Curs 8

8. Aplicaţii

8.1 Comanda motoarelor pas cu pas

- realizează conversia impulsurilor de comandă aplicate fazelor motorului într-o mişcare de rotaţie (deplasări unghiulare de mărime egală care reprezintă paşii motorului).
- Principala deosebire dintre motoarele pas cu pas şi cele de curent continuu (motoarele DC), este aceea că motoarele de curent continuu nu pot fi poziţionate cu acurateţe chiar dacă li se dă comanda de stop.
- Când se energizează bobina unui motor pas cu pas, arborele motorului (care defapt este un magnet permanent) se aliniază corespunzător cu polii bobinei energizate. Respectând această regulă se realizează rotirea motorului.
- Dublarea rezoluţiei motorului este un proces numit *semipas* (*half-stepping*)
- Există motoare pas cu pas unipolare şi bipolare

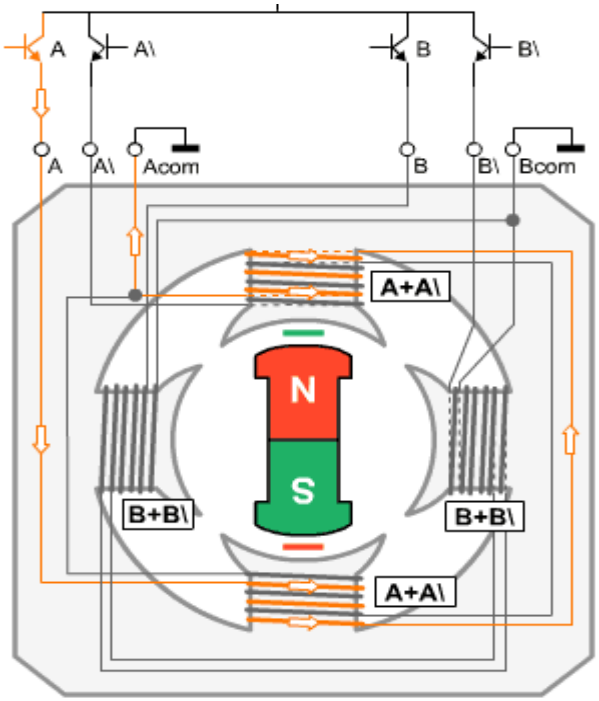
...Roboți

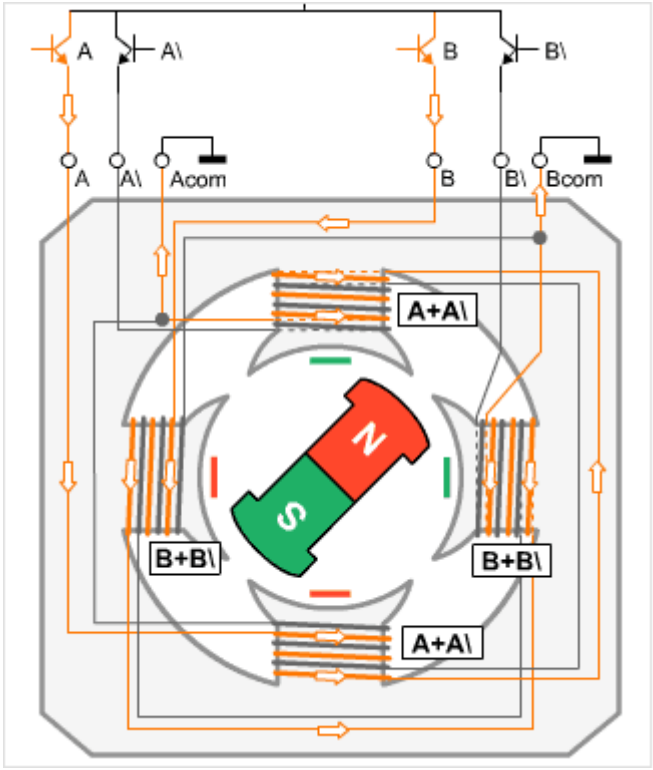
Tahografe...

8. Aplicaţii

8.1 Comanda motoarelor pas cu pas

- *Exemplu:* când este alimentată bobina A, polaritatea nord-sud este generată la “A+A\”, iar arborele se aliniază corespunzător. Când următoarea bobină este alimentată, arborele se rearanjează. Prin urmare, pentru a realiza deplasarea arborelui, trebuie construită o anumită secvenţă de alimentare a bobinelor.

