Wrocław, 13 maja 2018

Marcin Kotas, 235098 WT-P-7.15

prowadząca: mgr Aleksandra Postawka

Laboratorium Architektury Komputerów
(3) Funkcje

1 Treść ćwiczenia

1.1 Program pierwszy:

Pierwszy program zawierał funkcję zwracającą indeks pierwszego wystąpienia ciągu ‘abc’ w łańcuchu znaków — argumenty i wynik przekazywane w dowolny sposób.

1.2 Program drugi:

Program składał się z funkcji rekurencyjnej

$$f(n) = f(n-2) - 5f(n-1)$$
$$f(0) = 3$$
$$f(1) = 1$$

Argumenty i wynik przekazywane przez stos.

1.3 Program trzeci

Trzeci program zawierał implementację tej samej funkcji rekurencyjnej przekazującej argumenty przez rejestry.

2 Kod programu pierwszego

2.1 Wywoływanie funkcji

Funkcja przyjmuje argumenty według standardowej konwencji wywoływania. Pierwszy argument to długość ciągu znaków, a drugi to adres bufora zawierającego tekst. Funkcja zwraca wynik w rejestrze `rax`. Jeżeli nie znaleziono szukanego ciągu znaków, zwraca '-1'.

```
    mov %r8, %rdi
    mov $textin, %rsi
    call find_abc

    cmp $-1, %rax
```

2.2 Implementacja funkcji

Na początku funkcja zeruje licznik w rejestrze `rdx`. W pętli `loop` następuje porównywanie kolejnych znaków aż wykryta zostanie litera 'a'. Jeśli zostanie znaleziona, to następuje sprawdzenie, czy po niej następuje litera 'b'. Jeżeli nie, to następuje powrót na początek (bez inkrementacji licznika, aby uniknąć błędu w przypadku ciągu 'aabc'). Jeżeli wystąpiła litera 'b', następuje inkrementacja licznika i sprawdzenie czy następną literą jest 'c'. Jeśli tak, funkcja wychodzi z pętli i zwraca licznik obniżony o 3 (wskaźnik na 'a') w rejestrze `rax`. Jeżeli licznik przekroczy wartość w rejestrze `rdi` (długość tekstu podana jako argument funkcji), to szukana sekwencja znaków nie została odnaleziona i funkcja zwraca '-1'.

```
find_abc:
    mov $0, %rdx
loop:
    cmp %rdi, %rdx
    jge not_found
    mov (%rsi, %rdx, 1), %bl
    inc %rdx
    cmp $'a', %bl
    jne loop

    cmp %rdi, %rdx
    jge not_found
    mov (%rsi, %rdx, 1), %bl
    cmp $'b', %bl
    jne loop
    inc %rdx

    cmp %rdi, %rdx
    jge not_found
    mov (%rsi, %rdx, 1), %bl
    inc %rdx
    cmp $'c', %bl
    jne loop

    sub $3, %rdx
    mov %rdx, %rax
ret

    not_found:
    mov $-1, %rax
ret
```

3 Kod programu drugiego

3.1 Wywoływanie funkcji

Funkcja przyjmuje argumenty przez stos. Jedyny argument, jaki przyjmuje to 'n'. Wynik również zwracany jest przez stos.

```
    push %rax
    call func
    pop %rax
```

