Wrocław, 13 maja 2018

Marcin Kotas, 235098 WT-P-7.15

prowadząca: mgr Aleksandra Postawka

Laboratorium Architektury Komputerów
(4) Łączenie różnych języków programowania w jednym projekcie

1 Treść ćwiczenia

Zakres i program ćwiczenia:

Pierwszy program napisany w języku asemblera wywołuje funkcje napisane w języku C. Wczytanie argumentu całkowitoliczbowego i zmiennoprzecinkowego za pomocą funkcji bibliotecznej scanf, wywołanie funkcji napisanej w języku C:

```
double f(float x, int y);
   f(x, y) = x^2 + y^3
```

Dwukrotne wypisanie wyniku za pomocą pojedynczego wywołania funkcji printf `"%lf \n %lf"`.

Drugi program napisany w języku C zawiera wstawkę w języku asemblera, obliczającą wartość liczby danej za pomocą łańcucha znaków w reprezentacji siódemkowej, np. ”14352” i zapisanie wyniku w zmiennej globalnej w jęz. C. Można również przekazać długość.

Trzeci program napisany w języku C wywołuje funkcje napisane w języku asemblera. Funkcja obliczająca wartość liczby danej w zapisie tekstowym w reprezentacji, w której wagami pozycji są kolejne wartości silni (mniejsze wagi dla bardziej znaczących cyfr).

2 Program pierwszy

2.1 Sekcja danych

W sekcji danych umieszczone zostały zmienne definiujące argumenty funkcji `scanf` oraz `printf`.

```
    decimal: .asciz "%d"
    float: .asciz "%f"
    result: .asciz "%lf\n%lf\n"
.bss
    .comm num1, 8
    .comm num2, 8
```

2.2 Wywoływanie zewnętrznych funkcji

Do rejestru `rax` podawana jest liczba argumentów zmiennoprzecinkowych przekazywanych do funkcji. W rejestrach `rdi` i `rsi` znajdują się pierwsze dwa argumenty wywoływanej funkcji, zgodnie z konwencją wywoływania. Pierwsze wywołanie funkcji wczytuje liczbę zmiennoprzecinkową, a drugie liczbę całkowitą.
Podczas wywoływania funkcji z argumentami zmiennoprzecinkowymi przekazywane są one przez rejestry `xmm[0-7]`.
Funkcja w C ma postać
Wartość zmiennoprzecinkowa zwrócona przez funkcję znajduje się w rejestrze `xmm0`. Wartość ta jest kopiowana do rejestru `xmm1`, aby mogła zostać przekazana do `printf` dwa razy.
Zgodnie z `System V AMD64 ABI` (Interfejs binarny aplikacji 64 bitowych wykorzystywany m.in. przez Linux) stos przed wywołaniem funkcji powinien być zawsze wyrównany do 16 bajtów (adres stosu wielokrotnością 16). Z tego powodu przed wywołaniem funkcji `printf` następuje odjęcie 8 od `rsp` (po uruchomieniu programu stos przesunięty jest o 8 bajtów).
Po wykonaniu funkcji do stosu dodawane jest 8, aby przywrócić go do pierwotnego stanu.

```
    mov $0, %rax
    mov $float, %rdi
    mov $num1, %rsi
    call scanf

    mov $0, %rax
    mov $decimal, %rdi
    mov $num2, %rsi
    call scanf

    mov $1, %rax
    mov num2, %rdi
    movss num1, %xmm0
    call func
    # result in %xmm0
    movsd %xmm0, %xmm1

    # align stack to 16 bytes
    sub $8, %rsp

    mov $2, %rax
    mov $result, %rdi
    call printf

    add $8, %rsp
```

3 Program drugi

Kod w języku asemblera wstawiany jest w następującym formacie:

```
asm("instrukcje asemblerowe"
    : zwracane wartosci //opcjonalne
    : argumenty         //
    : uzywane rejestry  //
    );
```

3.1 Instrukcje

W instrukcjach asemblera do rejestrów należy odwoływać się z dwoma znakami `%%`. Do argumentów można odwoływać się za pomocą `%n`, gdzie `n` jest numerem argumentu. Rejestry numerowane są od 0 począwszy od wartości zwracanych (jeśli funkcja zwraca jedną wartość i przyjmuje dwa argumenty, to argument pierwszy będzie miał numer 1 itd.). Ten sposób odwoływania się do rejestrów jest potrzebny, jeśli nie sprecyzuje się, w których rejestrach mają być przechowywane wartości.

