Semestrální práce z

KIV/POT

Počítačová technika

Dělení 16 bitů / 16 bitů bez instrukce dělení

Martin Hron – A11B0379P
hronmar@students.zcu.cz

4. května 2013

1. Zadání

Dělení 16 bitů / 16 bitů = 16 bitů + 16 bitů (výsledek + zbytek) (bez použití instrukce dělení). Vstupy a výstupy hexadecimálně.

Protože simulátor procesoru H8S umožňuje též simulaci některých obvyklých služeb operačního systému (výstup znaku, výstup řetězce, ...), musí být programy napsány s jejich využitím. Vstupy proto budou zadávány z klávesnice, výstupy zobrazovány ve výstupním okně simulátoru.

V programu musí být ošetřeny vstupní hodnoty a musí být umožněn vícenásobný běh.

2. Programová dokumentace

2.1. Podprogramy

2.1.1. Podprogram dělení

Podprogram dělení provádí samotné dělení dvou 16 bitových čísel. Pro dělení se používá algoritmus odečítání dělitele od dělence a při každém odečtení se provede inkrementování pomocného registru pro výsledek. Na začátku podprogramu se vždy testuje, zda-li je dělitel větší než dělenec. Pokud ano, provede se skok na konec dělení, kde se přesune zbytek do pomocného registru pro zbytek a vyskočí se z podprogramu. Pokud ne, provede se algoritmus odečítání a pokračuje se další iterací.

Registr	Obsah
R2	Hodnota dělitele
E2	Hodnota dělence
R3	Hodnota výsledku
E3	Hodnota zbytku

2.1.2. Podprogram nulování registrů

Podprogram nulování registrů používá bitovou operaci XOR registru sám na sebe. Dochází tak k vynulování všech registrů kromě ER7, který se používá jako stack pointer.

2.1.3. Podprogram ošetření vstupu a převodu na hexadecimální hodnotu

Podprogram nejprve vezme první ASCII znak z paměti (bufferu) a uloží ho do registru R3L. Při této operaci se rovnou provede inkrementace ukazatele na buffer. Dále se provádí porovnání registru R3L s hodnotou 0x30 (0 v ASCII) a s hodnotou 0x46 (F v ASCII). Pokud se vstupní znak nevejde do tohoto rozmezí, rovnou se skáče na chybové hlášení o špatném

vstupu. Pokud se do rozmezí vejdeme, odečte se od registru 0x30 a porovnáme hodnotu s 0x9. Pokud je hodnota v registru větší než 9, skáčeme na návěští vstup_vetsi, kde se dále pracuje s hodnotou, jinak samovolně pokračujeme na zpracování čísla (návěští posun).

V části vstup_vetsi odečteme od hodnoty v registru hodnotu 0x7 a budeme porovnávat, jestli je daná hodnota v rozmezí od 0xA do 0xF. Pokud ne, skáčeme na chybové hlášení, pokud ano, skáčeme do části na zpracování čísla (návěští posun).

V části posun se nejprve provede logický součet registru R3L a R4L, což má za následek „přesunutí“ 4bitového čísla z R3L na poslední 4 bity v R4L (v R3L 0x5, v R4L 0x30 => 0x35). Jako další krok si do registru R3L přesunu další hodnotu z paměti a porovnám, jestli se nejedná o hodnotu 0x0A (\n v ASCII). Pokud ano, skáče se do části na zápis výsledného čísla do paměti. Pokud ne, provádí se další porovnání pomocného registru R6L s hodnotou 0x3 pro zjištění, jestli nejsou na vstupu více jak 4 ASCII znaky. Pokud jsou, skáče se na chybový výpis, jinak se provádí 4 bitové posuny doleva. To způsobí přesun HEXa znaku o jednu pozici doleva. Jako poslední se v této části provede inkrementace pomocného registru R6L a skáče se na začátek cyklu pro vstup.