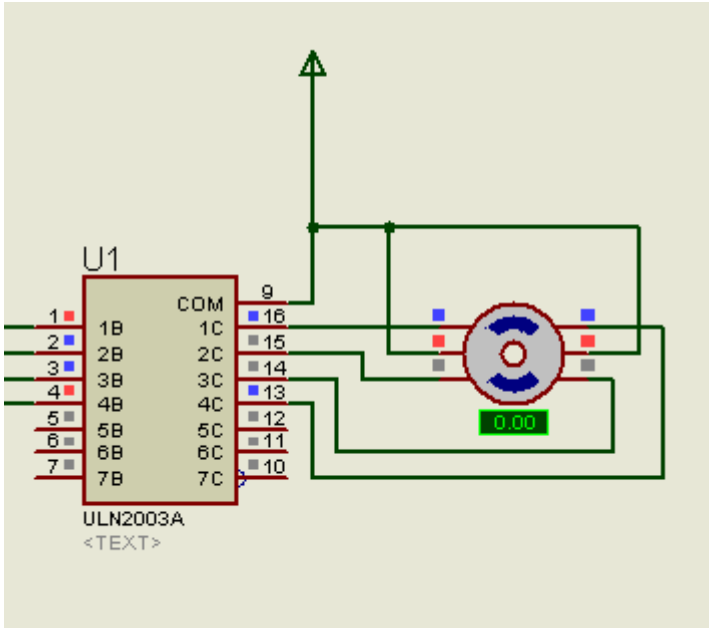




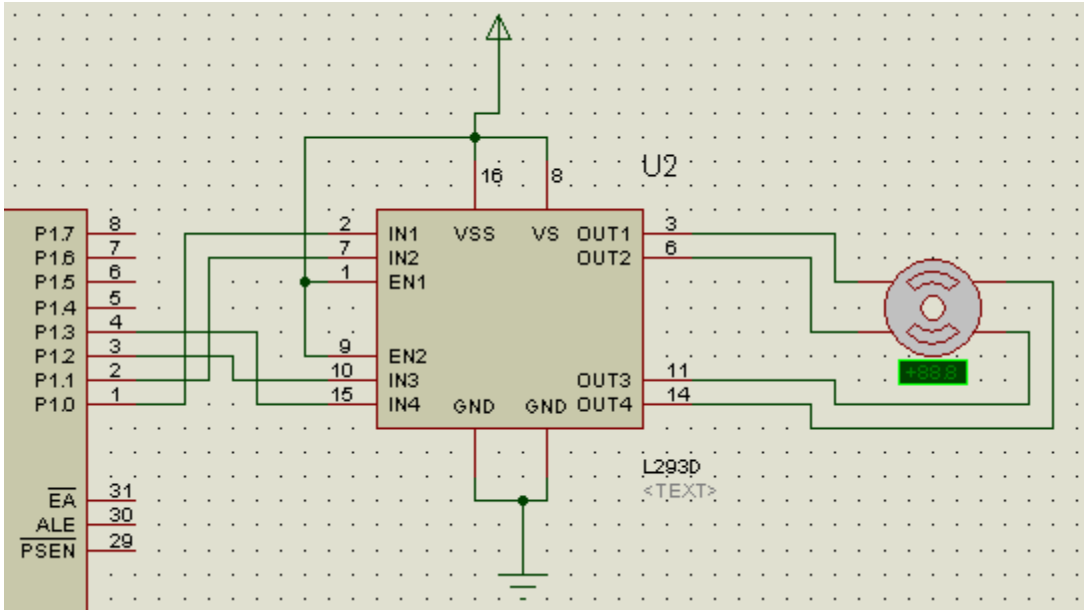
Aplicaţii

8.1. Comanda motoarelor pas cu pas

- Interfaţarea MPP cu microcontrolerul 80C51 se poate face prin două metode:
  - *Interfaţare folosind L293D* – punte H de comandă a motoarelor
  - *Interfaţare folosind ULN2003* – configuraţie Darlington de tranzistoare



a) Conectare motor unipolar



b) Conectare motor bipolar

Aplicaţii

8.1 Comanda motoarelor pas cu pas

- MPP pot fi comandate prin două secvenţe de comandă

a) Secvenţă pas complet

- În primul caz, două bobine sunt alimentate în acelaşi timp şi arborele motorului se deplasează. Ordinea în care bobinele trebuie alimentate este dată în tabelul de mai jos. Luând în discuţie cazul motorului unipolar

Secvenţă pas complet				
Step	**A**	**B**	**A**	**B**
0	1	1	0	0
1	0	1	1	0
2	0	0	1	1
3	1	0	0	1

b) Secvenţă semipas

- În acest caz, unghiul de rotire al arborelui se reduce la jumătate. Prin urmare numărul de paşi se dublează faţă de secvenţa cu pas complet

- Observatie: În cazul motoarelor bipolare, se activează de fiecare dată câte o bobină.

Aplicaţii

8.1 Comanda motoarelor pas cu pas

a) Secvenţă pas complet pt motor unipolar

```
#include <REG2051.H>
#define stepper P1
void delay();

void main(){
        while(1){
                stepper = 0x0C;
                delay();
                stepper = 0x06;
                delay();
                stepper = 0x03;
                delay();
                stepper = 0x09;
                delay();
        }
}

void delay(){
        unsigned char i,j,k;
        for(i=0;i<6;i++)
                for(j=0;j<255;j++)
                        for(k=0;k<255;k++);
}
```

```
ok      org 0H
stepper equ P1
main:
        mov stepper, #0CH
        acall delay
        mov stepper, #06H
        acall delay
        mov stepper, #03H
        acall delay
        mov stepper, #09H
        acall delay
        sjmp main
delay:
        mov r7,#4
wait2:
        mov r6,#0FFH
wait1:
        mov r5,#0FFH
wait:
        djnz r5,wait
        djnz r6,wait1
        djnz r7,wait2
        ret
 end
```

Aplicaţii

8.1 Comanda motoarelor pas cu pas

b) Secvenţă pas complet pt motor bipolar

```
#include <REG2051.H>
#define stepper P1
void delay();

void main(){
        while(1){
                stepper = 0x08;
                delay();
                stepper = 0x02;
                delay();
                stepper = 0x04;
                delay();
                stepper = 0x01;
                delay();
        }
}

void delay(){
        unsigned char i,j,k;
        for(i=0;i<6;i++)
                for(j=0;j<255;j++)
                        for(k=0;k<255;k++);
}
```

```
org 0H
stepper equ P1
main:
        mov stepper, #08H
        acall delay
        mov stepper, #02H
        acall delay
        mov stepper, #04H
        acall delay
        mov stepper, #01H
        acall delay
        sjmp main
delay:
        mov r7,#4
wait2:
        mov r6,#0FFH
wait1:
        mov r5,#0FFH
wait:
        djnz r5,wait
        djnz r6,wait1
        djnz r7,wait2
        ret
        end
```

Aplicaţii

8.1 Comanda motoarelor pas cu pas

Exerciţiu: Să se realizeze comanda motorului pas cu pas unipolar în regim de semipas

(schemă + cod C + cod asamblare + comentarii)

Aplicaţii

8.2 Comanda motoarelor de curent continuu

- În termeni de viteză, marime, cost, motoarele de curent continuu sunt întotdeauna preferate motoarelor pas cu pas. Şi la motoarele de curent continuu se poate controla viteza, direcţia de rotaţie etc.
- Pentru *conectarea* unui motor de curent continuu la microcontrolerul 80C51, se utilizează o punte de tip H, ca de exemplu circuitul integrat L293D, cu 16 terminale. Denumirea de punte H provine de la forma circuitului care controlează mişcarea motorului. Această configuraţie mai este denumită şi punte completă ("Full Bridge"). De regulă, o punte de tip H este formată din circuite întrerupătoare

