Kolegium Nauk Przyrodniczych
Uniwersytet Rzeszowski

Przedmiot:

Architektura Systemów Komputerowych

Temat sprawozdania

Dokumentacja projektu

Wykonał:

Bartłomiej Florek 125115

Prowadzący: Mgr inż. Jarosław Szkoła

Rzeszów 2024

Temat projektu: wyliczanie średniej na podstawie liczb wprowadzonych przez użytkownika z zabezpieczeniami.

```
finit
suma equ 0
licznik equ 0
```

Finit – inicjalizuje FPU, czyli koprocesor, dzięki któremu wyliczymy średnia na floatach.
Definicja zmiennych suma oraz licznik.

```
mov esi, ebx      ; esi = ebx

call liczba_iteracji

print:
      db "Ile liczb chcesz podac?: ", 0

liczba_iteracji:      ;esp [print][ret]
      call [esi+3*4]
      push esp       ;esp [----][zmienna][ret]
      call skanuj_liczbe
```

Na rejestr ESI zostanie odłożony rejestr EBX, po to aby później móc go wykorzystać.
Zostaje wywołana funkcja „liczba_iteracji”, która przed skokiem odłoży „print” na stos ESP.
W labelu „lilczba_iteracji” wywoływany jest print oraz robimy miejsce na stosie dla zmiennej przez instrukcje „push ESP”.

```
print2:
      db "%i", 0

skanuj_liczbe:        ;esp [%i][----][zmienna][ret]
      call [esi+4*4]
      add esp, 2*4    ;esp [zmienna][ret]
      call przypisanie
```

Call „skanuj_liczbe” zanim przeskoczy do tej etykiety, odłoży na stos „print2”. W etykiecie „skanuj_liczbe” wywołujemy metode scanf z rejestru ESI, a po zeskanowaniu liczby, usuwamy ze stosu print2 oraz wolną przestrzeń na ESP. Zmienna wprowadzona przez użytkownika zostanie przypisana do tego pierwszego printa.

```
print3:
      db "Ilosc liczb: %i", 0xA, 0

przypisanie:  ;esp [print3][zmienna][ret]
      call [esi+3*4]
      mov ecx, [esp+4] ; ILE PETLI SIE WYKONA

      cmp ecx, 0       ; porownanie czy wartosc nie jest ujemna
      jecxz domyslna      ; jesli ecx = 0 skocz do wartosci domyslnej
      jg poprawna           ; jesli > 0 skocz do negacji
      neg ecx          ; ecx = -ecx
```

Tak samo jak w poprzednich callach, zanim skoczymy, odkładamy na stos etykietę, która znajduje się zaraz pod tym wywołaniem, w tym przypadku „print3". W etykiecie „przypisanie" wyświetlamy jaką liczbę wprowadził użytkownik. Odkładamy na rejestr ECX (czyli rejestr licznikowy) naszą zmienną.
Następnie sprawdzamy przez instrukcje CMP, wartość na rejestrze ECX z liczba 0.
Instrukcja JECXZ, sprawdza czy wartość ECX jest równa 0, jeśli tak to skoczy do etykiety „domyślna", która ustawi wartość na 1 aby wprowadzić chociaż jedną liczbę.
Instrukcja JG, sprawdza czy wartość ECX, jest większa od 0, jeśli tak to skoczy do etykiety „poprawna".
Natomiast jeśli wartość na ECX jest mniejsza od 0 to wtedy wszystkie instrukcje skoku pominiemy i wykona się instrukcja NEG, która zaneguje tę wartość na wartość dodatnią i wykona się wszystko w etykiecie „poprawna"

```
poprawna:
      mov edi, suma    ;edi = 0
      mov ebx, licznik ;ebx = 0 licznik do dzielenia

      add esp, 2*4  ; esp [ret]

      jmp petla

domyslna:
      mov ecx, 1

      mov edi, suma
      mov ebx, licznik

      add esp, 2*4
      jmp petla
```

W etykiecie „poprawna" ustawiamy na rejestry EDI oraz EBX, odpowiednio sumę oraz licznik, które będą sumować wprowadzone liczby oraz inkrementować licznik. Następnie ze stosu usuwamy „print3" oraz „zmienna", które już wykorzystaliśmy i skaczemy bezwarunkowo przez instrukcje JMP do pętli. W etykiecie „domyslna" ustawiamy tylko wartość ECX na 1.

```
petla:
      push ecx ;esp [ecx][ret]

      call getaddr

      format:
            db "x = ", 0

      getaddr:   ;esp [zmienna][ecx][ret]
            call [esi+3*4]
            push esp   ;esp [----][zmienna][ecx][ret]
            call getaddr2
```

W pętli ustawiamy na ECX wprowadzoną przez użytkownika liczbę, czyli liczba iteracji. Uruchamiamy calla, odłożymy na stosie „format", w „getaddr" wywołujemy printa, i zostawiamy wolną przestrzeń na wprowadzoną liczbę. Uruchamiamy dobrze już nam znaną instrukcje call.

```
format2:
       db "%i", 0 ;%lf

getaddr2:   ;esp [%i][----][zmienna][ecx][ret]
       call [esi+4*4]
       add esp, 2*4   ;esp [zmienna][ecx][ret]
       call getaddr3
```

