Wrocław, dn. 9 marca 2017

Hanna Grodzicka, 226154

Pętle, podstawowe operacje logiczne i arytmetyczne

sprawozdanie z laboratorium przedmiotu „Architektura Komputerów 2”

Rok. akad 2016/2017, kierunek: INF

Prowadzący:

mgr inż. Aleksandra Postawka

1. Cel ćwiczenia

Celem ćwiczenia było napisanie szyfru Cezara. Program powinien wczytać podany przez użytkownika klucz szyfrujący, a następnie ciąg znaków. Dla wielkich liter i cyfr z ciągu wczytanych znaków powinien zostać dodany klucz. Następnie program powinien wypisać zaszyfrowane dane.
Ostatnie litery z alfabetu po dodaniu klucza, powinny przechodzić na początek alfabetu. Dla cyfr analogicznie - przejście przez ostatnią cyfrę tj. 9 oznacza powrót na początek, czyli na 0, 1, 2, …
Przykład:
klucz = 2
wpisany ciąg znaków: ABCabc123defXYZ890
wypisany ciąg znaków: CDEabc345defZAB012
Ćwiczenie wymaga zastosowania pętli i skorzystania z operacji arytmetycznych.

W ramach rozszerzenia programu:

- należało wykonać system kontroli wpisywanego klucza tak, aby przyjmował tylko cyfry;
- wpisany klucz odczytywać jako znaki z tablicy ascii;
- pozwolić na wpisywanie dużych wartości klucza szyfrującego.

2. Przebieg ćwiczenia

Zajęcia zostały rozpoczęte od objaśnienia programu i zasygnalizowania konieczności użycia takich operacji arytmetycznych jak dzielenie czy mnożenie.

```
.data
STDIN    = 0
STDOUT   = 1
SYSWRITE = 1
SYSREAD  = 0
SYSEXIT  = 60
EXIT_SUCCESS = 0
```

W sekcji `.data` deklarujemy stałe do późniejszego użycia przy wywołaniach funkcji systemowych, a także:

```
BUFLEN  = 128
```

W stałej `BUFLEN` przechowywana jest informacja o rozmiarze używanych w programie buforach (128 bajtów).

```
msg_key: .ascii "Podaj klucz: "
msg_key_len = .-msg_key

msg_error: .ascii "Wpisany klucz nie jest liczbą."
msg_error_len = .-msg_error

msg: .ascii "Wpisz ciag znakow: "
msg_len = .-msg
```

Powyższe trzy informacje (ciągi znaków ascii) będą wypisywane na ekranie. Pod każdą z nich zadeklarowana jest stała określająca długość tych informacji.

```
key_maxlen = 8
```

Ostatnią stałą w sekcji `.data` jest maksymalna długość klucza, którą będzie wpisywał użytkownik.

```
.bss
.comm keyin,    8
.comm textin,   128
.comm textout,  128
```

W sekcji `.bss` zadeklarowane są trzy bufory. Na `keyin` zarezerwowane jest 8 bajtów i będzie służył do przechowywania klucza, który będzie odczytywany jako ascii. Na `textin` i `textout` przeznaczone jest po 128 bajtów. Pierwszy z nich będzie przechowywał ciąg znaków wpisanych przez użytkownika, a do drugiego zostaną wpisane zaszyfrowane dane, które zostaną wypisane na ekranie.

```
.text
.globl _start

_start:
#wypisanie prośby o klucz
movq    $SYSWRITE, %rax
movq    $STDOUT, %rdi
movq    $msg_key, %rsi
movq    $msg_key_len, %rdx
syscall

#wczytanie klucza
movq    $SYSREAD, %rax
movq    $STDIN, %rdi
movq    $keyin, %rsi
movq    $key_maxlen, %rdx
syscall
```

```
dec     %rax                    #długość wczytanego klucza w rax
movq    $0, %rdi                #licznik
movq    $0, %r10                #wartość klucza szyfrującego
```