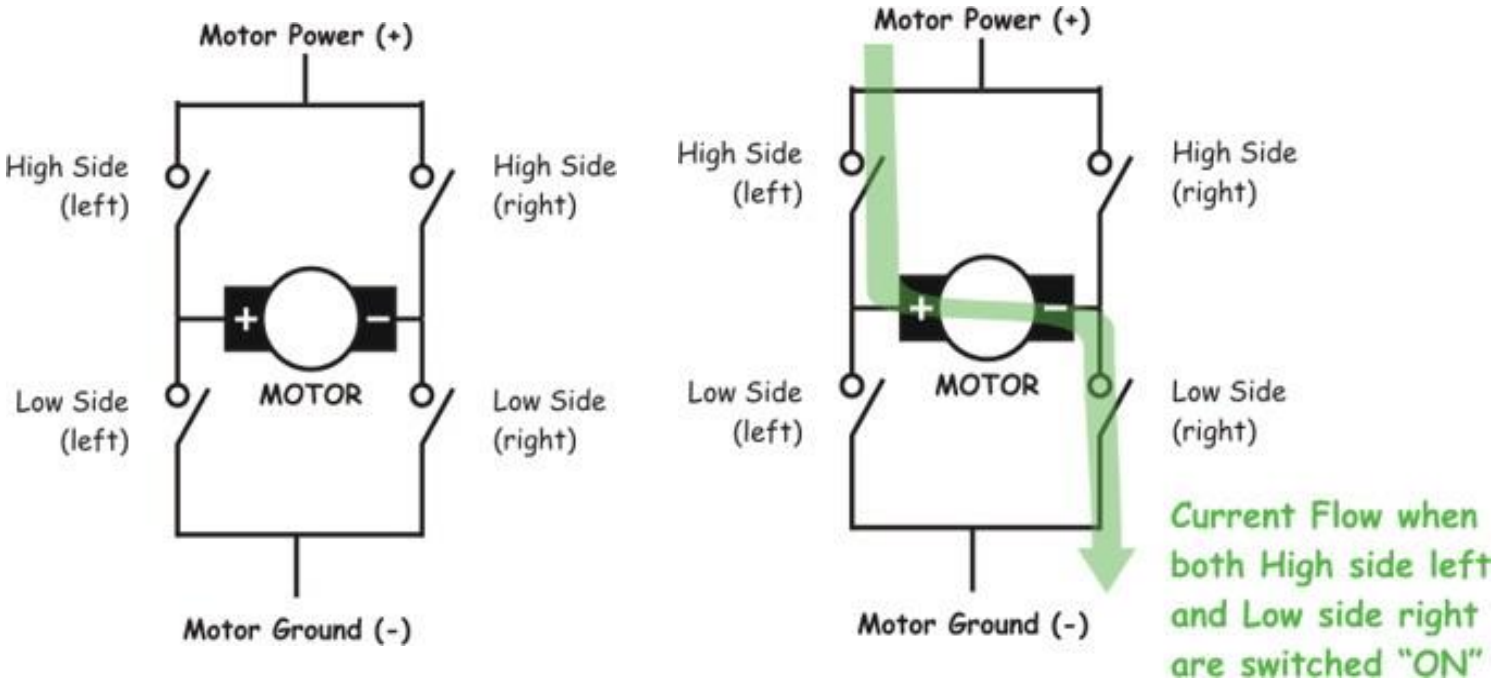
Motor de curent continuu folosit la Fiat Doblo

Motoare de curent continuu folosite la produse Panasonic

Aplicaţii

8.2 Comanda motoarelor de curent continuu

- Când întrerupătoarele sunt activate în pereche, motorul îşi va schimba direcţia în mod corespunzător. În acest sens, dacă se activează "High side left" şi "Low side right", curentul va străbate traseul de la alimentare către masă prin miezul motorului, rotind motorul în sensul acelor de ceasornic. În mod similar, când se activează "Low side left" şi "High side right", motorul va fi rotit în sensul invers acelor de ceasornic

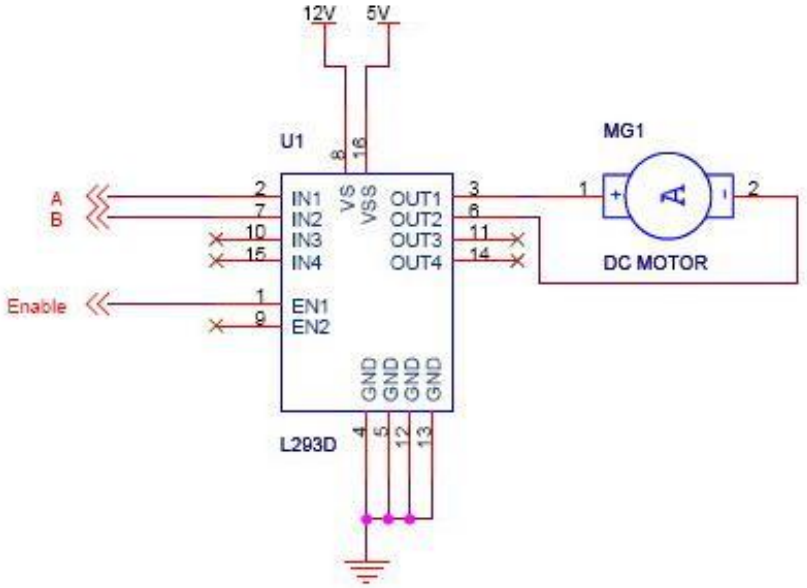


Aplicaţii

8.2 Comanda motoarelor de curent continuu

- Circuitul L293D este o punte H dublă, prin urmare cu acest circuit se pot interfaţa două motoare de curent continuu. Pentru conectarea unui astfel de motor, este nevoie de 3 terminale (A, B, Enable). Ieşirea OUT1 se contectează la borna pozitivă a motorului, iar ieşirea OUT2 se contectează la borna negativă.

