



ORGANIZACJA I ARCHITEKTURA KOMPUTERÓW

LABORATORIUM WT TP 07:30

---

## Sprawozdanie - laboratorium III

---

*Autor:*

Mateusz ŚLIWKA 241375

*Prowadzący:*

Mgr inż. Tomasz SERAFIN

Wrocław 10.04.2020

## Spis treści

<b>1</b>	<b>Wstęp . . . . .</b>	<b>2</b>
<b>2</b>	<b>Przebieg prac . . . . .</b>	<b>2</b>
<b>3</b>	<b>Napotkane problemy . . . . .</b>	<b>2</b>
<b>4</b>	<b>Opis implementacji . . . . .</b>	<b>3</b>
4.1	Operacje arytmetyczne . . . . .	4
4.2	Wyjątki . . . . .	6
<b>5</b>	<b>Opis uruchomienia programu . . . . .</b>	<b>7</b>

## 1 Wstęp

Powierzone zadanie polegało na przygotowaniu programu-kalkulatora realizującego działania zmiennoprzecinkowym pojedynczej/podwójnej precyzji. Kalkulator ten miał realizować dodawanie, odejmowanie, mnożenie oraz dzielenie. Program miał również uwzględniać różne sposoby zaokrąglania wyników. Oprócz kalkulatora należało zasymulować wystąpienie wyjątków.

## 2 Przebieg prac

Zadanie to było pierwszym zlecającym realizację programu pracującego na liczbach typu floating-point. Na początku należało więc zapoznać się z logiką działa jednostki X87 FPU w języku assembler. Po nabyciu wiedzy teoretycznej kolejnym etapem było stworzenie kalkulatora oraz przetestowania jego działania przy pomocy debuggera. Na początku w programie zaimplementowana została jedna precyzja oraz jedna metoda zaokrąglania. Po poznaniu budowy słowa sterującego program został rozbudowany o kolejne precyzje i zaokrąglania. Na końcu, w ostatnim etapie, do programu została dopisana symulacja wyjątków, której efekty również można zaobserwować w debuggerze gdb.

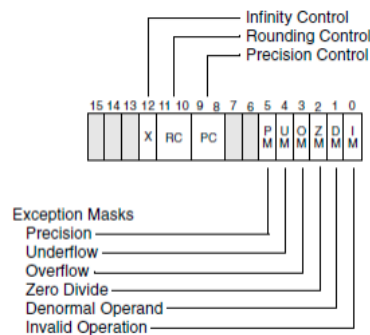
## 3 Napotkane problemy

Największym problemem było zrozumienie materiału teoretycznego. Rozszyfrowanie budowy słowa kontrolnego, budowa rejestrów dla liczb zmiennoprzecinkowych okazały się kluczowe i choć wymagały poświęcenia największej ilości czasu to potem ich dobra znajomość zdecydowanie uprościła rozbudowę programu o kolejne funkcje.

## 4 Opis implementacji

Wszystkie operacje kalkulatora oraz symulacja wyjątków zostały zaimplementowane w jednym programie. Na początku, przed skonstruowanie słowa kontrolnego możemy wybrać tryb zaokrąglania i precyzję.

Kluczowe w zrozumieniu jego konstruowania jest poznanie jego budowy, co świetnie opisuje poniższy schemat.



Rysunek 1: Budowa słowa sterującego <sup>1</sup>

```
#zaokraglenie
najblizsze: .short 0x000 #zaokragalnie do najblizszej, binarnie to 00000000 00000000
wdol: .short 0x400 #zaokraglanie do minus niesko, binarnie to 00000100 00000000
wgory: .short 0x800 #zaokraglanie do plus niesko binarnie to 00001000 00000000
obciecie: .short 0xc00 #zaokraglanie przez obciecie, binarnie to 00001100 00000000

#precyzja
pojedyncze: .short 0x000 #binarnie to 00000000 00000000
podwojna: .short 0x200 #binarnie to 00000010 00000000

#baza
slovo_sterujace: .short 0x103F #binarnie to 00010000 00111111
```

