# Raport Laboratorium 3

**Organizacja i Architektura komputerów**

**Autor: Uladzimir Kaviaka (257276)**

**Cel laboratorium:**

Zadanie polegało na napisaniu mnożenia dużych liczb, przyjęto następujące założenia:

- Wczytywanie/wypisanie realizowane za pomocą **scanf, printf**
- Liczby wpisywane/wypisywane w sposób szesnastkowy
- Liczby mogą być bardzo duże, ograniczenie 200 znaków
- Rozpatrzyć osobno przypadek wprowadzania małych liter

**Opis algorytmu:**

Wczytujemy dwie podane liczby, konwertujemy z formatu tekstowego na format hex, żeby zapisać w pamięci komputera do następnego mnożenia. Wykonujemy mnożenie tych liczb i wypisujemy wynik szesnastkowo.

**Implementacja programowa:**

**1. Rezerwacja pamięci pod liczby wprowadzane tekstowo, pod liczby już po konwersji, pod wynik. Rezerwacja zmiennych przechowujących długości naszych liczb, zmienne formatowania dla printf oraz zmienne do komunikacji z użytkownikiem.**

```
1   .data
2
3   msg1: .ascii "Podaj pierwsza liczbe\n\0"
4   msg2: .ascii "Podaj druga liczbe\n\0"
5
6   str1: .space 100 #dopuszczalne 200 znakow HEX, 2 znaki HEX = 1 bajt
7   str2: .space 100 #dopuszczalne 200 znakow HEX, 2 znaki HEX = 1 bajt
8   liczba1: .space 100 #bo .space 4 bajty wystarczy do 8 znakow 0xCAFF1234
9   liczba2: .space 100 #bo .space 4 bajty wystarczy do 8 znakow 0xCAFF1234
10  result: .space 200 #na wynik potrzebujemy n1+n2 liczb
11
12  format_s: .ascii "%s\0"
13  format_h: .ascii "%x\0"
14  newline: .ascii "\n"
15
16  outputIterator1: .int 0 #ile longow wczytano dla pierwszej
17  outputIterator2: .int 0 #ile longow wczytano dla drugiej
18  length1: .int 0
19  length2: .int 0
20
21  .text
```

Str1, Str2, to tekstowe reprezentacje naszych liczb. Liczba1, liczba2, to nasze liczby po konwersji na format szesnastkowy.

Zmienne length1, length2 to zmienne pomocnicze do poprawnego sprawdzania długości naszych liczb, będzie to pokazano w dalszej części raportu.

**2. Startowa komunikacja, wpisanie pierwszej liczby**

```
21   .text
22
23   .global main
24   #Wczytanie pierwszej liczby#
25   main:
26       pushl $msg1
27       call printf     #prosimy o podanie 1 liczby
28       pushl $str1
29       pushl $format_s
30       call scanf      #wczytujemy liczbe do liczba1
```

Umieszczamy na stosie naszą wiadomość i wypisujemy za pomocą **printf**. Następnie umieszczamy na stosie naszą zmienną do przechowywania wczytanej liczby w postaci tekstowej oraz format wczytywania, po czym wywołujemy **scanf.**

**3.Konwersja liczby z postaci tekstowej na format do obliczeń**

```
31      movl $0, %ecx   #rejestr indeksujacy nasza liczbe koncowa
32      movl $0, %ebx   #rejestr indeksujacy kolejne symbole tekstu
33  firstNum:
34      movl $0, %eax
35      movb str1(,%ebx,1), %al #pobieramy symbol
36      cmpb $0x0A, %al #sprawdzamy czy koniec linii (ascii 10 - znak konca linii)
37      jl finish #jezeli rowna sie to koniec wczytywania
38      incl %ebx   #przechodzimy do kolejnego symbolu
39      cmpb $0x30, %al #jezeli wartosc ASCII jest mniejsza niz 30 to znaczy mniejsze od symbolu '0'
40      jb end
41
42      #sprawdezanie malych liter#
43      cmpb $0x46, %al
44      jbe duze
45      cmpb $0x66, %al
46      jbe malalitera
47
```

Zapisujemy 0 w rejestrach **ecx** i **ebx** które będą iterowały nasze zmienne, **ecx** iteruje liczbę po konwersji, **ebx** iteruje liczbę wczytaną z konsoli.

