C dla mikrokontrolerów 8051

część 9

Większość kompilatorów języka C oferuje szereg różnych modeli pamięci. Który z nich wybrać? Jakie kryteria wyboru stosować? Jaki wpływ ma zastosowany model kompilacji na optymalizację kodu wynikowego oraz czas wykonywania programu przez mikrokontroler?

Konfiguracje pamięci mikrokontrolera 8051 dla programów napisanych w języku C, część 1

Podstawy: fizyczne lokalizacje segmentów pamięci

Prawdopodobnie najbardziej irytującym podczas pisania programów dla mikrokontrolerów 8051 i pochodnych jest fakt, że posiadają one kilka różnych obszarów pamięci zaczynających się od tego samego adresu. Inne mikrokontrolery, dla przykładu z rodziny 68HC11, mają ciągłą przestrzeń adresową, w której poszczególne obszary pamięci umieszczone są kolejno jeden za drugim, w zależności od tego, czy istnieją w danym mikrokontrolerze, czy też nie. Jest to zgodne z konfiguracją pojedynczego obszaru pamięci według założeń von Neumana.

W mikrokontrolerze 8051 pamięć podzielono na kilka części dostępnych na różne sposoby. Aby dobrze zrozumieć zasady użycia poszczególnych rodzajów pamięci i modeli kompilacji, musimy przyjrzeć się bliżej temu podziałowi i miejscom fizycznej lokalizacji komórek pamięci.

Pierwszy z segmentów nazywa się DATA. Mowa tu o wewnętrznej pamięci RAM mikrokontrolera zwanej segmentem danych. Zaczyna się on od adresu 0x00 i kończy pod adresem 0x7F (127 dziesiętnie). Główne przeznaczenie tego segmentu pamięci to przechowywanie zmiennych wykorzystywanych przez program w czasie pracy. Obszar ten dostępny jest jako adresowany bezpośrednio. Znajdą tu zastosowanie instrukcje asemblera z grupy MOV A,n - MOV n,A.

Od adresu 0x80 umieszczony jest tak zwany rejestr funkcji specjalnych SFR (*Special Function Register*), który również może być adresowany bezpośrednio. Powyżej adresu 0x80, do adresu 0xFF, rozciąga się część drugiego obszaru pamięci RAM, zwanego IDATA, który może być adresowany wyłączenie pośrednio. Znajdują tu zastosowanie instrukcje z grupy MOV A,@Ri (i=0 lub i=1).

Standardowe wersje mikrokontrolerów 8051 są wyposażone w 128 bajtów pamięci DATA. Dodatkowy segment IDATA pojawił się w momencie wprowadzenia do sprzedaży mikrokontrolerów z rodziny 8052. Segment ten nie jest częścią pamięci DATA. Rozciąga się od adresu 0x00 do 0xFF i może być adresowany wyłącznie pośrednio. Z związku z tym dobrze jest stosować go nie w celu zapamiętywania zmiennych, ale z przeznaczeniem na stos mikrokontrolera. Jednostka centralna będzie wtedy używała tego segmentu, adresując go za pośrednictwem wskaźnika stosu. Oczywiście pisząc programy w języku C, trudno jest wywrzeć wpływ na to, do czego kompilator będzie wykorzystywał obszar IDATA. Jest to możliwe tylko poprzez zmianę parametrów linkera, jednak wymaga to gruntownej wiedzy na temat zasad funkcjonowania kompilatora oraz daleko posuniętej ostrożności przy wprowadzaniu zmian. Opisaną tutaj zasadę należy stosować podczas pisania programów w asemblerze, natomiast w przypadku C zalecam zdać się na autorów kompilatora, chyba że jakieś specjalne powody zmuszają do innego podejścia.

