ZČU - Fakulta aplikovaných věd

Jednoduchá kalkulačka v mikroprocesoru H8S

Počítačová technika (PCT)

Matyáš Krejčí

V Plzni dne 10.4.2024

Obsah projektu:

[**1. Popis semestrální práce 3**](#_xc4hdcrgmpam)

[a. Zadání semestrální práce 3](#_86fdrk32bcdg)

[b. Podrobný popis 3](#_xs6o5tdlifr9)

[**2. Vývoj a použité prostředky 3**](#_64myiz6yxx2)

[**3. Návrh a implementace řešení 4**](#_aaedjq6rlqv3)

[1. Algoritmy a podprogramy 4](#_yt06rh9f6r0k)

[**4. Deklarované proměnné 7**](#_bxo0lxyxdqzh)

[**5. Obsah paměti 7**](#_ig2scs80enby)

[**6. Závěr 8**](#_im6hvr1o2rql)

## Popis semestrální práce

### Zadání semestrální práce

Úlohou semestrální práce je vytvoření jednoduché kalkulačky v mikroprocesoru H8S. Kalkulačka bude mít omezený vstup na 16bitová celá čísla. Bude umět klasické operace sčítání, odčítání, násobení a dělení.

### Podrobný popis

Uživatel do jednoho řádku zadá vstupní řetězec, například ve formátu *“11+23=”* a stiskne enter. Po stisknutí enteru se provede průzkum řetězce, převod řetězce na celočíselné hodnoty, porovnání operandu a výpočet. Výpočet je vytištěn do konzole a program znovu pobídne uživatele k zadání další hodnoty. Pokud je hodnota pouze enter, program se zacyklí v metodě *end.\**

## Vývoj a použité prostředky

K tvorbě semestrální práce s procesorem H8S bylo použito vývojové prostředí HEW (high-performance embedded workshop), simulátor procesoru H8S a přístupné předpřipravené projekty pro vývoj semestrální práce.

Ke vzniku projektu byl použit jazyk symbolických adres pro procesor H8S od firmy Renesas z roku 2006. Dále byla použita přiložená dokumentace a materiály Dr. Ing. Karla Dudáčka.

## Návrh a implementace řešení

Program při spuštění pobídne uživatele k vložení příkladu k výpočtu výpisem v konzoli. Po vložení bude program kontrolovat každý bit ze vstupu a bude si průběžně upravovat a ukládat data. Po tomto procesu dojde k výpočtu, výpisu do konzole a bude čekat na další vstup. Tento proces se tedy bude opakovat.

### Algoritmy a podprogramy

**\_start:**

Zde program začíná. Jelikož počítáme s tím, že kalkulačka bude fungovat opakovaně bez potřeby manuálního spuštění, nejprve vymaže použité / plné registry a až poté připraví registr pro zásobník, registr pro formátovaný výstup a registr pro uložení dat od uživatele. Vloží do registru ER1 dekadické řády, které později použijeme na roznásobování čísel. Připraví první bit vstupu do registru R4L, dále připraví ASCII hodnotu číslice ‘0’ do registru R4H, který bude sloužit pro porovnávání čísel a uloží do zásobníku ASCII hodnotu enteru, která později bude využita. Odtud program pokračuje do podprogramu *string\_check*.

**string\_check:**

Tato metoda porovnává registry R4H a R4L, kde jsou uložené výše zmíněné hodnoty. Nejdříve zkontroluje, zda-li se uživatel nezadal symbol písmena ‘e’, které znamená ukončení programu. Nasledně operátor (+-\*/), poté jestli se neshodují registry a pokud nedojde ke shodě, přičte do registru R4H hodnotu 0x01 a proces opět pokračuje. Takto se děje do doby, pokud se registry nerovnají nebo pokud program došel až do hodnoty 0x40 v registru R4H. Pokud se jedná o znak ‘=’, program se odkloní do metody *delete\_process*.

**bye:**

Pokud se program dostat do této metody, uživatel musel zadat znak ‘e’. Vypíše se řetězec poděkování za použití a dostává se do metody *loop*, kde se program zacyklí (dynamický stop).

**loop:**

Dynamický stop.

**process:**

Do metody *process* se opět dostáváme z metody *string\_check* a to tehdy, pokud se dojde k jednomu z operátorů. Jejím úkolem je zpracovat čísla ze zásobníku a roznásobovat je dekadickými řády uloženými registru ER1, tedy i posunutí pointeru v tomto zásobníku. Zároveň počítá počet průchodů, abychom později mohli dekrementací vrátit registr ER1 do původního stavu. Vynásobené číslice přičítá do registru ER5, čímž získáme první operand. Pokud se v procesu narazí na 0x0A, které jsme na počátku vložili do zásobníku, program přesměruje *program counter* (PC) do metody *before decrement*.

