Proměnné a data

Jak si doufám pamatujete, má náš procesor harwardskou strukturu. Má tedy jednu paměť pro data (ROM) a jinou pro proměnné (RAM) . Paměť RAM je poměrně malá, proto s ní musíme šetřit.

Na druhou stranu, při našich výpočtech (např. filtr FIR) potřebujeme do paměti uložit velké množství dat, která nebudeme měnit. Přímo se tedy nabízí tato data uložit do paměti ROM. K tomu nám slouží mechanismus PSV ( pokud nevíte, co to je, zpátky na stromy a doučit – viz druhý ročník).

Použití PSV překladačem vynutíme direktivou const (const jako konstantní).

Příklad použití máme ve file data.c Zcela úmyslně jsou všechny proměnné inicializovány nějakou hodnotou při jejich definici. Některé jsou ovšem const. Tyto proměnné se uloží do paměti ROM a nelze s nimi hýbat (takže bychom je spíše měli nazývat konstanty).

**Proměnné - konstanty, které chceme uložit do paměti ROM, musí být globální.**

V našem program to jsou

 const unsigned int ci = 0x5678 ;

 const unsigned int ai = 0x1234;

 const int bi = 65000;

 const char ac = 'D';

 const char bc[20] = "TOMAS";

Uncomentněte si řádek bi = 7; přeložte a podívejte se na chybu

../../../progr\_a\_vysv/data/data.c:44:9: error: assignment of read-only variable 'bi'

Je to jasné: do paměti ROM se nedají ukládat data, proměnná bi je const

Program přeložíme

Podíváme se do Window – Pic Memory Wievs – Program Memory

Od adresy 0x0200 máme kód, který jsme nenaprogramovali. Neděste se , „vono samo“ si to nuluje na začátku celou paměť RAM.

Pustíme si simulátor –Debugger .

Breakpoint si dáme na první inicializaci v main int i = 5; .Protože je tam příkaz – přiřazení, bude to fungovat. Pusíme si Window – Pic Memory Wievs – Program Memory. Když nyní budeme mačkat F7 , uvidíme dole kód assembleru, který se provádí při přiřazování.

Podíváme se do Variables, nacpeme si tam všechny proměnné, které jsme deklarovali jako const. Ve Variables vidíme adresy, na kterých jsou proměnné uloženy. Např. proměnná **ci** je na adrese 0x0300. Samozřejmě v Program Memory. Podíváme se tam a skutečně vidíme na adrese 0x0300 uložené číslo 0x5678 .



Proměnné, které jsou uloženy v Program Memory, poznáme podle písmenka P před jejich názvem. V závorce je způsob přístupu (PSV).

 V Debuggeru si mačkáme F7 a díváme se do Program Memory.

Až dolezeme do místa, kde se přiřazuje z ROM do RAM ( j = ai ; ) , vidíme v Program Memory kód, který používá PSV

 MOV 0x8300, W0

 MOV W0, [W14]

0x8300 je samozřejmě adresa 0x0300 s 15. bitem v 1 ( a doufám chápete, proč )

No a zkoušíme to dál, až nám bude úplně jasné, co program dělá.

Pokud máme konstantních dat hodně ( např. vzorky signálu, konstanty filtru FIR, …. ), je poněkud nešikovné je napsat na začátek programu, protože vlastní kód programu bude až kdovíkde dole. V tom případě si pomůžeme tak, že definici konstant uložíme do jiného file a tento file includneme. Příklad máme v data02.c , include je vloz.h .

Program si přeložte, pusťte v simulátoru a dívejte se, co to dělá. A hlavně se podívejte, jak je uděláno zadání velkého množství dat ve file vloz.h

Data jsou od adresy 0x0300 do 0x3792, na začátku je 0x55 0x66 0x77 0x88 aby se to dalo snadno najít. Možná to váš překladač uloží někam trochu jinam, tak si to najděte. Ve Variables si najděte adresu pole vzorky a na tuto adresu se podívejte do Program Memory. Překladač se pokouší převádět hexadecimální čísla zpět na assembler. Pokud se mu to nepodaří, napíše NOP. Neděste se tedy, když tyto instrukce vidíte v program Memory

main od programu začíná na adrese 0x37AA (zase je možné, že to váš překladač přeloží trochu jinak ). Mezi adresou 0x0300 a 0x37A8 jsou napsána naše data z file vloz.h .

Dále si můžeme prohlédnout Program Memory. Na adrese 0x0000 je skok na adresu 0x0200 – opcode 0x040200 , na této adrese pokračuje program, který jsme nenapsali a který vynuluje celou paměť RAM. Na adrese 0x0234 vidíme instrukci CALL main , kterou se volá náš program na adrese 0x037AA ( opcode 0x0237AA ) .