Temat: System przerwań, liczniki i wyświetlacz w STRC51. Ćwiczenie 3.

1. Przerwania na procesorze 80C51

Przerwania są mechanizmem umożliwiającym połączenie zdarzeń (sygnałów) z odpowiednim wykonaniem fragmentu programu - wywoływanymi niezależnie od aktualnie wykonywanego kodu. Procedura obsługi przerwania musi być zapisany w specjalny sposób. W języku maszynowym wymagane jest na początku procedury przerwania zapamiętanie stanu przerwanego programu, a na końcu przywrócenie tego stanu. Pisząc programy w językach wysokiego poziomu (takich jak C), część pracy przejmuje na siebie kompilator. Pakiet SDCC umożliwia włączanie takich procedur w następujący sposób:

```
void func_isr (void) interrupt 2
{
...
}
```

Należy ten zapis rozumieć jako przypisanie do podprogramu obsługi przerwania numer 2 kodu opisanego miedzy klamrami, słowo kluczowe using oznacza że używane będą zmienne wewnętrzne w rejestrze banku 1[1]. W najprostszym podejściu można zrezygnować z używania tej deklaracji.

Wnętrze podprogramu obsługi przerwania powinno być jak najbardziej zwięzłe. Szczególnie należy zwrócić uwagę czy przed pojawieniem się następnego przerwania, obecna obsługa zostanie zakończona (co mogło by powodować nieprzewidywalne działanie programu). Z tego powodu niedopuszczalne są pętle opóźniające wewnątrz procedur przerwania (problem zakleszczania się kodu i opuszczania przerwań).

Jednym ze sposobów uniknięcia złego pisania kodu, jest przekazywanie danych wytworzonych w przerwaniu do kolejki obsługiwanej poza procedurą obsługi przerwania.

W procesorze 80C51 możliwe jest stosowanie jako źródeł przerwania następujących sygnałów:

- zmiana stanu logicznego na wejściu INT0 (pin 12),
- zmiana stanu logicznego na wejściu INT1 (pin 13),
- przepełnienie licznika T0 (pin 14),
- przepełnienie licznika T1 (pin 15),
- odebranie znaku lub zakończenie wysłania znaku przez port transmisji szeregowej.

W wersjach bogatszych od podstawowego 80C51, obsługiwane są przerwania od takich peryferii jak:

-zegar czasu rzeczywistego,
-modułu obsługi interfejsu I2C,
-przetworników A/C,
-wielu innych (zależnych od wersji mikrokontrolera).

Każdy z podprogramów obsługi przerwań jest umieszczany w innym miejscu pamięci CODE:

numer przerwania	Nazwa przerwania	adres podprogramu obsługi
0	Wejście INT 0	0x0003
1	Licznik Timer 0	0x000B
2	Wejście INT 1	0x0013
3	Licznik Timer 1	0x001B
4	Łącze Serial	0x0023

[1] Banki – zbiory rejestrów, do dyspozycji programisty są cztery banki, należy unikać ustawiania wielu procedurom obsługi przerwania tego samego banku (ze względu na szybkość i zwięzłość kodu wynikowego jak i stabilność kodu).

Temat: System przerwań, liczniki i wyświetlacz w STRC51. Ćwiczenie 3.

Pisząc program w języku C, linker umieszcza odpowiednie fragmenty kodu w odpowiednich miejscach pamięci. Obserwując w jaki sposób został przetłumaczony kod zapisany w C do postaci ASM (kompilator tłumaczy przejściowo z C na ASM, potem na kod relokowalny .rel), można zauważyć duży nadmiar instrukcji w okolicach początku i końca kodu obsługi przerwania. Taki nadmiar może wpływać niekorzystnie na szybkość działania programu. Rozwiązaniem tego problemu jest pisanie podprogramów „gołych" (ang. nacked). Definicja takiego podprogramu wygląda tak:

```
void nakedInterrupt(void) interrupt 2 _naked {
 ...
}
```

Pisanie takiego typu podprogramów, jest stosunkowo trudne – wymaga znajomości asemblera dla tego procesora. W większości przypadków w kod takiej funkcji wprowadza się wstawki asemblerowe:

```
void nakedInterrupt(void) interrupt 2 _naked {
 _asm
   ...
 _endasm;
}
```

2. Sekcje krytyczne i dzielenie zasobów.

Pisząc programy wykorzystujące przerwania należy uwzględnić asynchroniczną naturę przerwań (względem programu wykonywanego w danej chwili). Zasoby modyfikowane przez podprogram przerwania i program główny mogą doprowadzać do niejednoznaczności – losowego działania programu.

