# Laboratorium Architektury Komputerów 2

Prowadzący: Mgr inż. Przemysław Świercz

Autor: Bartosz Szymczak, nr indeksu 252734

## Temat: Implementacja ROT13 w języku assemblera.

Zadaniem na pierwszym laboratorium było napisanie programu 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.

Założeniami powyższego programu są: maksymalna wielkość bufora równa 100 znakom, poprawność zamieniania małych liter na małe, dużych liter na duże oraz pozostawianie znaków niebędących literami alfabetu łacińskiego bez zmian.

## Użycie wczytywania wejścia i wyświetlania na ekranie.

W programie wykorzystałem bufor do wczytywania wejścia. Bufor umieściłem w segmencie danych niezainicjowanych.

*.section .bss*

*buff: .space BUFF_LEN*

*.lcomm my_buffer, 100*

Kompilator stworzy w segmencie .bss bufor o nazwie my_buffer i zarezerwuje dla niego 100 bajtów.

Następnie wykorzystałem wywołanie funkcji „read”, której argumentami jest systemowy deskryptor, adres oraz długość bufora.

*mov $SYSREAD, %eax*

*mov $STDIN, %ebx*

*mov $my_buffer, %ecx*

*mov $BUFF_LEN, %edx*

*int $SYSCALL32*

Po asemblacji i linkowaniu, proces czeka na podanie wejściowego łańcucha danych. Po otrzymaniu, zapisuje go w podanym adresie.

Do wyświetlania wiadomości na ekranie użyłem wywołania funkcji write, której argumentami jest systemowy deskryptor, adres wiadomości oraz jej długość.

*mov $SYSWRITE, %eax*

*mov $STDOUT, %ebx*

*mov $msg1, %ecx*

*mov $msg1_len, %edx*

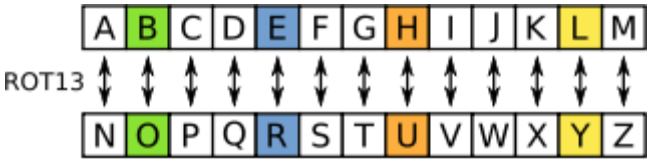
*int $SYSCALL32*

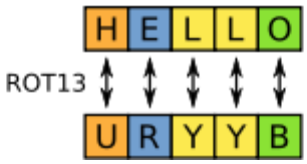
Po asemblacji i linkowaniu, proces wyświetla na ekranie wiadomość z podanego adresu.

Każdą funkcję systemową zakończyłem wywołaniem przerwania systemowego.

## Implementacja algorytmu ROT13

ROT13 to prosty sposób szyfrowania informacji, polegający na zamianie każdego znaku alfabetu łacińskiego na znak występujący 13 pozycji po nim. Jest to przykład szyfru Cezara, który używany był w Starożytnym Rzymie.




W trakcie implementacji należy pamiętać o tym w jakim systemie koduje się znaki we współczesnych komputerach. Jest to kod ASCII. W podanym kodzie litery od A do Z mają kody od 65 do 90 (bez polskich znaków), a litery od a do z mają kody od 97 do 122. W celu sprawdzenia, czy dany znak jest literą alfabetu łacińskiego należy sprawdzić czy kod znaku znajduje się pomiędzy powyższymi wartościami. Znaki ASCII kodowane są w jednym bajcie.

W programie oparłem się na pętli, która początkowo sprawdza czy dany znak jest dużą literą. Jeśli jest to zwiększamy jej wartość o 13. Po tej operacji należy sprawdzić czy zwiększona wartość dalej jest dużą literą. Jeśli nie jest, należy odjąć od niej 26 w celu otrzymania poprawnej wartości (alfabet łaciński ma 26 liter). W przypadku, gdy znak jest mniejszy od A to na pewno nie jest literą. Litera o kodzie większym niż Z może natomiast być małą literą. Dlatego jeżeli okaże się że znak jest większy niż Z, program sprawdzi czy znak jest większy niż a oraz mniejszy niż z. Program powtórzy operacje arytmetyczne także dla małych liter. Znak niebędący literą alfabetu łacińskiego zostanie przekazany bez zmian.

Licznikiem pętli jest rejestr ecx. Bufor jest przekazany do eax. Jeśli w ecx znajduje się wartość 0, pętla się kończy. Każde wykonanie pętli kończy się zmniejszaniem licznika o jeden oraz zwiększaniem wartości rejestru eax. W celu analizy każdego znaku, pobieram bajt z eax. Znak przekazuję do rejestru bl oraz dokonuję kolejnych sprawdzeń opisanych powyżej. Używam do tego odpowiednich komend porównań (cmpl dla liczb całkowitych, cmpb dla bajtów), skoków (jmp dla skoków bezwarunkowych, ja – skok jeśli większe (bez znaku), jb - skok jeśli mniejsze (bez znaku), jbe – skok jeśli mniejsze lub równe (bez znaku). Znak w rejestrze jest porównywany do wartości kodów danych liter alfabetu łacińskiego. Operacje arytmetyczne wykonałem komendami addb oraz subb, ponieważ operujemy na

bajtach. W przypadku jeśli znak spełnia określone kryteria, program wykonuje skok do danych etykiet: loop2 – sprawdzenie czy znak jest małą literą alfabetu łacińskiego, okay – zapisanie poprawnego znaku, ignore – iteracja głównej pętli, end – zakończenie pętli.