Obszar IDATA nakłada się na obszar rejestru SFR. Jawnie widać tu pewną sprzeczność: IDATA dzielący wspólnie przestrzeń adresową z SFR może być adresowany tylko pośrednio, natomiast do SFR mają dostęp instrukcje adresowania bezpośredniego! Aby było ciekawiej, obydwa te segmenty znajdują się w obszarze wewnętrznej pamięci RAM mikrokontrolera. Czyli patrząc na to od strony programisty 8051 - to do jakiego obszaru pamięci zapisywane czy z jakiego odczytywane są dane, zależy od sposobu w jaki zostanie zaadresowany ten sam obszar pamięci. Ten sam w znaczeniu fizycznie wpisywanego adresu, nie zaś segmentu pamięci mikrokontrolera. Pomoże to zrozumieć poniższy fragment programu asemblerowego:

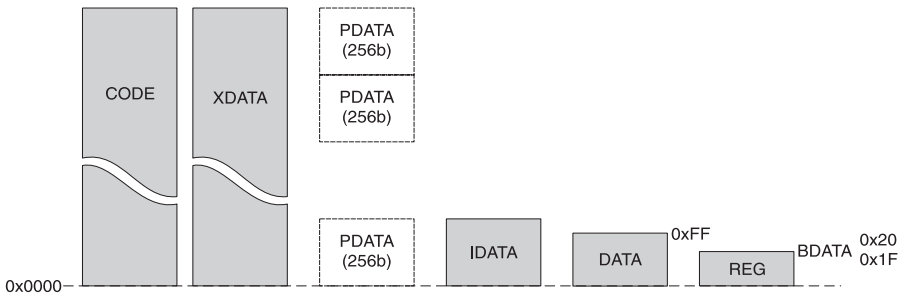
```
MOV 0A0H,#dana;zapis do obszaru
    ;SFR, w tym przypadku do P2
    ;(równoznaczny zapis to
    ;MOV  P2,#dana)
MOV R0,#0A0H
MOV @R0,#dana ;zapis do obszaru
    ;IDATA (nie do P2!)
```

Pierwsza linia przykładowego programu zapisuje dane do obszaru SFR, w tym przypadku jest to port P2. Druga i trzecia linia, mimo że powodują zapis bajtu pod ten sam adres, to jednak używane jest adresowanie pośrednie (adresowanie za pomocą rejestru R0) i bajt zostaje zapisany w segmencie IDATA, a nie jak poprzednio w SFR. Warto o tym pamiętać, tworząc własne aplikacje.

Trzeci segment pamięci, nazywany CODE, również rozpoczyna się od adresu 0x0000, ale zarezerwowany jest na pamięć programu. Typowo obszar ten zajmuje adresy od 0x0000 do 0xFFFF (65536 bajtów) i jeśli używany jest mikrokontroler 8031, to segment ten w całości podłączany jest z zewnątrz w postaci dodatkowego układu pamięci ROM. Niektóre mikrokontrolery, np. 8051, posiadają jednak wewnętrzną pamięć ROM, pełniącą tę samą rolę. Segment pamięci programu dostępny jest przez instrukcje wykorzystujące do adresowania licznik rozkazów PC oraz 16-bitowy rejestr DPTR. Oczywiście w segmencie CODE mogą być przechowywane wyłącznie wartości stałe, takie jak tablice danych, a przede wszystkim instrukcje programu wykonywanego przez mikrokontroler.

Czwarty segment pamięci, nazywany XDATA (czasami można się spotkać z określeniem XRAM), również znajduje się poza mikrokontrolerem. Zaczyna się od adresu 0x0000 i tak, jak segment CODE, kończy się pod adresem 0xFFFF. Jedna uwaga: omawiając przestrzeń adresową pamięci CODE czy XDATA, opisuję możliwość jej fizycznego rozszerzenia, a nie przymus zajmowania całego dostępnego obszaru przez dane urządzenie (układ) podłączone w tej przestrzeni adresowej.