**before\_decrement:**

Upravuje zásobník do správného tvaru, abychom poznali, kde končí hodnoty čísla, které uživatel zadal pomocí vložení 0x0A do zásobníku. Odtud se program dostane do *decrement\_operation*.

**continue:**

Funkce continue slouží pouze pro posun pointeru v registru ER2, načtení hodnoty bitu do R4L a opět krok do *string\_check*.

**delete\_process:**

Do této části se dostane pouze v případě, že byl načten znak ‘=’ a pouze maže registry ER4 a ER6, které následně budeme potřebovat. Z této metody program skáče do *process2*.

**pridej:**

Do této metody se program dostane, jestliže v registrech R4H a R4L nastala rovnost (číselná). Jejím úkolem je připravit číslice odečtením ASCII hodnoty ‘0’ a vložením do zásobníku. (např.: v registru máme obsah číslice 1, tedy 0x31, odečteme 0x30 a získáme číslice 1). Program pokračuje do *another*.

**another:**

Tato metoda nastává bezprostředně za metodou *pridej*. Vymaže předešlý registr R4L, který sloužil pro porovnávání čísel a naplní ho novou hodnotou registru ER2 (1bit). Odtud se opět program odchyluje do metody *string\_check*.

**decrement\_operation:**

Do této metody se dostáváme přes metodu *process*, která vypočítala počet číslic a na základě toho tato metoda dekrementuje pointer v dekadických řádech. Pokračuje pouze v případě, že se postupným odečítáním jedničky z registru ER3, kde máme napočítaný počet číslic vynuluje. Pokračuje do metody *continue*.

**process2:**

Plní naprosto stejnou funkci jako metoda process, ale používá k tomu registr ER5, do kterého ukládá druhý operand. Pokud se ze zásobníku popne hodnota 0x0A, kterou jsme opětovně vložili, *program counter* se přesměruje k metodě *reverse\_string*.

**reverse\_string:**

Zpětně procházíme string zadaný uživatelem a hledáme operátory (+-\*/) a podle tohoto se potom dostáváme do následující čtveřice.

**multiplication:**

Vynásobí registry ER6 a ER5. Po procesu se začne připravovat výpis v *lab1*.

**addition:**

Sečte registry ER5 a ER6, ve kterých jsou uloženy načtená čísla. Následně směřuje do metody *move*, která pouze nuluje registry a směřuje do *lab1*.

**substraction:**

Porovná hodnoty registrů ER5 a ER6 a podle toho rohoduje, do jaké ze dvou možných metod se vydá. Pokud je číslo v registru ER5 je větší než ER6, nastane metoda *sub\_fir\_low*, pokud naopak, tak *sub\_sec\_low*. Pokud nastane rovnost, ošetří to první podmínka.

**sub\_fir\_low:**

Odečte od registru ER5 registr ER6 a přesune výsledek do ER6. Pokračuje *lab1*.

**sub\_sec\_low:**

Odečte od registru ER6 registr ER5 a pokračuje do *lab1*.

**division:**

Podobně jako u odčítání nejdříve porovná čísla a podle toho se odchyluje do metody *div\_fir\_low* nebo *div\_sec\_low*. Pokud navíc nastane dělení nulou, program se přesune do metody *undef*.

**undef**:

Pokud program narazí na dělení nulou, vypíše hlášku “Undefined” a pokračuje na začátek a čeká na vstup.

**division\_fir\_low**:

Jelikož kalkulačka neumí řešit příklady s výsledkem s desetinnou tečkou pouze vyhodnotí výsledek jako nula .

**division\_sec\_low:**

Pokud je první číslo větší než druhé, využije program instrukci *divlxu* a posouvá se do metody *move*.

**move:**

Pouze maže registry registry ER4 a ER5 a pokračuje do *lab1*.

**lab1:**

Porovnává, jestli výsledek v registru ER6 je vetší nebo roven s hexadecimální hodnotou 2710 (10000dec). Pokud ano, přesune se do metody *lab\_jedna*. Pokud ne, pokračuje do *lab2*.

**lab\_jedna**:

Přičítá do registru R4L jedničku, která značít násobek desetitisícového řádu. Vrací se zpět do *lab1*.

**lab2:**

Porovnává, jestli výsledek v registru ER6 je vetší nebo roven s hexadecimální hodnotou 3E8 (1000dec). Pokud ano, přesune se do metody *lab\_dva*. Pokud ne, pokračuje do *lab3*.

