Wrocław 18 marca 2015

Kamil Cała
209954
Środa 7:15 TN

Sprawozdanie z laboratorium nr 1

Data laboratorium: 11.03.2015r
Rok akademicki 2014/2015, Informatyka
Prowadzący: Mgr. Aleksandra Postawka

Opis ćwiczenia

Celem tego ćwiczenie było dalsze zapoznanie się z działaniami arytmetycznymi, tworzeniem pętli, i operacjami na pamięci w GNU/Assembly. Dokonaliśmy tego poprzez wykonanie dwóch ćwiczeń polegających na konwersji reprezentacji liczb pomiędzy różnymi systemami liczbowymi.

Ćwiczenie 1

Ćwiczenie to polegało na odczytaniu z wejścia standardowego liczby w postaci dziesiętnej i po uprzedniej konwersji z użyciem schematu Hornera, wypisanie jej w postaci liczby o podstawie 3. Program podzieliłem na następujące etapy:

1. Wczytanie ciągu znaków z wejścia standardowego za pomocą przerwania syscall
2. Konwersja ciągu znaków zakodowanego w ASCII w liczbę zapisaną w pamięci
3. Konwersja tej liczby na jej reprezentację w systemie trójkowym przy użyciu schematu Hornera, czego wynikiem jest zapisany od tyłu ciąg znaków w ASCII
4. Odwrócenie ciągu znaków, aby prezentował liczbę w poprawnej formie
5. Wypisanie wyniku na wyjście standardowe

Konwersja ciągu znaków zakodowanego w ASCII w liczbę zapisaną w pamięci

Wczytanie ciągu znaków z wejścia standardowego odbywa się w standardowy sposób. Następnym krokiem jest zamiana tego ciągu, który jest zakodowany w ASCII na faktyczną liczbę. W tym celu należy każdy znak ciągu zamienić w wartość którą reprezentuje (w zależności od liczby i jej pozycji) po czym dodać ją do rejestru, który będzie przechowywał ostateczny wynik. W związku z tym inicjalizuję wszystkie wartości potrzebne do stworzenia pętli, oraz rejestr r15 który będzie przechowywał wynik

```
movq %rax,%r10              #length of input
dec %r10                    #decrease by 1 to skip '\n'
movq $0, %r11               #initialize counter
movq $0, %r15               #to store number
```

Pierwszym krokiem który należy podjąć w pętli jest skopiowanie kolejnej wartości do rejestru i konwersję jej z ASCII na wartość liczbową

```
movzbq user_input(,%r11,1), %r12
sub $'0', %r12                #substract value of '0' in order to normalize number to 0
```

Następnie należy sprawdzić na której pozycji w przetwarzanym ciągu aktualnie się znajdujemy

```
#which position are we on?
sub %r11, %r10
sub $1, %r10
```

Następnie należy przetwarzaną liczbę przemnożyć przez wartość jej pozycji, czyli 10^p. Odpowiedzialny jest za to poniższy wycinek kodu

```
#10^%r10
movq %r12, %r11                      #start with 1
cmp $0, %r10
je _lastdigit

_power:
    xor %rax, %rax
    movl $10, %eax
    mul %r11                        #multiply by 10
    movq %rax, %r11                 #move result to register
    dec %r10                        #decrease the counter
    cmp $0, %r10                    #if exponent still bigger/equal 0 start over
    jne _power
    jmp _afterpower

#The edge case for 10^0=1
_lastdigit:
    add %r11, %r12
```

Na koniec należy dodać obliczoną wartość do ostatecznego wyniku w r15, zwiększyć licznik, sprawdzić warunek kończący pętlę (licznik == długość ciągu znaków) i ewentualnie rozpocząć kolejną iterację.

```
_afterpower:
add %r11, %r15

inc %r11                             #decrease the counter
cmp %r10, %r11                       #check for loop's ending condition (every char literated)
jne _toint                           #if not met - jump to the beginning
```

Konwersja tej liczby na jej reprezentację w systemie trójkowym przy użyciu schematu Hornera, czego wynikiem jest zapisany od tyłu ciąg znaków w ASCII

W celu konwersji liczby przy użyciu schematu Hornera należy umieścić ją w rejestrze eax i dzielić przez 3 (gdyż konwertujemy na system trójkowy) aż w tym rejestrze pojawi się wartość 0. Operacja div dzieli liczbę znajdującą się w eax przez podany jawnie argument i zapisuje wynik w tym samym rejestrze. Jednocześnie zapisuje resztę z dzielenia w rejestrze edx i te właśnie wartość po każdym dzieleniu należy zapisać w pamięci, gdyż są wynikiem konwersji z tym zastrzeżeniem iż są one zapisane od tyłu.

```
#horner's method
xor %rax, %rax
xor %rdx, %rdx
movq %r15, %rax
_horner:
    CDQ                                         #sign-extend eax to edx
    movq $3, %rbx
    div %rbx                                    #divide the number by 3
    movl %edx, inverted(,%r10,1)                #the remainder is part of converted number
    cmp $0, %eax
    jne _horner
```

