Marta Głowacka, 234999 Wrocław, 26.03.2018r.

WT-N-10

LABOLATORIUM ARCHITEKTURY KOMPUTERÓW

1. Proste instrukcje programowe – pętle, warunki z użyciem instrukcji asemblera

1. Treść ćwiczenia

Zakres i program ćwiczenia:

- Wczytywanie w ASCII i zamiana na wartość liczbową
- Zamiana z systemu Ux na inny system Uy, gdzie x i y to podstawy systemu
- Pętle, warunki, dzielenie i mnożenie

Zrealizowane zadania:

- Wczytanie liczby w systemie U10 (ASCII)
- Obliczenie wartości bezwzględnej
- Sprawdzenie poprawności wejścia – czy na pewno wprowadzono liczbę, a nie ciąg znaków
- Sprawdzenie, czy liczba jest pierwsza
- Jeśli jest pierwsza, wypisanie jej w systemie U7 (ASCII)

2. Przebieg ćwiczenia

Wstęp

Cały program został wysłany w pliku *.s, poniżej omówione zostały fragmenty kodu – pominięto trywialne inicjowanie zmiennych funkcji systemowych na samym początku, gdyż omówione zostało ono we wcześniejszym sprawozdaniu.

Początek

W sekcji **.data** inicjujemy:

Znaki ascii początku liczb (posłuży nam do konwersji liczby na ascii i na odwrót) oraz nowej linii.

```
POCZ_LICZB = 0x30          #kod ascii pierwszej cyfry - 0 (48)
```

```
NOWA_LINIA = 0xA          #kod ascii nowej linii - (10)
```

Zmienne, które będą w naszym programie pewnego rodzaju stałymi, ale łatwo można je zmienić w tej sekcji, by program dalej działał poprawnie, lecz na innych systemach.

```
PODST_WEJ = 10       #podstawa systemu dziesiętna - liczba z wejścia
PODST_WYJ = 7        #podstawa systemu siódemkowa - na wyjście
CYFRA_UJEMNA = 5     #od 5 w systemie U-dziesiętnym zaczynają się liczby ujemne
```

Komunikaty, które będziemy wyświetlać w odpowiednich momentach.

```
komunikat: .ascii "Nie podano liczby\n"
komunikat_len = .-komunikat

komunikat_niepierwsza: .ascii "Nie jest to liczba pierwsza\n"
komunikat_niepierwsza_len = .-komunikat_niepierwsza

komunikat_pierwsza: .ascii "To liczba pierwsza\n"
komunikat_pierwsza_len = .-komunikat_pierwsza
```

Później alokujemy potrzebne bufory w sekcji **.bss**

```
.comm textin, 512    #bufor do ciągu wczytanego od użytkownika
.comm textinv, 512     #bufor do którego wpiszemy ciąg w odwrotenj kolejności
.comm textout, 512    #bufor z wyjściową liczbą wypisanej w poprawnej kolejności
.comm textmod, 512    #bufor do liczby bezwzględnej
```

Wczytanie liczby w systemie U10

Najpierw realizujemy wczytanie podanego ciągu znaków od użytkownika. Po etykiecie **_start**:

```
movq $SYSREAD, %rax
movq $STDIN, %rdi
```

```
movq $textin, %rsi
movq $BUFLEN, %rdx
syscall
```

Dokładnego omówienia tej części kodu dokonałam w poprzednim sprawozdaniu. Teraz uruchomimy pętle zamiany z ASCII na liczby (przy okazji sprawdzamy poprawność każdego znaku – i w razie potrzeby, przechodzimy do komunikatu o błędzie).