Np. Program główny wykonuje sekwencje:

```
if (p++ != 1){
  ...
}
```

A podprogram obsługi przerwania modyfikuje zmienną p w następujący sposób:

```
p--;
```

Jeżeli przerwanie "wystąpi" w chwili przed sprawdzeniem warunku '!=' może zostać podjęta mylna decyzja o prawdziwości warunku. Aby zabezpieczyć się przed tego typu sytuacjami, możliwe jest używanie sekcji krytycznych. Ogólna postać takiego zapisu wygląda następująco:

```
int funckaj() critical {
...
}
```

Oczywiście najważniejsze słowo tutaj to critical. Dla kompilatora oznacza to że podczas trwania funkcji należy tak skonstruować kod aby niemożliwe było obsługiwanie przerwań.

Nie należy umieszczać zbyt długo trwających fragmentów programu w sekcjach krytycznych – w trakcie ich trwania przerwania nie są obsługiwane ani kolejkowane (co może prowadzać do gubienia istotnych informacji).

3. Uruchamianie przerwań

System obsługi przerwań jest związany z bitem EA w rejestrze wewnętrznym procesora o nazwie: IE (ulokowanym pod adresem 0xA8). Aby z poziomu języka C można odwoływać się do tego rejestru należy napisać następującą definicję:

```
sbit at 0xAF EA;
```

Ustawiając zmienną `EA` na wartość równą 1 (zmienna tak zadeklarowana może przyjmować wartości: 0 lub 1), możemy zezwolić na przyjmowanie przerwań. W zależności które przerwanie ma zostać obsługiwane, należy ustawić odpowiednie bity w rejestrze `IE` o deklaracji:

```
sfr at 0xA8 IE;
```

O następujących polach:

7	6	5	4	3	2	1	0
EA	-	-	ES	ET1	EX1	ET0	EX0

`EA` – bit ogólnego zezwolenia na obsługę przerwań,
`ES` – bit zezwolenia na obsługę przerwań związanych z portem szeregowym,
`ET1` – bit zezwolenia na obsługę przerwań związanych z licznikiem `T1`,
`EX1` – bit zezwolenia na obsługę przerwań związanych z wejściem `/INT1`,
`ET0` – bit zezwolenia na obsługę przerwań związanych z licznikiem `T0`,
`EX0` – bit zezwolenia na obsługę przerwań związanych z wejściem `/INT0`,

Definicje pozostałych bitów wyglądają następująco:

```
sbit at 0xAC ES;
sbit at 0xAB ET1;
sbit at 0xAA EX1;
sbit at 0xA9 ET0;
sbit at 0xA8 EX0;
```

4. Liczniki

Jednym z najprostszych sposobów generowania przerwań jest ustawienie jednego z liczników w taki sposób aby jego przepełnienie (osiągniecie wartości maksymalnej) wywołało podprogram obsługi przerwania.

Układy liczników (w podstawowej wersji `80C51` mamy do dyspozycji dwa takie liczniki), mogą być ustawione w czterech trybach:

0 – tryb pracy 13 bitowy, stan licznika określają rejestry `TH0` i `TH1`, gdzie odliczanie przebiega od wartości wpisanej przed włączeniem licznika, do wartości 0 – co równocześnie może generować przerwanie,
1 – tryb pracy 16 bitowy, poza tą różnicą działający jak w trybie 0,
2 – tryb pracy 8 bitowy z przeładowaniem (po osiągnięciu wartości 0 następuje wpisanie do `TL0` zawartości z `TH0` – z równoczesnym wygenerowaniem przerwania i ponowne odliczanie),
3 – tryb pracy 8 bitowy z niezależnym odliczaniem `TH0` i `TL0` – bez możliwości przeładowywania.