V části zapis_vstup se provede zápis 16 bitového čísla z R4 na pozice do paměti, kam ukazuje registr ER5 a skáčeme z podprogramu.

V části chyba se provede vypsání chybové hlášky o zadání špatného vstupu, skáčeme na začátek programu a můžeme zadávat nové hodnoty.

Registr	Obsah
R0	Adresa pro výpis na konzoli
ER1	Adresa parametrického bloku par 4 („Špatný vstup“)
ER2	Ukazatel na buffer
R3/R3L	Hodnota jednoho znaku v ascii
R4/R4L	Výsledná hexadecimální hodnota
ER5	Ukazatel do paměti pro zapsání výsledné hodnoty
R6L	Pomocný registr pro zjištění počtu znaků na vstupu

2.1.4. Podprogram pro převedení výstupní hodnoty na ASCII

V podprogramu pro převedení výstupní hodnoty se nejprve porovná pomocný registr s hodnotou 4 z důvodu ukončení cyklu po převedení 4 čísel. Pokud tedy jsou převedeny 4 hodnoty skáče se na konec podprogramu. Pokud ne, přesune se 16 bitové hexadecimální číslo z R2 do R3 a v registru R2 se provedou 4 bitové posuny doprava, což způsobí „ztrátu“ poslední hexadecimální číslice (FA3C => 0FA3). Dále se pak inkrementuje pomocný registr R4 a v registru R3 se provede logický součin s hodnotou 0xF, což způsobí „získání“ pouze poslední hexadecimální číslice (0A4B => 000B). Následně provedeme porovnání hodnoty v R3 s 0x9. Pokud je hodnota větší, skáče se do části pricti, jinak se přičte hodnota 0x30 (v ASCII 0) a pokračuje se do části zapis_vystup.

Část pricti provede přičtení 0x37 (abychom se vešli v ASCII po přičtení do rozmezí od A - F) a skáče se do části zapis_vystup, kde se výsledná ASCII hodnota zapíše do paměti

na adresu v ER6, dekrementuje se adresa ukazatele do paměti a skáče se na začátek cyklu převodu výstupu.

Registr	Obsah
R2	Hodnota pro převedení na výstup
R3	Výsledná ASCII hodnota 1 znaku
R4L	Pomocný registr pro zjištění počtu převedených znaků
ER6	Ukazatel na adresu do paměti pro výstup

2.1.5. Podprogram zapsání LF (\n)

Podprogram nejprve inkrementuje ukazatel adresy na výstupní část o 4 a následně do registru R2L vloží hodnotu 0x0A (v ASCII \n). Tuto hodnotu pak zapíše do paměti na poslední byte bloku pro výstup, sníží adresu o 1 a vyskakuje z podprogramu.

Registr	Obsah
R2L	Hodnota 0x0A pro zapsání na konec bloku
ER6	Ukazatel na adresu do paměti pro výstupní blok

2.2. Datová část

Název	Adresa	Velikost	Význam
delenec_text	0xFF4000	-	text pro výzvu zadání dělence
delitel_text	0xFF400A	-	text pro výzvu zadání dělitele
vysledek_text	0xFF4014	-	text pro informační výpis vypsání výsledku
zbytek_text	0xFF401F	-	text pro informační výpis vypsání zbytku
chybka	0xFF4028	-	text pro informační výpis špatného vstupu
vystup_text	0xFF4037	5	pole pro výstupní text (výstup výsledku a zbytku)
buffer	0xFF403C	10	buffer pro načtení zadané hodnoty z konzole
par1	0xFF4050	4	parametrický blok pro dělence
par2	0xFF4054	4	parametrický blok pro buffer
par3	0xFF4058	4	parametrický blok pro dělitele
par4	0xFF405C	4	parametrický blok pro chybu
par5	0xFF4060	4	parametrický blok pro výsledek
par6	0xFF4064	4	parametrický blok pro zbytek
par7	0xFF4068	4	parametrický blok pro výstup
delenec	0xFF4070	2	pole pro hodnotu dělence
delitel	0xFF4074	2	pole pro hodnotu dělitele
vysledek	0xFF4076	2	pole pro výsledek
zbytek	0xFF4078	2	pole pro zbytek
stack	0xFF40B8	64	prostor pro zásobník

