Katedra Inżynierii Komputerowej
Politechnika Częstochowska

Laboratorium Programowania niskopoziomowego

Laboratorium 2

dr inż. Dziwiński Piotr

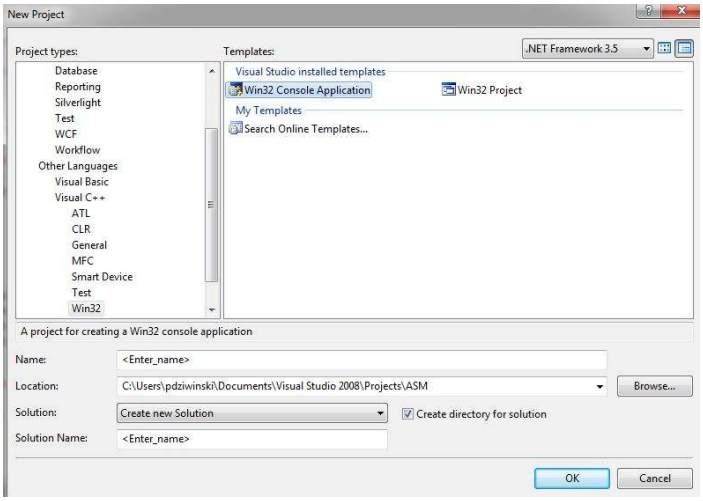
18 października 2011

1 Suma dwóch zmiennych

Należy policzyć sumę dwóch zmiennych **a** oraz **b** pobranych od użytkownika, przy wykorzystaniu wstawki asemblerowej.

Budowa prostego programu obliczającego sumę dwóch zmiennych

- Tworzymy w Visual Studio 2008 nowy projekt Visual C++ Win32 Application,



- Wprowadzamy fragment kodu w C++ pobierający dane od użytkownika

```
#include "stdafx.h"
#include <iostream>

using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
int a;
int b;
cout << "Podaj a:";
cin >> a;
cout << "Podaj b:";
cin >> b;
int wynik;
```

- Wprowadzamy wstawkę asemblerową

```
__asm {

}
```

Dostęp do zmiennych tworzonych w dowolnym miejscu kodu C++ jest bezpośredni - tzn. wartość z takiej zmiennej można kopiować bezpośrednio do rejestru przy użyciu instrukcji **mov**. Instrukcja ta przenosi wartość ze źródła (drugi argument) do przeznaczenia (pierwszy argument). Składnia instrukcji **mov** przedstawia się następująco:

```
mov reg,mem;
mov mem,reg;
mov reg,reg;
mov mem,mem;   // Uwaga - niedozwolony zestaw argumentów;
//mem - oznacza pamięć, zmienną, reg - oznacza rejestry procesora;
//przykład
mov eax,ebx;
mov eax,b;          //przeniesienie wartości zmiennej do rejestru eax;
mov b,eax;          //przeniesienie wartości rejestru eax do zmiennej b;
```

Tak więc, przeniesienie wartości ze zmiennej a do rejestru **eax** będzie wyglądało następująco:

```
        mov eax, a;
```

Do policzenia sumy dwóch zmiennych potrzebne są 2 rejestry **eax**, **ebx** - przenosimy wartości zmiennych do rejestrów:

```
        mov eax, a;
        mov ebx, b;
```

Sumowanie wykonuje się przy użyciu instrukcji **add** o następującej składni:

```
add cel,źródło;
//cel - mem lub reg, źródło - mem lub reg;
//cel oraz źródło nie może być równocześnie odniesieniem do pamięci;
//Przykład:
add eax, c; //c - zmienna;
```

Wykonujemy operację sumowania w asemblerze

```
        add eax, ebx;
```

Zwracamy wynik do zmiennej **wynik**

```
        mov wynik, eax;
```

Pozostaje wyświetlenie wyników na ekranie konsoli

```
cout << "Wynik wynosi: " << wynik;
```

Czy taki program będzie działał zawsze poprawnie? Otóż nie, jego poprawność zależy od tego co znajdowało się w rejestrach przed wywołaniem pierwszej instrukcji, czy zawartość rejestrów po ukończeniu wstawki asemblerowej jest wykorzystywana.

```
        mov eax, a;
```

Program należy uzupełnić o instrukcje, których zadaniem jest zapamiętanie stanu rejestrów przed przystąpieniem do właściwej pracy, oraz odtworzenie ich stanu po ukończeniu pracy:

- **push** - połóż na stosie,

```
        push eax;
        push ebx;
```

- **pop** - zdejmij ze stosu,

```
        pop ebx;
        pop eax;
```

Proszę uzupełnić programy o instrukcje zabezpieczające właściwe rejestry procesora.

2 Wyznaczanie maksymalnej i minimalnej wartości

W programie należy wyznaczyć wartość maksymalną dwóch zmiennych a oraz b, przy użyciu wstawki asemblerowej. Wstawkę asemblerową umieszczamy w osobnej funkcji C++
`int max(int a, int b) {}`.

Rozwiązanie zadania:

- Pobranie danych od użytkownika

```
        cout << "Podaj a: ";
        int a;
        cin >> a;
        cout << "Podaj b: ";Rr
        int b;
        cin >> b;
```

- Definicja metody wyznaczającej wartość maksymalną

```
int Max(int a,int b)
```

- Utworzenie zmiennej lokalnej z wynikiem

```
        int wynik;
```

- Wewnątrz wstawki asemblerowej pobieramy wartość zmiennej **a** do rejestru **eax**, oraz porównujemy ze zmienną **b**, co skutkuje zmianą stosownych flag. Instrukcja **cmp** działa tak jak instrukcja **sub**, z tą różnicą, że nie zwraca wyniku działania, a wpływa jedynie na flagi. Składnia instrukcji cmp przedstawia się następująco:

```
cmp cel,źródło;  // cel - źródło -> zmiana flag
//cel - reg,mem; źródło - reg, mem;
//cel oraz źródło nie może być równocześnie odniesieniem do pamięci;
```

```
        mov eax, a;
        cmp eax, b;
```

- Następnie sprawdzamy flagę **CF** (flagę przeniesienia) instrukcją **jnc** wykonując ewentualnie skok do etykiety **Mexit**. Zależnie od wyniku wcześniejszego wykonania instrukcji **cmp** wykonuje się instrukcja poprzedzająca lub tylko następna po etykiecie **Mexit**. Składnia instrukcji **jnc** przedstawia się następująco:

```
jnc etykieta;   //w programie musi się znaleźć gdzieś [etykieta:]
// skocz jeżeli flaga (CF = 0)
```

```
        jnc Mexit; //skocz do Mexit jeżeli flaga CF nie ustawiona
        mov eax, b; //jeżeli jest ustawiona to b jest większe
        //w wyniku realizacji instrukcji cmp wykonano pożyczkę CF=1
Mexit:
        mov wynik, eax;
```

- Pozostaje zwrócenie wyników

```
return wynik;
```

- Dodatkowe zabezpieczenie rejestru flag przed przystąpieniem do właściwych operacji:

```
pushf;          //zabezpieczenie rejestru flag na stosie
```

- Przywrócenie rejestru flag

```
popf            //przywrócenie rejestru flag ze stosu
```

Analogicznie należy napisać wstawkę asemblerową obliczającą wartość minimum w oddzielnej funkcji.