Aby ustawić w tryb pracy danego licznika należy, zadeklarować w `C` rejestr `TMOD`:

```
sfr at 0x89 TMOD;
```

Który posiada następującą strukturę:

7	6	5	4	3	2	1	0
GATE_1	C/T_1	M1_1	M0_1	GATE_0	C/T_0	M1_0	M0_0

`GATE_1` – bit bramkujący licznik T1 – ustawiony na 1 powoduje zliczanie sygnałów pod warunkiem ustawienia wejścia /INT1 (wyzerowanie tego bitu uniezależni zliczanie od stanu tego pinu),
`C/T_1` – bit wybierający sygnał zliczania, dla ustawionego na „1” pobierany z wejścia T1, dla wyzerowanego pobierany z generatora synchonizującego pracę procesora podzielonego przez 12,

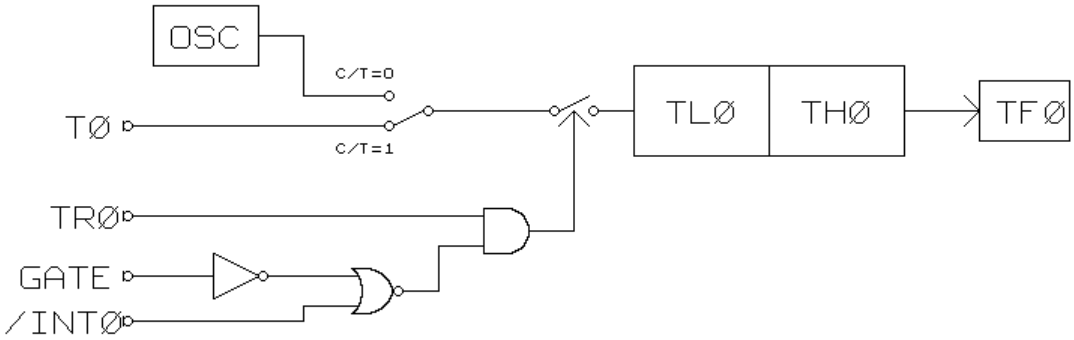
`M1_1, M0_1` – wybiera tryb pracy licznika:

M1_1	M0_1	
0	0	- tryb 0,
0	1	- tryb 1,
1	0	- tryb 2,
1	1	- tryb 3,

Ważne jest także aby ustawić bit TR0 zadeklarowany jako:

```
sbit at 0x8C TR0;
```

Który ustawiony, włącza układ licznikowy `T0`. Dla układu licznikowego `T1` analogicznie. Schemat na rys.1 pokazuje konfiguracje trybu pracy `1` licznika `0` (podobnie wyglądały by schematy dla pozostałych trybów, różnica wynikała by z konfiguracji `TL0` i `TH0`):



Rys. 1. Schemat wewnętrznej struktury generacji przerwań

Inne tryby pracy jak i działanie licznika T1 jest niemal identyczne (różnice wynikają z sposobu zliczania, a nie dostarczania sygnałów pobudzeń).

5. Przepełnienie liczników - źródło przerwań.

Jak widać na rysunku 1, przepełnienie jest sygnalizowane zmianą wartości bitu `TF0` (umieszczony w rejestrze `TCON` o deklaracji: `sbit at 0x8D TF0;`).

Można w tzw. poolingu analizować stan licznika (ewentualne przepełnienie), lub bardziej optymalnie włączyć obsługę przerwania od licznika ustawiając bit `ET0` w rejestrze `EA`.

Uwaga! Większość z rejestrów specjalnego znaczenia jest zdefiniowanych w pliku nagłówkowym `8051.h` w pakiecie `SDCC` (katalog: `sdcc\share\sdcc\include`).

Należy także zauważyć, że w trybie 0,1 liczniki wymagają ustawiania wartości do przeładowania za każdym razem po przeładowaniu (najczęściej robi się to w procedurze przerwania generowanego po przeładowaniu licznika). Natomiast w trybie 2 czynność taką wykonuje się podczas inicjowania licznika a po każdorazowym przepełnieniu następuje automatyczne odnowienie wartości.

