Raport Laboratorium OiAK 2 Zajęcia nr 1

Data wykonania	20.03.2021
Termin zajęć	Czwartek TP 17:05
Autor	Mateusz Kusiak Indeks:252805

1 Treść zadania

Program w języku assemblera w architekturze 32-bitowej na platformę Linux. Program powinien zapytać użytkownika o podanie zdania, wczytać je ze standardowego wejścia, a następnie wyświetlić na standardowe wyjście to samo zdanie zaszyfrowane za pomocą ROT13.

1.1 Przykład wywołania

```
$ ./rot13
Podaj zdanie: Ala ma kota
ROT13: Nyn zn xbgn
```

2 Objaśnienie

Poniżej znajduje się kod wraz z objaśnieniami:

```
SYSEXIT32 = 1
SYSCALL32 = 0x80
SYSREAD = 3
EXIT_SUCCESS = 0
SYSOPEN = 5
SYSWRITE = 4
STDOUT = 1
STDIN = 0
```

Listing 1: Przypisane numerów funkcji do wywołania potrzebnych w programie

```
.data
message_first: .ascii "Podaj zdanie: "  #zmienne potrzebne do zadania
message_first_length = . - message_first #dlugosc zmiennej
message_endl: .ascii "\n" #zmienna konca linii
message_endl_length = . - message_endl  #zmienna dlugosci konca linii
message_rot: .ascii "Rot13: "
message_rot_length = . - message_rot
message_user: .ascii "                "
message_user_length = . - message_user
```

Listing 2: Przypisane zmiennych w pamięci potrzebnych w programie

```
.global _start

_start:

mov $SYSWRITE, %eax      # funkcja do wywolania - SYSWRITE (wypisanie danych na ekranie)
mov $STDOUT, %ebx        # argument pierwszy systemowy deskryptor stdout
mov $message_first, %ecx         # argument drugi dres poczatku lancucha ascii
mov $message_first_length, %edx # argument trzeci dlugosc lancucha ascii
```

```
int $SYSCALL32  # wywolanie fukcji

movl $SYSREAD, %eax      # funkcja do wywolania - SYSREAD (pobranie danych)
movl $STDIN, %ebx        # argument pierwszy systemowy deskryptor stdin
movl $message_user, %ecx         # argument drugi adres poczatku lancucha ascii
movl $message_user_length, %edx # argument trzeci dlugosc lancucha ascii
int $SYSCALL32  # wywolanie fukcji
```

Listing 3: Wywołanie funkcji systemowych pokazujących Na ekranie napis "Podaj zdanie:" oraz pobranie wpisanego zdania od użytkownika

```
# licznik petli dla kazdego znaku z napisu wprowadzonego przez uzytkownika
movl $message_user, %eax
movl $message_user_length, %ecx
movl $0, %ebx
```

Listing 4: Przetworzenie zdania wejściowego odbywa się dla każdego znaku ACII oddzielnie dlatego potrzebna jest pętla która wykonuje się tyle razy ile znaków ma dany łańcuch.

```
# petla przeksztalcajaca lancuch znakow na ROT13

begin:  #flaga poczatku
cmpl $0, %ecx # warunek porownujacy czy sa jeszcze jakies znaki do przeksztalcenia
je ending # jezeli nie przejdz do konca programu
mov (%eax), %bl #kopiujemy jeden znak z rejestru eax do 8 bitowego bl
cmpb $'A', %bl  # Warunki sprawdzajace czy dany znak jest litera kod ASCII
jb add_adress   # Jezeli mniejsza wartosc od 65 to przeskocz do kolejnego
cmpb $'z', %bl  # porownaj znak z wartoscia 122='z'
ja add_adress # Jezeli wieksza wartosc od 122 przeskocz do kolejnego
cmpb $'a', %bl # warunek sprawdzajacy czy to mala litera
jae low_letter # jezeli wartosc wieksza lub rowna 'a'=97 przeskocz do low_letter

#przypadek wielkich liter
big_letter:
addb $13, %bl   #dodanie stalej 13 do kodu ascii
cmpb $'Z', %bl  #porownanie czy znak nie wyszedl ponad zakres wielkich liter
jbe add_adress  #jezeli nie przejdz do kolejnego znaku
subb $26, %bl   #w innym przypadku odejmij 26
jmp add_adress #przejdz do kolejnej litery

#przypadek malych liter
low_letter:
addb $13, %bl #dodanie stalej 13 do kodu ascii np. 97->110 = 'a'->'n'
cmpb $'z', %bl #porownanie czy znak nie wyszedl ponad zakres malych liter
jbe add_adress #jezeli nie przejdz do kolejnego znaku
subb $26, %bl #w innym przypadku odejmij 26

#funkcja dodajaca adres
```

```
add_adress:
movb %bl, (%eax) #aktualizacja znaku w pamieci na rot13
addl $1, %eax    #przeskoczenie do kolejnego znaku w lancuchu
subl $1, %ecx    #iteracja licznika petli
jmp begin
```

Listing 5: Kod zawarty w pętli który odpowiada za konwersję znaków na szyfr Rot13

Cała logika programu polega na pobieraniu pojedynczego znaku a następnie dzieki kodowi ASCII sprawdzanie warunków czy dany znak to litera oraz rozróżnianie czy jest to mały czy duży znak. Dzięki instrukcji warunkowych wykonujących skok możliwe jest porównywanie wartości. Graniczne wartości dla znaków łacińskiego alfabetu to 'A' = 65 do 'z' = 122. Co umożliwia porównanie liter jako wartości liczbowych a następnie po przez dodanie stałej 13 konwersję na kod Rot13 jeżeli dana litera wychodzi poza swój zakres tzn dla małych liter jest to 122, natomiast dla wielkich 90. W celu przejrzystości tego raportu każda linia w listingu wyżej jest komentowana i opisuje dokładnie dane przeskoki i warunki które są potrzebne do wykonania danego programu.

```
nding:
#wypisanie wterminalu napisu "Rot13:"
movl $SYSWRITE, %eax
movl $STDOUT, %ebx
movl $message_rot, %ecx
movl $message_rot_length, %edx
int $SYSCALL32

#wypisanie w terminalu napisu uzytkownika w kodzie rot13
movl $SYSWRITE, %eax
movl $STDOUT, %ebx
movl $message_user, %ecx
movl $message_user_length, %edx
int $SYSCALL32

#wypisanie w terminalu '\n'
movl $SYSWRITE, %eax
movl $STDOUT, %ebx
movl $message_endl, %ecx
movl $message_endl_length, %edx
int $SYSCALL32

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

Listing 6: Jest to zakończenie programu które wypisuje przekształcone zdanie w kodzie Rot13 a nastepnie wykonuje operację zakończającą program w języku assembler

3 Wnioski

W trakcie wykonywania tego programu na laboratorium dowiedziałem się jak poprawnie używać instrukcji warunkowych wykonujących skok. Bardzo ważene jest sprawdzenie poprawności wpisywanego kodu. Miałem problem z napisem który miał być już wyświetlony jako szyfr w Rot13 jednak na wyjściu w terminalu nadal widniał jako oryginalne zdanie wprowadzone przez użytkownika. Błędem okazało sie brak odpowiedniej operacji wykonującej zskopiowanie przetworzonego znaku do miejsca w panięci w której przechowywane jest zdanie użytkownika. Jednak po dodaniu tej instrukcji program działał poprawnie.