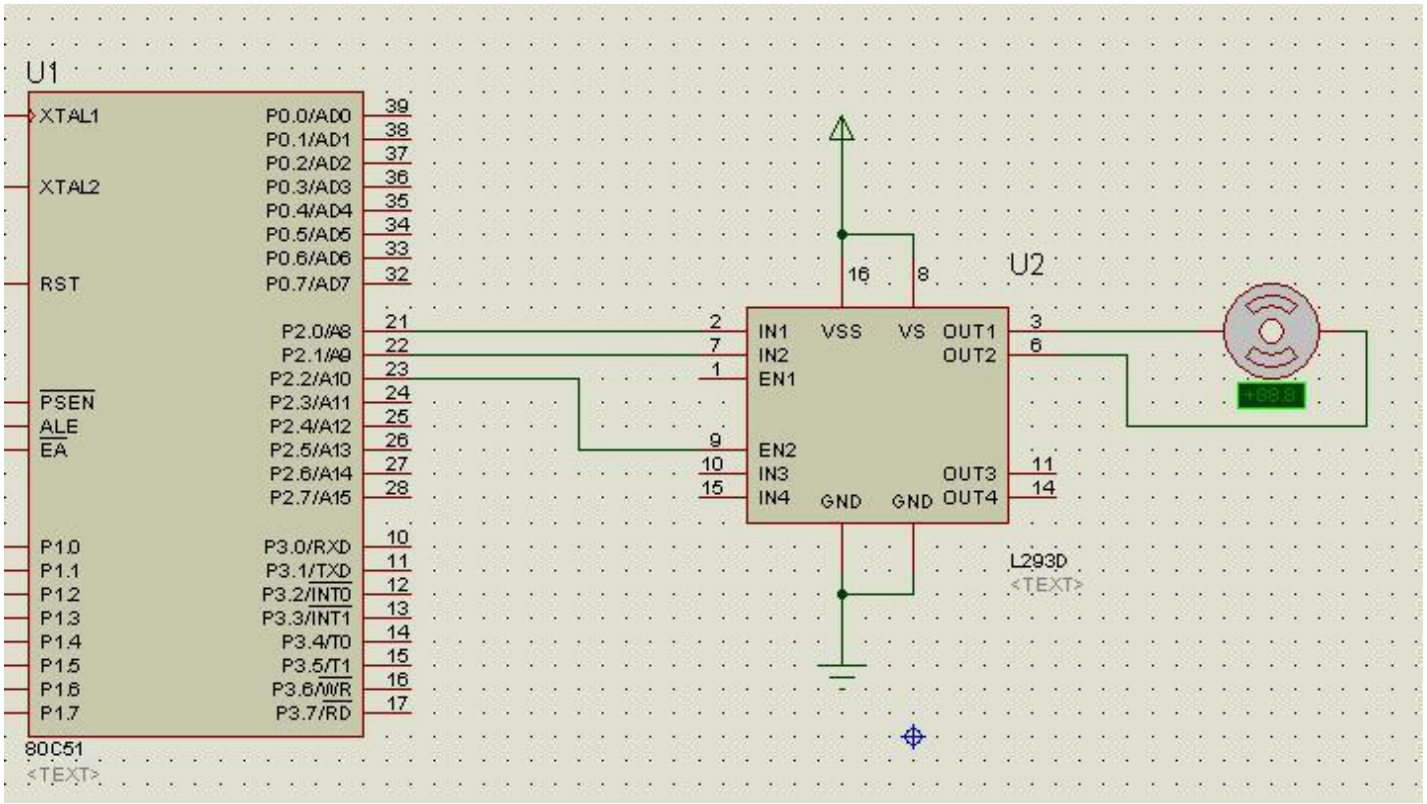
High Left	High Right	Low Left	Low Right	Description
On	Off	Off	On	Motorul se roteşte în sensul acelor de ceasornic
Off	On	On	Off	Motorul se roteşte în sensul invers acelor de ceasornic
On	On	Off	Off	Motorul se opreşte
Off	Off	On	On	Motorul se opreşte



Aplicaţii

8.2 Comanda motoarelor de curent continuu

- Schema de conexiune la 80C51 (Simulatorul Proteus):



Aplicaţii

8.2 Comanda motoarelor de curent continuu

```
#include <AT89X51.H>
#define L293D_A P2_0        //Terminal al portului aferent terminalului pozitiv al motorului
#define L293D_B P2_1        //Terminal al portului aferent terminalului negativ al motorului
#define L293D_E P2_2       // Terminal al portului aferent terminalului de Enable de la L293D

void rotate_f();           // Prototipuri de functii
void breaks();

void main(){                  //funcţia main
    while(1){                 //ciclu infinit
                rotate_f();           //Rotire în sensul acelor de ceasornic
                delay();              //Întârziere
                breaks();             //Stop
                delay();              //Întârziere
     }
}
void rotate_f(){
       L293D_A = 1;        // Trimite 1 la terminalul pozitiv al motorului
       L293D_B = 0;        // Trimite 0 la terminalul negativ al motorului
       L293D_E = 1;             //Activează L293D
}
void breaks(){
      L293D_A = 0;         // Trimite 0 la terminalul pozitiv al motorului
      L293D_B = 0;         // Trimite 0 la terminalul negativ al motorului
      L293D_E = 0;              //Dezactivează L293D
}
//implementare funcţie delay - a se vedea Aplicaţia 1
```

Aplicaţii

8.2 Comanda motoarelor de curent continuu

```
L293D_A   equ P2.0        ;L293D A - Terminal al portului aferent terminalului pozitiv al motorului
L293D_B   equ P2.1        ;L293D B - Terminal al portului aferent terminalului negativ al motorului
L293D_E   equ P2.2        ;L293D E – Terminal al portului aferent terminalului de Enable de la L293D
        org 0H
Main:
        acall rotate_f    ;Roteste motorul in sensul acelor de ceasornic
        acall delay       ;Motorul se roteste un anumit timp
        acall break       ;Oprire motor
        acall delay       ;Asteptare un anumit timp
        sjmp  Main        ;Ciclu

rotate_f:
        setb  L293D_A     ; Trimite 1 la terminalul pozitiv al motorului
        clr   L293D_B     ; Trimite 0 la terminalul negativ al motorului
        setb  L293D_E     ; Activează L293D
        ret               ; intoarcere din rutina
break:
        clr   L293D_A     ; Trimite 0 la terminalul pozitiv al motorului
        clr   L293D_B     ; Trimite 0 la terminalul negativ al motorului
        clr   L293D_E     ; Dezactivează L293D
        ret               ; intoarcere din rutina
```

Aplicaţii

8.2 Comanda motoarelor de curent continuu

Exerciţiul 1: Să se realizeze comanda motorului de curent continuu în sensul invers acelor de ceasornic (schemă + cod C + cod asamblare + comentarii)

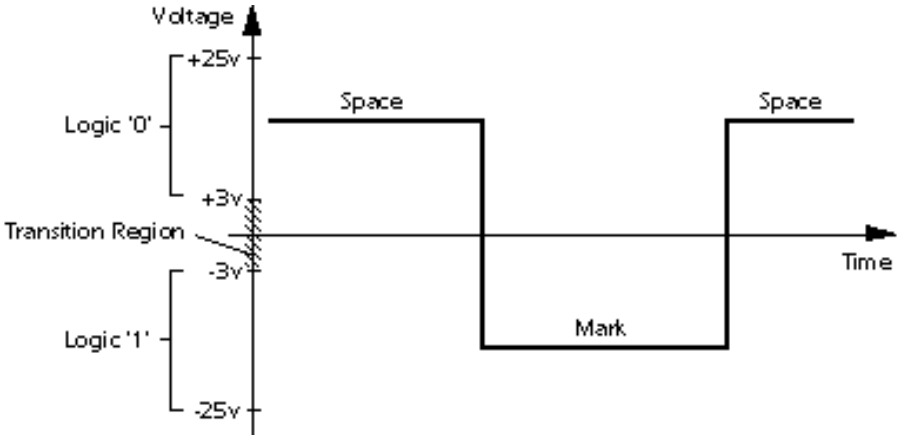
Exerciţiul 2: Să se realizeze un program care permite comanda motorului de curent continuu în sensul acelor de ceasornic, se va aştepta un anumit timp introdus printr-o metodă delay(), apoi motorul se va roti în sens invers acelor de ceasornic (schemă + cod C + cod asamblare + comentarii)

Aplicaţii

8.3 Comunicarea serială

- Interfaţa UART (Universal Asynchronous Receiver Transmitter) sau USART (Universal Synchronous Asynchronous Receiver Transmitter)
- Eficientă dpdv al costului, simplă şi fiabilă;
- Comunicare microcontroler cu alte microcontrolere, microcontroler cu PC etc.