Rejestr %rdi – licznik

Rejestr %r9 – liczba znaków (przyda się później)

Należy jednak odjąć 2 z zawartości obu rejestrów (znak \n oraz liczyć będziemy od zera)

Rejestr %rsi – kolejne potęgi, zaczynamy od 1

Rejestr %r8 – trzyma wynik końcowy

```
movq %rax, %rdi         #w rax siedzi liczba znaków, wrzucamy
                         to do rdi – posłuży nam za licznik
movq %rax, %r9         #w r9 przechowujemy liczbę znaków (w rdi
                        zostanie ona obniżona, bo to licznik)
sub $2, %r9           #nie bierzemy pod uwagę \n i liczymy od 0
sub $2, %rdi          #odejmujemy 2, bo: odejmujemy znak \n na
                     końcu i będziemy liczyć od zera, a nie od
                     jedynki
movq $1, %rsi          #na razie do rsi wrzucamy pierwszą
                     (dlatego 1) potęgę 10,
                     #później będziemy tu wrzucać kolejne potęgi
movq $0, %r8          #wynik końcowy, na razie wynosi 0
jmp petla_zamiany_na_liczby
```

Po takim przygotowaniu możemy przejść do pętli, która kolejno pobiera znaki z bufora textin:

```
petla_zamiany_na_liczby:
```

Pętla zakończy się, gdy przejdziemy wszystkie znaki (dojdziemy do indeksu 0).

```
cmp $0, %rdi             #wyskoczenie z petli jesli licznik
                          jest < 0,
```

Jeśli skończyliśmy pętle, to przechodzimy do kolejnego etapu zadania.

```
jl zamiana_na_bezwzgledna    #wykonuje skok, jeśli rdi < 0
```

```
movq $0, %rax                    #zerujemy rax (siedzi tu wynik z
                                 mnożenia)
movb textin(, %rdi, 1), %al      #odczyt (po jednym bajcie =
                                 8bitów) litery
                                 #do rejestru al (rax wyzerowany),
                                 #8 bitowy fragment rax
```

Rozkodowujemy ASCII na liczbę (mamy taką nadzieję, ale musimy to sprawdzić, czy znak okazał się liczbą).

```
sub $POCZ_LICZB, %al          #w al jest liczba, bo odjęliśmy
                               pocz_liczb od ascii
```

Sprawdzamy, czy wynik jest w odpowiednim przedziale – czy jest cyfrą.

Jeśli nie jest, wyświetlamy błędy.

```
cmp $PODST_WEJ, %al    #48 to 0 ... 57 to 9, więc x-48 jeśli
                    jest większe niż 10, czyli podstawa
                    systemu, znaczy, że wprowadzono złą liczbę
jge blad         #jge = jump if greater of equal


cmp $0, %al         #jeśli mniejsze od zera, to też nie
wprowadzono cyfry
            #druga (wyższa) strona przedziału ascii
jl blad             #jump if less
```

Wyliczamy kolejną potęgę 10. W rejestrze rax znajduje się nasza cyfra (wpisywaliśmy do al.). Mnożmy ją przez aktualną potęgę 10. Wynik dodajemy do %r8, czyli wartości globalnej (całościowego wyniku).

```
mul %rsi         #wynik wpisuje do rejestru rax (tak działa
mul)
            #mnożymy rsi (kolejne potęgi 10tki, najpierw 1)
razy rax
            #(znajduje się tu tylko wartość zapisana w al -
czyli nasza cyfra)
            #czyli np. 1 * 8, 10 * 7, 100 * 200 itd.
zapisujemy od końca
```

```
add %rax, %r8        #dodanie obecnego wyniku do globalnego (r8 trzyma wartość całej liczby)
```

Wyliczamy kolejną potęgę podstawy, czyli w tym przypadku 1, 10, 100 itd.

Działa to tak: która potęga (rsi) * 10 (PODST_WEJ) -> zapisanie wyniku do rsi

Trzeba pamiętać, że funkcja mul wymaga umieszczenia mnożnej w rejestrze rax, więc nie można od razu pomnożyć rsi*podstawy i zapisać w rsi.

```
movq %rsi, %rax
movq $PODST_WEJ, %rbx
mul %rbx
movq %rax, %rsi
```

Obniżamy licznik i kontynuujemy.

```
#zmniejszenie licznika, czyli rdi i powrot na poczatek petli
dec %rdi
jmp petla_zamiany_na_liczby
```

Zamiana na wartość bezwzględną

Przede wszystkim wypada zauważyć, że liczba dodatnia w U10 (taka, która ma na początku 0-4) nie wymaga żadnej konwersji, a jedynie ponownego wypisania na konsolę. Jednak, jeśli liczba okaże się ujemna (początkową cyfra to 5-9), należy wykonać konwersję na liczbę przeciwną do niej.

