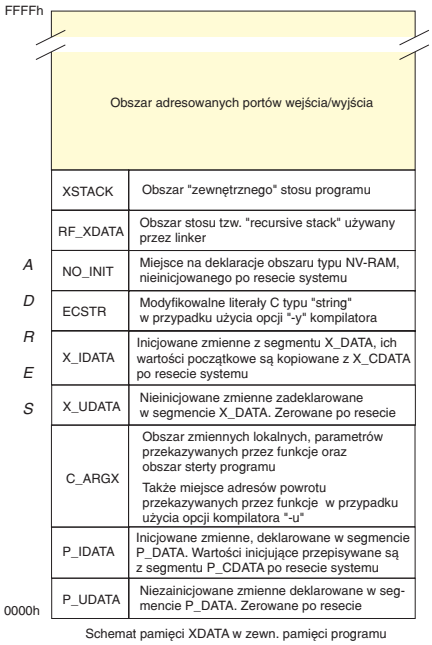
Realizacja projektów na 8051 przy pomocy oprogramowania firmy

W tym odcinku "Kursu" omówiono zagadnienia związane z obsługą wewnętrznej pamięci mikrokontrolerów serii '51, przy pomocy procedur dostępnych w pakiecie firmy IAR.



Schemat pamięci XDATA w zewn. pamięci programu

Rys. 3.

IDATA memory

Symbolem tym oznaczana jest wewnętrzna pamięć RAM procesora 8051 o adresach od 00h do 0FFh. Ponieważ pewna część procesorów tej rodziny posiada jedynie 128 bajtów pamięci użytkownika (00...7Fh), toteż kompilator umożliwia dostęp tylko do tej części pamięci, reszta jest zajęta przez SFR's, czyli rejestry specjalne procesora.

Wszystkie zmienne programowe zadeklarowane w obrębie tego segmentu pamięci są dostępne dla kompilatora poprzez adresowanie pośrednie (z wykorzystaniem wskaźników @R0 i @R1). Tak więc np. zapisanie stałej w obszarze tego segmentu zamiast w postaci:

```
MOV     45, #AA
```

kompilator wykona jako sekwencję 2 instrukcji:

```
MOV     R0, #45
MOV     @R0, #AA
```

Warto o tym pamiętać, szczególnie w sytuacjach kiedy czas wykonania rozkazu jest parametrem krytycznym.

DATA memory

Mianem tym kompilator określa wewnętrzną pamięć RAM procesora o adresach z zakresu 00h...7Fh, do której odwołania mogą być wykonane drogą adresowania pośredniego jak i bezpośredniego. W odróżnieniu od typu *IDATA,* wszystkie dane umieszczone w tym segmencie pamięci są dostępne poprzez drugi tryb adresowania.

Z punktu widzenia programisty, segment ten ma szczególne znaczenie przy definiowaniu zmiennych (danych), do których dostęp powinien być możliwie najkrótszy. Pisząc program, tego typu daną deklaruje się słowem kluczowym *data*.

W obrębie tego segmentu są dostępne także zmienne bitowe. Słowo kluczowe *bit* pozwala zadeklarować taką daną. Ten sposób deklarowania zapewnia najszybszy dostęp do zmiennych tego typu.

Kompilator C pozwala na definiowanie pól (rekordów) bitowych („bit-fields“) za pomocą standardowego słowa *struct*. Należy jednak pamiętać, że w tym przypadku kompilator nie umieszcza takich struktur w obszarze adresowania bitowego procesora 8051. Toteż, kiedy jest istotna szybkość wykonania instrukcji, należy wybierać metodę organizacji pojedynczych zmiennych bitowych, a nie struktur typu „bit-fields“.

BDATA memory

Obszar deklarowany jako BDATA to część wewnętrznej pamięci RAM procesora adresowanej bitowo - adresy 20...2Fh. Tu warto deklarować wszystkie zmienne tego typu, szczególnie jeżeli zależy nam na szybkim dostępie do nich.

PDATA memory

Pamięć tego typu jest dostępna poprzez adresowanie bezpośrednie zdefiniowanej strony zewnętrznej pamięci danych (typowe przy trybie dostępu do tej pamięci poprzez instrukcje *MOVX* wraz z adresowaniem poprzez wskaźniki *@R0* i *@R1*).