Następnie zerujemy rejestr **eax**, po czym umieszczamy w rejestrze **al** nasz pierwszy symbol tekstowy, następnie sprawdzamy czy jest to koniec tekstowej reprezentacji naszej liczby poprzez porównywanie z wartością ASCII **0x0A** co oznacza znak końca linii. Jeżeli równa się kończymy konwersję, po czym przesuwamy nasz iterator **ebx,** następnie porównujemy z 0x30 ASCII (‘0’) jeżeli mniejsze znaczy to nie jest poprawny symbol więc kończymy program. Następnie idzie sprawdzanie małych liter bo możemy wpisać A zarówno jak i a, więc najpierw sprawdzamy czy to jest duża litera a następnie jeżeli nie jest duża sprawdzamy czy jest mała. Następnie wykonujemy skoki:

```
48   duze:
49       subb $0x30, %al #odejmujemy 30 aby dostac liczbe z kodu ASCII
50       cmpb $0x0A, %al #jezeli mneijsza niz 10 to jest liczba
51       jb cyfra
52       subb $0x07, %al #przypadek gdy mamy litere
```

```
224  malalitera:
225      subb $0x50, %al
226      cmpb $0x0A, %al
227      jb cyfra
228      subb $0x07, %al
229      jmp cyfra
```

W przypadku gdy mamy dużą literkę to odejmujemy 0x30 (HEX), następnie sprawdzamy jeżeli wartość jest mniejsza od 10 to znaczy mieliśmy liczbę w przeciwnym przypadku mamy literkę, co potrzebuje odjąć 7. Bo mając F(46 hex – 30 hex = 16 hex (22 dec)) musimy odjąć 7 bo to będzie wtedy 15 co równoważne z F.

W przypadku małych liter analogicznie tylko przesuwamy pierwsze odejmowanie o 0x50(HEX) bo A(41 HEX) zaś a(61 HEX).

```
53  cyfra:
54      shll $4, liczba1(,%ecx,4) #przesuwamy w lewo o 4 pozycje zeby umiescic jeden symbol tekstowy
55      addl %eax, liczba1(,%ecx,4) #wstawiamy symbol tekstowy
56      movl $0, %edx #wyzerwoanie reszty
57      mov %ebx, %eax  #zapisujemy ebx jako dzielnik w eax w ebx ilosc symboli wczytanych
58      mov $8, %esi    # dzielna w ebp dzielmy razy 8 aby wiedziec ile longow wczytalismy
59      div %esi    #dzielimy reszta w edx
60      cmp $0, %edx    #jezeli reszta 0 znaczy wczytano caly long
61      je end
62      jmp firstNum
63  end:
64      cmpl $100, %ecx #sprawdzamy czy wpisalismy 100 longow bo 200 znakow ascii HEX to 100 long)
65      je finish
66      cmpb $0x0a, str1(,%ebx,1) #jezeli symbol po 8 kolejnych jest spacja nie zwiekszamy ilosc longow (outputItera
67      jl finish
68      add $1, outputIterator1 #iterator pokazuje ile liczb wczytalismy calkowicie
69      incl %ecx  #zwiekszamy indeks rejestru indeksujacego liczbe
70      jmp firstNum #skok do powtorzenia
71
72  finish:
73      mov $0, %eax
74      cmp $0, outputIterator1 #jezeli wpisano mniej niz 8 znakow to trzeba zwiekszyc o 1 ilosc liczb do wypisania
75      je addIterator1
76      add $1, outputIterator1 #jezeli kolejny symbol po 8 symbolach nie jest spacja to znaczy jeszcze jeden long
77
78      mov %ebx, length1 #zapisanie dlugosci liczby 1
```

Następnie gdy już mamy dobrze przekonwertowany nasz znak przechodzimy do metki cyfra gdzie będzie wstawianie do reprezentacji do obliczeń.

