Katedra Inżynierii Komputerowej
Politechnika Częstochowska

Laboratorium
Programowania niskopoziomowego

Laboratorium 4

30 października 2013

1 Suma elementów wektora danych.

Należy policzyć sumę **N** elementów wektora danych pobranych od użytkownika, przy wykorzystaniu wstawki asemblerowej. Wstawkę asemblerową umieszczamy w osobnej funkcji C++.

Rozwiązanie zadania:

- Utworzenie nowego programu, dołączenie właściwych bibliotek
- Pobranie rozmiaru tablicy od użytkownika

```
cout << "Podaj liczbe el tab. :";
int N;
cin >> N;
```

- Dynamiczny przydział pamięci dla tablicy danych typu integer

```
int *tab = new int[N];
```

- Pobranie elementów wektora danych

```
for(int i=0;i<N;i++)
{
        cout << "tab[" << i << "]:";
        cin >> tab[i];
}
```

- Utworzenie zmiennej suma, do której zostanie przeniesiony wynik sumowania składników wektora danych we wstawce asemblerowej

```
int suma;
```

- Wprowadzenie wstawki asemblerowej, zapamiętanie stanu rejestrów procesora

```
__asm {
        push eax;
        push ebx;
        push ecx;
```

- W kolejnym kroku musimy utworzyć pętlę bazującą na instrukcji **loop**. Najbliższym odpowiednikiem pętli opartej na **loop** w języku C++ jest pętla **do** ... **while**. Program realizujący zadanie sumowania elementów wektora danych w języku C++ może wyglądać następująco:

```
//Odpowiednik petli Loop
//Sumujemy od ostatniego elementu
int suma (int *tab, int N)
{
        int suma = 0;
        N--;
        do
        {
                suma+=tab[N];
        } while (N-->0);
        return suma;
}
```

Podobnie jak w przypadku pętli **do** ... **while**, w asemblerze sami musimy zorganizować licznik pętli oraz warunek zakończenia działania pętli. Licznikiem dla pętli **loop** jest rejestr **ecx**. Tak więc przystępujemy do przygotowania zawartości rejestrów przed uruchomieniem pętli.

- Wprowadzenie niezbędnych danych do odpowiednich rejestrów

```
xor eax, eax; //zerowanie rejestru eax (wynik sumy)
mov ecx, N;       //pobranie liczby elementów tablicy
// wartość początkowa licznika pętli
mov ebx, tab; //pobranie adresu pierwszego elementu
```

- Pętla sumowania

```
Suma_Loop:
        add eax, [ebx + 4*ecx-4]; //eax = eax + [...]
                //odpowiednik w C++ -> suma+=tab[N];
        loop Suma_Loop; //skocz do etykiety jeżeli ecx>0
```

Instrukcja loop przy każdym wywołaniu zmniejsza zawartość rejestru **ecx**. Odpowiednik powyższego kodu w języku C++ przedstawia się następująco:

```
N--;
do
{
        suma+=tab[N];
} while (N-->0);
return suma;
```

- Wyprowadzenie danych do zmiennej suma

```
mov suma, eax;  //wyprowadzenie wyników
```

- Przywrócenie stanu rejestrów procesora
- Wypisanie wyników na konsolę

```
cout << "Suma wynosi:" << suma << endl;
system("Pause");
```

Instrukcja **loop** przy każdym wykonaniu zmniejsza automatycznie wartość rejestru **ecx** o 1, następnie przenosi wykonanie programu do wskazanej etykiety (Suma_Loop). Nawias kwadratowy w instrukcji **add** oznacza dowołanie do danych, które znajdują się pod wskazanym adresem w pamięci. Wewnątrz nawiasu na pierwszym etapie jest obliczany adres, następnie są pobierane dane z wyznaczonego adresu. Mnożnik 4 wynika z ilości bajtów dla typu integer. Domyślnie są to 4 bajty, dlatego do kolejnych elementów wektora danych należy przemieszczać się o 4 bajty. Wartość (-4) wynika z indeksowania tablic od wartości 0. **Uwaga proszę spróbować wpisać add eax, [ebx + 4*(ecx −1)] i podejrzeć skompilowany kod asemblera.**

2 Wyznaczanie sumy elementów tablicy przy użyciu instrukcji jnz.

Tworzymy nową funkcję C++. Do realizacji zadania można wykorzystać zmodyfikowaną wersję program z rozdziału 1, zastępując instrukcję **loop** instrukcją **jnz**.

jnz etykieta; - skocz do wskazanej etykiety jeżeli nie równe (nie zero), czyli póki ZF != 0. Wpływ na flagę ZF ma przykładowo instrukcja **dec**. Jeżeli zmniejszy ona wartość jednego z rejestrów do zera - to wtedy ZF==0.