Rysunek 2: Stałe wykorzystywane do konstrukcji słowa sterującego

Po wyborze konkretnych stałych, dodaniu ich do siebie i wpisaniu do zmiennej `slovo_sterujace` zostaje ono załadowane przy pomocy komendy `fldcw`.

```
mov najblizsze, %eax #do eax wpisujemy tryb zaokraglania, w tym przypadku w dol
mov podwojna, %ebx #do ebx wpisujemy rodzaj precyzji na jakim operujemy
mov slovo_sterujace, %ecx #wpisanie slowa sterujacego do ecx
add %eax, %ebx #dodanie do siebiei rejestrow eax i ebx (zaokraglenie i precyzja), wynik w ebx
add %ebx, %ecx #dodanie do siebie rejestrow ebx i ecx (wyniku tego wyzej i slowa sterujacego), wynik w ecx
mov %ecx, slovo_sterujace #zapisanie ecx (nowego slowa sterujacego) jako slovo sterujace
fldcw slovo_sterujace #zaladowanie slowa sterujacego FPU
```

Rysunek 3: Załadowanie słowa sterujacego

Ważnym elementem jest również wybór trybu działania programu wraz ze wskazaniem typu precyzji na jakim chcemy działać poprzez odkomentowanie wybranych linii.

```
#WYBOR LICZB NA JAKICH DZIALAMY LUB TYLKO SYMULACJI WYJATKOW
jmp zaladuj_float #PROGRAM REALIZUJE DZIALANIA DLA POJEDYNCZEJ PRECYZJI
#jmp zaladuj_double #PROGRAM REALIZUJE DZIALANIA DLA PODWOJNEJ PRECYZJI
#jmp wyjatki #PROGRAM REALIZUJE SYMULACJE WYJATKOW

#WYBOR DZIALANIA DO ZREALIZOWANIA
dzialanie:
jmp dzielenie #w tym przypadku realizujemy dzielenie
```

Rysunek 4: Wybór trybu pracy programu

W tym momencie program jest gotowy do wykonywania operacji na wcześniej zadeklarowanych liczbach. Kolejnym etapem w realizacji programu było więc zaimplementowanie kolejnych działań

## 4.1 Operacje arytmetyczne

Każda z implementacji operacji arytmetycznych opiera się na wywołaniu odpowiedniej funkcji, którą tę operację realizuje. Każde z działań ma swoją funkcję.

- Dodawanie - **faddp**
- Odejmowanie - **fsubp**
- Dzielenie - **fdivp**
- Mnożenie - **fmulp**

---

<sup>1</sup>Źródło - Laboratorium architektury komputerów - materiały, Politechnika Wrocławska(str. 144)

```

dodawanie: #liczba1+liczba2
faddp #dodanie do siebie dwóch liczb tzn liczba2 z liczba 1 w st0, wynik do st0
jmp exit

```

Rysunek 5: Obsługa dodawania

Poniższe zrzuty ekranu pokazują wyniki wykonanych różnych operacji arytmetycznych na innych precyzjach oraz trybach zaokrąglania dla liczb:  $liczba1 = 5.513341$  oraz  $liczba2 = -1.21221$

```

(gdb) info float
=>R7: Valid 0xc001d5ddd10000000000 -6.683327198028564453
R6: Empty 0x4001b06d4a0000000000
R5: Empty 0x00000000000000000000
R4: Empty 0x00000000000000000000
R3: Empty 0x00000000000000000000
R2: Empty 0x00000000000000000000
R1: Empty 0x00000000000000000000
R0: Empty 0x00000000000000000000

Status Word: 0x3820 PE
TOP: 7
Control Word: 0x187f IM DM ZM OM UM PM
PC: Single Precision (24-bits)
RC: Round up
Tag Word: 0x3fff
Instruction Pointer: 0x00:0x080480bd
Operand Pointer: 0x00:0x00000000
Opcode: 0x0000
(gdb)

```

Rysunek 6: Pojedyncza precyzja, zaokrąglenie w gore, mnozenie

```

(gdb) info float
=>R7: Valid 0xc001d5ddd0c7f0d25800 -6.683327093609999103
R6: Empty 0x4001b06d4a1ad6451800
R5: Empty 0x00000000000000000000
R4: Empty 0x00000000000000000000
R3: Empty 0x00000000000000000000
R2: Empty 0x00000000000000000000
R1: Empty 0x00000000000000000000
R0: Empty 0x00000000000000000000

Status Word: 0x3820 PE
TOP: 7
Control Word: 0x1a7f IM DM ZM OM UM PM
PC: Double Precision (53-bits)
RC: Round up
Tag Word: 0x3fff
Instruction Pointer: 0x00:0x080480bd
Operand Pointer: 0x00:0x00000000
Opcode: 0x0000
(gdb)

