C dla mikrokontrolerów 8051

część 3

W trzeciej części kursu autor pokazuje jak w wygodny i przejrzysty sposób budować programy składające się z wielu fragmentów, które spełniają rolę biblioteki najczęściej stosowanych funkcji.

Budujemy program z klocków, czyli co to jest „project file"

W poprzedniej części artykułu wspominałem o zbiorach nagłówkowych oraz tak zwanych *project file*. Myślę, że zarówno jedne, jak i drugie będą dla nas bardzo użyteczne i wymagają kilku słów wyjaśnienia. Co to jest zatem *project file*?

Pod tą tajemniczą nazwą kryje się po prostu lista zbiorów składających się na nasz program. Wracając do przykładu z poprzedniego odcinka - jeśli na podstawie programu do obsługi wyświetlacza LCD utworzymy bibliotekę, to musimy ją w jakiś sposób dołączyć do programu głównego. Ten zbiór zawierający listę współpracujących ze sobą modułów, poddawanych działaniu kompilatora i linkera, stanowi tak zwany projekt (z angielskiego *project*). Czyli *project file* to po prostu lista zbiorów uwzględnianych przez kompilator i linker przy translacji kodu źródłowego na kod wynikowy. Na tej liście mogą się znaleźć nie tylko programy w języku C, ale również w języku asemblera. Dla nas jest to bardzo użyteczna informacja. Jest to sygnał, że nasz program nie musi być przygotowany wyłącznie w języku C. Po utworzeniu nowego projektu warto jest zajrzeć do zakładki menu *Project Options*. Można tam znaleźć szereg opcji kompilatorów C i asemblera, które mogą wymagać ustawienia. Na przykład, jeżeli stosujemy mikrokontroler AT89S8252, to można włączyć opcję *Dual DPTR*. Pomaga ona przy pewnych operacjach 16-bitowych. Podobnie można postąpić z definicją rejestrów 8051 dla modułów języka asembler tak, że nie trzeba będzie ich dołączać.

Budujemy pierwszą bibliotekę LCD4B.H

Stwórzmy na podstawie programu do obsługi wyświetlacza naszą pierwszą bibliotekę funkcji dla języka C. Po pierwsze, musimy zdefiniować zbiór nagłówka, który będzie zawierał wszystkie istotne dla pracy modułu parametry tak, aby nie trzeba było ich szukać w kodzie źródłowym biblioteki. Ten zbiór będzie też swego rodzaju łącznikiem pomiędzy programem głównym a biblioteką. Nasz zbiór *LCD4B.H* może mieć postać, jak poniżej:

```
#include <reg51.h>
// port, do którego podłączono
// wyświetlacz LCD
#define PORT    P2
// bity sterujące LCD
#define ENABLE  PORT^0
#define READ    PORT^3
#define REGISTER  PORT^2
// opóźnienie wielokrotności około
// 1 milisekundy dla kwarcu
// 7,3728MHz
void Delay (unsigned int k);
// zapis bajtu do LCD
void WriteByteToLcd(char X);
// zapis bajtu do rejestru
// kontrolnego LCD
void WriteToLcdCtrlRegister(char X);
// zapis bajtu do pamięci obrazu
void LcdWrite(char X);
// czyszczenie ekranu LCD
void LcdClrScr(void);
// inicjalizacja wyświetlacza LCD
// w trybie 4 bity
void LcdInitialize(void);
// ustawia kursor na współrzędnych
// x, y
void GotoXY(char x, char y);
// wyświetla tekst na współrzędnych
// x, y
void WriteTextXY(char x, char y,
char *S);
// wyświetla tekst od rozpoczynając
// od pozycji kursora
void WriteText(char *S);
// definiowanie znaków z tablicy
// wskazywanej przez ptr
void DefineSpecialCharacters(char
*ptr);
```

Jak łatwo zauważyć, jest to dokładne powtórzenie definicji nagłówków funkcji oraz stałych i zmiennych, które mają być dostępne również w innych modułach tego samego programu. Plik nagłówkowy najłatwiej jest w tym przypadku utworzyć, otwierając zbiór *LCD4B.C* i kasując wszystkie ciała funkcji, zostawiając tylko ich nagłówki. Pozostawić lub dopisać możemy również różne zmienne i stałe, których będziemy używać. Zbiór po edycji zapisujemy pod nazwą *LCD4B.H*. Bardziej właściwym wydaje się jednak pozostawienie tylko tych funkcji i procedur, które będą nam potrzebne.

