Aleksandra Ojdana, 198842 Wrocław, 09.05.2019
PT-N-17

Prowadzący: dr Piotr Patronik

Laboratorium Organizacja i Architektura Komputerów
Podstawowe operacje arytmetyczne liczb zmiennoprzecinkowych

1. Treść ćwiczenia

1.1 Zakres i program ćwiczenia

Celem ćwiczenia była realizacja następujących programów w języku asembler:

- wykonującego podstawowe operacji arytmetycznych (dodawanie, odejmowanie, mnożenie, dzielenie) dla liczb zmiennoprzecinkowych pojedynczej precyzji bez użycia instrukcji zmiennoprzecinkowych
- drukującego na standardowe wyjście wartości dziesiętnej liczby zmiennoprzecinkowej

W trakcie ćwiczeń obowiązywała znajomość wszystkich dotychczas poznanych komend debuggera, z szczególnym zwróceniem uwagi na: print, display, x.

Do wykonania wymaganych ćwiczeń wymagana była znajomość wykonywania operacji arytmetycznych na liczbach zmiennoprzecinkowych.

1.2 Zadania zrealizowane podczas ćwiczeń

Podczas ćwiczeń napisano algorytmy przebiegu programów, które później zostały wykorzystane do implementacji programów. Dodatkowo podczas ćwiczeń przetestowano wszystkie wymagane komendy debuggera, a także zapoznano się z dokumentacją opisującą flagi.

1.3 Zadania zrealizowane poza ćwiczeniami

Poza ćwiczeniami zrealizowano wszystkie wymagane programy w oparciu o wcześniej stworzone algorytmy. Zrealizowane programy:

- program dodający liczby zmiennoprzecinkowe pojedynczej precyzji
- program odejmujący liczby zmiennoprzecinkowe pojedynczej precyzji
- program mnożący liczby zmiennoprzecinkowe pojedynczej precyzji
- program dzielący liczby zmiennoprzecinkowe pojedynczej precyzji
- program wyświetlający wartość dziesiętną liczby zmiennoprzecinkowej pojedynczej precyzji

2 Przebieg ćwiczenia

2.1 Zagadnienia teoretyczne

Przed przystąpieniem do realizacji programów odświeżono wiedzę z zakresu licz zmiennoprzecinkowych tj. ich zapisu oraz wykonywania na nich operacji arytmetycznych. Zapoznano się także jeszcze raz z opisem funkcji debuggera: display, print i x.

1. Zapis liczb zmiennoprzecinkowych pojedynczej precyzji

Format pojedynczej precyzji liczb zmiennoprzecinkowych składa się z 32 bitów, z czego:
- pierwsze 23 najmłodsze bity nazywane są mantysą i stanowią dziesiętną część liczby. W zapisie znormalizowanym przed wartością dziesiętną domyślnie jest 1.
- kolejne 8 bitów to wykładnik, czyli liczba do jakiej potęgi zostanie podniesiona podstawa powiększona o obciążenie, czyli w przypadku pojedynczej precyzji jest to +127
- ostatni 32gi bit to bit znaku, informuję o tym czy liczba jest dodatnia, czy ujemna.

Przykładowo liczba: 1 1000 0010 0001 0000 0000 0000 0000 000 zapisana w formacie zmiennoprzecinkowym ma wartość w systemie dziesiętnym: -68 ponieważ:

Wykładnik = 1000 0010 → 133, potęga = 133 – 127 = 6
Mantysa = 0001 0000 0000 0000 0000 000 → 1,0001 0000 0000 0000 0000 000
Znak = 1 → 1 oznacza, że liczba jest ujemna, 0 dodatnia
Liczba (wartość bezwzględna)= 1,0001 0000 0000 0000 0000 000 * 2^6 = 10001 00, po zamianie na system dziesiętny otrzymujemy 68. Bit znaku jest równy 1 czyli jest to liczba ujemna, więc zapisana liczba równa się -68.

2. Operacje arytmetyczne na liczbach zmiennoprzecinkowych pojedynczej precyzji - algorytmy

2.2.1 Dodawanie

Zamiana liczby zapisanej w formacie zmiennoprzecinkowym na system o podstawie 2. Następnie sprawdzenie znaków. W przypadku takich samych znaków wykonanie operacji dodawania poprzez funkcję add, zamiana wyniku na zapis zmiennoprzecinkowy z pozostawieniem znaku. W przypadku różnych znaków:

- jeżeli pierwsza liczba jest dodatnia, wtedy wykonywane jest odejmowania drugiej liczby od pierwszej

- jeżeli pierwsza liczba jest ujemna, wtedy wykonywane jest odejmowanie pierwszej liczby od drugiej

W przypadku różnych znaków, bit znaku pobierany jest z większej liczby.