3.2 Używane rejestry

Zmienne przypisuje się do rejestrów w następującym formacie: `': "<rej>"(<zm>)'`, gdzie:

`<rej>` może być `'r'`, aby kompilator sam wybrał używany rejestr, lub symbolem rejestru (np. `'a'`, aby użyć rejestru `rax`).

Przed symbolem rejestru może być dodany modyfikator `'='`, który oznacza, że zmienna użyta będzie tylko do zapisu (zwracanie wyniku).

`<zm>` to zmienna zadeklarowana w programie. Jeżeli we wstawce asemblerowej używa się pełnych, 64 bitowych rejestrów, zmienna również powinna być 64 bitowa (typ long).

3.3 Kod

Wstawka asemblerowa traktuje ciąg znaków pod adresem `text` jako liczbę w reprezentacji siódemkowej i oblicza jej wartość dziesiętną.
Wynik przekazywany jest z rejestru `rax` do zmiennej `res`.
Adres ciągu znaków `text` przekazywany jest do rejestru `rdi`, a długość ciągu `length` do rejestru `rsi`.
Funkcje asemblerowe odwołują się do argumentów bezpośrednio przez zadeklarowane rejestry.

```
int length = 3;
long res;
char text[] = "123";
asm("mov $0, %%rax;\
    mov $0, %%r9;\
    mov $7, %%rcx;\
to_number:;\
    movb (%%rdi, %%r9, 1), %%bl;\
    cmp $'0', %%bl;\
    jl not_number;\
    cmp $'6', %%bl;\
    jg not_number;\
    sub $'0', %%bl;\
    mul %%rcx;\
    add %%rbx, %%rax;\
    inc %%r9;\
    cmp %%rsi, %%r9;\
    jl to_number;\
    jmp number;\
    not_number:;\
    mov $0, %%rax;\
    number:"
    :"=a"(res)
    :"D"(text), "S"(length)
);
printf("Result:%ld\n",res);
```

4 Program trzeci

Program wywołuje funkcję napisaną w jęz. asemblera, która oblicza wartość liczby danej w zapisie tekstowym w reprezentacji, w której wagami pozycji są kolejne wartości silni (mniejsze wagi dla bardziej znaczących cyfr).

4.1 Program główny

Na początku programu zadeklarowana zostaje funkcja zewnętrzna za pomocą słowa kluczowego `extern`. Podane muszą zostać typ zwracany przez funkcję, jej nazwa oraz typy argumentów przyjmowanych przez funkcję.

```
extern long func(char*, int);

int main(void)
{
    int length = 3;
    char text[] = "123";
    long res = func(text,length);
    printf("Result:%ld\n",res);
}
```

4.2 Funkcja w jęz. asemblera

Funkcja musi zostać zadeklarowana globalnie, co odbywa się na samym początku kodu.
Argumenty przekazane do funkcji znajdują się w `rdi(text)` oraz `rsi(length)`.
Funkcja pobiera kolejne znaki z tekstu, a następnie wymnaża przez aktualną wartość silni, począwszy od 1. Po wymnożeniu wartość dodawana jest do sumy w `rbx`, a wartość silni wymnażana jest przez wartość w rejestrze `r9`, który jest po tej operacji inkrementowany.
Pętla wykonuje się dopóki licznik w `rcx` nie przekroczy długości tekstu w `rsi`.
Na końcu suma z `rbx` kopiowana jest do `rax`, gdyż w tym rejestrze zwracany jest wynik funkcji. Jeżeli na którejś pozycji w tekście wystąpił znak inny niż cyfra, zwracany jest wynik `-1`.

```
.text
.global func
.type func, @function
func:
    mov $0, %rbx # suma
    mov $0, %rcx # iter tekstu
    mov $1, %r8  # silnia
    mov $2, %r9  # mnoznik silni
convert:
    mov $0, %rax # bufor
    movb (%rdi, %rcx, 1), %al
    cmp $'0', %al
    jl not_number
    cmp $'9', %al
    jg not_number
    sub $'0', %al
    mul %r8
    add %rax, %rbx

    mov %r8, %rax
    mul %r9
    mov %rax, %r8
    inc %r9
    inc %rcx
    cmp %rsi, %rcx
    jl convert
    mov %rbx, %rax
ret
not_number:
    mov $-1, %rax
ret
```

5 Wnioski

Wszystkie programy uruchomiły się poprawnie. W programie drugim zaszła potrzeba wyrównania stosu do 16 bajtów podczas wywoływania funkcji `printf` z argumentami zmiennoprzecinkowymi. Mimo iż zgodnie z `System V AMD64 ABI` stos musi zawsze być wyrównany przed wywoływaniem jakiejkolwiek funkcji, często nie powoduje to błędu.