Teraz kolej na właściwą implementację *LCD4B.C*. Na początku modułu umieszczamy dyrektywę *#include LCD4B.H*. W ten sposób wszystkie definicje z pliku nagłówkowego dostępne będą również w pliku źródłowym. W kolejnym kroku usuwamy segment *main()*. W zasadzie biblioteka jest już gotowa. Należy jeszcze spróbować skompilować *LCD4B.C*, aby sprawdzić, czy któreś z definicji nie powtarzają się i czy nie ma w nich błędów.

Łączymy bibliotekę z programem głównym

Program główny to program w języku C. Stanowi on jeden ze składników projektu. Podobnie, składnikiem projektu musi być postać źródłowa biblioteki funkcji wyświetlacza oraz plik nagłówka. Posłużmy się przykładem z poprzedniego odcinka i napiszmy ten sam program, ale korzystając z utworzonej biblioteki.

Po uruchomieniu RIDE, w pasku menu na górze ekranu odnajdziemy *Project*. Wybierzmy tę opcję, a następnie *New*. Utwórzmy nowy zbiór projektu i nazwijmy go *Pierwszy*. Program sam nada mu domyślne rozszerzenie.*PRJ*, a na dole ekranu zostanie otwarte okienko projektu. Następnie wybierzmy *File New* i utwórzmy zbiór o rozszerzeniu.*C* - to będzie nasz program główny. Jego treść powinna być następująca:

```
#include "lcd4b.h"
// program główny
void main(void)
{
 char ix = 1, iy = 1, x, y, i = 0;

 LcdInitialize();
 DefineSpecialCharacters(&CGRom);
 LcdClrScr();
 while (1)
```

```
 {
  WriteTextXY(x, y, " ");
  if (ix == 1) x++; else x-;
  if (iy == 1) y++; else y-;
  if (x == 19) ix = 0;
  if (x == 0) ix = 1;
  if (y == 3) iy = 0;
  if (y == 0) iy = 1;
  WriteTextXY(x, y, 0x01);
  Delay(50);
 }
}
```

Uwaga: wybranie *File Open* lub *File New* nie powoduje dodania otwartego lub nowego zbioru do projektu.

Po wpisaniu tych kilku linijek instrukcji, zapisujemy program główny poprzez *File Save As* pod nazwą *TEST.C*. Zapamiętaliśmy program główny, teraz musi się on stać częścią projektu. W tym celu naciskamy klawisze skrótu ALT + INSERT i w otwartym okienku wskazujemy zbiór *TEST.C*. Zbiór powinien się pojawić w okienku *Project* na dole ekranu. Będzie on widoczny po wskazaniu symbolu „+". W ten sam sposób musimy dołączyć zbiór *LCD4B.C*, ponieważ musi on być kompilowany równocześnie z programem głównym. Zbioru *LCD4B.H* nie trzeba dołączać. Zostanie on dołączony automatycznie w czasie kompilacji, ponieważ jest wymieniony jak parametr dyrektywy *#include*. Po naciśnięciu klawisza F9, co odpowiada wywołaniu *Make All*, nasz projekt powinien skompilować się bezbłędnie, a rezultat w postaci zbioru wynikowego „Pierwszy.hex" powinien zostać zapisany na dysku.

Łączymy moduł języka C z asemblerem

Podobnir jak łączyliśmy moduły napisane w języku C, możemy dołączyć do programu głównego w C moduły napisane w języku asembler. Kompilator RC-51 oferuje nam 3 sposoby dołączania funkcji napisanych w języku asembler:

1. Za pomocą instrukcji **ASM**.

Instrukcja *asm* pozwala na umieszczenie kodu języka asembler w postaci liczb szesnastkowych w pliku źródłowym programu. Znajdzie się on pod adresem wynikającym z bieżącego stanu kodu. Spodziewane są wartości jednobajtowe obiektów, za wyjątkiem adresów zmiennych zewnętrznych (*code* lub *xdata*), które wymagają 2 bajtów. Nie jest to metoda zbyt wygodna. Sami musimy bowiem spełnić rolę asemblera tłumacząc mnemoniki na kody szesnastkowe.