Obszar SFR

Znaczenie tego obszaru wewnętrznej pamięci RAM procesorów 8051 jest oczywiste dla programistów. Opisywany kompilator C zawiera zbiory nagłówkowe definiujące rejestry specjalne w zależności od zastosowanego rodzaju mikroprocesora. Zbiory te maja nazwy *IO*.H*, gdzie znak * określa ostatnie cyfry charakterystyczne wybranej kostki, np. *IO552.H* zawiera definicje rejestrów specjalnych dla procesora 80552. Dzięki temu są możliwe także (podobnie jak w przypadku kompilatorów niskiego poziomu) odwołania do poszczególnych bitów niektórych rejestrów specjalnych np.:

```
P1.7 = 0
```

co w efekcie wyzeruje najstarszy bit portu P1.

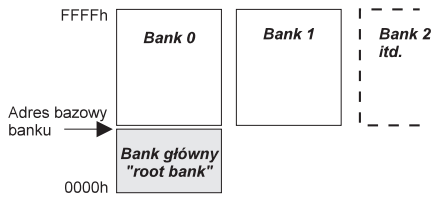
XDATA memory

Definicja ta deklaruje segment danych umieszczony w zewnętrznej pamięci RAM o adresach 0000h...0FFFFh. W obszarze tym może być definiowana (i stosowana) także pamięć typu NV-RAM oraz obszary wejścia-wyjścia. **Rys.3** przedstawia strukturę pamięci segmentu XDATA.

Wspomniany wcześniej typ pamięci NV-RAM jest ważny dla użytkownika, który w ten sposób może zabezpieczyć wybraną część danych po jego wyzerowaniu lub wyłączeniu procesoras.

BANKED memory model

Rys. 4 przedstawia mapę pamięci programu (CODE) przy korzystaniu z modelu typu banked. Górna sekcja przestrzeni adresowej, rozpoczynająca się od tzw. adresu bazowego banku,



Rys. 4.

może zawierać wiele banków przełączanych przez mikroprocesor podczas wykonywania programu (Bank 0, 1, 2 itd.). Aby umożliwić pracę w tym trybie, kompilator i linker generują dodatkowy adres - tzw. adres banku. Adres ten składa się z 16-bitowego adresu oraz dodatkowo numeru banku. Fizycznie pierwsza część adresu wystawiana jest typowo na szynę adresowa mikroprocesora 8051, numer banku natomiast, przekazywany jest za pośrednictwem portu P1 kontrolera lub innego, wybranego przez użytkownika, w przypadku innej wersji procesora (80552, 80451, itp).

Dolna sekcja przestrzenie adresowej nazywana „root bank" (bank główny), a adresowana od 0000h do adresu początkowego banku nie jest przełączalna.

W „root banku" kompilator umieszcza wszystkie obiekty programu, z wyjątkiem funkcji usługowych. Mogą być one umieszczone w przełączalnej części adresu, ale z równym powodzeniem mogą znajdować się w banku podstawowym. Pod pojęciem obiektów kryją się:
- funkcje usługowe bibliotek standardowych C;
- wszystkie typy stałych;
- wszystkie typy inicjujące zmienne;
- procedury obsługi przerwań;
- kod STARTUP.

Przy odwołaniach do tych elementów, kompilator zawsze używa adresu z obszaru dolnego - nie przełączalnego banków (ang. non bankable area). Rozmiar banku jest ograniczony adresem końcowym banku podstawowego -"root" a maksymalnym 16-bitowym adresem, jakim posługuje się procesor 8051 czyli FFFFh. Typowy rozmiar banku podstawowego zawiera się w granicach 16kB..48kB.

Kompilator przy odwołaniach do banków przełączalnych generuje 3-bajtowy wskaźnik, w którym pierwszy bajt określa numer banku, pozostałe dwa bajty zawierają 16-bitowy adres odwołania w wybranym banku pamięci programu. **Rys. 5** ilustruje strukturę wskaźnika przy adresowaniu pamięci programu w trybie przełączania banków.

Sposób pisania programu przez użytkownika dla tego modelu pamięci programu nie różni się zbytnio od modelu „*large*" . Istnieje jednak kilka różnic, o których warto pamiętać analizując kod źródłowy. O nich napiszemy w dalszej części artykułu.

Rozmiar banku a rozmiar segmentu kodu

Każdy skompilowany moduł zawiera segment zwany CODE. Obszar ten zawiera instrukcje odnoszące się do danego segmentu. Kompilator nie ma możliwości dzielenia, np. na 2 części, tej części segmentu i umieszczenia ich w różnych bankach pamięci programu. Dlatego pisząc programy należy pamiętać o tym, że rozmiar pojedynczego segmentu kodu musi być mniejszy od zdefiniowanego rozmiaru banku. W praktyce sytuacja taka występuje prawie zawsze.