W zestawie rozkazów 8051 można znaleźć tylko jedną metodę dostępu do



Rys. 1

całego segmentu XDATA za pomocą pojedynczego rozkazu. Służą do tego instrukcje wykorzystujące do adresowania rejestr DPTR. Niemniej jednak cały obszar XDATA (od adresu 0x0000 do 0xFFFF) może być też dostępny w trybie stronicowania. Do adresowania pamięci w obszarze 256-bajtowej strony jest tu używany ośmiobitowy rejestr R0 lub R1. W tym przypadku starszą część adresu (numer strony) trzeba ustawić „ręcznie" np. poprzez wpisanie odpowiedniej wartości do P2. Trzeba przy tym pamiętać, że P2 nie bierze aktywnego udziału podczas takiego adresowania (nie pełni funkcji szyny systemowej). Równie dobrze może być wykorzystany inny rejestr lub nawet tylko jego część. Obszar ten nazywany jest PDATA (**rys. 1**).

Pojawia się pytanie: w jaki sposób jednostka centralna mikrokontrolera 8051 rozróżnia fizycznie inne i o innym przeznaczeniu obszary pamięci? W jaki sposób kod instrukcji programu pobierany jest spod adresu CODE:0x0000, zamiast DATA:0x00? Odpowiedź tkwi w budowie mikrokontrolera. Gdy CPU żąda dostępu do segmentu DATA, załączany jest wewnętrzny RAM przez wewnętrzny sygnał odczytu READ - odpowiednik tego sygnału wyprowadzany na zewnątrz (wyprowadzenie READ mikrokontrolera) nie zmienia się.

Odczyt i zapis bajtu z wykorzystaniem akumulatora w trybie adresowania bezpośredniego można uzyskać za pomocą poleceń:

```
MOV A,40H
;odczyt bajtu spod adresu 0x40
MOV 40H,A
;zapis bajtu pod adres 0x40
```

Jest to podstawowy tryb dla modelu pamięci SMALL.

Odczyt bajtu z wykorzystaniem trybu adresowania pośredniego z segmentu IDATA za pośrednictwem akumulatora i rejestru R0 wygląda następująco:

```
MOV R0,#0A0H
;odczyt bajtu z segmentu IDATA
;znajdującego się pod adresem
MOV A,@R0
;0xA0 do akumulatora
```

Ten tryb adresowania używany jest do dostępu do pośrednio adresowanych komórek pamięci IDATA leżących powyżej 0x80 i jest alternatywną metodą dostępu do danych leżących poniżej tego adresu.

W obrębie segmentu DATA znajduje się również obszar nazywany BDATA. Jest to szesnaście bajtów (128 bitów) zajmujących przestrzeń adresową od 0x20 do 0x2F w obszarze adresowania bezpośredniego. Specjalną cechą tego obszaru jest to, że oprócz instrukcji MOV mają zastosowanie również instrukcje operujące na pojedynczych bitach i wykorzystujące specjalny tryb adresowania pojedynczych bitów.

Zewnętrzna pamięć ROM (segment CODE) nie jest załączana podczas dostępu do RAM czy XDATA (XRAM). Jej wyborem steruje sygnał PSEN (*Program Store Enable*) - zmiana poziomu wyprowadzenia PSEN na niski uaktywnia pamięć programu. Nazwa wyprowadzenia sygnału jest jednocześnie sugestią, że główną rolą pamięci ROM jest przechowywanie instrukcji programu.

Pewną ciekawostkę stanowi fakt, że jeśli mikrokontroler posiada wewnętrzną pamięć ROM (FLASH, EPROM), to w cyklu dostępu do tej pamięci stan zewnętrznego, wyprowadzenia PSEN nie zmienia się dotąd, aż przekroczony zostanie obszar adresowania wewnętrznego ROM. Po tym fakcie mikrokontroler wykonuje normalny cykl dostępu do zewnętrznej pamięci programu wyprowadzając przez porty P0 i P2 adresy oraz pobierając instrukcje do wykonania z zewnętrznej pamięci ROM. W takiej sytuacji, jeśli zewnętrzna pamięć ROM jest ulokowana od adresu 0x0000, to bajty leżące poniżej końca adresu wewnętrznej pamięci ROM (na przykład dla 4 kB będzie to adres 0x1000) nie będą dostępne.