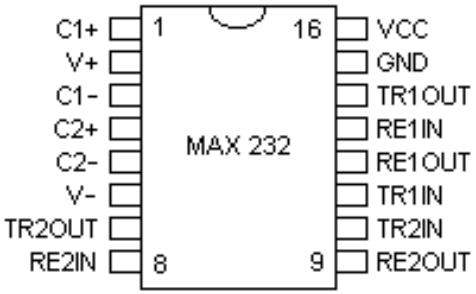
- RS-232 (Recommended Standard 232) este un standard pentru semnale seriale binare care conectează un DTE (Data terminal equipment) şi un DCE (Data Circuit-terminating Equipment)
- Acest standard defineşte nivelele de tensiune care corespund la 1 şi 0 logic
- Semnale valide sunt de la +/- 3 V până la +/- 25 V; “Space” – semnificaţie ON; “Mark” – semnificaţie OFF;
  - 0 logic este definit de la +3V la +25V
  - 1 logic este definit de la -3V la -25V

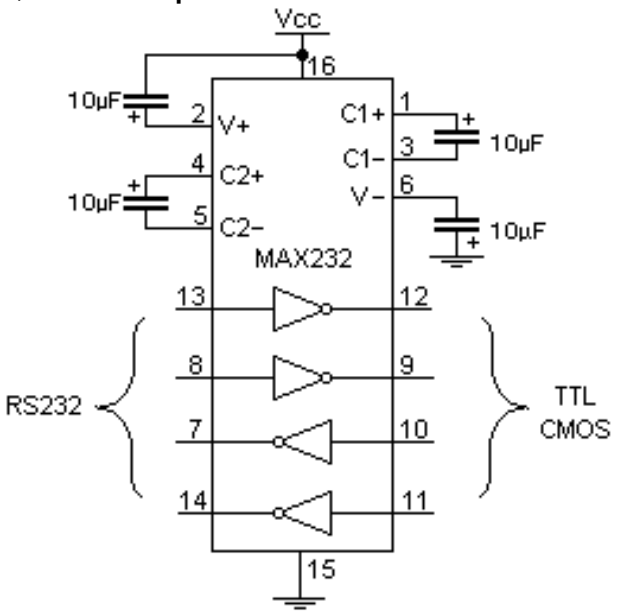


Aplicaţii

8.3 Comunicarea serială

- Convertoare de nivel RS-232: de regulă, circuitele integrate funcţionează bazându-se pe nivele de tensiune TTL sau CMOS care nu pot fi folosite să comunice cu protocolul RS-232; prin urmare este necesar a folosi un aşa-numit convertor de tensiune sau nivel care poate converti nivel TTL în RS-232 şi nivel RS-232 în TTL
- Cel mai des folosit convertor de nivel RS-232 este MAX232;
- acest circuit poate genera nivele de tensiune RS-232 (de la -10V la 10V) de la o sursă de 5V;
- Include două receptoare şi două transmiţătoare, fiind capabil de comunicare UART/USART full-duplex;

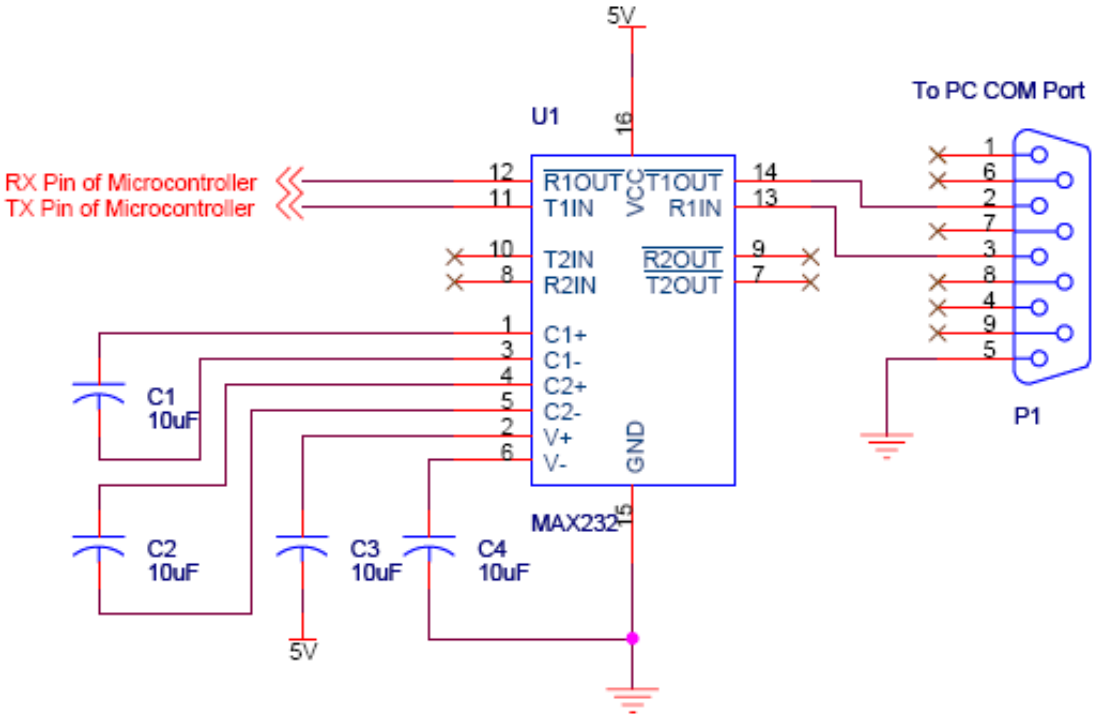




Aplicaţii

8.3 Comunicarea serială

- Conectarea MAX232 la microcontrolere: pentru a comunica prin intermediul interfeţei UART sau USART sunt utilizate 3 semnale:
  - RXD (receive)
  - TXD (transmit)
  - GND (ground)
- Pentru a conecta orice microcontroler la MAX232, avem nevoie de aceste 3 semnale