Wywołania funkcji typu „banked" i „non-banked"

W każdym modelu pamięci, oprócz „banked model", adres powrotny, jak i nowy adres wywołania następnych funkcji, jest zawsze 16-bitowy (2 bajty). Taki sposób adresowania odnosi się także do odwołań funkcji tzw. „lokalnych". Aby dokonywać odwołań do funkcji tego typu, kompilator musi wiedzieć, że miejsce odwołania do tej funkcji oraz adres odwołania znajdują się w obszarze tego samego banku. Wywołania funkcji umieszczonych w innych bankach pamięci wymagają zachowania „trzeciego" bajtu adresu powrotu. Z tego też powodu oraz z faktu programowego zachowywania rozszerzonego (3-bajtowego) adresu powrotu, wywołania do funkcji lokalnych odbywają się szybciej. Fakt ten nabiera szczególnego znaczenia, jeżeli czas wykonania określonej funkcji jest parametrem krytycznym.

W trybie programowania pamięci typu „banked", użytkownik może zmusić kompilator do wygenerowania kodu funkcji typu „non banked". Kompilator umieszcza wtedy taki kod w banku głównym - root. Do zadeklarowania takiego segmentu służy słowo kluczowe „RCODE" („-RRCODE").

Dla przykładu rozpatrzmy dwie funkcje F1() i F2(), gdzie druga zostaje wywołana w ciele pierwszej funkcji. Pomimo, że obie zostały zdefiniowane w różnych modułach kodu źródłowego, to podczas konsolidacji linker umieści je w tym samym banku pamięci programu. W takim przypadku deklaracja funkcji F2 powinna wyglądać następująco:

```
non_banked void F2(void)
{
.............
/* tu kod funkcji */
.............
}
```

natomiast prototyp funkcji F2() w module F1() wywołującym ją będzie następujący:

```
extern non_banked
void F2(void);
```

a samo wywołanie:

```
void F1(void)
{
F2();
}
```

Oczywiście moduł, który zawiera funkcję wywołującą F1(), powinien być skompilowany z deklaracją segmentu typu -RRCODE.

Wywołania funkcji obsługi przerwań w modelu „banked"

Wywołania te są zawsze typu „non-banked". Dlatego przy korzystaniu z modelu „banked" wszystkie odwołania do funkcji obsługi przerwań powinny znajdować się w banku głównym - „root". IAR zaleca umieszczanie ich w oddzielnych modułach i kompilowanie z opcją -R, która automatycznie deklaruje segment z takimi odwołaniami jako RCODE.

Dla przykładu poniższe wywołanie kompilatora w postaci:

```
ICC8051 -mb -RRCODE isr
```

kompiluje moduł ISR.C zawierający funkcje obsługi przerwań (od ang. Interrupt Service Routines). Oczywiście korzystając z wersji kompilatora dla Windows - „Embedded Workbench" - opcje te wybiera się w menu kompilatora.

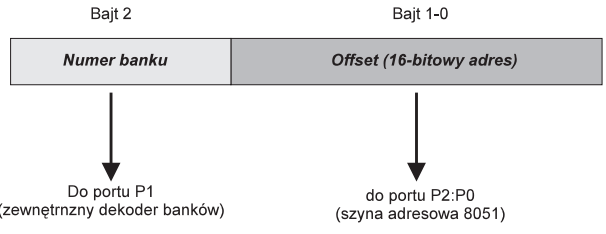
Modyfikacja portu obsługi pamięci w trybie „banked"

Domyślnym portem, poprzez który następuje przełączanie banków pamięci programu, jest P1. Użytkownik może zdefiniować inny port. W tym celu należy zmodyfikować znajdujący się w pakiecie plik źródłowy L18.S03. Zawiera on definicje i funkcje używane przez kompilator w trybie „bank mode". Dla ułatwienia zbiór zawiera dokładne linie komentarza, toteż modyfikacja nie jest kłopotliwa, szczególnie jeżeli pracujemy w środowisku shell a Workbench.

Po każdej zmianie tego programu należy moduł skompilować, a następnie powstały zbiór obiektowy skopiować jako CL8051B.R03.

Sławomir Surowiński, AVT

System udostępniła redakcji firma RK-System.



Rys. 5.