Podłączone do mikrokontrolera z zewnątrz segmenty XDATA oraz CODE nie są ze sobą w konflikcie. Ich rozdział jest przeprowadzany przez zewnętrzne sygnały sterujące. Jak wcześniej wspomniałem, dostęp do obszaru CODE aktywowany jest za pomocą PSEN. Dostępem do obszaru XDATA sterują zewnętrzne sygnały READ (odczyt) i WRITE (zapis). Wyprowadzenie PSEN nie bierze udziału w cyklu dostępu do danych zawartych w segmencie XDATA.

Aby odróżnić polecenia dostępu do danych zawartych w segmencie XDATA od pozostałych, wprowadzono specjalne instrukcje zawierające w swej nazwie literę X:

```
MOV DPTR,#08000H
;zapamiętaj daną zawartą
;w akumulatorze pod adresem
MOVX A,@DPTR
;0x8000 w pamięci XDATA
```

Ten alternatywny tryb dostępu do pamięci XDATA jest podstawowym dla modelu COMPACT. Zauważ, że jeśli Port 2 podłączony jest do starszych linii adresowych, może on pracować jako przełączany przez aplikację kontroler stron pamięci.

Istotną do zapamiętania informacją jest to, że wyprowadzenie PSEN przyjmuje stan niski, jeśli pobierany jest kod instrukcji, natomiast wyprowadzenia READ i WRITE podczas wykonywania przez CPU rozkazu MOVX. Litera X w symbolu polecenia języka asembler mikrokontrolera 8051 oznacza rozkaz związany z urządzeniem podłączonym z zewnątrz, aktywowanym za pomocą READ lub WRITE.

Dostępne modele pamięci

Pisząc program dla mikrokontrolera 8051, pierwsza decyzja, którą musisz podjąć to taka, jaki model pamięci wybierzesz. Podczas gdy programista komputera PC dokonuje wyboru pomiędzy modelami TINY, SMALL, MEDIUM, COMPACT, LARGE i HUGE, aby kontrolować jak używane są segmenty pamięci RAM komputera PC, programista 8051 planujący swoją aplikację musi podjąć decyzję na podstawie tego, gdzie znajdują się dane niezbędne podczas pracy mikrokontrolera.

Popularnie używane kompilatory Keil i Raisonance aktualnie obsługują następujące konfiguracje pamięci:

1. ROM - największy rozmiar zbioru obiektu, który może powstać po kompilacji, to 64kB, jakkolwiek znacznie większe rozmiary pamięci ROM (do 1 MB - kompilator Keil, do 4 MB - kompilator Raisonance) mogą być obsługiwane w trybie przełączanych banków pamięci (BANKED) opisanym w dalszej części artykułu. Zmienne mogą być deklarowane przy użyciu słowa kluczowego *code* umieszczającego je w pamięci progra-

mu mikrokontrolera. Nazwa zmienne jest tu myląca, ponieważ zadeklarowana, na przykład tablica może pełnić rolę wzorca - nigdy zaś zmiennej.

2. RAM - dostępne są trzy modele pamięci: SMALL, COMPACT i LARGE:
- SMALL - wszystkie zmienne zostają umieszczone w wewnętrznej pamięci mikrokontrolera
- COMPACT - zmienne zostają zapamiętane w segmencie pamięci PDATA, adresowanej przez port P0 (z bankami przełączanymi przez P2). Używany jest tryb adresowania pośredniego. Wewnętrzne rejestry mikrokontrolera są w dalszym ciągu używane do przechowywania lokalnych zmiennych i parametrów.
- LARGE - zmienne i parametry przechowywane są w zewnętrznej pamięci adresowanej za pośrednictwem @DPTR. Wewnętrzne rejestry mikrokontrolera w dalszym ciągu używane są do przechowywania zmiennych i parametrów.
- BANKED (przełączane banki pamięci) - program może zajmować do 1 MB - kompilator Keil lub 4 MB - kompilator Raisonance. Pamięć przełączana jest w formie „stron" o rozmiarze 64kB każda za pomocą innych niż właściwe dla P0 i P1 wyprowadzeń mikrokontrolera albo też za pomocą zatrzasków *latch* umieszczonych w przestrzeni adresowej, powyżej adresu 0xFFFF. Każdy 64 kB blok pamięci programu musi posiadać ustawiony tak zwany blok wspólny (COMMON AREA) dla biblioteki funkcji przełączającej banki pamięci.

