

Raport Laboratorium OiAK 2 Zajęcia nr 3

Data wykonania	10.05.2021
Termin zajęć	Czwartek TP 17:05
Autor	Mateusz Kusiak Indeks:252805

1 Treść zadania

Proszę napisać program w języku asemblera w architekturze 32 bit. Program powinien zapytać użytkownika o dwie całkowite liczby. Użytkownik wpisuje liczby w notacji hexadecymalnej (szesnastkowo). Program powinien wypisać na standardowe wyjście wynik mnożenia tych dwóch liczb. Uwagi:

- liczby są podawane z klawiatury, a więc jako tekst, trzeba je sobie przekonwertować do obliczeń
- podane liczby mogą być bardzo duże, w szczególności możemy się umówić na limit 200 znaków (np CF00A1 - to jest 6 znaków)
- należy wykorzystać odpowiedni algorytm mnożenia dużych liczb (np. mnożenie przez części)
- sposoby zaokrąglania FPU (wg FPU control word): nearest (even if tie), down, up, to zero

1.1 Przykład wywołania

```
$ ./hex_calc
Podaj liczbe: 21AABB
Kolejna liczba:
```

2 Objaśnienie

Poniżej znajduje się kod wraz z objaśnieniami:

```
.data
message_first: .ascii "Podaj_liczbe:_" #zmienne potrzebne do zadania
message_first_length = . - message_first #d u g o zmiennej potrzebna do funkcji
message_second: .ascii "Kolejna_liczba:_" #zmienne potrzebne do zadania
message_second_length = . - message_second
message_endl: .ascii "\n"
message_endl_length = . - message_endl
message_user_one: .ascii "*****"
message_user_one_length = . - message_user_one
message_user_two: .ascii "*****"
message_user_two_length = . - message_user_two
space_one = (message_user_one_length/2)
space_two = (message_user_two_length/2)
equal_len = (space_one+space_two)*4
equal_space = equal_len*4
equal_space_write = equal_space/2
```

Listing 1: Sekcja .data w programie, stworzenie potrzebnych treści, które umożliwią poprawne działanie programu.

```

.bss
number_one:
.space space_one
number_two:
.space space_two
equal:
.space equal_len
equal_write:
.space equal_space_write

```

Listing 2: Przypisane zmiennych w pamięci potrzebnych w programie które są pobierane od użytkownika oraz konwertowane podczas działania programu.

```

mov $SYSWRITE, %eax      # funkcja do wywołania – SYSWRITE (wypisanie danych na ekranie)
mov $STDOUT, %ebx        # argument pierwszy systemowy deskryptor stdout
mov $message_first, %ecx  # argument drugi adres początku łańcucha ascii
mov $message_first_length, %edx # argument trzeci długość łańcucha ascii
int $SYSCALL32 # wywołanie funkcji

movl $SYSREAD, %eax
movl $STDIN, %ebx
movl $message_user_one, %ecx
movl $message_user_one_length, %edx
int $SYSCALL32

mov $SYSWRITE, %eax
mov $STDOUT, %ebx
mov $message_second, %ecx
mov $message_second_length, %edx
int $SYSCALL32

movl $SYSREAD, %eax
movl $STDIN, %ebx
movl $message_user_two, %ecx
movl $message_user_two_length, %edx
int $SYSCALL32

```

Listing 3: Początek właściwego programu. Wyświetlenie wiadomości oraz pobranie danych od użytkownika.

```
# petla przekształcająca lancuch znakow ASCII na HEX
```

```
movl $0, %ecx  
movl $0, %ebx
```

```
begin:  
movl $0, %eax  
movb message_user_one(,%ebx,1), %al  
cmpb $0x30, %al  
jb end  
subb $0x30, %al  
cmpb $0x0A, %al  
jb number  
subb $0x7, %al
```

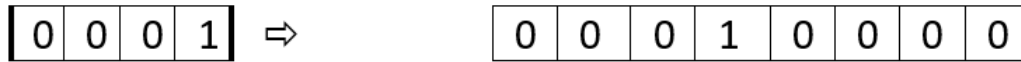
```
number:  
shll $4, number_one(,%ecx,4)  
addl %eax, number_one(,%ecx,4)  
incl %ebx  
cmpl $message_user_two_length, %ebx  
je end  
jmp begin
```

```
end:
```