W getaddr2, wywołujemy scanf, i po wprowadzeniu przez użytkownika liczby usuwamy ze stosu. Nasza zmienna zostanie zapisana w „zmienna".

```
format3:
       db "wprowadzona liczba: %i", 0xA, 0

getaddr3:     ;esp [format3=][zmienna][ecx][ret]
       call [esi+3*4]

       mov eax, [esp+4]  ; do testu czy wartosc jest ujemna
       test eax, eax     ; test
       jge negat         ; skok jesli niej jest ujemna
       neg eax           ; eax = -eax
```

W getaddr3, wywołujemy printa i wypisujemy wprowadzona liczbę. Na rejestr EAX, ustawiamy nasza zmienna, i wykonując instrukcje TEST, która ustawi flagę SF na 1 jeśli EAX jest mniejszy od 0. Instrukcja JGE skoczy do etykiety „negat" jeśli spełnione są warunki, czyli jeśli wartość na rejestrze EAX jest dodatnia, natomiast jeśli nie jest dodatnia, wykona się jej negacja.

```
negat:
        add edi, eax ; edi = edi + zmienna
        add esp, 2*4  ;esp [ecx][ret]

        pop ecx   ;esp [ret]
        inc ebx   ;ebx++
        loop petla

        push edi   ;esp [edi][ret]
        call wypisz_sume
```

W etykiecie „negat” dodajemy do rejestru EDI, naszą zmienna zapisaną na rejestrze EAX, i usuwamy ze stosu „format” oraz „zmienna”. Ze stosu przez instrukcje POP usuniemy wartość ECX. INC EBX zwiększy nasz licznik do dzielenie za każda wprowadzoną liczbę, oraz LOOP petla, która sprawdzi czy wartość na ECX nie jest równa 0, jeśli nie jest to zmniejszy jej wartość oraz skoczy do etykiety petla. Jeśli ECX jest równy 0 to na stos wypchniemy EDI, czyli nasza sumę wszystkich wprowadzonych liczb.

```
laczna_suma:
      db "suma = %i", 0xA, 0

wypisz_sume:   ;esp [laczna_suma=][edi][ret]
      call [esi+3*4]

      fild qword [esp]   ;st = [st0] = [edi]

      add esp, 2*4  ;esp [ret]

      push ebx      ;esp[ebx][ret]
      call wypisz
```

W „wypisz_sume”, wywołujemy printa który wyświetli nasza sumę. FILD qword [esp] sprawi, że na koprocesor zostanie odłożona liczba z rejestru EDI. Następnie stos zostaje wyczyszczony, a wartość z rejestru EBX, czyli naszego licznik zostaje wypchnięta.

```
ile_liczb:
      db "ile liczb: %i", 0xA, 0

wypisz:     ;esp [ile_liczb][ebx][ret]
      call [esi+3*4]

      fild qword [esp]    ;st = [st0, st1] = [ebx, edi]

      add esp, 2*4  ;esp [ret]

      fdiv
      sub esp, 8
      fstp qword [esp]

      call koniec
```

Następują te same instrukcje co w przypadku wyświetlania sumy, czyli wywołanie printa, odłożenie na stos koprocesora wartość z rejestru EBX, oraz usuniecie ze stosu printa oraz EBX. FDIV operuje na koprocesorze, a dokładniej dzieli st1(nasza suma) przez st0(licznik). Następnie przygotowujemy miejsce na stosie aby zapisać wynik operacji dzielenia zmiennoprzecinkowej. FSTP qword [esp] zapisuje wartość z wierzchu stosu FPU na stosie ESP.

```
srednia:
      db "srednia: %0.2f", 0xA, 0

koniec:    ;esp [srednia][][][ret]
      call [esi+3*4]
      add esp, 3*4

      push 0       ;esp [0][ret]
      call [esi+0*4]
```

W etykiecie „srednia” wypisujemy średnia w zmiennoprzecinkowym formacie, z dokładnością do 2 miejsc po przecinku. W etykiecie koniec, wywołujemy printa, usuwamy ze stosu zbędne już miejsca i printa, a na koniec pushujemy na stos 0, aby móc wyjść z programu oraz wywołujemy funkcje exit z rejestru ESI.

Program jest również dostarczony w wersji .exe, który może zostać uruchomiony poprzez przejście do cmd, wejście do folderu, w którym się znajduje ten plik, i wpisanie jego nazwy, wtedy program się uruchomi w konsoli.

Projekt znajduje się na githubie: https://github.com/flor3kk/Assembly/tree/main/projekt

Aby uruchomić projekt w programie ConTEXT, musimy utworzyć folder, do którego skopiować program asmloader.exe oraz plik projekt_ask.asm a następnie wcisnąć F9 (kompilacja) oraz F10 (uruchomienie).