Aplicaţii

8.3 Comunicarea serială

- La 80C51: folosim Timer 1 pentru a genera rata de baud 9600 la un tact de 11.0592MHz
- Pentru a comunica prin UART folosim registrele:
  - TMOD, SCON, TCON, TH1 şi TL1 (registre pentru Timer 1, determinând rata de baud a UART)

```
#include <reg51.h>

void serial_init(){
      // setează Timer 1 la modul 8-bit Auto-Reload
         TMOD = 0x20;
      // activează recepţia
      // setează modul portului serial la 8-bit UART
         SCON = 0x50;
      // setează rata de baud la 9600 şi tact 11.0592 MHz
         TH1  = 0xFD;
         TL1  = 0xFD;
      // porneşte timerul
         TR1  = 1;
}
```

```
Serial_Init:
        mov TMOD,#20H
        mov SCON,#50H
        mov TH1,#0FDH
        mov TL1,#0FDH
        ;Start Timer
        setb TR1
        ret
```

Aplicaţii

8.3 Comunicarea serială

- Pentru a trimite date la portul serial trebuie să mutăm datele în SBUF (serial buffer register) şi aşteptat a.î. flagul Transmit Interrupt să fie setat
- La recepţie, se aşteaptă ca Receive Interrupt flag să fie setat, apoi se citeşte data din registrul SBUF

```
 void serial_send(unsigned char dat){
// aşteptare până când ultima dată este trimisă complet
        while(!TI);
// Flagul de Transmit Interrupt este pus pe 0 (clear)
        TI = 0;
// Mutare data de transmis in SBUF
        SBUF = dat;
}
unsigned char serial_read(){
// aşteptare după Receive interrupt flag
        while(!RI);
// dacă flagul este setat, atunci setare la 0 (clear)
        RI = 0;
// se citeşte data din SBUF
        return SBUF;
}
```

```
Serial_Send:
        jnb TI,Serial_Send
        clr TI
        mov SBUF,A
        ret

Serial_Read:
        jnb RI,Serial_Rea
    d
        clr RI
        mov A,SBUF
        ret
```

Aplicaţii

8.4 Programarea în C a întreruperilor microcontrolerului 80C51

- O întrerupere este un semnal asincron care necesită o anumită atenţie;
- opreşte CPU-ul microcontrolerului, acesta oprind execuţia taskului curent pentru a trata întreruperea;
- după ce întreruperea a fost tratată CPU-ul microcontrolerului reia execuţia taskului oprit.

	Interrupt	Source	Priority Number
1	RESET	RST	N/A
2	External Interrupt 0	IE0	0
3	Timer 0 Interrupt	TF0	1
4	External Interrupt 1	IE1	2
5	Timer 1 Interrupt	TF1	3
6	UART Interrupt	RI or TI	4

Activarea unei întreruperi:

1. Iniţializarea surselor de întreruperi, cum ar fi temporizatoarele (Timers), întreruperile externe sau UART;
2. Setarea biţilor din registrul IE care corespund surselor de întreruperi care doresc a fi activate

   Exemplu: activarea întreruperilor portului serial sau UART: ES = 1
3. Activarea întreruperii globale prin setarea bitului EA din registrul IE (EA = 1)

Aplicaţii

8.4 Programarea în C a întreruperilor microcontrolerului 80C51

Programarea rutinei de tratare a întreruperii (ISR, interrupt service routine)

- Rutina de tratare a întreruperii (ISR) este acea rutină pe care microcontrolerul o apelează de fiecare dată când apare o întrerupere; poate apărea sub forma unei metode în C;

```
void your_ISR_name(void) interrupt interrupt_priority_number
{
        //your routine here
}
```

- Programarea unei întreruperi introduse de modulul **c**ontor/**t**emporizator (counter/timer)

1. Iniţializarea modulului C/T prin încărcarea în TMOD a unei valori corespunzătoare
2. Încărcarea în THx şi TLx a unor valori iniţiale de numărare
3. Setarea biţilor ET1 şi ET0 din registrul IE pentru a activa întreruperile Timer 1 şi Timer 0
4. Activarea întreruperilor globale prin setarea bitului EA din registrul IE
5. Pornirea modulului prin setarea TRx.

Aplicaţii

8.4 Programarea în C a întreruperilor microcontrolerului 80C51

- *Exemplu:* Să se scrie un program C care va genera un semnal dreptunghiular cu o perioadă de 20ms la pinul P3.0. Frecvenţa de lucru: 11.0592 MHz.

```
#include<reg51.h>

sbit pulse = P3^0;

void toggle_pin(void) interrupt 3 {
      pulse = ~pulse;  //complementare P3.0
}
void main(void) {
      TMOD = 0b00010000; //Timer 1, mode 1 (16-bit)
      TL1 = 0x00;     // încărcare TL cu 0x00
      TH1 = 0xDC;     // încărcare TH cu 0xDC
      ET1 = 1;      // activare întrerupere Timer 1
      EA = 1;       // activare întrerupere globală
      TR1 = 1;      //start Timer1
      while(1);    //ciclu infinit
}
```

Observaţie:

- Flagul pentru Timer nu este monitorizat în program => minimizare încărcare MCU
- Flagul pentru Timer este automat pus la 0 de către hardware de fiecare dată după ce ISR este apelată.

$$YYXX_{hexadecimal} = 65536_{decimal} - (delay \div \frac{12_{decimal}}{XTAL})$$

$$YYXX_{hexadecimal} = 65536_{decimal} - \left(10 \times 10^{-3} seconds \div \frac{12_{decimal}}{11.0502 \times 10^{6} Hz}\right)$$

$$YYXX_{hexadecimal} = 56320_{decimal} = DC00_{hexadecimal}$$

Aplicaţii

8.4 Programarea în C a întreruperilor microcontrolerului 80C51

- Programarea unei întreruperi introduse de UART

1. Setarea Timer 1 pentru a funcţiona în modul 8-bit auto reload
2. Încărcarea în registrul TH1 a unei valori corespunzătoare baud rate
3. Încărcare registrul SCON pt a seta UART să funcţioneze în modul dorit
4. Pornire Timer 1 prin setare TR1
5. Ştergere flag RI sau TI
6. Activarea întreruperii pe portul serial prin setarea ES (ES = 1)
7. Activarea întreruperii globale prin setarea EA (EA = 1)

```
void your_uart_isr(void) interrupt 4 {
   if(TI)          //verificare dacă intreruperea e cauzată de Transmit Flag
   {
        //your routine here
        TI=0;          //clear TI
   }
   else     // verificare dacă intreruperea e cauzată de Receive Flag
   {
        //your routine here
        RI=0;          //clear RI
   }
}
```