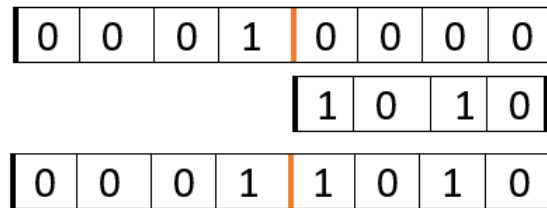
Listing 4: Konwersja danych pobranych od użytkownika (ASCII) na binarny kod szesnastkowy.
Poniżej został przedstawiony algorytm, który został zaimplementowany w kodzie.



Rysunek 1: Konwersja z ASCII do binarnego za pomocą odjęcia stałej(przykład dla znaku A = 10d) dla liczb stała odejmowana jest inna, ponieważ tak ułożone są symbole w kodzie ASCII.



Rysunek 2: Przetworzenie kolejnych bitów liczby. Następnie przesunięcie 4 bitów o 4 bity w lewo aby dać miejsce dla 4 bitów młodszych liczby szesnastkowej.



Rysunek 3: Sklejenie dwóch kolejnych liczb szesnastkowych w postaci binarnej.

#mnozenie wielkich liczb

```

movl $0, %ebx
multiplication_x:
cmpl $4, %ebx
jz ending
movl $0, %ecx
movl $0, %esi

multiplication_s:
movl $0, %edi
cmpl $4, %ecx
jz end_s
movl number_one(,%ebx,4), %eax
movl number_two(,%ecx,4), %edx
mull %edx
addl %ebx, %ecx
addl equal(,%ecx,4), %edx
movl %eax, equal(,%ecx,4)
incl %ecx

```

```

adcl equal(,%ecx,4) , %edx
adcl $0, %edi
adcl %esi, %edx
adcl $0, %edi
movl %edi, %esi
movl %edx, equal(,%ecx,4)
subl %ebx, %ecx
jmp multiplication_s
end_s:
incl %ebx
jmp multiplication_x

```

Listing 5: Blok kodu służący do mnożenia dwóch liczb w systemie binarnym. Dzięki niemu wyświetlany jest poprawny wynik. Poniżej rozpisano algorytm w postaci matematycznej.

$$A * B = C$$

$$\sum_x (a_x * B^x * B) = C$$

$$\sum_x [a_x * B^x (\sum_y b_y * B^y)] = C$$

$$\sum_x (\sum_y a_x * b_y * B^{x+y}) = C$$

Zerowanie rejestru EBX, będzie on iteratorem pętli zewnętrznej. Sprawdzenie czy nastąpił koniec drugiej liczby. Następnie zerowanie rejestru ECX, będzie on iteratorem pętli wewnętrznej. Rejestr ESI będzie przeniesieniem na pozycję "+2". Sprawdzenie czy pętla przeszła przez liczbę drugą. Wykonuje instrukcję mull EDX, która mnoży zawartość EAX i EDX. Zwraca 8 bajtów, gdzie wyższe zostają w EDX a niższe w EAX. Następnie dodajemy X + Y bez przeniesienia. I kopiuje do zmiennej wyniku. Następnie zaimplementowana jest obsługa przeniesienia. Zamiast uwzględniać przeniesienie co każdą wyższą liczbę, można zapamiętać czy na danej pozycji nastąpiło przeniesienie. Jeśli jest przeniesienie zostaje ono wyzerowane, w rejestrze ESI(przeniesienie z poprzedniej iteracji). Kopiujemy EDI do ESI, ponieważ przeniesienie przechowywane w ESI zostało wykorzystane. Kopiuje obliczoną pozycję na odpowiednie miejsce w wyniku.

```

#konwersja wyniku na ascii
ending:
movl $equal_len, %ecx
movl $equal, %ebx
movl $equal_write, %eax

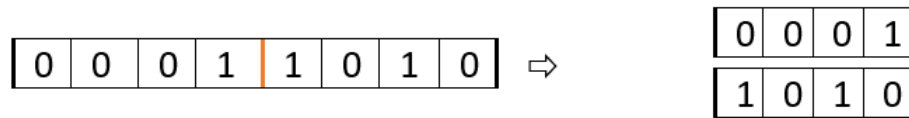
converter:
mov (%ebx), %al
movb %al, %bl
and $15, %al
cmppb $9, %al
jbe num_in_num
addb $55, %al
jmp second
num_in_num:
addb $48, %al

second:
shr $4, %bl
cmppb $9, %bl
jbe num_in_num_2
addb $55, %bl
jmp overwrite
num_in_num_2:
addb $48, %bl

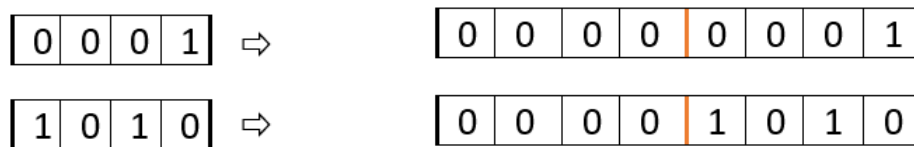
overwrite:
movb %al, (%eax)
incl %eax
movb %bl, (%eax)
incl %eax
addl $1, %ebx
subl $1, %ecx
cmpl $0, %ecx
tu:
je ending-program
jmp converter