Powyżej zaczyna się wykonywanie programu. Następuje wykonanie po sobie dwóch funkcji systemowych - wypisanie prośby o klucz, a następnie wczytanie klucza szyfrującego.

```
wczytaj_klucz:
```

Od tej etykiety klucz pobrany jako ciąg znaków ascii będzie zmieniany na wartość liczbową i zapisywany do rejestru `%r10`. Instrukcje w obrębie etykiety będą działały w pętli.

```
movq    $0, %rbx
```

Rejestr `%rbx` będzie służył do pobrania pojedynczego znaku z `keyin` zaczynając od najstarszej pozycji.

```
imulq   $10, %r10
```

Przed dodaniem każdej cyfry, która będzie pobrana przez rejestr `%bl`, a następnie dodana do `%r10`, suma z `%r10` będzie mnożona przez `$10`. Jest to rozwiązanie zrobione na wzór schematu Hornera.

```
#pobranie jednego znaku (bajtu) z keyin do %bl
movb    keyin(, %rdi, 1), %bl

#sprawdzanie błędów - czy w keyin są tylko cyfry?
cmp     $'0', %bl
jl      wypisz_blad
cmp     $'9', %bl
jg      wypisz_blad
```

Powyższe porównania ustawiają flagi, przez które może nastąpić skok do etykiety `wypisz_blad`, jeśli pobrany symbol z `textin` nie jest cyfrą.

```
subb    $'0', %bl
```

Od pobranego znaku odejmowana jest wartość symbolu `$'0'`, aby z kodu ascii uzyskać wartość liczbową cyfry.

```
addq    %rbx, %r10
```

Cyfra dodawana jest do `%r10`, w którym sumowana jest pełna liczba wpisana do `textin`.

```
incq    %rdi              #zwiększenie licznika
cmp     %rax, %rdi        # ? licznik==długość klucza z textin
jl      wczytaj_klucz     #skok, jeśli %rdi < %rax

jmp dalej
```

Po wczytaniu i przekonwertowaniu na wartość liczbową klucza, następuje ominięcie poniższego wypisania informacji o błędnym podaniu klucza i przeskok pod etykietę `dalej`.

```
wypisz_blad:
movq    $SYSWRITE, %rax
movq    $STDOUT, %rdi
movq    $msg_error, %rsi
movq    $msg_error_len, %rdx
syscall
jmp koniec
```

Wypisanie informacji o błędzie, wykonywane jeśli wpisany klucz nie jest cyfrą. Po tej wykonaniu tej funkcji następuje przeskok do etykiety `koniec`, w której program jest kończony przez wywołanie funkcji systemowej.

```
dalej:
#wypisanie prośby o ciąg znaków do zaszyfrowania
movq    $SYSWRITE, %rax
movq    $STDOUT, %rdi
movq    $msg, %rsi
movq    $msg_len, %rdx
syscall

#wczytanie ciągu znaków
movq    $SYSREAD, %rax
movq    $STDIN, %rdi
movq    $textin, %rsi
movq    $BUFLEN, %rdx
syscall

movq    $0, %rdi                #licznik
movq    $0, %r8                 #rejestr do przechowywania
                                #dzielnika

loop:
```

Od etykiety `loop` będą wczytywane kolejne znaki pobrane wcześniej do `textin` i odpowiednio szyfrowane.

```
movq    $0, %rbx
movb    textin(, %rdi, 1), %bl

cmp     $'A', %bl
jl      czy_to_cyfra
cmp     $'Z', %bl
jg      wypisz
```