Dokonujemy wstępnego przygotowania rejestrów. Naszym licznikiem będzie rejestr r9 – liczba cyfr pomniejszona o 1, już wcześniej odpowiednio przygotowany. Rejestr rsi trzyma kolejne potęgi 10.

```
zamiana_na_bezwzgledna:


movq $1, %rsi
movq %r9, %rdi        #liczba cyfr później dalej jest przydatna (index)
movq %r9, %r11        #liczba cyfr później dalej jest przydatna (index)
dec %r9            #liczba cyfr pomniejszona o 1 - licznik (index)
movq %r8, %rcx        #kolejne liczby, które są wynikiem dzieleń przez 10
```

Chcemy uzyskać pierwszą cyfrę wprowadzonej liczby. Należy podzielić ją przez 10^y, gdzie y to (liczba cyfr-1). Aby uzyskać y, wykonujemy następującą pętlę.

Przykładowo, jeśli wprowadziliśmy 1, chcemy uzyskać 1, jeśli 5, to 1, jeśli 54, to 10, jeśli 656, to 100, jeśli 7676, to 1000.

```
#pętla dająca 1, 10, 100, 1000 itd.
petla_pow:
#Porownanie indeksu, aby wybrac odpowiednia potege 10 w zaleznosci od liczby cyfr
cmp $0, %r9
```

Jeśli pętla się skończyła, następny etap to dzielenie przez uzyskaną liczbę postaci 10^y.

```
jl dzielenie
```

Wyliczenie kolejnej potęgi podstawy, czyli w tym przypadku 1, 10, 100 itd.

która potęga (rsi) * 10 (PODST_WEJ) (rbx) -> zapisanie wyniku do rsi

Ale funkcja mul wymaga umieszczenia mnożnej w rejestrze rax, więc nie można od razu pomnożyć rsi*podstawy (rbx) i zapisać w rsi. W rsi siedzi prawidłowy wynik postaci 1 lub jedynki z zerami: 10...

```
movq %rsi, %rax
movq $PODST_WEJ, %rbx
mul %rbx
movq %rax, %rsi

dec %r9            #obniżenie indeksu
jmp petla_pow
```

Dzielimy liczbę przez 10^(liczba cyfr - 1) – wynik zapisany jest w rsi, aby uzyskać pierwszą cyfrę wprowadzonej liczby, bo chcemy sprawdzić, czy jest dodatnia, czy ujemna.

W r8 jest cała liczba. Dzieląc przez rsi, otrzymamy pierwszą cyfrę -> wynik do rax, a stamtąd do rcx.

```
dzielenie:
movq %r8, %rax
div %rsi
movq %rax, %rcx
```

Porównujemy pierszą cyfrę z 5. Jeżeli jest od niej mniejsza, to po prostu ją wypisujemy.

```
cmp $CYFRA_UJEMNA, %rcx         #jesli 1. cyfra jest mniejsza
od 5, to wypisz całą liczbę
                    #dodatnią - jest to zarazem jej moduł
jl wypisz
```

Jeśli nie jest, przechodzimy do zamiany na liczbę przeciwną.

Aby to zrobić, należy skorzystać, ze wzoru:

$$-x = \bar{x} - x + 1$$

Gdzie $\bar{x}$ to liczba postaci (9)9…

Np. chcą zamienić 56 robimy:

(9)99-(9)56 = (0)43

(0)43 + 1 – (0)44

Można zauważyć, że jest to tożsame z wyrażeniem 100-56=(0)44. Jedynkę dodaliśmy wcześniej. Poniżej wykonujemy zamianę właśnie tą metodą.

Rejestr rdi to licznik. Rsi – kolejne potęgi 10.