Tabela 1. Lokalizacja wartości zwracanych przez funkcje

Rozmiar (typ parametru)	Lokalizacja
bit (1 bit)	flaga Carry
char (1 byte)	R7
int (2 bajty)	R6:R7
generic pointer (3 bajty)	R1:R2:R3
float (4 bajty)	R4:R5:R6:R7
double (6 bajtów)	R2:R3:R4:R5:R6:R7
long double (7 bajtów)	R1:R2:R3:R4:R5:R6:R7

Tabela 2. Lokalizacja parametrów przekazywanych do funkcji

Numer argumentu	char lub wskaźnik 1-bajtowy	int lub wskaźnik 2-bajtowy	long lub float	wskaźnik typu generic
1	R7	R6:R7	R4:R5:R6:R7	R1:R2:R3
2	R5	R4:R5	R4:R5:R6:R7	R1:R2:R3
3	R3	R2:R3		R1:R2:R3

2. Za pomocą dyrektywy **#pragma ASM**.

Dyrektywa ta pozwala na umieszczenie kodu w postaci mnemoników asemblera w pliku źródłowym programu. Blok ten powinien się zaczynać od *#pragma ASM* i kończyć *#pragma ENDASM*. Metoda ta, podobnie jak powyższa, polecana jest jednak do niewielkich procedur.

3. Za pomocą modułu w języku asmeblera.

Moim zdaniem jest to najlepsza i najbardziej efektywna metoda zarówno do małych, jak i dużych procedur w języku asembler. Pozbawiona jest wad poprzedników - wymaga jednak opanowania kilku podstawowych reguł przekazywania parametrów. Metodę tę omówimy najszerzej. Będziemy też stosować ją we wszystkich przykładach łączenia modułów asemblera z modułami w języku C.

Jak przekazywane są parametry funkcji?

Parametry funkcji możemy podzielić na dwie grupy. Pierwszą z nich będą stanowić parametry zwracane jako rezultat działania funkcji, drugą pobierane przez nią z zewnątrz (argumenty). Oba rodzaje parametrów przekazywane są przez bank rejestrów. Trzeba być więc ostrożnym, wykorzystując go do innych celów.

W przypadku wartości zwracanych przez funkcję sytuacja jest bardzo prosta, ponieważ funkcja może zwracać tylko jedną wartość. Wystarczy więc znajomość **tab. 1**. Chyba że wartość zwracana nie mieści się w banku rejestrów. Wówczas jest ona umieszczana na stosie mikrokontrolera. Inaczej jest w przypadku, gdy zachodzi konieczność przekazania argumentów do funkcji.

Przez bank rejestrów może być przekazane do trzech argumentów. Jeśli jest ich więcej, wówczas używany jest stos. Metoda przekazywania argumentów przez bank rejestrów jest domyślną dla kompilatora. Można ją jednak zmienić, stosując polecenie *#pragma NOREGPARMS*. Wówczas argumenty funkcji przekazywane są przez stos lub zapamiętywane w segmencie zmiennych pseudostatycznych. Położenie tego segmentu zależy od wybranego modelu pamięci (jeśli wybrany został model SMALL, LARGE lub COMPACT, wówczas znajduje się on odpowiednio w obszarze DATA, XDATA lub PDATA). Nie polecam jednak używania tego polecenia, ponieważ wówczas lokalizacja argumentów może być dosyć trudna.

Nazwy funkcji języka C a asembler

W celu ułatwienia analizy programu w języku asembler oraz uniknięcia błędów tak zwanych *runtime*, nazwy funkcji podczas kompilowania programu są nieco modyfikowane.

- **void func1(void)** wygeneruje symbol *FUNC1*. Nazwa funkcji bez parametrów lub z parametrami nieprzekazywanymi przez rejestry jest przenoszona do zbioru.*OBJ* bez zmian. Małe litery w nazwie funkcji zamieniane są na duże.
- **void func2(char)** wygeneruje symbol *_FUNC2*. Dla funkcji z argumentami przekazywanymi w rejestrach, na początku nazwy dodawany jest znak podkreślenia. Znak ten jest identyfikatorem funkcji pobierających argumenty z rejestrów. Małe litery w nazwie funkcji zamieniane są na duże.
- **void func3 (void) reentrant** wygeneruje symbol *?FUNC3*. Funkcje typu *reentrant* (to znaczy takie, które mogą być wywoływane jednocześnie przez wiele innych funkcji) mają nazwę poprzedzoną znakiem zapytania. Określa on funkcję typu *reentrant*.