2.3. Kódová část

V kódové časti programu se nejprve inicializuje zásobník a poté volá podprogram pro nulování registrů. Dále se vypíše na konzoli výzva pro zadání dělence a konzole čeká na

vstupní řetězec. Po načtení vstupního řetězce do paměti se do registru ER2 přesune adresa bufferu a do ER5 adresa dělence z paměti. Po té se už jen volá podprogram pro ověření a převod vstupu na hexadecimální číslo. Následně se opět volá podprogram pro nulování registrů, především kvůli přehlednosti co v kterém registru je, a celý proces načtení a převodu se opakuje pouze s tím rozdílem, že se pracuje s dělitelem.

V další části se provede přesun adresy dělence do ER4 a dělitele do ER5 a následně jejich hodnoty do E2 (dělenec) a R2 (dělitel). Dochází k volání podprogramu pro dělení a následnému zápisu výsledku a zbytku do paměti na adresy výsledku a zbytku.

V poslední části se vždy nejprve přesunou adresy bloku výstupního textu (ER6) a výsledku nebo zbytku (ER5) do registrů a do výstupního bloku se na poslední pozici zapíše znak „\n“. Do registru R2 se vždy přesune hexadecimální hodnota, která se má vypisovat a zavolá se podprogram pro převod čísla na ASCII. Pak už jen proběhne vypsání příslušné hlášky a daného výsledku resp. zbytku. Na konci celého programu je zakomentována řádka pro neustálý běh programu tj. skok na _start. Nulování registrů v průběžných krocích je především kvůli přehlednosti, co v kterém registru je, a aby nedocházelo k případným chybám při vytváření programu.

(pozn. protože zásobník je používán pouze pro ukládání návratové adresy z podprogramu, není nutné uvádět konkrétní obsah, konkrétní návratové adresy jsou dokumentovány ve zdrojovém kódu)

3. Uživatelská dokumentace

Program je intuitivní na ovládání a není potřeba nějak rozsáhle popisovat ovládání. Uživatel se řídí pokyny konzole a na výzvu „Dělenec:“ nebo „Dělitel:“ zadá 16 bitové číslo v hexadecimálním tvaru např. A8F0. Všechna „písmena“ (A - F) musí být velká, jinak program vyhodnotí zadané číslo jako špatný vstup. Pokud uživatel zadá více než 4 hexadecimální čísla nebo jiný znak než hexadecimální číslo, program opět vyhodnotí vstup jako špatný. Při zadání špatného vstupu dojde k opětovnému zadání obou vstupů. Nepředpokládá se, že by uživatel zadal vstup delší než 9 znaků. V případě, že se tak stane, dojde k přetečení vyhrazeného bloku paměti pro buffer a program přestane fungovat jak má.

Výstup programu je taktéž na konzoli, kde se nejprve vypíše příslušná hláška („Výsledek:“ nebo „Zbytek:“) a za hláškou se vždy vypíše dané 16 bitové číslo v hexadecimálním tvaru např. 00A9. Celý program je možno upravit tak, aby docházelo k neustálému opakování běhů programu (odkomentování řádky jmp _start a zakomentování řádky bra end).

4. Závěr

Program v jazyce symbolických adres by měl splňovat zadání a měl by být schopný více běhů. Vstup by měl být ošetřen proti zadání nesmyslných hodnot. Program by se jistě dal řešit efektivněji, například zvolením jiného algoritmu pro dělení nebo efektivnějším využitím registrů, aby nebylo třeba registry před každým důležitým kroku nulovat.