuously. Use 8-bit data and 1 stop bit.

Solution:

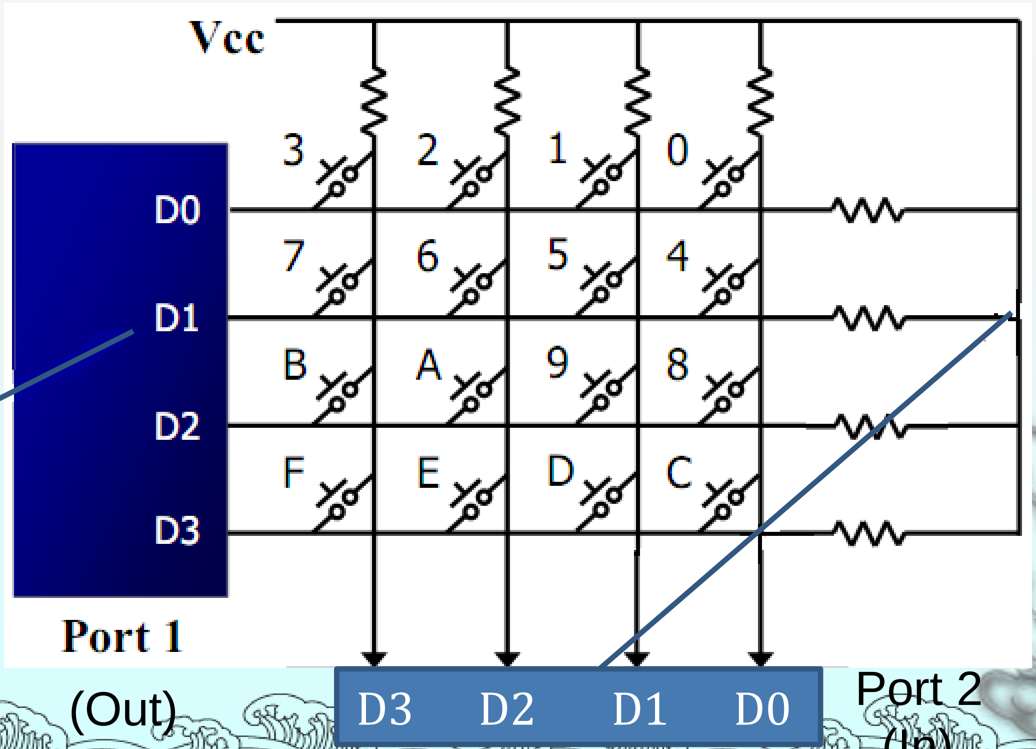
```
#include <reg51.h>
void main(void){
   TMOD=0x20; //use Timer 1, mode 2
   TH1=0xFA; //4800 baud rate
   SCON=0x50;
   TR1=1;
   while (1) {
      SBUF=‘A’; //place value in buffer
      while (TI==0);
      TI=0;
     }
 }
```

Chapter 13
Real-world Interfacing
LCD, ADC, and DAC

Scanning and Identifying the Key

- A 4x4 matrix connected to two ports
  - The rows are connected to an output port and the columns are connected to an input port

Matrix Keyboard Connection to ports



(Out)

Port 2 (In)

If all the rows are grounded and a key is pressed, one of the columns will have 0 since the key pressed provides the path to ground

If no key has been pressed, reading the input port will yield 1s for all columns since they are all connected to high (Vcc)

§ 9.1.3　矩阵式键盘接口设计

- **矩阵式键盘按键识别方法有：**
  行扫描法和线路反转法。

1、行扫描法

设置行线为输出，列线为输入，当无按键按下时，列输入全为“1”。设计时，将某一行输出为“0”，读取列线值，若其中某一位为“0”，则表明行、列交叉点处的按键被按下，否则无按键按下；继续扫描下一行（将下一行输出为“0”），直至全扫描完为止。