**lab\_dva:**

Přičítá do registru R4H jedničku, která značí násobek řádu tisíců. Vrací se zpět do *lab2*.

**lab3:**

Porovnává, jestli výsledek v registru ER6 je vetší nebo roven s hexadecimální hodnotou 64 (100dec). Pokud ano, přesune se do metody *lab\_tri*. Pokud ne, pokračuje do *lab4*.

**lab\_tri:**

Přičítá do registru R5L jedničku, která značí násobek řádu stovek. Vrací se zpět do *lab3*.

**lab4:**

Porovnává, jestli výsledek v registru ER6 je vetší nebo roven s hexadecimální hodnotou 0A (10dec). Pokud ano, přesune se do metody *lab\_ctyri*. Pokud ne, pokračuje do *lab5*.

**lab\_ctyri:**

Přičítá do registru R5H jedničku, která značí násobek řádu desítek. Vrací se zpět do *lab4*.

**lab5:**

Porovnává, jestli výsledek v registru ER6 je vetší nebo roven s hexadecimální hodnotou 1 (1dec). Pokud ano, přesune se do metody *lab\_pet*. Pokud ne, pokračuje do *first\_char*.

**lab\_pet:**

Přičítá do registru E5 jedničku, která značí násobek řádu desítek. Vrací se zpět do *lab5*.

Zbytek metod pouze připravuje výpis podle toho, zdali je registr prázdný nebo není. Následně přičítá k číslu ascii ‘0’ (0x30) a přesune se do metody pokus, která vypisuje proměnnou par\_result, která byla naplněna čísly. Zárověň také vypíše tabulátor, který naformátuje správně konzoli. Program tedy vypíše výsledek a čeká na další vstup a celý proces se opakuje.

## Deklarované proměnné

Následující tabulka uvádí seznam proměnných, které byly použity v projektu. Dále byly také použity systémové akce GETS a PUTS.

| Název | Reference v paměti | Velikost |
| --- | --- | --- |
| vstup | FF4078 | 100b. |
| prompt | FF4128 | 50b. |
| dec\_c | FF4000 | 100b. |
| stck | FF4194 | 100b. |
| result | FF40DC | 50b. |
| minus | FF4114 | 4b. |
| undefined | FF413A | 8b. |
| tab | FF4145 | 4b. |
| goodbye | FF4115 | 4b. |

*Tabulka 1.1 - přehled proměnných*

**Vstup** obsahuje řetězec znaků, které uživatel vložil k výpočtu.

**Prompt** obsahuje výpisový řetězec znaků pro pobídnutí uživatele k zadání příkladu.

**Dec\_c** obsahuje dekadické řády od 10.000 do 1, kterými se násobí jednotlivé cifry.

**Stck** je 100 bitový zásobník uložený jako v registru ER7 (stack pointer)

Do proměnné **result** vkládáme výsledek výpočtu.

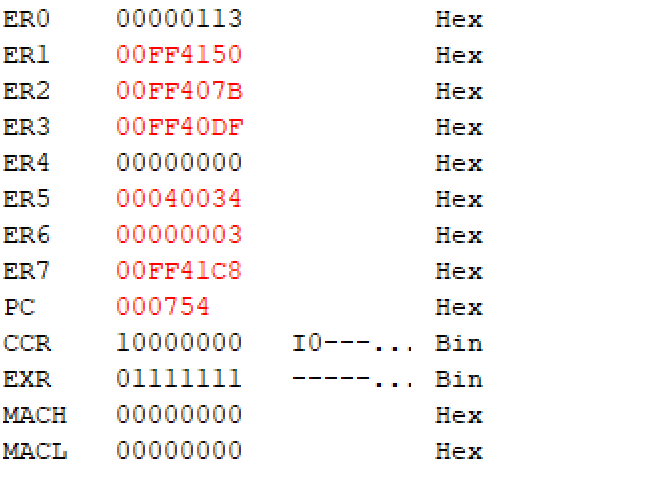
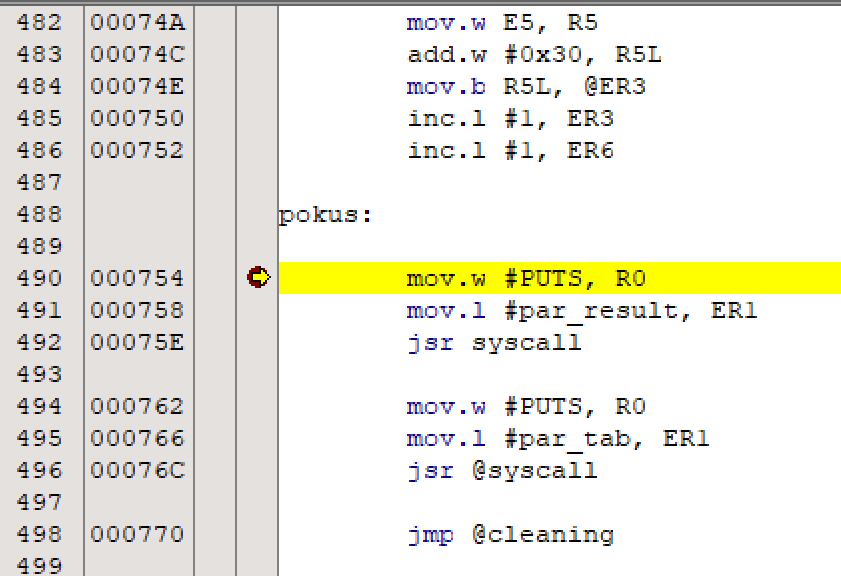
Proměnná **minus** pouze obsahuje znak ‘-’.