Firma Raisonance wprowadziła do swojego kompilatora dodatkowy model pamięci TINY, który jest identyczny z modelem SMALL z tą różnicą, że podczas kompilowania programu generowane są instrukcje ACALL i AJMP, zamiast LCALL i LJMP. Limituje to rozmiar obszaru pamięci programu do 2 kB i jest użyteczne szczególnie dla mikrokontrolerów, które nie obsługują lub nie potrzebują instrukcji LCALL i LJMP, czyli mających do 2 kB pamięci ROM w swojej strukturze (AT89C2051, 87C751 itp.) oraz nieposiadających na zewnątrz wyprowadzeń PSEN, READ i WRITE. W Keilu można korzystać z dyrektywy ROM, np. #PRAGMA ROM(SMALL) spowoduje używanie wyłącznie rozkazów ACALL i AJMP.

Możliwe jest również łączenie poszczególnych modeli pamięci tak, aby zmusić kompilator do lokowania zmiennych i danych w określonych segmentach pamięci, pod określonym adresem.

Tab. 1.

Nazwa segmentu pamięci	Zalecany do...	Nie zalecany do...
DATA Rozmiar 128 bajtów, domyślny dla modelu SMALL	• Często używanych danych wymagających szybkiego dostępu. • Procedur obsługi przerwań, które powinny być wykonywane bardzo szybko, powinny używać obszaru DATA, poprzez lokalną deklarację funkcji jako stosującej model SMALL. • Często wywoływanych podprogramów pobierających czy przekazujących dużą liczbę parametrów. • Stosu funkcji typu *re-entrant*.	• Zmiennych tablicowych i struktur zawierających więcej niż kilka - kilkanaście bajtów.
IDATA 128 lub 256 bajtów, nie jest przypisany do żadnego z modeli kompilacji	• Zmiennych tablicowych i struktur o ograniczonym do około 32 bajtów rozmiarze. **Uwaga:** suma rozmiarów struktur i zmiennych tablicowych nie powinna przekraczać 64 bajtów. • Stosu mikrokontrolera (lokowany jest on w obszarze IDATA i adresowany pośrednio przy pomocy SP).	• Dużych tablic i struktur oraz zmiennych z wymaganym krótkim czasem dostępu.
CODE 64kB	• Stałe programu. • Duże tablice konwersji • PLUS oczywiście instrukcje programu!	• Zmiennych.... To jest ROM, więc nie mogą być tu zapisywane żadne zmienne.
PDATA 256 bajtów, obszar domyślny dla modelu COMPACT	• Funkcji obsługi przerwań o niezbyt krytycznym czasie wykonywania. • Zmiennych typu *char*, niedużych tablic i struktur o wymaganym krótkim czasie dostępu. • Doskonała do zmiennych, które muszą być monitorowane w czasie rzeczywistym podczas uruchamiania programu przy pomocy ICE.	• Dużych tablic i struktur o rozmiarze przekraczającym 256 bajtów • Bardzo często używanych danych oraz zmiennych wykorzystywanych przez procedury obsługi przerwań • Zapamiętywania zmiennych typu single, float, double itp.
XDATA do 64kB, domyślny dla modelu LARGE	• Dużych tablic i struktur o rozmiarze powyżej 256 bajtów. • Zmiennych o niezbyt krytycznym czasie dostępu. • Zmiennych rzadko używanych. • Doskonała do zmiennych, które muszą być monitorowane w czasie rzeczywistym podczas uruchamiania programu przy pomocy ICE.	• Bardzo często używanych danych. • Zmiennych wykorzystywanych przez procedury obsługi przerwań. • Zapamiętywania zmiennych typu single, float, double itp.