Na dysku powstaje co najmniej trzy rodzaje zbiorów. Pierwszy z nich to przetłumaczony na język asemblera program źródłowy (.LST), drugi to mapa zmiennych i stałych (adresy ich rozmieszczenia w pamięci 'M51), trzeci to zbiór wynikowy.HEX lub.BIN (albo oba te zbiory). Jeśli mamy wątpliwości co do sposobu, w jaki wykonany zostanie nasz program napisany w języku C, to wówczas analiza kodu wynikowego w języku asembler może je wyjaśnić.

Nareszcie coś dla praktyków

Spożytkujmy nowo zdobytą wiedzę. Przyda nam się zarówno ta o tworzeniu zbiorów nagłówkowych typu *.H*, jak i o łączeniu modułów i przekazywaniu argumentów do funkcji w języku C oraz pobieraniu rezultatów ich działania. Za przykład niech posłuży nam program do odczytywania standardowej klawiatury PC. Jeszcze drobna uwaga. Bez wdawania się w szczegóły funkcjonowania kompilatora języka asembler, moduł z tego przykładu można potraktować jako pewnego rodzaju szablon. W skrócie struktura takiego pliku jest następująca:

```
<nazwa segmentu> SEGMENT CODE
EXTRN DATA (<nazwa zmiennej>)
EXTRN CODE (<nazwa funkcji w C>)
PUBLIC <nazwa udostępnianej funkcji
w asemblerze>
RSEG <nazwa segmentu >
.....implementacja....
END
```

Kiedy warto stosować moduły napisane w asemblerze? Po pierwsze wtedy, gdy zależy nam na szybkości działania. Drugi powód może zaprzeczyć idei stosowania języków wysokiego poziomu, do których należy również C, jednak niektóre funkcje jest znacznie łatwiej zaimplementować w asemblerze niż w C. Tak jest na przykład z odczytywaniem klawiatury PC. Dane z niej wprowadzane są szeregowo, więc konieczne jest użycie instrukcji przesuwania zawartości rejestru roboczego w prawo lub w lewo. Najprościej jest to zrobić, zapamiętując wartość bitu portu, do którego podłączone jest wejście/wyjście danych klawiatury w fladze przeniesienia C (na przykład C = P1^0 lub w asemblerze MOV C,P1.0). Później wystarczy już tylko instrukcja przesunięcia w prawo lub w lewo danych w rejestrze A z uwzględnieniem flagi C - wykonana ośmiokrotnie - aby odczytać cały bajt danych. W języku C pojawia się pewien problem. Otóż zmienne bez znaku (unsigned) i ze znakiem (signed) są różnie przez kompilator przesuwane. Różnica polega na uwzględnianiu flagi przeniesienia w przypadku zmiennych typu signed. Co innego w asemblerze, mamy pełną kontrolę nad postacią źródłową programu i sposobem jego wykonywania, niezależnie od typu deklarowanych zmiennych.

Łącząc moduł napisany w języku C z modułem w asemblerze, staram się zachować pewien styl programowania. Oczywiście możesz sobie wybrać własny. Moja propozycja jest jednak następująca:

1. Program główny zawsze napisany jest w języku C i do niego dołączane są moduły w języku asembler.

2. Każdą zmienną (każdą komórkę pamięci używaną przez asembler) deklaruję w programie głównym w języku C. Czasami zmiennych można używać zamiennie albo w asemblerze, albo w C. Oszczędzamy w ten sposób pamięć mikrokontrolera, której i tak nie ma zbyt wiele.

Aby poinformować kompilator języka C, że dana funkcja jest zewnętrzną, czyli pochodzi z innego modułu, używa się słowa kluczowego *extern*. Oto przykładowe deklaracje funkcji zewnętrznych:

```
extern void TX_byte(char x);
extern char RX_byte(void);
```

Każda funkcja i zmienna w języku C, jeżeli nie jest poprzedzona słowem *static*, ma zasięg globalny, to znaczy może być wykorzystywana przez inne moduły. Upraszcza to korzystanie z nich z poziomu modułu przygotowanego w języku asembler. Wystarczy bowiem tylko poinformować kompilator, że dana funkcja czy zmienna jest zewnętrzna. Inaczej jest w przypadku modułów w asemblerze. Każda funkcja musi zostać zadeklarowana jako dostępna z zewnętrz. Służy do tego słowo *PUBLIC*. Do pobrania natomiast danych z modułów w języku C, słowo *EXTRN*. Oto przykłady deklaracji dostępu na poziomie modułu napisanego w asmeblerze:

