V následující stati se zaměříme na vzájemnou komunikaci mezi Cčkovým file a aseemblerovským file. Pustíme si MPLABX, do Source file si dáme oba file, které máme v tomto adresáři , tedy asembler.s i Ccko.c Oba file samozřejmě překládáme najednou. Pustíme si simulátor, uděláme reset procesoru a mačkáme F7, abychom „vlezli“ i do podprogramů.

**Pozor ! Oba file musí mít různé názvy - blabla.c a blabla.s je špatně.**

Cckový file bude „hlavní program“. Je v něm funkce main, dále jsou zde nastavená konfigurační slova (fuses). Z hlavního programu budeme volat podprogramy, které jsou ve file assembler.s

Na začátku file Ccko.c jsou deklarace funkcí. Není tam deklarována funkce bla, a to proto, že ji používáme poněkud „prasácky“ – jednou bez parametrů, podruhé s parametrem, kterým je řetězec. Úplně správně je samozřejmě vše nadefinovat, ale chceme ukázat, že překladač si poradí i s takovouto nedokonalostí.

V Cckovém file voláme podprogram bla() . Tento podprogram je v assemblerovském file uvozen návěštím \_bla: Podtržítko je zásadně důležité, bez něj Cčkový program assemblerovský podprogram neuvidí. Dále je nutné, aby \_bla bylo nadeklarováno jako .global – to znamená, že tento identifikátor je viditelný i mimo assemblerovský file.

Mačkáme F7 a dáváme se, jak se podprogram volá. Nedělá nic, ale to nám nevadí, sledujeme pouze mechanismus volání a návratu.

Dále máme příklad předání parametrů do funkce. To vidíme u podprogramu parametr. Parametr může být pouze a jedině int (no, vlastně ne, časem uvidíme, že může být i fractional), a při volání podprogramu se tupě zapíše do registru W0. Dejte si do VARIABLES registr WREG0, dále proměnnou i, a dívejte se, co to dělá.

Další příklad je podprogram dvapar, který používá dva parametry integer. Parametry se prostě nacpou do registrů W0 a W1 . Pokud by v závorce bylo více parametrů, pak by se parametry nacpaly postupně do dalších registrů W2, W3, W4, .....W7. Dál než do W7 to nejde. Samozřejmě, protože registry jsou int, také parametry musí být int. Při debugování můžete mačkat F8, tím se podprogram udělá najednou jako jedna instrukce. Až budete u dvapar, mačkejte F7, tím program vleze dovnitř podprogramu.

Podprogram soucet ukazuje způsob předání čísla z assemblerovského podprogramu do Cčkového programu. To, co je při return v registru Wo, se prostě předá ven jako hodnota funkce soucet (a u nás se přiřadí do proměnné i). Překladač tedy na konec doplní tuto cčkovou konstrukci:

W0 = W0 + W1

return W0 ;

 }

Funkce vrací hodnotu, která je v registru W0 , s jiným registrem nelze pracovat.

A překladač umí ještě onačejší věci. Při použití řetězce jako parametr - při volání

bla("To je muj retezec" );

se do registru W0 předá pointer na řetězec – v naší assemblerovské terminologii se tam předá adresa začátku řetězce. Tato adresa je upravena tak, aby měla na 15. bitu hodnotu 1, tedy je nachystána pro použití mechanismu PSV. Podíváme se opět na registr W0, u čísla v něm odstraníme 1 na 15. bitu a máme adresu v paměti programu. Pustíme si Windov – Pic memory views – Program memory a najdeme si uvedenou adresu. Vidíme, že tady je opravdu uložen řetězec "To je muj retezec" . Formát si přitom přepněte na hex (dole pod oknem). Pokud necháte formát na code, bude se debugger snažit přeložit čísla zpátky na instrukce assembleru a většinou mu vyjde nop, takže nic neuvidíte. Samozřejmě čísla ve sloupci opcode jsou vždy platná, vidíme tam ASCII kódy písmenek.

Samozřejmě, s řetězcem chceme pracovat. Podprogram retezec ukazuje, jak se to dělá. Řetězec prostě čteme pomocí PSV. Cčkový řetězec končí 0, proto je na konci test na 0 a po jejím přečtení podprogram končí. V registru DSRPAG je uloženo správné číslo (podívejte se ). Netuším, zdali ho tam překladač nacpe pouze v případě, že používáme PSV, nebo zda ho tam „pro jistotu“ nacpe vždy, i když nevoláme žádný program používající PSV. Můžete to zkusit – vymažte volání podprogramu retezec a podívejte se, co je v registru DSRPAG.

Jo aha – pokud netušíte, k čemu je DSRPAG, tak zpátky na stromy a zopakovat PSV – máte to v <http://ozeas.sdb.cz/panska/mikroproc/33EV32/programovani/asm_progr_a_vysv/PSV/>

Stejně jako podprogramy můžeme sdílet mezi Cčkovým a assemblerovským file proměnné. Musí být deklarovány globálně. Ve file \*.c máme proměnnou promenna , do které jsme uložili číslo 0x1234 . Dále voláme funkci spromennou. Tam je zcela normální použití direktivy MOV.

MOV #\_promenna , W1 uloží do W1 adresu proměnné promenna, # znamená číslo

MOV \_promenna , W2 uloží do W2 obsah buňky proměnna .

Dále pokračujte v kapitole fixed point na <http://ozeas.sdb.cz/panska/mikroproc/33EV32/programovani/progr_a_vysv/fixed_point/>