2.2.2 Odejmowanie

Zamiana liczby zapisanej w formacie zmiennoprzecinkowym na system o podstawie 2. Następnie sprawdzenie znaków. W przypadku takich samych znaków wykonanie operacji dodawania poprzez funkcję add, zamiana wyniku na zapis zmiennoprzecinkowy, znak pobierany jest z argumentu o większej wartości. W przypadku różnych znaków:

- jeżeli pierwsza liczba jest dodatnia, wtedy wykonywane jest dodawanie, bit znak jest dodatni

- jeżeli pierwsza liczba jest ujemna, wtedy wykonywane jest dodawanie, bit znaku jest ujemny

2.2.3 Mnożenie

Zamiana liczby zapisanej w formacie zmiennoprzecinkowym na system o podstawie 2. Następnie wykonanie operacji mnożenia.

2.2.4 Dzielenie

Zamiana liczby zapisanej w formacie zmiennoprzecinkowym na system o podstawie 2. Następnie wykonanie operacji dzielenia.

3. Wyszczególnione komendy debuggera oraz nowe instrukcje wykorzystane w programach

Opisy komend oraz sposób ich wykorzystania sprawdzano poleceniem help w debugerze.

print – wyświetla zawartość pod wskazanym adresem
display – wyświetla zawartość pod wskazanym adresem, za każdym razem jak program się zatrzyma
x/Nbx adres – wyświetla zawartość danego adresu w formacie szesnastkowym, N odpowiada liczbie wyświetlonych bajtów
x/Nwf - wyświetla zawartość danego adresu w formacie float, N odpowiada liczbie słów maszynowych
x/Ni - wyświetla zawartość danego adresu w formie instrukcji
shl – funkcja przesuwająca wartość zapisaną w danym rejestrze w lewo o zadana liczbę bitów [1]
btc – sprawdzenie jednego wskazanego bitu z ciągu bitów i zapisanie go w fladze CF [1]
jnb – sprawdzenie wartości flagi CF, jeżeli równa się zero wykonuje się skok pod wskazany adres [1]

2.2 Uruchomienie programu

Procedura uruchamiania wszystkich programów odbyła się na takiej samej zasadzie jak podczas wykonywania ćwiczeń na poprzednich zajęciach, z wykorzystaniem pliku Makefile. Ta część ćwiczeń została opisana w sprawozdaniu do laboratorium „Podstawy uruchomienia programów asemblerowych na platformie Linux/x86”.

2.3 Konstrukcja pliku źródłowego

Konstrukcja plików źródłowych wyglądała podobnie jak w plikach źródłowych z poprzednich ćwiczeń. Opis w sprawozdaniu „Podstawy uruchomienia programów asemblerowych na platformie Linux/x86”.

3. Wnioski

Zaimplementowano programy wykonujące podstawowe operacje arytmetyczne na liczbach zmiennoprzecinkowych. Programy działają poprawnie tylko dla liczb całkowitych, w programach brakuje sprawdzenia, czy liczba posiada wartości dziesiętne. Ponadto wykonano program wyświetlający na standardowe wyjście wartość dziesiętną liczby zmiennoprzecinkowej.
Podczas realizacji zadań problemem okazało się samo stworzenie algorytmu, który można było zaimplementować w języku asembler bez nadmiarowego kodu, czyli w taki sposób, aby kod był czytelny.

4. Literatura

1. IA-32 Intel Architecture Software Developer’s Manual Volume 2: Instruction Set Reference

5. Programy

Dodawanie liczb zmiennoprzecinkowych

CCESS = 0

```
.data
liczba1: .byte 0b01000010, 0b01100000,0b00000000, 0b00000000
liczba2: .byte 0b11000010, 0b01100000,0b00000000, 0b00000000

.global _start
_start:

mov $1, %esi
mov $0, %edi

#zapis wykładnika do rejestru
movb liczba1(%esi,%edi,4), %bh
add $0b10000001, %bh

movb liczba2(%esi,%edi,4), %bl
add $0b10000001, %bl

#przesunięcie mantysy
mov liczba1, %ecx
and $0b00000000011111111111111111111111, %ecx
add $0b00000000100000000000000000000000, %ecx
shl %ecx, %bh

pop %ecx

mov liczba2, %ecx
and $0b00000000011111111111111111111111, %ecx
add $0b00000000100000000000000000000000, %ecx
shl %ecx, %bh

pop %ecx

#zapis znaku do rejestru
mov $0, %esi
mov $0, %edi

movb liczba1(%esi,%edi,4), %ah
and $0b10000000, %ah

movb liczba2(%esi,%edi,4), %al
and $0b10000000, %al
```

```
#sprawdzenie znaku
cmp %ah, %al
je same_sign_add

cmp %ah, %al
jg one_two_sub

#wykonanie operacji zależnie od znaku
two_one_sub:
push %eax
push %ebx
sub %eax, %ebx
jmp end

one_two_sub:
push %eax
push %ebx
sub %ebx, %eax
jmp end

same_sign_add:
push %eax
push %ebx
add %ebx, %eax
jmp end

end:
mov $SYSEXIT, %eax
mov $EXIT_SUCCESS, %ebx
int $0x80
```