Pętla musi wykonać się liczba cyfr razy np. 534 -> 1000.

```
movq $1, %rsi         #w rsi była pierwsza cyfra, już jest
niepotrzebna, bo zrobiono porównanie z 5


petla_dopelnienie:


cmp $0, %rdi
```

Jeśli skończyliśmy wyliczać 10^y, to przechodzimy do odjęcia:

```
jl wylicz_odwrotnosc
```

Następuje wyliczanie kolejnych potęg 10 i wpisywanie ich do rejestru rsi.

```
movq %rsi, %rax
movq $PODST_WEJ, %rbx
mul %rbx
movq %rax, %rsi         #wpisujemy potęgę do rsi

```

```
dec %rdi        #obniżenie indeksu
jmp petla_dopelnienie
```

Wyliczamy prawidłową wartość do wyświetlenia, czyli odejmujemy od liczby (znajduje się w rejestrze r8)

```
wylicz_odwrotnosc:
sbb %r8,%rsi        #rsi-> rsi – r8 = (999...+ 1) - liczba
jmp odwrotnosc_na_ascii_przygotowanie
```

Wypisanie wartości bezwzględnej na ASCII.

Najpierw rutynowe przygotowanie rejestrów. Musimy zwiększyć zawartość rejestru r11, który trzymał indeks ostatniej cyfry. Teraz chcemy mieć dokładną liczbę cyfr.

```
odwrotnosc_na_ascii_przygotowanie:


inc %r11
movq $0, %rdi
movq $0, %r9
mov %rsi, %rax            #z rsi do rax, aby wykonać dzielenie
jmp odwrotnosc_na_ascii
```

Pętla zamiany na ASCII i wpisanie do bufora textmod – wpisanie do bufora odbywa się od końca, więc konieczne będzie późniejsze przekonwertowanie z bufora textmod.

W rsi znajduje się wyliczona liczba bezwzględna.

```
odwrotnosc_na_ascii:
```

Zerujemy resztę. W rax znajduje się liczba bezwzględna – potrzebne do wykonania dzielenia.

```
movq $0, %rdx
div %rbx;
```

Zakodowanie liczby w ASCII i wpisanie do bufora (licznik to rdi).

```
add $POCZ_LICZB, %rdx        #dodanie 48 - zamiana na ascii
mov %dl, textmod(, %rdi,1)
inc %rdi    #zwiększenie licznika pętli
cmp %r11, %rdi    #porównanie licznika pętli z wartością ile
razy pętla ma się wykonać, a ma się
        #wykonać tyle razy, ile jest cyfr w systemie
wyjściowym - liczba tych cyfr w r11
```

```
jle odwrotnosc_na_ascii
jmp odwrocenie_odwrotnosci_przygotowanie
```

Później następuje odwrócenie kolejności cyfr w analogiczny sposób. Jedyne co się zmienia to bufor, do którego wpisujemy, teraz do bufora textout, z którego zostanie wypisany tekst na wyjście. Inaczej wygląda nieco warunek zakończenia pętli – teraz porównujemy czy licznik jest równy liczbie cyfr – wtedy pętla się zakończy, bo wszystkie cyfry zostały wpisane.

Do bufora textout zapisujemy od końca.

```
odwrocenie_odwrotnosci_przygotowanie:


movq $0, %rcx
movq %rdi, %rsi        #zapisanie do licznika rsi tego, co
bylo w liczniku rdi
            #a jest tam liczba cyfr zamienionej liczby na
system wyjsciowy
dec %rsi        #odejmujemy ostatni przypadek, w którym wynik
okazuje sie juz zerowy
            #tego zera nie należy rozpatrywać
jmp odwrocenie_odwrotnosci


odwrocenie_odwrotnosci:
movq textmod(,%rsi,1), %rax    #zapisznie do raxa ostatniej
cyfry wyniku
movq %rax, textout(, %rcx, 1)     #z raxa do wynikowego bufora
# w ten sposób tworzona jest dobra kolejność


inc %rcx    #zwiększenie licznika pętli
dec %rsi    #zmniejszenie licznika, który liczy cyfry
cmp %rdi, %rcx    #porównanie licznika pętli z wartością ile
razy pętla ma się wykonać, a ma się wykonać tyle razy, ile
jest cyfr w systemie wyjściowym - liczba tych cyfr
jle odwrocenie_odwrotnosci
jmp wypisz_odwrotnosc
```