Wybór najlepszego modelu pamięci

Model TINY nie nastręcza żadnych trudności przy wyborze. Stosuje się go raczej do bardzo małych programów. Dla większości aplikacji wykonywanych dla mikrokontrolera 8051 wystarczający jest model SMALL. Stosując go można również używać zewnętrznej pamięci znajdującej się w segmencie PDATA. Dostęp do niej uzyskuje się za pomocą instrukcji MOVX A,@Ri i MOVX @Ri,A.

SMALL - pamięć RAM, 128 bajtów

Używając modelu SMALL, należy zredukować do minimum liczbę zmiennych globalnych używanych w programie. Pozwoli to linkerowi na nakładkowanie funkcji w taki sposób, aby aplikacja pracowała efektywnie. W przypadku mikrokontrolerów z serii 8052/8032 deklaracje zmiennych w przestrzeni IDATA powyżej adresu 0x80 mogą pozwolić aplikacji na uzyskanie dodatkowej przestrzeni do przechowywania zmiennych. Należy jednak pamiętać, że obszar ten używany jest również na stos mikrokontrolera.

Model SMALL można także stosować przy kompilacji nawet bardzo dużych programów, umieszczając obiekty duże i takie, do których nie jest wymagany bardzo szybki dostęp, w zewnętrznej pamięci RAM. Dobrze jest tam również ulokować zmienne, które muszą być dostępne w czasie rzeczywistym podczas uruchamiania urządzenia z mikrokontrolerem przy pomocy emulatora, ponieważ emulatory (takie jak produkowane przez Hitex albo Raisonance) mają bezpośredni dostęp do tego segmentu pa-

List. 1.
```
#pragma COMPACT
/* model SMALL */
void SmallFunc() small
{
  printf("%s\n","Hello!");
}
/* model LARGE */
void LargeFunc() large
{
  printf("%s\n","Hello!");
}

/* program główny */
void main()
{
  SmallFUnc();
  LargeFunc();
}
```

mięci. Ten model najlepszy jest również dla aplikacji o krytycznym czasie wykonywania, jako że gwarantuje on najszybszy dostęp do zmiennych i parametrów przez funkcje, podczas gdy duże obszary danych mogą zostać umieszczone poza układem mikrokontrolera.

COMPACT - pamięć RAM 256 bajtów poza układem, 128 lub 256 bajtów w układzie

COMPACT to model pamięci dostosowany do programów, gdzie dla przykładu wewnętrzny RAM mikrokontrolera przeznaczony jest na zmienne systemu operacyjnego. Model ten jest rzadko używany dla całego programu. Najbardziej użyteczna kombinacja to jego połączenie z modelem SMALL używanym lokalnie dla procedur obsługi przerwań. COMPACT stosuje się przede wszystkim do programów zawierających dużą liczbę zmiennych, które nie wymagają krótkiego czasu dostępu. Odbywa się on bowiem za pomocą instrukcji MOVX A,@Ri wykorzystującej tryb adresowania pośredniego przy pomocy rejestru R0 lub R1. COMPACT może być również bardzo użyteczny dla aplikacji wymagających stosu o dużym rozmiarze, co może oznaczać konieczność umieszczenia go w zewnętrznej pamięci RAM poza układem mikrokontrolera.

LARGE - pamięć RAM do 64kB poza układem, 128 lub 256 bajtów w układzie

Model LARGE pozwala na niezbyt szybki dostęp do bardzo dużego obszaru pamięci RAM i jest przypuszczalnie najłatwiejszy w stosowaniu. Podobnie jak poprzednio, niezbyt często używa się go w niezależnie - najczęściej jest stosowany w połączeniu z modelem SMALL. Tak jak

List. 2.
```
#pragma COMPACT
void fast_func() SMALL
{
..... kod.....
}
```

w COMPACT, nadal używane są rejestry mikrokontrolera.

Wybór optymalnego segmentu dla danych

Podsumowując: mikrokontroler 8051 oferuje pięć segmentów pamięci dostępnych dla danych, z których każdy ma swoje pewne specyficzne cechy. W **tab. 1** przedstawiamy kilka wskazówek, ułatwiających dobór modelu do aplikacji.