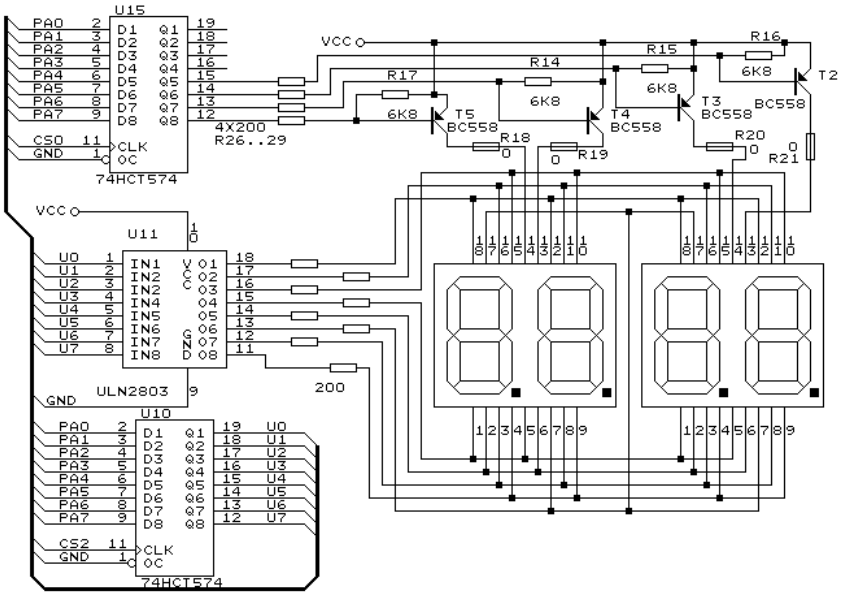
Wpisywanie ilości zliczonych impulsów dokonuje się przez wpisanie bezpośrednio do rejestrów `TL0` i `TH0` (i odpowiednio `TL1` i `TH1`). Wartość wpisywana do tych rejestrów nie jest liczbą impulsów jakie muszą być zliczone lecz od jakiej wartości maja one być zliczane aż do wartości maksymalnej (w przypadku trybów 16 bitowych, ilości zliczonych impulsów wynosi: `65535-THL0/1`, a w trybie 8 bitowym: `255-TH0/1`).

Deklaracje rejestrów są następujące:

Licznik	Starszy bajt	Młodszy bajt
T0	`sfr at 0x8C TH0;`	`sfr at 0x8A TL0;`
T1	`sfr at 0x8D TH1;`	`sfr at 0x8B TL1;`

6. Wyświetlacz LED

Schemat rys.2, przedstawia schemat wyświetlacza LED podłączony do magistrali procesora za pośrednictwem rejestrów `U10` i `U15`.



Rys. 2. Schemat obwodu sterującego wyświetlaczem LED

Są one umieszczone w przestrzeni pamięci danych. W języku `C` można zadeklarować te rejestry jako:

```
xdata at 0x8000 unsigned char U15;
xdata at 0xFFFF unsigned char U10;
```

Rejestry te są tylko do zapisu (nie można wykonywać operacji typu U10 &= 0x01; i podobnych). W tabeli pokazano definicje i maski jakie trzeba wysłać do układu U10 dla poszczególnych segmentów wyświetlacza.

Dla włączenia odpowiedniego segmentu można skorzystać z następujących definicji:	Nazwy segmentów są następujące:

`#define SEG_A 0x02`
`#define SEG_B 0x04`
`#define SEG_C 0x40`
`#define SEG_D 0x10`
`#define SEG_E 0x08`
`#define SEG_F 0x01`
`#define SEG_G 0x20`
`#define SEG_H 0x80`
	`  a`
	`---`
	`f| g |b`
	`---`
	`e|   |c`
	`---`
	`  d  .h`

Dla łatwiejszego definiowania kształtu wyświetlanych znaków warto użyć narzędzi języka C, w następujący sposób:

```
#define CYFRA_1 SEG_B | SEG_C
#define CYFRA_2 SEG_A | SEG_B | SEG_G | SEG_E | SEG_D
...
```

Po takim zdefiniowaniu symboli, dla przykładu aby wyświetlić znak o kształcie ‘2’ na pozycji cyfry 4 należy wykonać następujący kod:

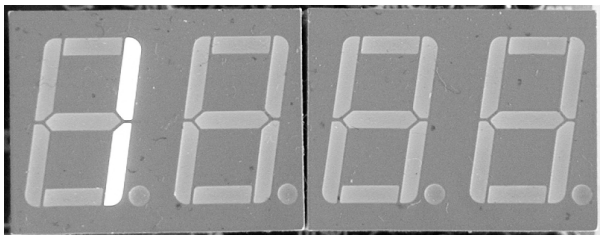
```
U15=0xBF;   //włączenie wyświetlacza na pozycji 4
U10=CYFRA_2;//włączenie takich segmentów aby cyfra wyglądała jak 2
```

Temat: System przerwań, liczniki i wyświetlacz w STRC51. Ćwiczenie 3.