或同理可设置列线为输出，行线为输入。设计时，将某一列输出为“0”。

§ 9.1.3 矩阵式键盘接口设计

- **线路反转法需要两个双向I/O口分别接行、列线。步骤如下：**

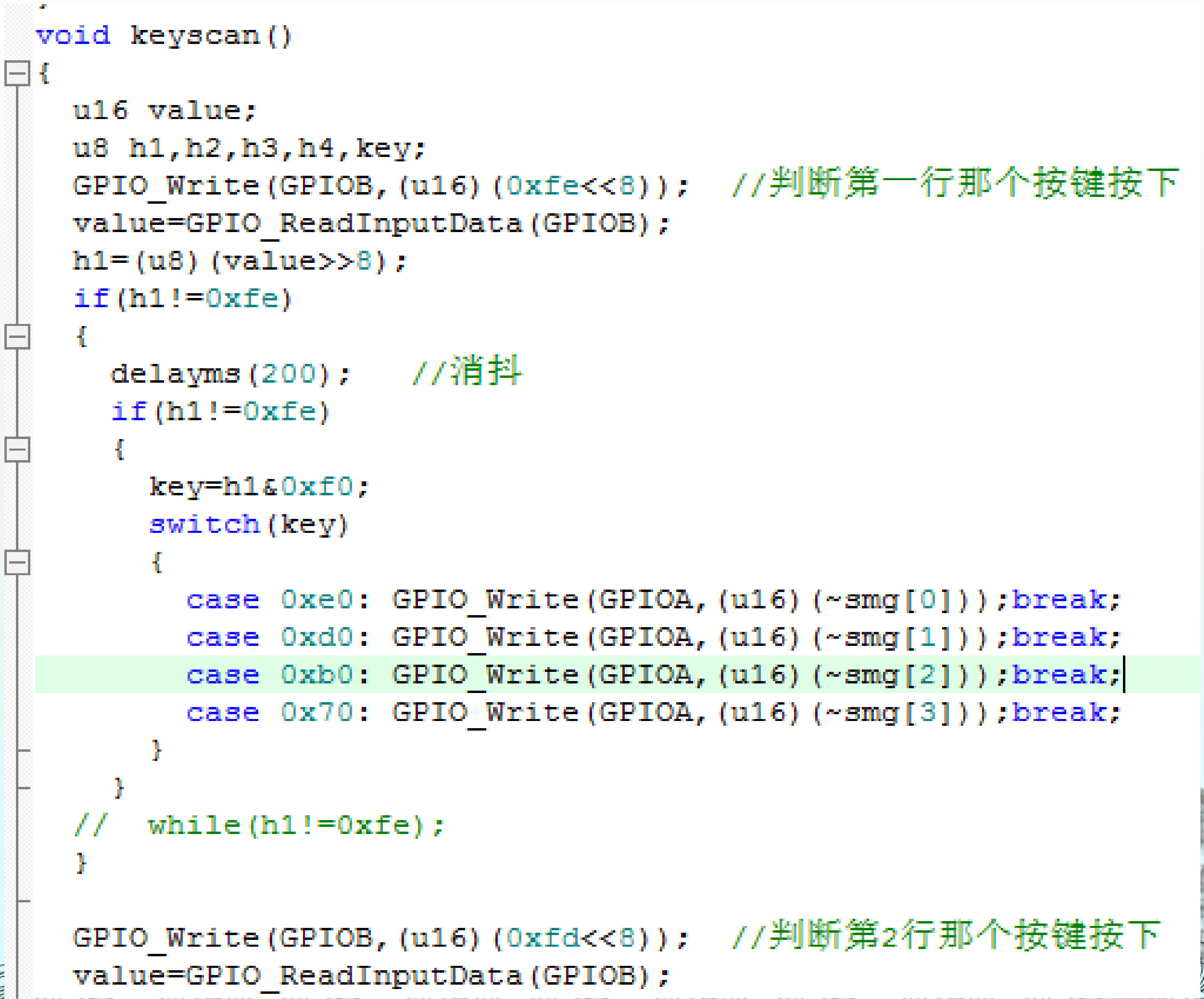
（1）由行线输出全“0”，读入列线，判有无键按下（若有一个列为“0”则表明有键按下）。

