8051 seria druga

Adrian Jałoszewski

1 Port szeregowy - wysyłanie ciągu znaków w pętli głównej

Inicjalizacja:

```
INIT:   MOV TMOD, #20H
        MOV TL1, #245    ; 2400 baud
        MOV TH1, #245
        MOV SCON, #50H
        SETB TR1
```

Podprocedura wysyłająca dane z akumulatora przez port szeregowy.

```
SEND:   MOV SBUF, A
        JNB TI, $
        CLR TI
        RET
```

Dane do wysłania, zakończone zerem:

```
DATA_TO_TRANSFER:       ;"Hello world!"
        DB 72
        DB 101
        DB 108
        DB 108
        DB 111
        DB 32
        DB 87
        DB 111
        DB 114
        DB 108
        DB 100
        DB 33
        DB 0
```

Pętla główna:

```
MAIN:   MOV DPTR, #DATA_TO_TRANSFER
        MOV A, #0
REPEAT:
        PUSH A
        MOVC A, @A+DPTR
        JZ CLEAR_STACK
        ACALL SEND
        POP A
        INC A
        SJMP REPEAT
CLEAR_STACK:
        POP A
        SJMP MAIN
```

2 Port szeregowy - echo

Podprocedura odbierająca dane do akumulatora:

```
RECV:   JNB RI, $
        MOV A, SBUF
        CLR RI
        RET
```

Pętla główna:

```
MAIN:   ACALL RECV
        ACALL SEND
```

Echo selektywne wyświetlające litery od dużego 'A' do dużego 'Z'

```
MAIN:   ACALL RECV
        CJNE A, #0x5a, NOT_Z
        SJMP SEND_LETTER
NOT_Z:  JNC MAIN            ; >= 'Z'
        CJNE A, #0x41, NEXT
NEXT:   JNC SEND_LETTER ; >= 'A'
        SJMP MAIN
SEND_LETTER:
        ACALL SEND
        SJMP MAIN
```

Echo selektywne na przerwaniach, w inicjalizacji dodatkowo jest `MOV IE, #0x90`, dla włączenia przerwań:

```
HANDLE_SERIAL_INTERRUPT:
        PUSH A
        JNB TI RECEIVED
        CLR TI
        SJMP END_SERIAL_INTERRUPT
RECEIVED:
        MOV A, SBUF
        CLR, RI
        CJNE A, #0x5a, NOT_Z
        SJMP SEND_LETTER
NOT_Z:  JNC END_SERIAL_INTERRUPT            ; >= 'Z'
        CJNE A, #0x41, NEXT
NEXT:   JNC SEND_LETTER ; >= 'A'
        SJMP END_SERIAL_INTERRUPT
SEND_LETTER:
        MOV SBUF, A
END_SERIAL_INTERRUPT:
        POP A
        RETI
```

3 Wyświetlacze - statyczne wyświetlanie

Tablica przekodowań na wyświetlacz siedmiosegmentowy:

```
LED_TABLE:      ;0...9
        DB 0xc0
        DB 0xf9
        DB 0xa4
        DB 0xb0
        DB 0x99
        DB 0x92
        DB 0x82
        DB 0xf8
        DB 0x80
        DB 0x90
```

Kod inicjalizujący program - ustawia timer 16 bitowy oraz przerwania od niego, również inicjalizuje poszczególne cyfry.

```
INIT:   MOV     TMOD,   #0x1
        MOV     IE,     #0x82
        MOV     TH0,    #60
        MOV     TL0,    #176
        SETB    TR0
        MOV     DPTR,   #LED_TABLE
        MOV     R0,     #0
        MOV     R1,     #1
        MOV     R2,     #2
        MOV     R3,     #3
        MOV R5, #0
        SJMP $
```

Obsługa przerwania od timera:

```
T0_ISR:
        PUSH A
        CLR     TR0
        MOV     TH0,    #60
        MOV     TL0,    #176
        SETB    TR0
        CALL    CHANGE_LED
        POP A
        RETI
```