```

Rysunek 7: Podwojna precyzja, zaokrąglenie w gore, mnozenie

```

(gdb) info float
=>R7: Valid 0x4001d737b6bb12902000 +6.7255509999999999391
R6: Empty 0x4001b06d4a1ad6451800
R5: Empty 0x00000000000000000000
R4: Empty 0x00000000000000000000
R3: Empty 0x00000000000000000000
R2: Empty 0x00000000000000000000
R1: Empty 0x00000000000000000000
R0: Empty 0x00000000000000000000

Status Word: 0x3820 PE
TOP: 7
Control Word: 0x127f IM DM ZM OM UM PM
PC: Double Precision (53-bits)
RC: Round to nearest
Tag Word: 0x3fff
Instruction Pointer: 0x00:0x080480b9
Operand Pointer: 0x00:0x00000000
Opcode: 0x0000
(gdb)

```

Rysunek 8: Podwojna precyzja, zaokrąglenie do najbliższej, odejmowanie

```

(gdb) info float
=>R7: Valid 0x400189a2dd0000000000 +4.301130771636962891
R6: Empty 0x4001b06d4a0000000000
R5: Empty 0x00000000000000000000
R4: Empty 0x00000000000000000000
R3: Empty 0x00000000000000000000
R2: Empty 0x00000000000000000000
R1: Empty 0x00000000000000000000
R0: Empty 0x00000000000000000000

Status Word: 0x3820 PE
TOP: 7
Control Word: 0x1c7f IM DM ZM OM UM PM
PC: Single Precision (24-bits)
RC: Round toward zero
Tag Word: 0x3fff
Instruction Pointer: 0x00:0x080480b5
Operand Pointer: 0x00:0x00000000
Opcode: 0x0000
(gdb)

```

Rysunek 9: Pojedyncza precyzja, zaokrąglenie przez obcięcie, dodawanie

Przy wybranych liczbach, po zmianie precyzji efekt widoczny jest przez wywołanie komendy **info float** w debuggerze gdb. Przy pomocy tej komendy możemy zobaczyć zmianę precyzji liczby oraz efekt działania zaokrąglania.

## 4.2 Wyjątki

Wyjątki zostały zaimplementowane w podobny sposób jak działania. Wszystkie wyjątki wywołane zostały przy pomocy operacji *fdiv*. Wybrany został specjalny, wywołujący wyjątek zestaw liczb. Są to:

- Dla NaN wykonujemy operacje 0/0
- Dla +0 wykonujemy operacje 0/liczba1 (dodatnia)
- Dla -0 wykonujemy operacje 0/liczba2 (ujemna)
- Dla -inf wykonujemy operacje liczba2(ujemna)/0
- Dla +inf wykonujemy operacje liczba1(dodatnia)/0

Po wykonaniu tych operacji pokolei debugger zwrócił nam informacje o takim stanie rejestrów, co potwierdza prawidłowe zaimplementowanie algorytmów.

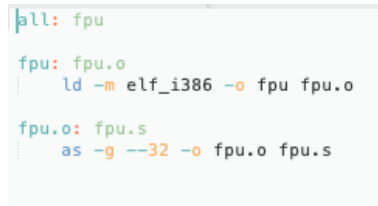
```
(gdb) info float
R7: Special 0xffffc000000000000000 Real Indefinite (NaN)
R6: Zero    0x00000000000000000000 +0
R5: Zero    0x80000000000000000000 -0
R4: Special 0xffff8000000000000000 -Inf
=>R3: Special 0x7fff8000000000000000 +Inf
R2: Empty   0x00000000000000000000
R1: Empty   0x00000000000000000000
R0: Empty   0x00000000000000000000

Status Word:      0x1805  IE  ZE
                  TOP: 3
Control Word:     0x1a7f  IM DM ZM OM UM PM
                  PC: Double Precision (53-bits)
                  RC: Round up
Tag Word:         0x96bf
Instruction Pointer: 0x00:0x080480ff
Operand Pointer:   0x00:0x00000000
Opcode:           0x0000
```

Rysunek 10: Stan rejestrów po wywołaniu wyjątków

## 5 Opis uruchomienia programu

Program uruchamiany jest przy pomocy makefile, który asembduje i linkuje program. Przy asemblacji użyte zostały takie opcje jak `-32` (informacje o kodzie dla systemu 32bit) oraz `-g` (generowanie flag dla debuggera). Do komendy linkowania dołożona została informacja o wybranym trybie emulacji `elf_i386`.



```
all: fpu

fpu: fpu.o
    ld -m elf_i386 -o fpu fpu.o

fpu.o: fpu.s
    as -g -32 -o fpu.o fpu.s
```

Rysunek 11: Zawartość pliku makefile