（2）若有键按下,再将读入的列值从列线输出,读取行线的值。

（3）第一步读入的列值与第二步读入的行值运算，
从而得到代表此键的唯一的特征值。

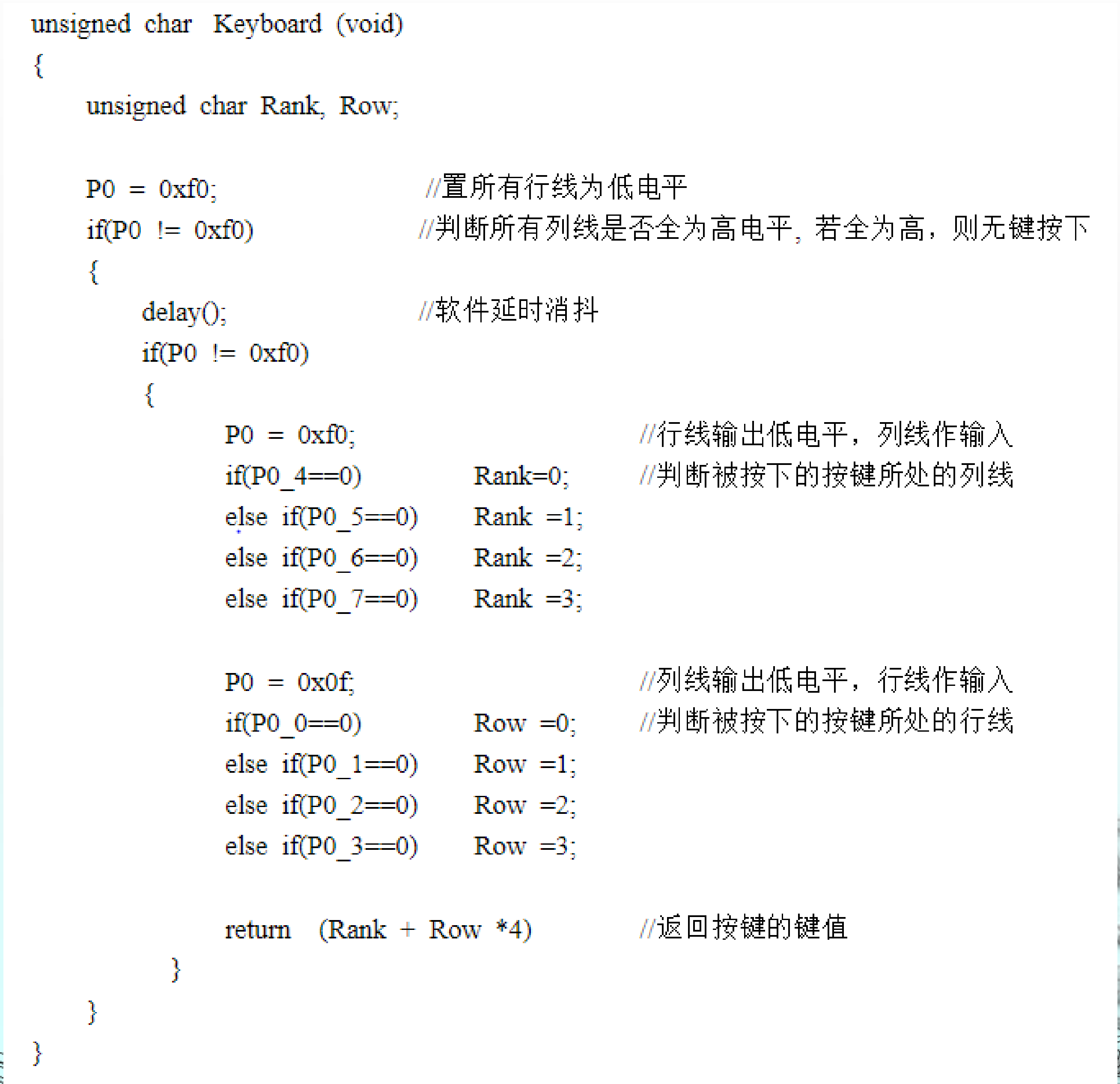
优点：判键速度快，两次即可。

STM32矩阵键盘查询程序



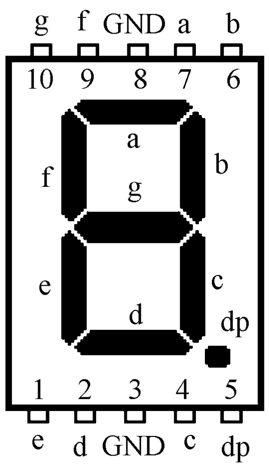
```
void keyscan()
{
  u16 value;
  u8 h1,h2,h3,h4,key;
  GPIO_Write(GPIOB,(u16)(0xfe<<8));  //判断第一行那个按键按下
  value=GPIO_ReadInputData(GPIOB);
  h1=(u8)(value>>8);
  if(h1!=0xfe)
  {
    delayms(200);   //消抖
    if(h1!=0xfe)
    {
      key=h1&0xf0;
      switch(key)
      {
        case 0xe0: GPIO_Write(GPIOA,(u16)(~smg[0]));break;
        case 0xd0: GPIO_Write(GPIOA,(u16)(~smg[1]));break;
        case 0xb0: GPIO_Write(GPIOA,(u16)(~smg[2]));break;
        case 0x70: GPIO_Write(GPIOA,(u16)(~smg[3]));break;
      }
    }
  //  while(h1!=0xfe);
  }

  GPIO_Write(GPIOB,(u16)(0xfd<<8));  //判断第2行那个按键按下
  value=GPIO_ReadInputData(GPIOB);
```

矩阵键盘翻转法

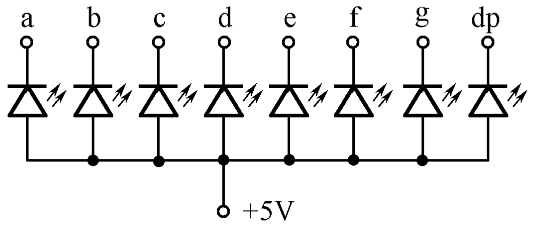


```
unsigned char  Keyboard (void)
{
    unsigned char Rank, Row;