Tutaj najpierw przesuwamy w lewo o 4 pozycje co potrzebne do jednej liczby szesnastkowej (bo max F = 1111). Następnie pamiętamy że po konwersji jest nasza liczba w rejestrze **al.** Więc dodajemy na potrzebną pozycję naszą liczbę. Następnie 56-60 linijki są do sprawdzenia czy wpisaliśmy jeden long, do tego potrzebujemy zrobić dzielenie, więc zerujemy rejestr **edx** gdzie będzie reszta z dzielenia. Następnie wpisujemy w **eax** ilość przekonwertowanych symboli i jeżeli reszta z dzielenia przez 8 jest 0 znaczy wczytaliśmy całego longa i przechodzimy do sekcji **end**: gdzie sprawdzamy czy nie ma przekroczenia zakresu (podane w założeniach) następnie sprawdzamy jeżeli symbol po 8 symbolach(całym long) jest nową linią to znaczy że długość mamy 1 a nie 2 i znaczy to już koniec konwersji naszej liczby. Jeżeli nie to zwiększamy o 1 nasz **outputIterator1** który odpowiada za długość naszej liczby oraz zwiększamy o 1 rejestr indeksujący liczbę po konwersji i wykonujemy skok do **firstNum (punkt 3 raportu)** co równoważne że przechodzimy do konwersji następnej liczby zapisanej tekstowo.

Jeżeli przechodzimy do sekcji **finish**, co jest równoważne z konwersją całej liczby tekstowej na postać do obliczeń. Tutaj sprawdzamy jeżeli było wpisano mniej niż 8 znaków, co odpowiada że **outputIterator1** będzie 0, to trzeba zwiększyć długość liczby o 1. bo (0xCCFF1100) ma długość 1 a nie 0. Więc przechodzimy do sekcji **addIterator1**

```
214   addIterator1:
215       mov %ebx, length1
216       add $1, outputIterator1
217       jmp step
218
```

Dodajemy 1 do długości naszej liczby. I przechodzimy do **step**(sekcji wczytania drugiej liczby).

Jeżeli nie wejdziemy do tej sekcji to znaczy że wpisano więcej niż jeden cały long więc zwiększamy o 1 naszą długość liczy ( **outputIterator1**).

**4. Wczytanie drugiej liczby i konwersja.**

**Analogicznie jak dla liczby 1.**

**5. Wykonanie mnożenia**

```
137   multiply:
138       clc #wyczyszczenie flag
139       pushf #odlozenie czystych flag na stos
140       push outputIterator1 #do pobrania do esi
141       push $0
142
143   outerLoop:
144       pop %edi #index wyniku
145       inc %edi
146       pop %esi #licznik outerLoop
147       cmp $0,%esi
148       jz showresult
149       dec %esi
150       movl liczba1(,%esi, 4), %eax #wpisujemy aktualny fragment luczby
151       push %esi #licznik na stos
152       push %edi #index wyniku na stos
153       mov outputIterator2, %esi #dlugosc liczba2 = licznik wewnetrznej petli
154
155   innerLoop:
156       push %eax #frament liczby1 na stos
157       dec %esi
158       dec %edi
159       movl liczba2(,%esi, 4), %ebx  #wycinam fragment liczby2 i wkladam do ebx
160       mull %ebx #ebx*eax = edx|eax
161       addl %eax,result(,%edi,4) #dodanie eax do wyniku
162       inc %edi
163       jc overflow1 #sprawdzanie nadmiaru
164   back1:
165       addl %edx,result(,%edi,4) #dodanie drugiej porcji iloczynu po mull
166       inc %edi
167       jc overflow2 #czy jest nadmiar
168   back2:
169       pop %eax
170       cmp $0,%esi #czy koniec wewnetrznej petli
171       jz outerLoop
172       jmp innerLoop
173
174   overflow1:
175       adcl $0,result(,%edi,4) #dodanie nadmiaru
176       clc
177       jmp back1
178
179   overflow2:
180       adcl $0,result(,%edi,4) #dodanie nadmiaru
181       clc
182       jmp back2
```

Najpierw trzeba wyczyścić wszystkie flagi żeby nie miały wpływu na kolejne kroki mnożenia. Po czym odkładamy te flagi na stos.

