Raport Laboratorium OiAK 2 Zajęcia nr 2

Data wykonania	26.04.2021
Termin zajęć	Czwartek TP 17:05
Autor	Mateusz Kusiak Indeks:252805

1 Treść zadania

Proszę napisać program w języku assemblera w architekturze 32-bitowej na platformę Linux. Program powinien zapytać użytkownika o podanie dwóch liczb rzeczywistych, operacji do wykonanie (+ - * /) oraz sposobu zaokrąglania wyniku FPU. Program powinien wypisać na standardowe wyjście wynik danej operacji zaokrąglony w wybrany sposób.

- do wczytywania / wyświetlania można używać funkcji z libc (printf, scanf)
- liczby wpisywane są w sposób dziesiętny, np. 1.23456789
- wynik wyświetlamy też w sposobie dziesiętnym
- sposoby zaokrąglania FPU (wg FPU control word): nearest (even if tie), down, up, to zero

1.1 Przykład wywołania

```
$ ./calculator

1 - dodawanie
2 - odejmowanie
3 - mnozenie
4 - dzielenie
Podaj operacje:
```

2 Objaśnienie

Poniżej znajduje się kod wraz z objaśnieniami:

```
.section .data
hellostring:
.ascii "Wpisz dzialanie matematyczne np. (13.2 / 2): \nWprowadzone dane:\0"
error_msg:
.ascii "SYNTAX ERROR!!!\n\0"
temp_for_scanf:
.ascii "%f %c %f"
equals:
.ascii "Wynik: %f\n\0"
is_zero_message:
.ascii "Nie mozna dzielic przez zero!\n\0"
round_msg:
.ascii "Wybierz rodzaj przyblizenia: \n1.Nearest \n2.Down \n3.Up \n4.To_zero\n5.Default

#definicja slow kluczowych dla operacji zaokraglania
old_rounding: .word 0
nearest_rounding: .word 0x0000
down_rounding: .word 0x0400
```

```
up_rounding: .word 0x0800
zero_rounding: .word 0x0C00

temp_two_for_scanf:
.ascii "%d"
```

Listing 1: Sekcja .data w programie, stworzeine potrzebnych treści i formantów dla funkcji printf oraz scanf oraz przypisanie wartości dla słów kuczowych potrzebnych do jednostki sterującej FPU.

```
.section .bss
    .lcomm input_one, 32
    .lcomm input_two, 32
    .lcomm input_sing, 8
    .lcomm type_of_rounding, 8
```

Listing 2: Przypisane zmiennych w pamięci potrzebnych w programie które są pobierane od użytkownika.

```
#wpisanie podstawowej konfiguracji zaokraglenia FPU
fstcw old_rounding

#dwa bity okreslaja zaokraglenie w s owie sterujacym jednostki FPU
#jednak zmiana dwoch bitow i pozostawienie reszty nietknietej potrzebuje operacji
movw old_rounding, %ax
andb $0b11110011, %ah
orw %ax, nearest_rounding
orw %ax, down_rounding
orw %ax, up_rounding
orw %ax, zero_rounding
```

Listing 3: Początek właściwego programu. Ustawienie poprawnych wartości słów kluczowych za pomocą funkcji orw oraz załadowanie podstawowej konfiguracji do jednostki FPU.

```
#wypisanie komunikatu powitalnego do uzytkownika
pushl $hellostring
call printf

#wczytanie danych od uzytkownika:
 #pierwsza liczba
 #znak arytmetyczny (rodzaj operacji)
 #druga liczba
pushl $input_one
pushl $input_sing
pushl $input_two
pushl $temp_for_scanf
call scanf

#pokazanie menu zaokraglania
pushl $round_msg
```

```
call printf

#zczytanie liczby odpopiadajacej zaokraglaniu
pushl $type_of_rounding
pushl $temp_two_for_scanf
call scanf
```

Listing 4: Wyświetlenie menu i pobranie danych od użytkownika wpisanych do terminala. Opiera się to na ustawieniu argumentów na stosie a następnie wywołaniu odpowiednio funkckji Scanf i Printf z biblioteki libc.

```
#porownywanie znaku wpisanej liczby znaku
#z odpowiednia etykieta
#i przeskoczenie do odpowiedniego mmiejsca w kodzie
movl $input_sing, %eax
mov (%eax), %bl
cmpb $'+', %bl
je addition
cmpb $'-', %bl
je subtraction
cmpb $'*', %bl
je multiply
cmpb $'/', %bl
je divide
```

Listing 5: Blok złożony z funkcji porównujących, a następnie przeskakujących do odpowiednich etykiet w kodzie w zależności od tego co wprowadził użytkownik, w tym przypadku znaku arytmetycznego.

```
#ogolny blad ktory obsluguje wyjatki takie jak
#nieprawidlowa operacja arytmetyczna
error:
pushl $error_msg
call printf
jmp end

#komunikat dzielenia przez 0
zero:
pushl $is_zero_message
call printf
jmp end
```