Odejmowanie liczb zmiennoprzecinkowych

```
SYSEXIT = 1
EXIT_SUCCESS = 0

.data

liczba1: .byte 0b01000010, 0b01100000,0b00000000, 0b00000000
liczba2: .byte 0b11000010, 0b01100000,0b00000000, 0b00000000

.global _start
_start:

mov $1, %esi
mov $0, %edi

#zapis wykładnika do rejestru
movb liczba1(%esi,%edi,4), %bh
add $0b10000001, %bh
```

```
movb liczba2(%esi,%edi,4), %bl
add $0b10000001, %bl


#przesunięcie mantysy
mov liczba1, %ecx
and $0b00000000011111111111111111111111, %ecx
add $0b00000000100000000000000000000000, %ecx
shl %ecx, %bh

pop %ecx

mov liczba2, %ecx
and $0b00000000011111111111111111111111, %ecx
add $0b00000000100000000000000000000000, %ecx
shl %ecx, %bh

pop %ecx

#wykonanie operacji
add_plus:
push %eax
push %ebx
add %eax, %ebx
jmp end

#zapis znaku do rejestru
mov $0, %esi
mov $0, %edi

movb liczba1(%esi,%edi,4), %ah
and $0b10000000, %ah

movb liczba2(%esi,%edi,4), %al
and $0b10000000, %al

#sprawdzenie znaku
cmp %ah, %al
je same_sign_add

cmp %ah, %al
jg add_minus

same_sign_add:
#znak pobierany z większej liczby

end:
mov $SYSEXIT, %eax
mov $EXIT_SUCCESS, %ebx
int $0x80
```

Mnożenie/Dzielenie liczb zmiennoprzecinkowych

```
SYSEXIT = 1
EXIT_SUCCESS = 0

.data

liczba1: .byte 0b01000010, 0b01100000,0b00000000, 0b00000000
liczba2: .byte 0b11000010, 0b01100000,0b00000000, 0b00000000

.global _start
_start:

mov $1, %esi
mov $0, %edi

#zapis wykładnika do rejestru
movb liczba1(%esi,%edi,4), %bh
add $0b10000001, %bh

movb liczba2(%esi,%edi,4), %bl
add $0b10000001, %bl


#przesunięcie mantysy
mov liczba1, %ecx
and $0b00000000011111111111111111111111, %ecx
add $0b00000000100000000000000000000000, %ecx
shl %ecx, %bh

pop %ecx

mov liczba2, %ecx
and $0b00000000011111111111111111111111, %ecx
add $0b00000000100000000000000000000000, %ecx
shl %ecx, %bh

pop %ecx

#wykonanie operacji
push %eax
push %ebx
mov %ebx, %edx
mul %eax  #div %eax, %ebx – dla dzielenia
jmp end

#zapis znaku do rejestru
mov $0, %esi
mov $0, %edi

movb liczba1(%esi,%edi,4), %ah
```

```
and $0b10000000, %ah

movb liczba2(%esi,%edi,4), %al
and $0b10000000, %al

#sprawdzenie znaku
cmp %ah, %al
je same_sign

#liczba ujemna
jmp end

same_sign:
#liczba dodatnia

end:
mov $SYSEXIT, %eax
mov $EXIT_SUCCESS, %ebx
int $0x80
```

Wyświetlanie części dziesiętnej liczby zmiennoprzecinkowej

```
EXIT =1
READ =3
WRITE =4
STDOUT =1
STDIN =0
RETURN =0

.data

liczba1: .byte 0b11111111, 0b01111111,0b01111100, 0b11111111
one: .ascii "1"
one_len = . -one
zero: .ascii "0"
zero_len = . - zero
new_line: .ascii "\n"
new_line_len = . -new_line

.global _start
_start:


#pobranie liczby dziesietnej z liczby
mov liczba1, %ecx
and $0b00000000011111111111111111111111, %ecx

mov $23, %edi
loop_start:
cmpl $0, %edi
je loop_end
dec %edi
```

```
btc %edi, %ecx
jnb print_one

mov $zero_len, %edx
mov $zero, %ecx
mov $STDOUT, %ebx
mov $WRITE, %eax
int $0x80
jmp loop_start

print_one:
mov $one_len, %edx
mov $one, %ecx
mov $STDOUT, %ebx
mov $WRITE, %eax
int $0x80
jmp loop_start


loop_end:
mov $new_line_len, %edx
mov $new_line, %ecx
mov $STDOUT, %ebx
mov $WRITE, %eax
int $0x80

mov $EXIT, %eax
mov $RETURN, %ebx
int $0x80
```