Wojciech Adles, 209853 Wrocław, 24.05.17r

ŚR-N-15

Laboratorium Organizacji i Architektury Komputerów

Laboratorium 4:

Łącznie różnych języków programowania w jednym projekcie

1. Treść ćwiczenia

- Napisać program w języku Asemblera który wczyta za pomocą funkcji bibliotecznej scanf liczbę stałoprzecinkową, która posłuży na argument funkcji rekurencyjnej napisanej w języku C, liczącej n-ty wyraz ciągu zadanego wzorem $n_i=n_{i-1}+2n_{i-2}$. Funkcja ta zwróci wynik w formacie double, który trzeba wypisać na konsolę za pomocą funkcji printf **(2 pkt)**
- Napisać program w języku C który wywoła funkcję napisaną w języku Asemblera obliczającą wartość zadanej liczby w systemie pozycyjnym o wagach równych kolejnych wartości silni **(1,5 pkt)**
- Napisać wstawkę w języku Asemblera która konwertuje liczbę zapisaną w ciągu ASCII na liczbę w systemie o podstawie 9. **(1,5 pkt)**

2. Przebieg ćwiczenia

2.1 Wywołanie funkcji zewnętrznej napisanej w języku C

Na obecnym laboratorium nowym zagadnieniem było wywoływanie w kodzie asemblera funkcji zewnętrznej, napisanej w tym przypadku w języku C. Aby linker wiedział gdzie szukać danej funkcji, należy w sekcji *.data* zasygnalizować mu że jest ona zewnętrzna poprzez dyrektywę *.extern function.*

Poniżej znajduję się kod programu pierwszego napisanego w asemblerze, który będę omawiał w dalszej części sprawozdania.

```
.section .data

SYSEXIT = 60
formatd: .asciz "%d"
formatf: .asciz "%f \n"
decimal: .long 0
flt: .double 0
n1: .double 1.5
n2: .double 2.5

.extern f

.section .text
.globl _start
_start:
        movq $0, %rax                   #0 floating point arguments
        movq $formatd, %rdi             #load format string
        movq $decimal, %rsi             #set storage to address of x
        call scanf
        movq $2, %rax                   #2 floating point arguments
        movq $0, %rdi                   #clear rdi
        movl decimal, %edi
        movsd n1, %xmm0
        movsd n2, %xmm1
        call f
        movsd %xmm0, flt
        movq $1, %rax                   #one float number as argument
        movq $formatf, %rdi             #load format string
        call printf
        addq $8, %rsp
        movq $SYSEXIT, %rax
        syscall                         #Exiting program
```

Listing 1: lab4_1.s

Jak widać w powyższym kodzie, po pobraniu zadanej przez użytkownika liczby kroków funkcją *scanf* ładujemy ją do rejestru *%edi* po uprzednim jego wyczyszczeniu. Deklarujemy również 2 argumenty typu zmiennoprzecinkowego, kopiując wartość *$2* do rejestru *%rax.* Następnie inicjujemy rejestry *%xmm0* i *%xmm1* wartościami początkowymi danego ciągu za pomocą instrukcji *movsd* należącej do zestawu instrukcji SSE2, przeznaczonych między innymi do operacji na rejestrach XMM. Następnie wywołujemy zewnętrzną funkcję instrukcją *call*, która zwróci nam wynik typu double w młodszych 64 bitach rejestru *%xmm0*. Wynik ten kopiujemy do zmiennej *flt* i następnie wypisujemy na konsoli wywołaniem funkcji bibliotecznej *printf*. Poniżej znajduje się kod funkcji *f* napisanej w języku C.

```
double f(int steps, double n1, double n2);

double f(int steps, double n1, double n2){
        double result;

        result = n1 * n2;
        n1 = n2;
        n2 = result;
        steps--;
        if(steps > 0){result = f(steps, n1, n2);}

        return result;
}
```

Listing 2: func.c

Konsolidując plik wykonywalny użyłem następujących poleceń zawartych w pliku *Makefile*.

```
all: lab4_1

lab4: lab4_1.o func.o
        ld -o lab4_1 -dynamic-linker /lib64/ld-linux-x86-64.so.2 func.o
lab4_1.o -lc
lab4.o: lab4_1.s
        as -gstabs lab4_1.s -o lab4_1.o
func.o: func.c
        gcc -c func.c
```

Listing 3: Makefile

2.2 Program napisany w języku C z funkcją w języku Asemblera

Procedura przy wywoływaniu funkcji zewnętrznej napisanej w asemblerze z poziomu programu napisanego w języku C, nie rożni się od wywoływania zewnętrznej funkcji napisanej w C. Należy zadeklarować ją przed jej wywołaniem z dyrektywą *extern*, podobnie jak w przypadku z poprzedniego zadania. Kod krótkiego programu głównego znajduje się poniżej.

```c
#include <stdio.h>
#include <string.h>

extern int asmfunc(char *string, int len);

int main(int argc, char *argv[]){

        int result = 0;
        int len = strlen(argv[1]);

        result = asmfunc(argv[1], len);
        printf("Wartość tej liczby to: %d\n", result);

        return 0;
}
```