Wypisanie odwrotności po dodaniu „/n” na koniec ciągu ASCII. W rejestrze r11 jest długość ciągu znaków.

```
wypisz_odwrotnosc:


inc %r11
movb $NOWA_LINIA, textout(, %r11, 1)        #w r11 to liczba cyfr wyniku
                    #bo r11 trzymal tylko do indeksu, a nie liczbe cyfr
inc %r11                    #bo dodaliśmy /n


movq $SYSWRITE, %rax
movq $STDOUT, %rdi
movq $textout, %rsi
movq %r11, %rdx        #długość ciągu znaków
syscall
```

Po tym następuje zakończenie programu, bo jeżeli liczba okazała się ujemna, należało tylko wyświetlić jej moduł. Liczba ujemna nie może być pierwsza, więc nie sprawdzamy warunku pierwszości.

Należy natomiast sprawdzić warunek pierwszości dla liczby dodatniej.

Czy liczba jest pierwsza?

Wcześniej wypisujemy jej moduł na ekranie (bez zmiany na wartość, jak zostało wcześniej wyjaśnione). Później sprawdzamy warunek pierwszości.

```
#Wypisuje po prostu wprowadzoną liczbę dodatnią - jest to zarazem jej moduł
#---------------------------------------------
wypisz:
movq $SYSWRITE, %rax
movq $STDOUT, %rdi
movq $textin, %rsi
movq $BUFLEN, %rdx        #długość ciągu znaków
syscall
```

Sprawdzenie warunku pierwszości odbywa się w następujący sposób: sprawdzamy dzielniki liczby od 2 do n-1. Jeśli reszta z dzielenia jest 0, znaczy, że liczba nie jest pierwsza. Jeśli reszta z żadnego dzielenia nie okaże się zerem, liczba jest pierwsza.

```
#-------------------------------------------CZY--PIERWSZA
#liczba n jest w r8

movq $2, %rsi             #zaczynamy sprawdzac dzielniki od 2
```

Należy uwzględnić wyjątki (2 oraz 1).

```
cmp %rsi, %r8             #wyjątek dla 2
je zapis_do_ascii_i_zamiana

cmp $1, %r8            #wyjątek dla 1
je komunikat_o_niepierwszosci
```

Sprawdzamy kolejne dzielniki i wyświetlamy odpowiedni komunikat.

```
czyPierwsza:

movq $0, %rdx
movq %r8, %rax
div %rsi
cmp $0, %rdx
je komunikat_o_niepierwszosci

inc %rsi
cmp %r8, %rsi
jl czy pierwsza
```

Jeśli liczba jest pierwsza, zamieniamy ją na system U7

```
jmp zapis_do_ascii_i_zamiana
```

Zamiana na system U7 (liczby pierwszej)

Aby dobrze zrozumieć program, należy przypomnieć sobie jak działa zamiana z systemu na inny system. Jeśli chcemy zamienić liczbę zapisaną w reprezentacji U10 na U7, to należy wykonać dzielenia, aż nie uzyskamy 0. Kolejne reszty to cyfry w systemie U7 zapisane od końca (tak działa zasada). Później należy je odwrócić do bufora textout z bufora textinv.

Po przygotowaniu rejestrów, rozpoczynamy pętlę dzielącą.

```
zapis_do_ascii_i_zamiana:


movq %r8, %rax        #zapisanie wyniku globalnego do raxa
movq $PODST_WYJ, %rbx    #zapisanej podstawy wyjściowej do rbx (rbx zawsze trzyma podstawy)
movq $0, %rcx        #wyzerowanie rcx - licznik
jmp zamiana_na_system_wyj
```

Pętla dzieląca liczbę w U10 przez 7.

```
zamiana_na_system_wyj:
movq $0, %rdx
div %rbx       #dzielenie bez znaku liczby z rejestru rax
               (wynik globalny) przez rbx
               #(podstawa wyjściowego systemu), zapis wyniku
               do rax, a reszta z dzielenia do rdx
#Reszta zapisana w rdx to część wyniku po konwersji
```

Zamiana na Ascii.

```
add $POCZ_LICZB, %rdx        #Dodanie kodu znaku pierwszej liczby - zakodowanie w ascii
mov %dl, textinv(,%rcx,1)    #zapisanie z dl do bufora w odwrotnej kolejności
inc %rcx                    #zwiększenie licznika
```

