Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Semestrální práce z předmětu POT

zadání č. 29

Vypracoval: Jiří Velek A20B0269P

Datum: 30.3.2021

**1. Zadání úlohy**

Program vypočítá a vypíše 25 členů Fibonacciho posloupnosti

Každý program musí obsahovat alespoň jeden kódový a jeden datový segment. Dále musí být řešení každé z úloh navrženo tak, aby obsahovalo alespoň jeden podprogram, která je v programu volán. Proto musí být správně inicializován a použit zásobník.

**2. Popis algoritmu řešení**

Program je rozdělen na tři části:

1. Inicializace registrů, příprava běhu programu
2. Výpočet Fibonacciho čísla iterativním způsobem
3. Převod čísla na ASCII
4. Výpis čísla

**3. Popis programu**

**3.1 Použité proměnné**

datový segment DATA:

**number** – číslo převedené do ASCII

**newline –** znak pro novou řádku

**p\_number –** ukazatel na number (pro výpis)

**p\_newline –** ukazatel na novou řádku (pro výpis)

**stack –** zásobník

pozice proměnných v paměti:

|  |  |
| --- | --- |
| Proměnná | Adresa v paměti |
| number | FF4000 |
| newline | FF4064 |
| p\_number | FF4068 |
| p\_newline | FF406C |
| stack | FF40D0 |

**3.2 Zásobník**

Zásobník je inicializován v registru ER7 a má velikost 100 B.

**3.3 Hlavní program**

Hlavní část programu začíná návěštím \_start. Inicializuje se zásobník, na ER0 se vloží n (počet Fibonacciho čísel). Zavolá se procedura *fib*, která vypočítá a vypíše prvních n Fibonacciho čísel. Na konci programu se spustí nekonečná smyčka.

**3.4 Procedury**

***strlen:***

procedura předpokládá řetězec v ER0 a vrací délku daného řetězce také v ER0

***strrev:***

procedura předpokládá řetězec v ER0, za pomocí smyčky a dvou pomocných registrů převrátí pořadí znaků v daném řetězci

***convert:***

procedura předpokládá celé číslo v ER0, a ukazatel na dostatečně velký buffer v ER1

převede číslo na řetězec, pomocí postupného dělení

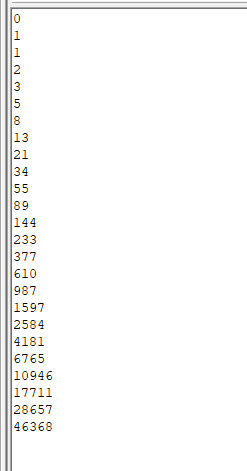
**4. Obsluha Programu**

Program je napsaný pro procesor H8S, přeložitelný a spustitelný na emulátoru HEW pro Windows.

Vstupní data jsou uložena ve zdrojovém souboru a jsou tedy známa již během překladu.

Výstup programu je zajištěn pomocí simulovaného výstupu simulátoru HEW.

**5. Výstup programu**



**6. Zdrojový kód programu**

.h8300s

.data

.equ syscall, 0x1FF00

.equ PUTS, 0x114

.align 2

number: .space 100 ; FF4000

newline: .asciz "\n" ; FF4064

.align 2

p\_number: .long number ; FF4068

p\_newline: .long newline ; FF406C

.align 1

.space 100

stack: ;FF40D0

.text

; ER0 = string

strlen:

mov.l ER0, ER2

xor.l ER0, ER0

strlen.loop:

mov.b @ER2, R1L

beq strlen.done

inc.l #1, ER0

inc.l #1, ER2

bra strlen.loop

strlen.done:

rts

;ER0 = string

strrev:

push.l ER0

jsr @strlen

;ER0 = strlen(str)

pop.l ER2

mov.l ER2, ER1

add.l ER0, ER2

dec.l #1, ER2

;ER1 = start

;ER2 = end

strrev.loop:

; while(end > start)

cmp.l ER1, ER2

bls strrev.done

mov.b @ER1, R0L

mov.b @ER2, R0H

mov.b R0L, @ER2

mov.b R0H, @ER1

inc.l #1, ER1

dec.l #1, ER2

bra strrev.loop

strrev.done:

rts

; ER0 = number

; ER1 pointer to string

; returns string in ER1

convert:

push.l ER1

mov.w #10, R2

convert.loop:

divxu.w R2, ER0

; R0 = podil

; E0 = zbytek

push.w R0

mov.w E0, R0

add.b #'0', R0L

mov.b R0L, @ER1

inc.l #1, ER1

pop.w R0

xor.w E0, E0

cmp.l #0, ER0

bne convert.loop

;convert.done:

mov.b #0, R0L

mov.b R0L, @ER1 ; je treba zakoncit null charakterem

pop.l ER0

jsr @strrev

rts

fib:

xor.l ER1, ER1

mov.l #1, ER2

xor.w E0, E0

fib.loop:

; while(E0 < R0)

cmp E0, R0

beq fib.done

push.l ER2

push.l ER1

push.l ER0

mov.l ER1, ER0

mov.l #number, ER1

jsr @convert

mov.w #PUTS, R0

mov.l #p\_number, ER1

jsr @syscall

mov.l #p\_newline, ER1

jsr @syscall

pop.l ER0

pop.l ER1

pop.l ER2

push.l ER2

add.l ER1, ER2

pop.l ER1

inc.w #1, E0

bra fib.loop

fib.done:

rts

.global \_start

\_start:

mov.l #stack, ER7

mov.w #25, R0

jsr @fib

lab3:

jmp @lab3