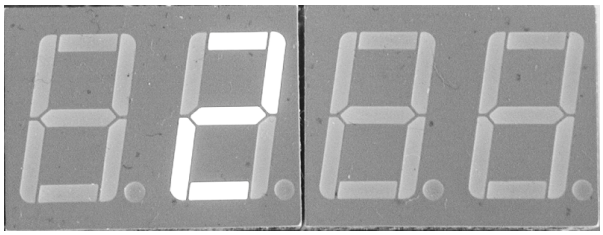
Włączanie poszczególnych cyfr wyświetlacza odbywa się przez wysyłanie do układu `U15` wartości z wyzerowanym bitem (jednym z czterech najstarszych bitów):

7	6	5	4	3	2	1	0
Cyfra 3	Cyfra 4	Cyfra 1	Cyfra 2	-	-	-	-
0x7F	0xBF	0xDF	0xEF	-	-	-	-

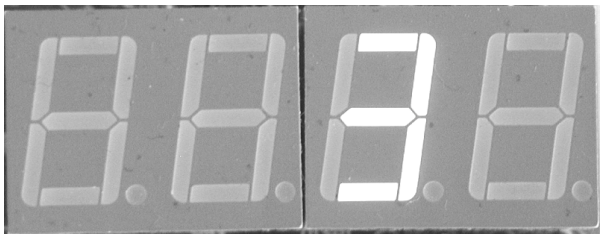
W dolnej linii tabeli podano przykładowe wartości do wysłania na układ `U15` aby włączyć możliwość wyświetlenia tylko jednej z cyfry. Wyłączenie więcej niż jednego z podanych bitów spowoduje włączenie dwóch lub więcej cyfr – co z reguły jest nie pożądane.
Jednym ze sposobów wypisywania na wyświetlaczu informacji jest wyzerowanie kolejno jednego z czterech bitów wysyłanych do `U15` i odpowiedniego wysterowania bitów wysyłanych do `U10`, takie rozwiązanie wymaga ciągłego – niewidzialnego dla oka – gaszenia i zapalania cyfr na kolejnych pozycjach (przemiatanie).
Przy bardzo dużej szybkości „przemiatania" wymagane jest przed zmianą cyfry (wyzerowanie następnego bitu) wpisanie do `U10` samych zer – zapobiega to „smurzeniu" wyświetlacza.
Na rysunkach 3,4,5,6 pokazano kolejne fazy budowania napisu 1234, w całości pokazanego na rysunku 7. Czas przejścia między kolejnymi fazami musi być bardzo krótki – rzędu 10ms – aby nie były widoczne przejścia.



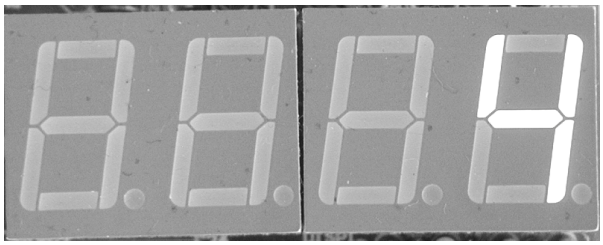
Rys. 3.



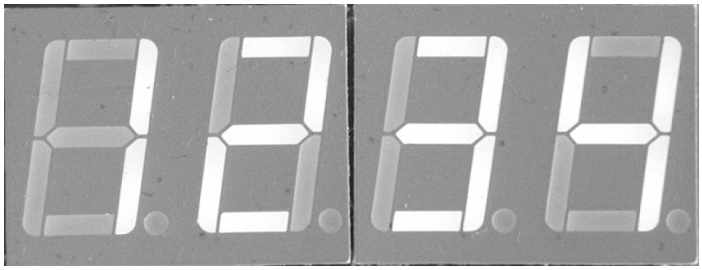
Rys. 4.



Rys. 5.



Rys. 6.



Rys. 7.

7. Zadanie do wykonania

Napisać program który pokazuje na wyświetlaczu efekt liczenia co około sekundę od 0 do 9999 z mrugającym przecinkiem przy drugiej cyfrze. Po osiągnięciu wartości maksymalnej liczymy od poczatku. Uwaga dla ułatwienia można liczy w systemie hexalnym (wartość maksymalna wynosi wówczas FFFF).

Ocena:

- 3.5 - program działający bez użycia przerwań,
- 4 - program działający bez użycia przerwań poprawnie napisany i skomentowany,
- 4.5 - program działający z użyciem przerwań,
- 5 - program działający z użyciem przerwań poprawnie napisany i skomentowany.

Za „smużenie” obcinane będzie pół oceny.
Prowadzący zastrzega sobie możliwość innego oceniania w przypadkach szczególnych.

UWAGA!

Pod pojęciem „poprawne napisanie” rozumiane jest:
-przejrzyście napisany kod, opatrzony komentarzami (sensownymi),
-zużycie jak najmniejszej ilości zasobów,
-wybranie optymalnego algorytmu,
-wynik działania kodu bez zastrzeżeń.