## Kod źródłowy

```
SYSEXIT32 = 1

EXIT_SUCCES = 0

SYSCALL32 = 0x80

SYSWRITE = 4

SYSREAD = 3

BUFF_LEN = 100

STDIN = 0

STDOUT = 1


.global _start              #wskazanie punktu wejścia do programu


.section .bss               #segment danych niezarejestrowanych

buff: .space BUFF_LEN       #inicjowanie buforów

.lcomm my_buffer, 100



.text

msg1: .ascii "Podaj zdanie: "  #etykieta napisu wraz z dyrektywą rezerwującą pamięć dla napisu

msg1_len = .-msg1           #długość napisu

msg2: .ascii "ROT13: "

msg2_len = .-msg2


_start:

#wypisanie napisu msg1

mov $SYSWRITE, %eax         #wywołanie funkcji write

mov $STDOUT, %ebx           #pierwszy argument funkcji, będący deskryptorem stdout
```

```
mov $msg1, %ecx              #drugi argument funkcji, będący adresem początkowym napisu
mov $msg1_len, %edx          #trzeci argument funkcji, będący długością łańcucha
int $SYSCALL32               #wywołanie przerwania systemowego

#wczytanie napisu do buforu
mov $SYSREAD, %eax           #wywołanie funkcji read
mov $STDIN, %ebx             #pierwszy argument funkcji, będący deskryptorem stdin
mov $my_buffer, %ecx         #drugi argument funkcji, będący adresem bufora
mov $BUFF_LEN, %edx          #trzeci argument funkcji, będący długością bufora
int $SYSCALL32

mov $my_buffer, %eax         #kopiowanie adresu bufora do akumulatora
mov $BUFF_LEN, %ecx          #kopiowanie długości bufora do ecx

#główna pętla - sprawdzenie czy znak jest dużą literą
loop1:
cmpl $0, %ecx                #sprawdzenie czy ecx jest większy od 0
jz end                       #jeśli jest zerem to skok do end
mov (%eax), %bl              #kopiowanie do bl wartości z eax
cmpb $'A', %bl               #porównanie bl do wartości znaku A
jb ignore                    #jeśli jest mniejszy to skok do ignore
cmpb $'Z', %bl               #porównanie bl do wartości znaku Z
ja loop2                     #jeśli jest większy to skok do drugiej pętli
addb $13, %bl                #dodanie wartości 13 do bl
cmpb $'Z',%bl                #porównanie bl do wartości znaku Z
jbe okay                     #jeśli jest mniejszy lub równy to skok do okay
subb $26, %bl                #odjęcie wartości 26 od bl
jmp okay                     #skok do okay

#sprawdzenie czy znak jest małą literą
loop2:
```

```
cmpb $'a',%bl              #porównanie bl do wartości znaku a
jb ignore                  #jeśli jest mniejszy to skok do ignore
cmpb $'z',%bl              #porównanie bl do wartości znaku z
ja ignore                  #jeśli jest większy to skok do ignore
addb $13, %bl              #dodanie wartości 13 do bl
cmpb $'z',%bl              #porównanie bl do wartości znaku z
jbe okay                   #jeśli jest mniejszy lub równy to skok do okay
subb $26, %bl              #odjęcie wartości 26 od bl

#zapisanie sprawdzonego znaku do rejestru
okay:
movb %bl, (%eax)           #kopiowanie bl do eax

#iteracja głównej pętli
ignore:
incl %eax                  #zwiększenie eax
decl %ecx                  #zmniejszenie ecx
jmp loop1                  #skok do pętli głównej

#koniec pętli
end:

#wypisanie napisu msg2
mov $SYSWRITE, %eax
mov $STDOUT, %ebx
mov $msg2, %ecx
mov $msg2_len, %edx
int $SYSCALL32

#wypisanie zmodyfikowanego bufora
mov $SYSWRITE, %eax
```

```
mov $STDOUT, %ebx
mov $my_buffer, %ecx
mov $BUFF_LEN, %edx
int $SYSCALL32

#zakończenie programu
mov $SYSEXIT32, %eax          #wywołanie funkci sysexit
mov $EXIT_SUCCES, %ebx        #argument funkcji, będący kodem wyjścia programu
int $SYSCALL32
```

## Źródła


https://pl.wikipedia.org/wiki/ROT13

http://jedrzej.ulasiewicz.staff.iiar.pwr.wroc.pl/Architektura-Komputerow/lab/Architektura-92.pdf?fbclid=IwAR0YgC3llqrhyZq0AhrX_m4nsbTVcTV0vc2QAXyiPqkVcuErlPXrI_zhIhw

https://pl.wikipedia.org/wiki/ASCII

https://pl.wikibooks.org/wiki/Asembler_x86/Instrukcje/Skokowe

https://chromium.googlesource.com/chromiumos/docs/+/HEAD/constants/syscalls.md#x86-32_bit