Przy każdej iteracji sprawdzamy, czy wynik nie jest już zerowy.

```
cmp $0, %rax                    #czy wynik dzielenia jest juz zerowy
jne zamiana_na_system_wyj            #jeśli nie, rób pętle zamiany na system dalej
```

Jeśli pętla się zakończyła, odwracamy cyfry w buforze.

```
jmp odwrocenie_kolejnosci_przygotowanie        #jeśli tak, odwróć kolejność
```

Odwrócenie kolejności, aby liczba nie była zapisana od końca. Odbywa się to w analogiczny sposób jak odwracanie w buforze w poprzednim przykładzie z wartością bezwzględną (dokładny kod załączony do sprawozdania).

Wyświetlenie wyniku - zamienionej liczby na system wyjściowy.

W buforze textout jest umieszczona wartość w U7 w odpowiedniej kolejności. Należy ją jedynie wypisać na ekran podobną instrukcją jak wartość bezwzględną.

```
wyswietl_wynik:


movb $NOWA_LINIA, textout(, %rcx, 1)        #w rcx liczba cyfr wyniku
inc %rcx                    #zwiększenie rcx, bo teraz jeszcze wpisaliśmy \n


movq $SYSWRITE, %rax
movq $STDOUT, %rdi
movq $textout, %rsi
movq %rcx, %rdx                     #długość ciągu znaków
syscall
```

Skoro liczba była zamieniana na U7, znaczy, że okazała się piewsza. Wypiszmy stosowny komunikat.

```
jmp komunikat_o_pierwszosci
```

Etykiety z odpowiednimi informacjami do wyświetlenia.

```
#-----------------------------------------------
#wyświetlanie komunikatu o błędzie
blad:
movq $SYSWRITE, %rax
movq $STDOUT, %rdi
movq $komunikat, %rsi
movq $komunikat_len, %rdx
syscall
jmp exit


#-----------------------------------------------
komunikat_o_niepierwszosci:
movq $SYSWRITE, %rax
movq $STDOUT, %rdi
movq $komunikat_niepierwsza, %rsi
```

```
movq $komunikat_niepierwsza_len, %rdx
syscall
jmp exit

#--------------------------------------------
komunikat_o_pierwszosci:
movq $SYSWRITE, %rax
movq $STDOUT, %rdi
movq $komunikat_pierwsza, %rsi
movq $komunikat_pierwsza_len, %rdx
syscall
jmp exit
```

Zakończenie programu.

```
exit:
#--------------------------------------------
#zakończenie programu
movq $SYSEXIT, %rax
movq $EXIT_SUCCESS, %rdi
syscall
```

Wnioski

Należy pamiętać, że dzielenie i mnożenie używają rejestru rax i wynik jest również w nim zapisywany. Należy wartość mnożoną i dzieloną przenieść wcześniej do rejestru rax.

Efektywne używanie rejestrów (np. niezerowanie ich wtedy, gdy nie ma potrzeby) jest w cenie.

W całym sprawozdaniu używam terminu odwrotność do liczby przeciwnej, w matematyce ma to nieco inne znaczenie i może wprowadzać w błąd – należałoby zatem zmienić jedynie etykiety.

Dobra ideą wydaje się być wpisywanie cyfr w bufor textout od razu od końca, zamiast najpierw do bufora trzymającego cyfry w odwrotnej kolejności – jednak nie udało mi się tego dokonać, gdyż program wypisywał mi wtedy tylko pierwszą cyfrę.

Źródła:

http://zak.ict.pwr.wroc.pl/materials/architektura/laboratorium%20AK2/Dokumentacja/Programowanie/Linux-asm-lab-2015.pdf

Tabela instrukcji na stronie:

http://zeszyt.olo.ovh/2016/02/28/architektura-komputerow-2-laboratorium-nr-1-podstawy-pisania-programow-w-jezyku-asembler/

Debugowanie (nauka):

http://students.mimuw.edu.pl/~zbyszek/asm/en/slides/debug.pdf