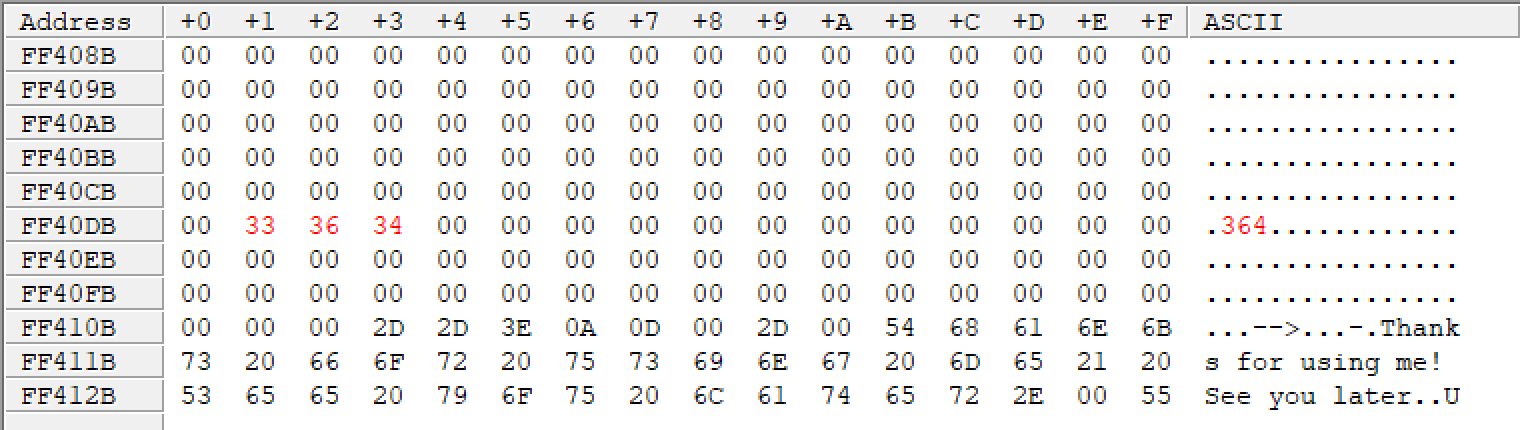
**Undefined** je řetězec v případě, že se jedná o dělení nulou.

**Tab** obsahuje pouze enter k odřádkování.

**Goodbye** je proměnná, která nese hlášku na rozloučení.

## Obsah paměti





Program jsme zastavili na řádce 490, těsně před výpisem výsledku na obrazovku. Výsledek příkladu “123+241=” máme uložený v registru ER3, tedy na adrese FF40DF, kam směřuje pointer adresy.

## Závěr

Vypracování projektu pro mne nebyla jednoduchá záležitost. Zvyky z vyšších programovacích jazyků nebyly lehké implementovat v jazyku symbolických adres (JSA), ale bylo poměrně zajímavé a zábavné prostudovávat strukturu JSA.

Některé operace mi zabraly dlouhou dobu, než jsem je dovedl do dosavadního stavu. Program byl testován na různých možnostech příkladů a je tedy funkční. Ze zadání mu chybí jedna vlastnost a to dělení s výsledkem s desetinnou tečkou.

Ve shrnutí mě tedy vypracování tohoto projektu bavilo, ale vyskytlo se mnoho situací, kdy jsem si dlouhou dobu nevěděl rady a zůstával i po hodinách studia na stejném místě.