Następnie umieszczamy na stosie długość pierwszej liczby i liczbę 0. Następnie w zewnętrznej pętli **outerLoop,** pobieramy 0 ze stosu do rejestru **edi** co będzie odpowiadało pozycji wyniku oraz zwiększamy od razu o 1. Następnie pobieramy ze stosu naszą wartość **outputIterator1** do rejestru **esi** który będzie licznikiem naszej pętli **outerLoop.** Sprawdzamy jeżeli **esi** jest zerem(długość liczby 0 to kończymy). Jeżeli nie to dekrementujemy **esi** i wpisujemy do eax fragment liczby pierwszej. Po czym odkładamy licznik pętli oraz pozycję wyniku na stos. Następnie do rejestru esi wpisujemy długość drugiej liczby (**outputIterator2**). I przechodzimy do pętli wewnętrznej gdzie najpierw odkładamy na stos **eax** w którym jest fragment naszej pierwszej liczby, następnie zmniejszamy **esi** oraz **edi** o 1. Następnie do **ebx** wpisujemy fragment drugiej liczby i wykonujemy mnożenie po czym inkrementujemy **edi**. Jeżeli wystąpił nadmiar to przechodzimy do **overflow1** i wykonujemy dodanie naszego nadmiaru używając **adcl $0** co nie dodaje wartości ale dodaje nadmiar, po czym obowiązkowo należy wyczyścić flagi i powracamy. Następnie dodajemy do kolejnej pozycji wyniku zawartość rejestru **edx.** Bo po wykonaniu mnożenia rozkazem mull, wynik jest zawarty w dwóch rejestrach **eax|edx** po czym inkrmenetujemy znów edi i sprawdzamy czy wystąpił teraz nadmiar(jeżeli tak przechodzimy do **overflow2)** następnie ściągamy ze stosu **eax** czyli fragment naszej liczby pierwszej który umieściliśmy na stosie na początku wewnętrznej pętli. Następnie sprawdzamy czy jest koniec wewnętrznej pętli jeżeli nie przechodzimy do następnego kroku wewnętrznej pętli jak nie to wracamy do zewnętrznej pętli.

**6. Wypisanie wyniku.**

Ostatnie działanie to wypisanie wyniku, skoro że mamy wartość jak by odwróconą w pamięci po mnożeniu np.:

CCFF1111BBCCFFAA*1234 = 00000E93 9302B6CE 8F9DE288

zaś wynik mamy w pamięci w ten sposób:

**0x8f9de288, 0x9302b6ce, 0x00000e93**

To musimy wypisywać od końca:

```
185  #Wypisanie wyniku#
186  showresult:
187      decl %edi
188      mov %edi, %eax
189
190  nextresult:
191      cmp $-1, %eax
192      je endprogram
193      movl result(,%eax,4), %ebx
194      push %eax   #odkladamy na stos zeby nie zepsuc wartosc
195      push %edx   #rezerwajca miejsca zeby wypisac long
196      push %edx
197      pushl %ebx  #push liczby do wykorzystania printf
198      pushl $format_h #push formatu
199      call printf
200      pop %eax    #pobieramy odlozone wartosci ze stosu
201      pop %eax
202      pop %eax
203      pop %eax
204      pop %eax    #pobiearmy wartosc ze stosu eax
205      dec %eax
206      jmp nextresult
207
208
209  endprogram:
210      pushl $newline
211      call printf
212      call exit
```

Najpierw dekrementujemy **edi** który odpowiadał pozycji w wyniku. Po czym umieszczamy długość wyniku w **eax.** Następnie w pętli póki nasza długość nie jest -1, dlaczego -1? bo jeżeli mamy długość wyniku 2, to znaczy musimy wypisać longi z 0, 1 i 2 pozycji więc jeżeli tu wstawić zero stracimy jeden long. Następnie umieszczamy fragment wyniku w **ebx**, po czym odkładamy nasz licznik na stos. Następnie żeby wypisać long musimy zarezerwować miejsce po czym umieszczamy na stosie nasz fragment wyniku, umieszczamy na stosie format wypisywania szesnastkowy, wywołujemy **printf**. Po czym ściągamy nasze wartości ze stosu w odwrotnej kolejności, dekrementujemy **eax** i przechodzimy do wypisania następnego longa.

**Wnioski:**

Zadanie nie było zbyt trywialne i banalne więc nauczyłem się sporo, najpierw trzeba było wykonać konwersję z tekstu na hex ale tutaj nie było dużego problemy jeżeli umieć posługiwać się tabelą ASCII oraz rozumieć jak to trzeba umieścić w pamięci. Mnożenie trzeba było dużo razy sprawdzać bo często miałem błędy jak zrzut pamięci albo segmentation fault, jednak użycie debuggera GDB pozwoliło znaleźć i wyeliminować te błędy. Ćwiczenie naprawdę jest ciekawe i zmusza do umiejętności korzystania z GDB, co jest obowiązkowe przy pisaniu programów w assemblerze.