ORGANIZACJA I ARCHITEKTURA KOMPUTERÓW

LAB 3

08 CZERWCA 2019
ŚRODA, TN 17:05
AUTOR: WOJCIECH KUR
PROWADZĄCY: DR INŻ. PIOTR PATRONIK

Spis treści

1. Treść ćwiczenia

1.1. Zakres i program ćwiczenia

1.1.1. Napisanie programu dodającego/odejmującego/mnożącego/dzielącego liczby zmiennoprzecinkowe pojedynczej precyzji – bez użycia instrukcji zmiennoprzecinkowych.

1.1.2. Napisanie programu drukującego na standardowe wyjście wartość dziesiętną liczby zmiennoprzecinkowej.

1.2. Zrealizowane zadania

1.2.1. Stworzenie programu dodającego/odejmującego oraz mnożącego liczby zmiennoprzecinkowe pojedynczej precyzji.

2. Przebieg ćwiczenia

2.1. Konstrukcja pliku źródłowego „adder”

2.1.1. Listing 1: kod źródłowy adder.s

```
.data
EXIT =                  1
READ =                   3
WRITE =                  4
STDIN =                  0
STDOUT =                   1
SYSCALL =                  0x80

sign1:
.byte 1
exponent1:
.byte 2
mantissa1:
.byte 1, 2, 4, 1

sign2:
.byte 0
exponent2:
.byte 2
mantissa2:
.byte 1, 1, 3, 1

signout:
.byte 0
exponentout:
.byte 0
mantissaout:
.byte 0, 0, 0, 0

.text
.global _start
_start:
add $0b01111111, exponent1
add $0b01111111, exponent2

mov exponent1, %al
mov exponent2, %bl
cmp %al, %bl
jg set_sign_2
jl set_sign_1
mov mantissa1, %eax
mov mantissa2, %ebx
cmp %eax, %ebx
jg set_sign_2
```

```
jl set_sign_1
je preparations

set_sign_1:
mov sign1, %al
mov %al, signout
jmp preparations
set_sign_2:
mov sign2, %al
mov %al, signout

preparations:
mov $0, %cl
mov exponent1, %al
mov exponent2, %bl
cmp %al, %bl
jl increment_exp2
je end_shift

increment_exp1:
inc %al
inc %cl
cmp %al, %bl
jg increment_exp1

mov mantissa1, %esi
shr %cl, %esi
mov %esi, mantissa1
jmp end_shift

increment_exp2:
inc %bl
inc %cl
cmp %bl, %al
jg increment_exp2

mov mantissa2, %esi
shr %cl, %esi
mov %esi, mantissa2

end_shift:
mov %al, exponentout
mov $0, %al

mov sign1, %bl
mov sign2, %cl
mov mantissa1, %esi
mov mantissa2, %edi

cmp %bl, %cl
jg negate_sign2
```

```
je adder

neg %edi
inc %edi
jmp adder

negate_sign2:
neg %esi
inc %esi
adder:

add %esi, %edi
cmp %bl, %cl
je check_shift
cmp $1, signout
jne check_shift
negate:
neg %edi
inc %edi
check_shift:

cmp $0, %edi
je end

mov %edi, mantissaout
shr $24, %edi
cmp $1, %edi
jl decrement_exp
je end

inc %al
mov mantissaout, %esi
shr $1, %esi
mov %esi, mantissaout
jmp end

decrement_exp:
dec %al
mov mantissaout, %esi
shl $1, %esi
mov %esi, mantissaout
mov %esi, %edi
shr $24, %edi
cmp $1, %di
jl decrement_exp
end:
add %al, exponentout
mov mantissaout, %esi
shl $8, %esi
mov %esi, mantissaout
```

```
mov $EXIT, %eax
mov $0, %ebx
int $SYSCALL
```

2.1.2. Opis programu

W sekcji zmiennych data zadeklarowane są modele kolejno: pierwszej liczby zmiennoprzecinkowej, drugiej liczby zmiennoprzecinkowej oraz wyniku w tym samym formacie. Format ten to: 1 bajt dla znaku, 1 bajt dla wykładnika oraz 4 bajty dla mantysy. Mantysa została rozszerzona o 4 bajt w celu ułatwienia obliczeń (jedynka z przodu) i nie jest uwzględniana w wyniku końcowym. Ostatni bit w 3 bajcie służy do ewentualnego zaokrąglenia. Poprawny zapis mantysy w inpucie to: [trzecie 8 bitów][drugie 8 bitów][pierwsze 8 bitów][jedynka z przodu]. Jest to spowodowane notacją little endian.