Przykładowy kod programu w asemblerze realizujący pętlę:

```
        mov ecx,a;        //wart. pocz. dla ecx
label:
        //....  //inne instrukcje w pętli
        dec ecx;          //zmniejsz licznik pętli
        //instrukcja dec ustawi flagę ZF jeżeli ecx=0;
        jnz label;        //skocz do label jeżeli ecx>0, czyli ZF!=0;
                //jako indeks pętli może być użyty inny rejestr.
```

3 Zerowanie tablicy danych.

Proszę zmodyfikować tak wcześniejszy program aby przy użyciu pętli **loop** wyzerować tablicę **tab** o rozmiarze **N**. Zmodyfikowany kod programu umieszczamy w nowej funkcji C++.

4 Poszukiwanie maksymalnego elementu tablicy

Proszę napisać wstawkę asemblerową wyznaczającą maksymalny element tablicy **tab** o rozmiarze **N**. Zadanie realizujemy w kolejnej funkcji C++.

Rozwiązanie zadania

- Na pierwszym etapie należy napisać funkcję w C++ realizującą to zadanie. Należy zastosować pętlę **do** ... **while** tak, aby kod był jak najbardziej zbliżony do wersji asemblerowej.

```
int max_tab(int *tab, int N)
{
        int max; //wartość maksymalna
        max = tab[N-1]; //zakładam, że ostatni el. max
        N--;     //element wcześniej
        do
        {
          N--; //element wcześniej
          if (max<tab[N])
                  //porównanie przedostatni i ostatni (1 obieg)
                max=tab[N]; //jeżeli nie max to zapamiętaj nowy
        } while(N>0); //dopóki nie dojdzie do 0,
        // dla zerowego się wykona
        return max;
}
```

Przymiarka do instrukcji asemblerowych

```
int max_tab1(int *tab, int N)
{
        int max;
        max = tab[N-1]; //max -> eax, tab -> esi
        N--;     // N -> ecx
        do               // etykieta:
        {
          N--;
```

```
            if (max<tab[N])   //cmp oraz jnc
                    //porównanie przedostatni i ostatni (1 obieg)
                max=tab[N];   //eax = ...
        } while(N>0); //loop etykieta
        // dla zerowego się wykona
        return max;
}
```

- Utworzenie nowej funkcji wraz ze wstawką asemblerową

```
int max_tab_asm(int *tab,int N)
{
        int max;
        __asm
        {
```

- Przygotowanie odpowiednich rejestrów

```
        mov ecx,N;        //rozmiar tablicy
        mov esi,tab; //adres pierwszego elementu
        mov eax, [esi + 4*ecx-4];   //max = tab[N-1];
        dec ecx;          // na przedostatni //N--;
```

- Utworzenie odpowiedniej pętli przy wykorzystaniu instrukcji **loop**

```
Lab_loop:
                //instrukcje wewnątrz pętli

        loop Lab_loop;   //while(N>0)
```

- Dopisanie kodu odpowiedzialnego za warunek **if**(max<tab[N]) wewnątrz pętli **loop**

```
        cmp eax,[esi +4*ecx-4]; //if(max<tab[N])
        jnc Lab_pomin;
        mov eax,[esi +4*ecx-4]; //{max = tab[N]}
Lab_pomin:
```

Powyższy kod odpowiada fragmentowi w języku C++

```
            if (max<tab[N])   //cmp oraz jnc
                    //porównanie przedostatni i ostatni (1 obieg)
                max=tab[N];   //eax = ...
```

- Pozostało zwrócenie maksymalnej wartości z rejestru **eax** do właściwej zmiennej w C++

```
        mov max, eax;    //wyprowadzenie maksymalnej wartości
```

5 Poszukiwanie minimalnego elementu tablicy

Proszę napisać nową funkcję C++, w której umieszczamy zmodyfikowaną wersję wcześniej napisanego programu tak, aby wyszukiwał minimalny element tablicy. Do budowy pętli należy wykorzystać instrukcję **jnz**.

6 Wyznaczanie iloczynu skalarnego wektorów

Iloczyn skalarny wektorów oblicza się zgodnie z następującym wzorem:

$$w = \sum_{i=1}^{N} a_i \cdot b_i \tag{1}$$

gdzie: **a**,**b** - są wektorami, N - liczba elementów wektora.

W zadaniu należy napisać w asemblerze funkcję wyznaczającą iloczyn skalarny dwóch wektorów o wartościach oraz rozmiarze pobranym od użytkownika.

Wskazówki: Do przemieszczania się po wektorach można przykładowo wykorzystać dwa rejestry **esi** oraz **edi**, iloczyn składowych wektora umieszczamy w rejestrze **eax**, sumę możemy umieścić w rejestrze **ebx**. Zakładamy, że nie nastąpi przepełnienie w przypadku wykonywania operacji arytmetycznych. Uwaga na użycie rejestru **edx** przy wykonywaniu mnożenia. Podczas tej operacji jest on zerowany (jeżeli nie nastąpi przepełnienie) lub zostaje tam umieszczona wartość nadmiaru z operacji mnożenia.