3.2 Implementacja funkcji

Funkcja na początku odkłada na stosie obecną wartość rejestru `rbp`, po czym kopiuje wartość `rsp` do `rbp`. Następnie odejmuje od `rsp` 16 bajtów, aby zrobić miejsce na lokalne zmienne. W ten sposób następne wywołanie funkcji przekaże na stos adres powrotu poniżej zmiennych lokalnych. Procedura ta nazywana jest prologiem funkcji.
Następnie funkcja przenosi argument (znajdujący się nad adresem powrotu — `16(%rbp)`) do adresu zarezerwowanego na zmienne lokalne (`-8(%rbp)`). Od tego momentu wszystkie operacje związane z 'n' odbywać się będą względem tego adresu. Najpierw sprawdzane jest, czy 'n' nie jest równe '0' lub '1'. Jeśli tak, to funkcja zwraca odpowiednio '3' lub '1'.
Jeśli nie, to następuje dekrementacja i odłożenie na stos 'n' (`-8(%rbp)`), a następnie wywołanie funkcji. Wynik zwrócony przez funkcję $f(n-1)$ jest kopiowany do rejestru `rax` i wymnażany przez 5. Następnie wynik mnożenia kopiowany jest do `-16(%rbp)`, aby nie został nadpisany przez kolejne wywołanie funkcji.
Wartość 'n' zostaje kolejny raz dekrementowana w celu wywołania drugi raz funkcji rekurencyjnej. Wynik $f(n-2)$ kopiowany jest do rejestru `rbx`, a następnie odejmowana jest od niego wartość zapisana w `-16(%rbp)`.
Wartość zwracana przez funkcję (1, 3, lub $f(n-2) - 5f(n-1)$) kopiowana jest do `16(%rbp)`. Dzięki temu wartość zwracana znajduje się bezpośrednio nad adresem powrotu, więc po wyjściu z funkcji można wykonać na stosie operację 'pop' i uzyskać wynik działania funkcji.
Na końcu funkcji następuje epilog — przywrócenie poprawnej wartości `rsp` oraz pobranie ze stosu wcześniejszej wartości `rbp`.

```
func:
    push %rbp
    mov %rsp, %rbp
    sub $16, %rsp
    mov 16(%rbp), %rbx
    mov %rbx, -8(%rbp)
    # -8(%rbp) = n
        cmp $0, -8(%rbp)
        je is0
        cmp $1, -8(%rbp)
        je is1

        decq -8(%rbp)
        push -8(%rbp)
        call func # f(n-1)
        pop %rax
        mov $5, %rbx
        mul %rbx
        mov %rax, -16(%rbp)
        # -16(%rbp) = 5f(n-1)

        decq -8(%rbp)
        push -8(%rbp)
        call func
        pop %rbx
        # %rbx = f(n-2)

        sub  -16(%rbp), %rbx
        mov %rbx, 16(%rbp)
        jmp end
    is0:
        movq $3, 16(%rbp)
        jmp end
    is1:
        movq $1, 16(%rbp)
    end:
    mov %rbp, %rsp
    pop %rbp
ret
```

4 Kod programu trzeciego

4.1 Wywoływanie funkcji

Funkcja przyjmuje argument i zwraca wynik przez rejestry — odpowiednio `r8` i `r9`.

```
    movq %rax, %r8
    call func
    mov %r9, %rax
```

4.2 Implementacja funkcji

Funkcja przekazuje argumenty przez rejestry, więc musi na początku skopiować ich poprzednią wartość, aby móc ją przywrócić przed zakończeniem. W funkcji wykorzystywany jest również rejestr `r10` do przechowywania wyniku zwróconego przez funkcję $f(n-1)$. Z tego powodu funkcja na początku kopiuje wartości rejestrów `r8` oraz `r10` do pamięci lokalnej (zarezerwowanej przez przesunięcie `rsp`).
Dalej funkcja działa na tej samej zasadzie, co w programie 2.
Na końcu funkcji przywracane są poprzednie wartości `r8` oraz `r10`, po czym następuje normalny epilog funkcji. Wynik znajduje się w rejestrze `r9`.

```
func:
    push %rbp
    mov %rsp, %rbp
    sub $16, %rsp
    mov %r8, -8(%rbp)
    mov %r10, -16(%rbp)
        cmp $0, %r8
        je is0
        cmp $1, %r8
        je is1

        dec %r8
        call func
        mov $5, %rax
        mul %r9
        mov %rax, %r10
        # %r10 = 5f(n-1)

        dec %r8
        call func
        # %r9 = f(n-2)

        sub %r10, %r9
        jmp end
    is0:
        movq $3, %r9
        jmp end
    is1:
        movq $1, %r9
    end:
    mov -8(%rbp), %r8
    mov -16(%rbp), %r10
    mov %rbp, %rsp
    pop %rbp
ret
```

5 Wnioski

Wszystkie programy uruchomiły się poprawnie. W programie drugim nie trzeba było kopiować ‘n’ do `-8(%rbp)`, tylko odnosić się do ‘n’ zawsze przez `16(%rbp)`. W ten sposób od `rsp` wystarczyłoby odejmować 8 bajtów na przechowanie wyniku $5f(n-2)$.
W programie trzecim warto dostosować wywoływanie do konwencji — argument w `rdi`, a wynik w `rax`. Nie istnieje również potrzeba modyfikowania wartości `rsp`, gdyż można odkładać używane rejestry na stos przed wywołaniem funkcji. Modyfikacje te zwiększyłyby czytelność kodu.