Funkcja odświeżająca wyświetlacze, R5 trzyma stan wyświetlacza aktywnego:

```
CHANGE_LED:
        MOV P0, #0xFF
        CJNE R5, #3, NOT3
        MOV P1, #8
        MOV A, R0
        MOVC A, @A+DPTR
        MOV     P0, A
        MOV R5 #2
        RET
NOT3:   CJNE R5, #2, NOT2
        MOV P1, #4
        MOV A, R1
        MOVC A, @A+DPTR
        MOV     P0, A
        MOV R5 #1
        RET
NOT2:   CJNE R5, #1, NOT1
        MOV P1, #2
        MOV A, R2
        MOVC A, @A+DPTR
        MOV     P0, A
        MOV R5 #0
        RET
NOT1:   MOV R5, #3
        MOV P1, #1
        MOV A, R3
        MOVC A, @A+DPTR
        MOV     P0, A
        MOV R5 #3
        RET
```

4 Licznik

Kod inicjalizujący program - ustawia timer 16 bitowy oraz przerwania od niego, również inicjalizuje poszczególne cyfry, również inicjalizuje rejestr R5 wartością 201, gdyż mając częstotliwość odświeżania 50Hz przełączamy między czterema wyświetlaczami co 5ms, więc po upływie 200 iteracji mamy upływ sekundy:

```
INIT:   MOV     TMOD,   #0x1
        MOV     IE,     #0x82
        MOV     TH0,    #60
        MOV     TL0,    #176
        SETB    TR0
        MOV     DPTR,   #LED_TABLE
        MOV     R0,     #0
        MOV     R1,     #0
        MOV     R2,     #0
        MOV     R3,     #0
        MOV     R4,  #201
        MOV R5,  #0
        SJMP  $
```

Obsługa przerwania od timera:

```
T0_ISR:
        PUSH  A
        CLR     TR0
        MOV     TH0,    #60
        MOV     TL0,    #176
        SETB    TR0
        DJNZ  R4,  NEXT
        MOV R4,  #201
        CALL  UPDATE_COUNTER
        NEXT:   CALL    CHANGE_LED
        POP  A
RETI
```

Funkcja uaktualiniająca wartości R0, R1, R2, R3, na podstawie wartości R6 zwiększa kolejną wartość o 1 lub nie.

```
UPDATE_COUNTER:
        MOV R6,  #1
        MOV A,  R0
        ACALL  GET_MODULO_TEN
        MOV R0,  A
        MOV A,  R1
        ACALL  GET_MODULO_TEN
        MOV R1,  A
        MOV A,  R2
        ACALL  GET_MODULO_TEN
        MOV R2,  A
        MOV A,  R3
```

```
        ACALL GET_MODULO_TEN
        MOV R3, A
        RET
```

Funkcja odpowiedzialna za zmianę wartości, zapisuje do R6 1, jeżeli nastąpiło przepełnienie:

```
GET_MODULO_TEN:
        CJNE R6, #1, RETURN_MODULO_TEN
        MOV R6, #0
        INC A
        CJNE A, #10, RETURN_MODULO_TEN
        MOV A, #0
        MOV R6, #1
RETURN_MODULO_TEN:
        RET
```

5 Zegarek

Aby zrobić z tego zegarek należy zamienić co drugie wywołanie funkcji GET_MODULO_TEN na GET_MODULO_SIX

```
UPDATE_COUNTER:
        MOV R6, #1
        MOV A, R0
        ACALL GET_MODULO_TEN
        MOV R0, A
        MOV A, R1
        ACALL GET_MODULO_SIX
        MOV R1, A
        MOV A, R2
        ACALL GET_MODULO_TEN
        MOV R2, A
        MOV A, R3
        ACALL GET_MODULO_SIX
        MOV R3, A
        RET
```

Funkcja GET_MODULO_SIX:

```
GET_MODULO_SIX:
        CJNE R6, #1, RETURN_MODULO_TEN
        MOV R6, #0
        INC A
        CJNE A, #6, RETURN_MODULO_SIX
        MOV A, #0
        MOV R6, #1
RETURN_MODULO_SIX:
        RET
```