Listing 6: Sekcja która wypisuje na ekranie błędy oraz kończy działanie programu. Zostały uwzględnione nieodpowiednio wprowadzone dane przez użytkownika oraz podzielenie przez 0.

```
#dodawanie
addition:
fld input_one   #zaladowanie do stosu FPU pierwszej liczby
fld input_two   #zaladowanie do stosu FPU drugiej liczby
faddp           #dodanie dwoch liczb, zapisz w st(1) i pop st(0)
#operacje skokowe zarzadzajace obsluga przyblizenia
jmp round_menu

#odejmowanie
subtraction:
fld input_one   #zaladowanie do stosu FPU pierwszej liczby
fld input_two   #zaladowanie do stosu FPU drugiej liczby
fsubp           #odjecie dwoch liczb st(0)-st(1) i pop st(0)
jmp round_menu

#mnozenie
multiply:
fld input_one   #zaladowanie do stosu FPU pierwszej liczby
fld input_two   #zaladowanie do stosu FPU drugiej liczby
fmulp           #wymnozenie dwoch liczb i zapisz w st(1) i pop st(0)
jmp round_menu
```

Listing 7: Sekcja która załadowuje liczby do stosu FPU oraz wykonuje operacje dodawania (faddp), odejmowania(subp), mnożenia(mulp) - działanie funkcji zostało napisane w postaci komentarza w kodzie powyżej.

```
#dzielenie
divide:
fld input_one
ftst #funkcja testujaca liczbe przechowywana w st(0) i odpowiednio ustawia flagi C0 C2
fnstsw %ax #przechowanie slowa statusowego w rejestrze ax
fwait
sahf #zapisuje rejestr ah we flagach
jpe error
ja continue
jb continue
jz zero #jesli dzielnik = 0 blad!

#jesli nie dzielimy przez 0 to wykonuj dalej
continue:
fld input_two
fdivp           #podzielenie dwoch liczb st(1)/st(0)
```

Listing 8: Sekcja dzielenia która musi obsłużyć wyjątek dzielenia przez 0. Wypisuje odpowiedni komunikat. Za pomocą funkcji ftst zostaje sprawdzona wartość liczbowa w st(0) a następnie podejmowane odpowiednie kroki.

```
round_menu:
movl $type_of_rounding, %eax
mov (%eax), %bl
cmpb $1, %bl
je nearest
cmpb $2, %bl
je down
cmpb $3, %bl
je up
cmpb $4, %bl
je chop
cmpb $5, %bl
je default
jmp error
```

Listing 9: Sekcja obsługująca odpowiednie zaokrąglenie wybrane przez użytkownika. Została rozwiązana bardzo podobnie do tej, w której użytkownik wybierał rodzaj działania matematycznego.

```
#zaokraglanie
#najblizsza liczba
nearest:
fldcw nearest_rounding
frndint
jmp show

#do gory (+nieskonczonosc)
up:
fldcw up_rounding       #zaladowanie slowa do jednostki FPU
frndint                 #zaokraglenie do intigera
jmp show

#do dolu (-nieskonczonosc)
down:
fldcw down_rounding
frndint
jmp show

#po przez obciecie
chop:
fldcw zero_rounding
frndint

#domyslne
default:
```

Listing 10: Sekcja obsługująca odpowiednie zaokrąglenie wybrane przez użytkownika. Wykonywanie programu zostaje przeniesione w odpowiednie miejsce w celu zaokrąglenia liczby w

wybrany przez używkownika sposób. Funkcją fldcw zostaje załadowaneodpowiednie dla danego zaokrąglenia słowo kluczowe jednostki sterującej FPU a następnie wykonana instrukcja frndint czyli zaokrąglenie liczby do Intiger. Zostaje opcja default która wyświetla wynik w postaci liczby zmienno przecinkowej.

```
show:
fstpl (%esp)
pushl $equals
call printf      #wypisanie wyniku
pushl $0

end:
#funkcja z bibioteki libc konczaca program
call exit
```

Listing 10: Sekcja wyświetlająca wynik i kończąca program. Dzięki funkcji fstpl liczba zostaje załadowana dla funkcji printf jako argument.

3 Wnioski

W trakcie wykonywania tego programu na laboratorium dowiedziałem się jak poprawnie używać instrukcji dla procesorów FPU. Poznałem sposoby zaokrąglania liczb zmiennoprzecinkowych jak również "obsługę wyjątków" (przy dzieleniu przez 0). Po przez ciągłą pracę nad programem zauważałem że dość mocno moge optymalizować kod, gdy zobaczyłem ten sam blok funkcji powtarzający się w programie. Pomocne okazały się funkcje Scanf i Printf z biblioteki libc. Stworzenie prostego interfejsu tekstowego także stwarzało pewne problemy ale po odpowiednim sprawdzeniu kodu i dodaniu instrukcji skokowych błędy zostały one poprawione.