- zmienna pobierana z modułu napisanego w języku C:
  ```
  EXTRN DATA (temp)
  ```
- deklaracja dostępu do funkcji zaimplementowanej w module w języku C, niezbędne jest przekazanie parametrów do funkcji, stąd też kompilator języka C dodaje znak podkreślenia przed jej nazwą zgodnie z wcześniej opisywanymi regułami:
  ```
  EXTRN CODE (_Delay)
  ```
- deklaracje funkcji udostępnianych przez moduł w języku asembler - zwróćmy uwagę na zmianę nazw w stosunku do tych, użytych w C (podobnie jak *_Delay*):
  ```
  PUBLIC _TX_byte
  PUBLIC RX_byte
  ```

Deklaracja obiektów zewnętrznych składa się z trzech członów: słowa kluczowego *EXTRN*, określenia miejsca w obszarze adresowym mikrokontrolera, gdzie umieszczony jest obiekt (*CODE*, *DATA* itp.) oraz nazwy obiektu w nawiasach. Deklaracja obiektów udostępnianych na zewnątrz zawiera tylko słowo kluczowe *PUBLIC*.

Na koniec bardzo ważna uwaga. Moduł napisany w języku C musi mieć inną nazwę niż napisany w języku asembler. Dlaczego? Podczas kompilowania tworzone są różne zbiory tymczasowe. Mają one taką samą nazwę jak zbiór źródłowy, lecz inne rozszerzenie (na przykład.*AOF*). Gdy oba moduły, w C i asemblerze, będą miały tę samą nazwę, to kompilator najpierw przetworzy zbiór PCKBD.C i utworzy z niego zbiór *PCKBD.AOF*, a następnie to samo zrobi ze zbiorem *PCKBD.A51*. Nietrudno zauważyć, że ostatni kompilowany zbiór nałoży się na już istniejący. Linker próbując zbudować z obiektów zbiór wynikowy, nie znajdzie potrzebnych danych i wyświetli komunikat o brakujących funkcjach czy zmiennych, mimo iż teoretycznie wszystko jest w porządku.

Program do odczytu klawiatury PC

Program składa się z dwóch części. Główny napisany jest w języku C, a funkcje do komunikacji z klawiaturą napisane są w asemblerze. Są to dwa odrębne moduły. Pierwszy z nich nazywa się *PCKBD-C.C*, a drugi *PCKBD-ASM.A51*. Oba pliki składają się na projekt o nazwie *PCKBD* (*.PRJ*). Do projektu dołączyłem również wcześniej utworzoną przez nas bibliotekę zawierającą funkcje obsługi wyświetlacza LCD. Myślę, że przyda nam się jeszcze niejednokrotnie.

Oczywiście program główny jest tylko przykładowym. Wyświetla kody naciśniętych klawiszy, podczas gdy w praktyce można je wykorzystać w zupełnie inny sposób do wprowadzania danych czy sterowania własnym urządzeniem. Funkcja *main()* zawiera dosyć rozbudowany warunek *switch*. Jest to połączenie tej instrukcji z konstrukcją *if....else*.

Zwróćmy uwagę na wywołanie funkcji *_Delay* w module w języku asembler. Jako argument funkcji wymagana jest liczba typu *unsigned int*. Jest ona wpisywana do rejestrów R6:R7 tuż przed poleceniem *ACALL _Delay*. Funkcjami wymieniającymi dane z modułem w języku C są również *RX_byte* i *TX_byte*. Ponieważ zarówno argument dla *TX_byte*, jak i liczba zwracana przez *RX_byte* są typu *char*, przekazywane są one przez rejestr R7 (patrz tab. 1 i 2).

Układ elektryczny jest bardzo prosty. Wystarczy w zasadzie dowolny płytka testowa z mikroprocesorem z rodziny 8051. Klawiatura wymaga rezystorów *pull-up* o wartości około 5,1kΩ. W układzie modelowym używałem AVR Starter Kit firmy Atmel Corp. z mikrokontrolerem AT89S8252. Program po skompilowaniu zajmuje nieco więcej niż 1,5kB. Z powodzeniem można go więc zmieścić również w malutkim AT89C2051.

Jacek Bogusz, AVT
jacek.bogusz@ep.com.pl

Dodatkowe informacje

Ewaluacyjną wersję pakietu firmy Raisonance oraz postać źródłową programu prezentowanego w artykule zamieściliśmy na CD-EP8/2002B.