```

Listing 6: Sekcja która kowertuje ciąg liczb binarnych na kod ASCII. Jest zapętłona dla każdego znaku.



Rysunek 4: Należy pobrać z ciągu liczby szesnastkowej po 8 bitów reprezentujące dwie liczby w systemie szesnastkowym.



Rysunek 5: Następnie każdą liczbę przesunąć o odpowiednią ilość nitów tak aby 4 bity znajdowały się na początku (najmłodszych 4 bitach)



Rysunek 6: Należy dodać odpowiednią stałą tak aby była ona w standardzie ASCII.

```
ending_program:
mov $SYSWRITE, %eax    # SYSWRITE (wypisanie danych na ekranie)
mov $STDOUT, %ebx      # argument pierwszy systemowy deskryptor stdout
mov $equal_write, %ecx  # argument drugi adres początku lancucha ascii
mov $equal_space_write, %edx # argument trzeci dlugosc lancucha ascii
int $SYSCALL32 # wywołanie fukcji

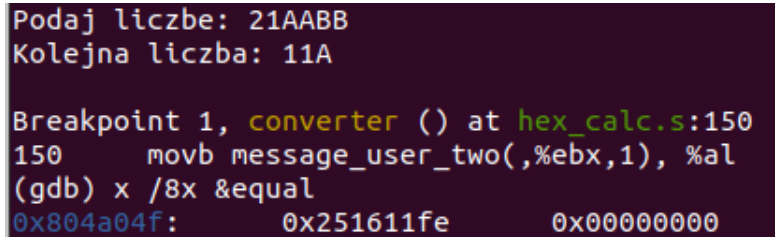
movl $SYSEXIT32, %eax
movl $EXIT_SUCCESS, %ebx
int $SYSCALL32
```

Listing 7: Ostatnia sekcja programu, która wypisuje wynik i kończy program.

3 Wnioski

W trakcie wykonywania tego programu na laboratorium miałem problemy z napisaniem parsera danych z kodu ascii do binarnego. Następnie wymagający okazał się algorytm obsługujący mnożenie wielkich liczb. Po wielu próbach napisania konwertera z kodu binarnego gotowego już wyniku w formacie szesnastkowym na kod ASCII okazał się największym problemem. Program po wglądzie w gdb mnoży liczbę prawidłowo jednak do pełni funkcjonalności brakuje mu obsługi wyświetlania wyniku na ekranie terminala. Mógłbym zmienić rodzaj algorytmu, który zmienia liczby binarne na kod ASCII jednak nie miałem pomysłu jak to zrobić (podany przeze mnie algorytm wyżej nie działa poprawnie), prawdopodobnie po spędzeniu większej ilości czasu możliwe by było napisanie poprawnego programu.

4 Przykład obliczenia i wyniku w GDB



```
Podaj liczbe: 21AABB
Kolejna liczba: 11A

Breakpoint 1, converter () at hex_calc.s:150
150      movb message_user_two(,%ebx,1), %al
(gdb) x /8x &equal
0x804a04f:      0x251611fe      0x00000000
```

Rysunek 7:

$$21AABB * 11A = 251611FE$$