Następuje wyliczenie znaku końcowego oraz wyrównanie wykładników (mniejszy do większego) po czym mantysy zostają dodane z uwzględnieniem liczby ujemnej (zamiana na liczbę przeciwną). Po wykonaniu operacji dodawania liczba zostaje znormalizowana, a wykładnik odpowiednio zwiększony bądź zmniejszony.

2.2. Konstrukcja pliku źródłowego „multiplier”

2.2.1 Listing 2: kod źródłowy multiplier.s

```
.data
EXIT =                  1
READ =                   3
WRITE =                  4
STDIN =                  0
STDOUT =                   1
SYSCALL =                  0x80

sign1:
.byte 1
exponent1:
.byte 2
mantissa1:
.byte 2, 0, 0, 1

sign2:
.byte 0
exponent2:
.byte 2
mantissa2:
.byte 2, 0, 0, 1

signout:
.byte 0
exponentout:
.byte 0
mantissaout:
.byte 0, 0, 0, 0

.text
.global _start
_start:
add $0b01111111, exponent1
add $0b01111111, exponent2

mov sign1, %al
mov sign2, %bl
xor %al, %bl
mov %bl, signout

mov exponent1, %al
mov exponent2, %bl
sub $0b01111111, %bl
add %al, %bl
jc errors
mov %bl, exponentout
```

```
mov mantissa1, %eax
mov mantissa2, %esi
mul %esi
breakpoint:
cmp $0, %edx
je last_check
shrd $24, %edx, %eax

last_check:
mov %eax, %esi
shr $24, %esi
cmp $1, %esi
je end

mov exponentout, %bl

last_check_loop:
shr $1, %eax
inc %bl
mov %eax, %esi
shr $24, %esi
cmp $1, %esi
jne last_check_loop

mov %bl, exponentout
mov %eax, mantissaout
jnc end
errors:
mov $0b11111111, %al
mov %al, exponentout
mov $0x00000000, %eax
mov %eax, mantissaout

end:
shl $8, %eax
mov %eax, mantissaout

mov $EXIT, %eax
mov $0, %ebx
int $SYSCALL
```

2.2.2. Opis programu

Program przyjmuje takie same modele liczby zminnoprzecinkowej jak podczas dodawania. Znak wyniku zostaje wyliczony za pomocą rozkazu XOR. Wykładniki zostają dodane oraz odjęte zostaje obciążenie w celu przywrócenia liczby do prawidłowego stanu. Następnie mantysy zostają wymnożone, wynik skrócony do 24bitów (23 bity + 1 w celu ewentualnego zaokrąglenia) oraz znormalizowany.

3. Podsumowanie

Zadanie zostało wykonane w połowie i choć w przypadku dodawania jak i mnożenia nie została zaimplementowana obsługa wszystkich wyjątków to program udało się zredukować o część instrukcji. Jest to dowodem na to, iż stosowanie pewnych założeń oraz konkretnych instrukcji (lub zasad działania procesora) pozwala zoptymalizować działanie programu.

4. Literatura

4.1. http://zak.ict.pwr.wroc.pl/materials/architektura/laboratorium%20AK2/wzorzec%20sprawozdania.pdf, wzorzec sprawozdania

4.2. http://jedrzej.ulasiewicz.staff.iiar.pwr.wroc.pl/Architektura-Komputerow/lab/Architektura-63.pdf, laboratorium architektury komputerów – materiały dr Jędrzeja Ułasiewicza

4.3. https://pl.wikibooks.org/wiki/Asembler_x86, teoria oraz prosty program krok po kroku

4.4. http://quantum-mirror.hu/mirrors/pub/gnusavannah/pgubook/ProgrammingGroundUp-1-0-booksize.pdf?fbclid=IwAR0b_yzxqe1Ib9ANvA1BX5r4fFWKh6TarAiws4QtwKpikw2nGNYaYwcYwfI, Programming from the Ground Up

4.5. http://www.cs.umd.edu/~meesh/cmsc311/links/handouts/ia32.pdf

4.6. http://zak.ict.pwr.wroc.pl/materials/architektura/laboratorium%20AK2/Dokumentacja/Intel%20Penium%20IV/IA-32%20Intel%20Architecture%20Software%20Developers%20Manual%20vol.%202%20-%20Instruction%20Set%20Reference.pdf, dokumentacja intela (Instruction Reference vol.2)