Raport Laboratorium 1

Organizacja i Architektura komputerów

Autor: Uladzimir Kaviaka (257276)

Cel laboratorium:

Zadanie polegało na napisaniu algorytmu które szyfruje wpisany tekst za pomocą algorytmu szyfrowania ROT13. Przy tym przyjęto następujące założenia:

- Polskie litery pozostają bez zmian (input: żół, output: żół)
- Algorytm zamienia wielkie litery na wielkie, małe na małe
- Ograniczenie wpisywanego tekstu 100 symboli.

Opis algorytmu:

Szyfrowanie ROT13 polega na tym, że mamy np. literę A która w kodzie ASCII jest przedstawiana kodem 65. ROT13 dodaje do tej wartości wartość 13, więc zaszyfrowana litera A to jest N (ascii:78)

Szyfrowanie zamienia litery na litery, czyli nie może w wyniku pojawić się symbol który nie jest literą.

Implementacja programowa:

1. Rezerwacja pamięci pod zmienną tekstową która będzie szyfrowana

```
13  .bss
14  input: .space input_length #zarezerwowanie w pamieci ciagu pustego o dlugosci 100
15
16  .data
17  msg_start: .ascii "Podaj zdanie: "
18  msg_start_length = . - msg_start
19
20  msg_rot13: .ascii "Rot13: "
21  msg_rot13_length = . - msg_rot13
```

Wydzielenie pamięci pod zmienną tekstową o długości 100 linijka 14 Definicja zmiennych tekstowych oraz ich długości

2. Wypisanie tekstu początkowego: Podaj zdanie o wczytanie tekstu Wczytywanie tekstu

```
25  _start:
26  mov $SYSWRITE, %eax
27  mov $STDOUT, %ebx
28  mov $msg_start, %ecx
29  mov $msg_start_length, %edx
30  int $SYSCALL
31
32  mov $SYSREAD, %eax
33  mov $STDIN, %ebx
34  mov $input, %ecx
35  mov $input_length, %edx
36  int $SYSCALL
```

26-30 linijka: Wypisanie tekstu „Podaj zdanie”
32-36 linijka: Wczytywanie danych z terminalu

3. Wyliczanie długości tekstu i skopiowanie długości do rejestru

```
38   mov %eax, %edi
39   dec %edi
40   xor %ebp, %ebp
```

linijka 38 – w rejestrze eax po SYSREAD znajduje się długość wczytanego tekstu więc kopiujemy do rejestru edi.
Linijka 39 – dekrementacja edi ponieważ po SYSREAD zostaje znak nowej linii, poprzez dekrementację usuwamy go i dostaniemy rzeczywistą długość tekstu.
Linijka 40 – zerowanie rejestru ebp, który będzie służył jako indeks w tekście.

4. Sprawdzanie czy litera znajduje się w zakresie dużych liter

```
42   petla:
43
44   cmpl $0, %edi
45   jz done
46
47   cmpb $65, input(%ebp)
48   jb not_letter
49
50   cmpb $90, input(%ebp)
51   ja not_big_letter
52
53   not_big_letter:
```

linijka 44 – porównanie zawartości rejestru edi, gdzie jest przechowywana długość wczytanego tekstu z zerem.
Linijka 45 – jeżeli zero to wykonuje skok
Linijka 47 – porównanie symbolu z kodem w ASCII 65 (początek dużych liter) jeżeli jest mniejsza znaczy to nie litera, skok
Linijka 50 – porównanie symbolu z kodem w ASCII 90 (koniec dużych liter) jeżeli większa znaczy to nie jest duża litera wykonujemy skok

5. Sprawdzanie czy litera znajduje się w zakresie małych liter. Kodowanie uwzględniając pożyczkę.

```
53   not_big_letter:
54
55
56   cmpb $122, input(%ebp)
57   ja not_letter
58
59   addb $0x0D, input(%ebp)
60   cmpb $90, input(%ebp)
61   jbe no_correction
62   cmpb $122, input(%ebp)
63   jbe no_correction
64
65   subb $26, input(%ebp)
66
67   no_correction:
```

Linijka 56 – sprawdzanie symbolu, jeżeli jest większy niż 122 (koniec małych liter) to skok
Linijka 59 – tu będzie algorytm wykonywany dla symboli które są w zakresie małych/dużych liter, dodajemy 0x0D(13) do symbolu na danej pozycji
Linijka 60 – sprawdzamy symbol z 90, jeżeli jest mniejsza lub równa to nie trzeba edytować (uwzględniać pożyczkę)
Linijka 122 – sprawdzamy symbol z 122, jeżeli jest mniejszy lub równy to też nie uwzględniamy pożyczkę.
Linijka 65 – edytowanie pozycji (uwzględnienie pożyczki) np. dla symbolu ‘o’ ASCII = 111, jeżeli dodamy 13 wynik będzie 124 ASCII co nie jest literą więc odejmujemy od tego 26(ilość liczb małych/dużych)

6. Warunek pętli

```
67   no_correction:
68
69   not_letter:
70   inc %ebp
71
72   cmp %edi, %ebp
73   jl petla
```

Jeżeli nie ma przeniesienia lub symbol nie jest literą możemy spokojnie przechodzić do następnej pozycji poprzez inkrementację rejestru ebp (linijka 70)
Linijka 72 – porównuje zawartość rejestru edi gdzie jest przechowywana długość tekstu, oraz ebp(aktualna pozycja)
Linijka 73 – wykonuje skok jeżeli ebp jest mniejsze niż edi

7. Wypisanie wyniku końcowego i zatrzymanie programu

```
75   mov %eax, %edx
76   mov $SYSWRITE, %eax
77   mov $STDOUT, %ebx
78   mov $input, %ecx
79   int $SYSCALL
80
81   done:
82   mov $SYSEXIT32, %eax
83   mov $SUCCESSEDEXIT, %ebx
84   int $SYSCALL
85
```

Linijka 75 – wpisanie zawartośći eax(długość tekstu po SYSREAD) do rejestru edx który jest odpowiedzialny za długość wypisywanego tekstu
Wypisanie wyniku szyfrowania i wyjście z programu

Wnioski:
Ćwiczenie pozwoliło zrozumieć zasady działania instrukcji warunkowych, wykonanie skoków oraz jak realizować pętle w języku assemblera. Pozwoliło zrozumieć jak zwracać się do komórek pamięci i rejestrów oraz robić operacje na tych komórkach/rejestrach.