Aplicaţii

8.4 Programarea în C a întreruperilor microcontrolerului 80C51

Exemplu: Să se scrie un program care va permite microcontrolerului 80C51 să primească date via UART şi să mute datele recepţionate la pinii portului 1. Se foloseşte Mode 1, 8-bits, 1 bit de start, 1 bit de stop, fără bit de paritate, 9600bps, tact 11.0592 MHz.

```
void my_uart_isr(void) interrupt 4 {
    if(TI)  //verifică dacă întreruperea e cauzată de Transmit Flag
          TI = 0;          //clear TI
    else     // dacă întreruperea e cauzată de Receive Flag
    {
          P1 = SBUF;
          RI = 0;         //clear RI
    }
}
void main(void) {
    TMOD = 0x20;     //Timer 1, mode 2 (auto-reload)
    TH1 = 0xFD;      //încarcă TH cu -3 sau FDh
    SCON = 0x50;     //UART mode 1, receive enabled
    TR1 = 1;         //start Timer1
    RI = 0;          //clear RI
    ES = 1;            //activează întreruperea UART
    EA = 1;            // activează întreruperea globală
    while(1);         //loop forever
}
```

Observaţie:

- Spre deosebire de întreruperile de la contoare / temporizatoare, TI sau RI trebuie puse la 0 în ISR

Aplicaţii

8.4 Programarea în C a întreruperilor microcontrolerului 80C51

- Programarea în cazul întreruperilor externe
- Microcontrolerul 80C51 are două intrări pt întreruperi externe, fiind localizate la pinii P3.2 şi P3.3;
- pinii P3.2 şi P3.3 sunt denumiţi şi INT0 şi INT1;
- Întreruperile externe sunt activate fie pe front, fie pe nivel;
- dacă o întrerupere este activată pe nivel, atunci întreruperea apare când un semnal “low” este aplicat la intrarea întreruperii externe
- dacă o întrerupere este activată pe front, atunci întreruperea apare când o tranziţie “high-to-low” este aplicată la intrarea întreruperii externe

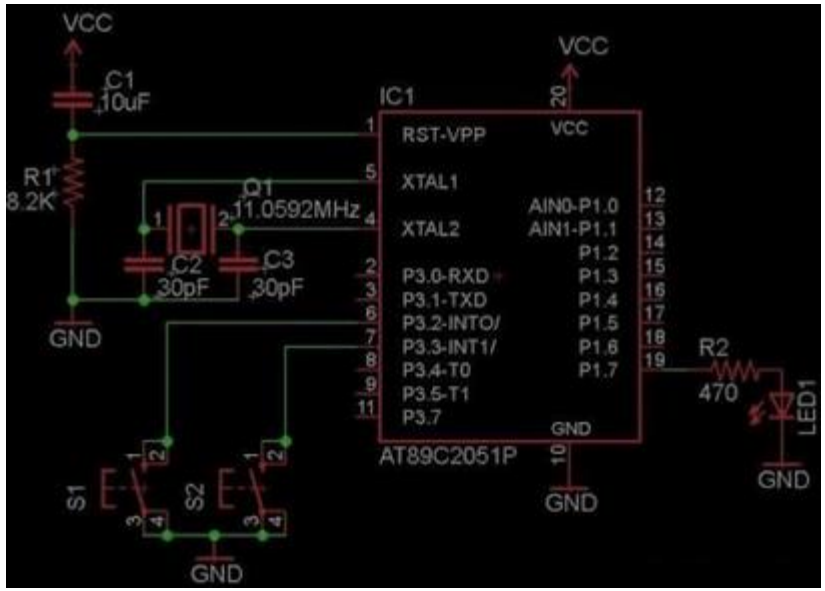
- Paşii care trebuie realizaţi:

1. Iniţializare a întreruperilor externe care vor fi folosite; selectarea dacă întreruperea externă este activată pe front sau pe nivel se realizează prin setarea la 1 sau 0 a IT0 pt întreruperea externă 0 sau IT1 pentru întreruperea externă 1;
2. Activarea unei anumite întreruperi externe care va fi folosita prin setarea EX0 sau EX1 a registrului IE;
3. Activarea întreruperilor globale prin setarea EA din registrul IE.

Aplicaţii

8.4 Programarea în C a întreruperilor microcontrolerului 80C51

- Exemplu: Dat fiind circuitul de mai jos, să se scrie un program care va porni LED-ul când S1 este apăsat şi
- va opri LED-ul când S2 este apăsat.




```
#include<reg51.h>
sbit LED = P1^7;
//ISR pentru întreruperea externă 0
void turn_on(void) interrupt 0 {
    LED=1;
}
//ISR pentru întreruperea externă 1
void turn_off(void) interrupt 2 {
    LED=0;
}
void main(void) {
    LED=0;        //Turn LED off
    IT0=1;  //setare întrerupere externă 0 activată pe front
    IT1=1;  //setare întrerupere externă 1 activată pe front
    EX0=1;          //activare întrerupere externă 0
    EX1=1;          //activare întrerupere externă 1
    EA=1;           //activare întrerupere globală
    while(1);        //loop forever
}
```

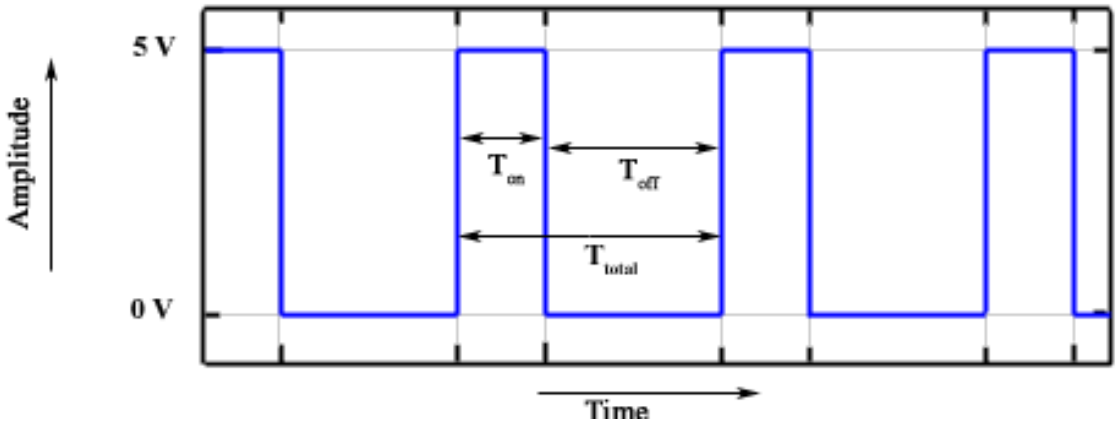
Observaţie:

- Flagul pentru întrerupere externă nu este monitorizat în program => minimizare încărcare MCU
- Flagul pentru întrerupere externă este automat pus la 0 de către hardware de fiecare dată după ce ISR este apelată.