Jeśli wczytany znak jest mniejszy niż kod ascii litery `$'A'`, to należy jeszcze wykonać sprawdzenie czy znak nie jest cyfrą, dlatego następuje w tym miejscu skok do etykiety `czy_to_cyfra`. Jeśli kod znaku jest wyższy od `$'Z'`, to nie zostaje zakodowany i przechodzi po etykietę `wypisz`, które nie modyfikuje znaku.

```
#kod od tego miejsca się wykona <=> duża litera w %bl
subb    $'A', %bl           #uzyskanie liczbowej reprezentacji
                            #litery (od 0 do 26)
addq    %r10, %rbx          #dodanie klucza do litery

#liczenie modulo 26
movq    %rbx, %rax          #dzielna zawsze musi być w %rax
movq    $0, %rdx
movq    $26, %r8            #dzielnik w %r8
idiv    %r8, %rax           #wynik operacji dzielenia zapisywany
                            #jest %rax, a reszta w %rdx

movq    $0, %rbx
movb    %dl, %bl
```

Po uzyskaniu reszty z dzielenia sumy pobranej litery (w kodzie liczbowym od 0 do 26) z wcześniej wyliczonym kluczem, należy dodać wartość, która ustawi literę do odczytania jako ascii.

```
add     $'A', %bl
jmp     wypisz
```

Analogiczną operację wykonujemy dla wykrycia i zakodowania cyfr z `textin`:

```
czy_to_cyfra:
cmp     $'0', %bl
jl      wypisz
cmp     $'9', %bl
jg      wypisz
```

```
subb    $'0', %bl
addq    %r10, %rbx

#liczenie mod10
movq    %rbx, %rax
movq    $0, %rdx
movq    $10, %r8
idiv    %r8, %rax

movq    $0, %rbx
movb    %dl, %bl
addb    $'0', %bl
```

Jeśli pobrany znak nie mieści się w przedziale [0, 9], to zostaje wypisany bez żadnych zmian. W innym wypadku zostaje zostaje odjęta wartość `$'0'` w ascii, by uzyskać wartość liczbową cyfry. Następnie dodany zostaje klucz i wykonywane jest dzielenie przez `$10`. Do reszty z dzielenia dodawane jest `$'0'`, aby otrzymać cyfrę w kodzie ascii.

```
wypisz:
movb    %bl, textout(, %edi, 1)
inc     %rdi                    #przejście do następnego znaku z
                                #bufora
cmp     $'\n', %rdi             #sprawdza czy nie pobrano '\n'
jne     loop                    #pobranie kolejnego znaku

movq    $SYSWRITE, %rax         #wypisanie zaszyfrowanego ciągu
movq    $STDOUT, %rdi
movq    $textout, %rsi
movq    $BUFLEN, %rdx
syscall
```

```
koniec:
movq    $SYSEXIT, %rax
movq    $EXIT_SUCCESS, %rdi
syscall
```

Powyższe operacje wykonywane są dla wszystkich znaków, niezależnie od tego czy przeszły szyfrowanie (wielkie litery i cyfry), czy nie.

3. Podsumowanie i wnioski

   Przy korzystaniu z operacji mnożenia i dzielenia należy pamiętać, że ich wyniki zajmują różną liczbę bajtów niż te, które biorą w nich udział np. 32b * 32b = 64b, 64b / 32b = 32b.

   Aby wykonać dzielenie w assmebly x86_64 (GAS, AT&T Syntax) dzielną trzeba zawsze umieścić w rejestrze `%rax`, a `%rdx` wyzerować. Operacji dokonujemy przy użyciu `idiv`. W jej wyniku w `%rax` zwracana jest wartość z dzielenia, a w `%rdx` reszta.

   Do operacji mnożenia służy `imul`.

4. Bibliografia

- https://en.wikibooks.org/wiki/X86_Assembly/GAS_Syntax
- https://en.wikibooks.org/wiki/X86_Assembly/Arithmetic
- J.Barlett *Programming from the ground up*, 2003
- http://sticksandstones.kstrom.com/appen.html
- https://en.wikipedia.org/wiki/X86_instruction_listings
- http://zak.ict.pwr.wroc.pl/materials/architektura/laboratorium%20AK2/Dokumentacja/Intel%20Penium%20IV/IA-32%20Intel%20Architecture%20Software%20Developers%20Manual%20vol.%202%20-%20Instruction%20Set%20Reference.pdf