Należy jeszcze dodać do każdej wartości w buforze przechowywującym wynik tej operacji dodać wartość ‘0’, aby ponownie zakodować ciąg znaków w ASCII

```
#take inverted back to ascii chars
xorq %r11, %r11
#iterate each char
_toascii:
    xorq %r12, %r12
    mov inverted(,%r11,1), %r12
    add $'0', %r12                  #add '0' value to code the number
    mov %r12, inverted(,%r11,1)
    inc %r11
    cmp %r11, %r10
    jne _toascii
```

Odwrócenie ciągu znaków, aby prezentował liczbę w poprawnej formie

Tę operację wykonałem przy użyciu dwóch liczników. Jeden zaczyna się w zerze i zwiększa o jeden z każdą iteracja pętli, a drugi zaczyna z wartością równą długości ciągu znaków i po każdej iteracji maleje. Pętla wykonuje się, póki rosnący licznik nie osiągnie wartości równej długości ciągu znaków.

```
#invert back to normal
```

```
xorq %r11, %r11                     #the increasing counter
movq %r10, %r13                     #the decreasing counter
sub $1, %r13
_invert:

    xor %r12, %r12
    mov inverted(,%r13,1), %r12     #take char iterating from back...
    mov %r12, output(,%r11,1)       #...and put it back, iterating from beginning
    inc %r11
    dec %r13
    cmp %r10, %r11


    jl _invert
```

Wypisanie wyniku na wyjście standardowe

Na koniec wystarczy dodać znak końca linii na końcu ciągu znaków.

```
movq $'\n', output(,%r10,1)          #append newline char at the end of the output
```

Wypisanie wyniku na wyjście standardowe odbywa się w standardowy sposób.

Ćwiczenie 2

Drugie ćwiczenie było mniej skomplikowane. Dla ułatwienia zakładamy w nim że dane wejściowe zawsze mają prawidłowy format. Działanie programu polega na pobraniu z wejścia standardowego ciągu znaków reprezentującego liczbę w systemie szesntastkowym, gdzie A=10, B=11, …, F=16 Następnie liczba zostaje przekonwertowana na system czwórkowy korzystając z faktu iż systemy te posiadają skojarzoną bazę. Wystarczy więc każdy znak z ciągu wejściowego zapisać na dwóch pozycjach w ciągu wyjściowym.

Ponownie więc iterujemy po całym ciągu wczytanym z wejścia standardowego. Po wczytaniu znaku do rejestru eax trzeba rozpoznać czy jest on cyfrą czy wielką literą (tylko takie są akceptowane).

```
cmp $'9', %eax              #we're assuming correct input
    jle _number
    jg _letter
```

W zależności od tego jaką formę ma przetwarzany znak, sprowadzamy go do jego faktycznej wartośći w adekwatny sposób. Dla liczb odejmujemy wartość ‘0’, a dla liter ‘A’, po czym dodajemy do nich jeszcze 10.

```
_number:
```

```
    subl $'0', %eax          #convert ascii to int
    jmp _after
_letter:
    subl $'A', %eax          #convert ascii to int
    addl $10, %eax           #A = 10, B = 11 and so on
    jmp _after
_after:
```

Następnie należy obliczyć miejsce w którym przetwarzana liczba znajdzie się w buforze wyjściowym. Z racji tego że każda pozycja w systemie szesnastkowym w systemie czwórkowym zajmuje dwie pozycje, to poszukiwana przez nas pozycja jest określona prostym wzorem:

*pozycja_wyjściowa = pozycja wejściowa*2*

```
#calculate output write position
push %r11
push %rax
movl $2, %eax                       #output_position is input_position*2
mul %r11
movl %eax, %r11D
pop %rax
```

Po obliczeniu pozycji, dzielimy przetwarzaną liczbę przez 4. Wynik dzielenia zapisujemy na obliczonej pozycji, a resztę na pozycji kolejnej.

```
CDQ                                 #sign extend eax to edx
movl $4, %ebx                       #to convert to base 4
div %ebx
add $'0', %eax                #back to ASCII
movl %eax, output(,%r11,1)          #quoitient is older bytye
inc %r11
add $'0', %edx                      #back to ASCII
movl %edx, output(,%r11,1)          #remainder is lower byte
```

Po zakończeniu działania pętli w buforze wyjściowym znajduje się już gotowy wynik. Wystarczy tak jak wcześniej dodać na końcu znać końca linii i wypisać go na wyjście standardowe

Wnioski

Po wykonaniu tego ćwiczenia można stwierdzić ze operacje które w językach wyższego poziomu są banalne w wykonaniu, na tym poziomie abstrakcji mogą sprawiać kłopoty. Dobrym tego przykładem jest odczytanie liczby z wejścia standardowego, które wymaga pracochłonnej w napisaniu konwersji z ciągu znaków na wartość liczbową.

Zgodnie z oczekiwaniami można też zauważyć że konwersja pomiędzy systemami liczbowymi jest o wiele prostsza jeżeli mają one bazy skojarzone.