Aplicaţii

8.5 Generarea unor semnale PWM folosind 80C51

- PWM (Pulse Width Modulation) reprezintă o tehnică de generare a unui semnal dreptunghiular a cărui factor de umplere este schimbat pt a obţine o ieşire în tensiune variabilă;
- este folosit în aplicaţii de control: controlul vitezei, a puterii, în comunicaţii etc



$$T_{total} = T_{on} + T_{off}$$

$$D = \frac{T_{on}}{(T_{on} + T_{off})} = \frac{T_{on}}{T_{total}}$$

$$V_{out} = D \times V_{in}$$

$$\boxed{V_{out} = \frac{T_{on}}{T_{total}} \times V_{in}}$$

- Se observă că tensiunea de ieşire variază în funcţie de **factorul de umplere D**
- Din ultima ecuaţie rezultă că tensiunea de ieşire poate fi variată prin variaţia valorii T_{on} (adică a perioadei cât semnalul generat este pe 1)
- dacă T_{on} =0 atunci V_{out} este 0
- dacă T_{on} = T_{total} atunci V_{out} este V_{in} adică maxim

Aplicaţii

8.5 Generarea unor semnale PWM folosind 80C51

- Implementarea se bazează pe modulele contor/temporizator (timers) şi pe alternarea high/low la pinul la care se doreşte generarea semnalului
- vom folosi Timer 0, Mode 0
- pentru nivel “high” vom încărca o valoare X în TH0, iar pt nivel “low” va fi incărcată în TH0 valoarea 255 – X, a.î. totalul e 255
- pwm_width = 0 => tensiunea de ieşire va fi 0V, iar pt pwm_width = 255 => tensiunea de ieşire va fi 5V; poate fi setată orice valoare de la 0 la 255;

```
#define PWMPIN P1_0
 //pinul pe care se generează semnal PWM
unsigned char pwm_width;
bit pwm_flag = 0;

void pwm_setup(){
  TMOD = 0; // Timer 0, Mode 0
  pwm_width = 160; /*valoare ce determină
tensiunea de ieşire*/
  EA = 1; // activare întreruperi
  ET0 = 1; // activare întreruperi Timer 0
  TR0 = 1; // pornire Timer
}
```

```
void timer0() interrupt 1 {
  if(!pwm_flag) { //Start al nivelului High
     pwm_flag = 1;    //Setare flag
     PWMPIN = 1;      //Setare PWM o/p pin
     TH0 = pwm_width;      //Load timer
     TF0 = 0;  //Ştergere flag de întrerupere
     return;  // întoarcere din ISR
   }
   else {  //Start al nivelului Low
      pwm_flag = 0;    //Ştergere flag
      PWMPIN = 0;  //Pune pe 0 PWM o/p pin
      TH0 = 255 - pwm_width;  //Load timer
      TF0 = 0; //Ştergere flag de întrerupere
      return;      // întoarcere din ISR
    }
}
```

Aplicaţii

8.5 Generarea unor semnale PWM folosind 80C51

```
void timer0() interrupt 1 {
  if(!pwm_flag) {          //Start al nivelului High
     pwm_flag = 1;         //Setare flag
     PWMPIN = 1;           //Setare PWM o/p pin
     TH0 = pwm_width;      //Load timer
     TF0 = 0;              //Ştergere flag de întrerupere
     return;               // întoarcere din ISR
   }
   else {                  //Start al nivelului Low
      pwm_flag = 0;        //Ştergere flag
      PWMPIN = 0;          //Pune pe 0 PWM o/p pin
      TH0 = 255 - pwm_width;  //Load timer
      TF0 = 0;             //Ştergere flag de întrerupere
      return;              // întoarcere din ISR
    }
}
```

Sisteme Încorporate

- Header reg51.h:

```
/*BYTE Register*/
sfr P0   = 0x80;
sfr P1   = 0x90;
sfr P2   = 0xA0;
sfr P3   = 0xB0;
sfr PSW  = 0xD0;
sfr ACC  = 0xE0;
sfr B    = 0xF0;
sfr SP   = 0x81;
sfr DPL  = 0x82;
sfr DPH  = 0x83;
sfr PCON = 0x87;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr TL0  = 0x8A;
sfr TL1  = 0x8B;
sfr TH0  = 0x8C;
sfr TH1  = 0x8D;
sfr IE   = 0xA8;
sfr IP   = 0xB8;
sfr SCON = 0x98;
sfr SBUF = 0x99;

/*BIT Register*/
/*  PSW   */
sbit CY   = 0xD7;
sbit AC   = 0xD6;
sbit F0   = 0xD5;
sbit RS1  = 0xD4;
sbit RS0  = 0xD3;
sbit OV   = 0xD2;
sbit P    = 0xD0;

/*  TCON  */
sbit TF1  = 0x8F;
sbit TR1  = 0x8E;
sbit TF0  = 0x8D;
sbit TR0  = 0x8C;
sbit IE1  = 0x8B;
sbit IT1  = 0x8A;
sbit IE0  = 0x89;
sbit IT0  = 0x88;

/*  IE    */
sbit EA   = 0xAF;
sbit ES   = 0xAC;
sbit ET1  = 0xAB;
sbit EX1  = 0xAA;
sbit ET0  = 0xA9;
sbit EX0  = 0xA8;

/*  IP    */
sbit PS   = 0xBC;
sbit PT1  = 0xBB;
sbit PX1  = 0xBA;
sbit PT0  = 0xB9;
sbit PX0  = 0xB8;

/*  P3    */
sbit RD   = 0xB7;
sbit WR   = 0xB6;
sbit T1   = 0xB5;
sbit T0   = 0xB4;
sbit INT1 = 0xB3;
sbit INT0 = 0xB2;
sbit TXD  = 0xB1;
sbit RXD  = 0xB0;

/*  SCON  */
sbit SM0  = 0x9F;
sbit SM1  = 0x9E;
sbit SM2  = 0x9D;
sbit REN  = 0x9C;
sbit TB8  = 0x9B;
sbit RB8  = 0x9A;
sbit TI   = 0x99;
sbit RI   = 0x98;
```