    P0 = 0xf0;                  //置所有行线为低电平
    if(P0 != 0xf0)              //判断所有列线是否全为高电平，若全为高，则无键按下
    {
        delay();                //软件延时消抖
        if(P0 != 0xf0)
        {
            P0 = 0xf0;                              //行线输出低电平，列线作输入
            if(P0_4==0)         Rank=0;             //判断被按下的按键所处的列线
            else if(P0_5==0)    Rank =1;
            else if(P0_6==0)    Rank =2;
            else if(P0_7==0)    Rank =3;

            P0 = 0x0f;                              //列线输出低电平，行线作输入
            if(P0_0==0)         Row =0;             //判断被按下的按键所处的行线
            else if(P0_1==0)    Row =1;
            else if(P0_2==0)    Row =2;
            else if(P0_3==0)    Row =3;

            return   (Rank + Row *4)                //返回按键的键值
         }
    }
}
```

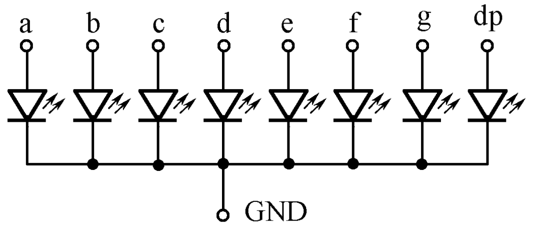
§ 9.2.1　LED显示原理



（a）外形



（b）共阳极



（c）共阴极

上图中的a～g七个笔划（段）及小数点dp均为发光二极管。数码管显示器根据公共端的连接方式，可以分为共阴极数码管（将所有发光二极管的阴极连在一起）和共阳极数码管（将所有发光二极管的阳极连在一起）。