Listing 4: lab4_2.c

Program wywołuje się z jednym argumentem, który jest liczbą której wartość chcemy zdekodować za pomocą asemblerowej funkcji. Można zauważyć, że funkcja *asmfunc* przyjmuje dwa argumenty. Jednym z nich jest zadana liczba, a drugim jej długość. Ma to na celu uproszczenie kodu asemblerowego, w którym pojedyncze bajty ze znakami ASCII danej liczby kopiowane są określoną ilość razy (otrzymaną właśnie jako drugi parametr funkcji) do bufora instrukcjami *rep movsb*. Instrukcja *movsb* kopiuje pojedynczy bajt znaku ASCII do adresu znajdującego się w rejestrze *%rdi.* Połącznie tej instrukcji z instrukcją *ret* sprawia, że *movsb* wykona się zadaną w rejestrze *%rcx* ilość razy, automatycznie zwiększając (bądź zmniejszając) adres docelowy o wartość zależną od instrukcji kopiującej. Kierunek kopiowania jest ustalany stanem flagi kierunku (DF – direction flag), którą czyścimy uprzednio instrukcją *cld* przez co wartości adresu są zwiększane.

```
.section .bss

.comm asciibuff, 64

.section .text
.type asmfunc @function
.globl asmfunc
asmfunc:
        push %rbp
        push %r13
        push %r14
        push %r15
        movq %rsp, %rbp

        movq $0, %r14                           #Register for final value

        movq %rsi, %rcx                         #Lenght of the string to %rcx
        movq %rsi, %rdx                         #Save value of string length
        movq %rdi, %rsi                         #Copy pointer to %rsi

        leaq asciibuff, %rdi                    #Move address of buff %rdi
        cld
        rep movsb

        movq $1, %rcx                           #Value of initial digit
        movq %rcx, %rax                         #position to %rcx and %rax
        movq %rdx, %r13
factorial:
        push %rcx
        movq $0, %r15
        dec %r13                                #%r13 is offset for buff
        mov asciibuff(,%r13,1), %r15b           #Copy ascii sign to %r15b
        sub $'0', %r15b

mul:
        cmp $1, %rcx
        je end
        dec %rcx
        mulq %rcx
        jmp mul
end:
        mulq %r15
        addq %rax, %r14
        pop %rcx
        inc %rcx
        movq %rcx, %rax
        cmp $0, %r13
        jne factorial

        movq %r14, %rax

        movq %rbp, %rsp
        pop %r15
        pop %r14
        pop %r13
        pop %rbp
        ret
```

Listing 5: Lab4_2.s

Wracając jednak do początku kodu, odkładamy na stos rejestry których wartość jako element wywoływany musimy zachować, a z których chcemy korzystać w ciele funkcji. Wykonujemy opisaną wcześniej operację kopiowania i przechodzimy do właściwej części funkcji tj. obliczanie wartości danej liczby w zadanym systemie. Kopiujemy do rejestrów *%rcx* i *%rax* wartość *$1* która jest numerem pozycji cyfry której wartość aktualnie obliczany. Informacja zawarta w rejestrze *%rcx* będzie potrzebna przy obliczaniu wartości silni, gdzie podczas jej obliczania będzie dekrementowana, natomiast wartość w rejestrze *%rax* posłuży przy mnożeniu kolejnych liczb w silni.

Przechodząc zatem do etykiety *factorial*, odkładamy numer obliczanej aktualnie pozycji na stos, aby posłużył nam przy obliczaniu wartości cyfry na następnej pozycji i czyścimy rejestr %r15, do którego skopiujemy cyfrę z danej pozycji. Odejmujemy od niej odpowiednią wartość, konwertując ja z kodu ASCII na odpowiadającą jej wartość. W etykiecie mul mnożymy kolejne liczby tak aby otrzymać wartość silni. Gdy dojdziemy do końca jej obliczania przechodzimy do etykiety end. Tutaj mnożymy wartość obliczonej silni z wartością cyfry na danej pozycji i dodajemy wynik to sumy z poprzednich pozycji. Gdy zsumujemy wartości wszystkich liczb (wartość w *%r13* wyniesie 0) wychodzimy z pętli kopiując otrzymaną liczbę do rejestru *%rax*, przez który zostanie zwrócona jako wynik funkcji. Przywracamy zachowane wartości rejestrów i wykonujemy instrukcję powrotu *ret*. Po wyjściu z funkcji drukujemy jej wynik na konsoli i kończymy wykonywanie programu.

3. Wnioski

Podczas laboratorium udało mi się napisać działający fragment programu pierwszego wczytujący dane za pomocą funkcji scanf, dużo czasu zabrała mi poprawna konsolidacja programu, tak aby było możliwe używanie funkcji bibliotecznych języka C. Zaznajomiłem się również z zasadami działania rejestrów XMM i instrukcjami SEE. Dwa pierwsze programy dokończyłem w domu, ostatniego niestety nie udało mi się napisać.