Uwagi dotyczące stosowania modelu COMPACT

Przestrzeń XDATA jest adresowana przez DPTR, który umieszcza połówki 16-bitowego adresu w portach P0 i P2. Model COMPACT używa również R0 jako 8-bitowego wskaźnika, który umieszcza adres w tylko w porcie P0...P2. Jest on pod pełną kontrolą użytkownika i ponieważ jego wyprowadzenia połączone są z liniami adresowymi układu pamięci, pełni rolę przełącznika jej stron. Kompilator nie posiada informacji na temat stanu portu P2 i dopóki użytkownik nie ustawi jego wartości będzie ona niezdefiniowana, zazwyczaj równa 0xFF. Domyślnym dla zmiennych modelu COMPACT jest obszar PDATA adresowany za pomocą R0. Linker łączy zmienne XDATA oraz PDATA i umieszcza te ulokowane w obszarze PDATA od adresu 0x00. Niekoniecznie jest to intencją programisty - czasami zmienne mogą znajdować na którejś z kolejnych stron pamięci. Kompilator jednak używając jako domyślnego segmentu PDATA adresuje go za pośrednictwem R0 nie ustawiając tym samym wartości portu P2 odpowiedniej dla pożądanej strony pamięci RAM. Tak więc w rezultacie program COMPACT nie będzie pracował poprawnie.

Kiedy używa się kompilatora Keil bardzo ważne jest aby wartość PPAGE zawartą w zbiorze *startup.a51* ustawić na znaną wartość - dobrym wyborem jest 0x00. Stała PPAGEENABLE musi być ustawiona na „1" aby włączyć tryb stronicowania pamięci. Zaniedbanie tych nastaw zaowocuje bardzo niebezpiecznymi wynikami, jako że dane będą lokowane w zależności od przypadkowej wartości portu P2.

Kompilator Raisonance ustawia port P2 i zezwala na tryb stronicowania automatycznie. Podczas pracy linkera parametr PDATA(ADDR) musi być ustawiony aby powiedzieć linkerowi pod jakim adresem znajduje się obszar PDATA.

Wybór modelu pamięci

Model pamięci, zarówno dla narzędzi firmy Keil jak i Raisonance, wybiera się przy pomocy polecenia #*pragma* umieszczonego w pierwszej linii programu. Format polecenia jest następujący:

```
#pragma <Model Pamięci>
```

np.

```
#pragma LARGE
```

Domyślnie używany jest model SMALL i jak wspomniano wcześniej, może on mieć zastosowanie również do całkiem sporych programów, zapewniając pełną funkcjonalność segmentów PDATA i XDATA dla danych o niezbyt krytycznym czasie dostępu. Kompilatory C pozwalają również na lokalne definiowanie modeli pamięci przyporządkowanych do indywidualnych funkcji. Konsekwencją tego jest fakt, że w obrębie pojedynczego modułu, funkcje mogą zostać zadeklarowane jako SMALL, COMPACT lub LARGE (**list. 1**).

Program napisany w języku C może zawierać wszystkie mniej ważne funkcje skompilowane jako COMPACT oraz funkcje krytyczne pod względem czasu wykonywania (np. obsługi przerwań) skompilowane jako SMALL. Może to jednak w połączeniu z użyciem polecenia #*pragma* doprowadzić do niezamierzonych efektów pracy linkera, do komunikatów typu MULTIPLE PUBLIC DEFINTION (wielokrotna definicja funkcji). Powód jest taki, że podczas kompilowanie modułów jako COMPACT kompilator tworzy odniesienia do biblioteki funkcji właściwej dla tego modelu a funkcje kompilowane z wykorzystaniem modelu SMALL będą korzystały z biblioteki funkcji dla SMALL. Podczas pracy linkera, dla przykładu dwie definicje *putchar()* pochodzące z dwóch różnych bibliotek mogą zostać odnalezione.

Rozwiązaniem jest ustawienie jednego globalnego modelu i następnie użycie atrybutu SMALL opisywanego w poprzedniej sekcji do ustawienia modelu pamięci lokalnie (**list. 2**).

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

Źródło:
Embedded Systems Academy
http://www.esacademy.com/