Katedra Inżynierii Komputerowej
Politechnika Częstochowska

Laboratorium Programowania niskopoziomowego

Laboratorium 3

dr inż. Dziwiński Piotr

18 października 2011

1 Implementacja instrukcji warunkowej if

Proszę napisać wstawkę asemblerową realizującą warunek logiczny taki, jak w poniższym kodzie programu

```
int if_1(int a, int b)
{
        int x=0;
        if (a<b)
                x = b - a;
        return x;
}
```

Rozwiązanie:

```
__asm {
        mov ebx,b;
        cmp a,ebx;
        jnl pomin;
        sub ebx,a;
        mov x, ebx;
pomin:
}
```

Proszę napisać wstawkę asemblerową realizującą warunek logiczny taki, jak w poniższym kodzie programu

```
int if_2(int a, int b)
{
        int x=0;
        if (a<b)
                x = b - a;
        else
                x = a - b;
        return x;
}
```

Rozwiązanie:

```
__asm {
        mov eax,a;
        mov ebx,b;
        cmp eax,ebx;
        jnl pomin;
        sub ebx,eax;
        mov x, ebx;
        jmp pomin2;
pomin:
        sub eax,ebx;
        mov x, eax;
pomin2:
}
```

2 Zadanie 1

Proszę napisać wstawkę asemblerową realizującą warunek logiczny taki, jak w poniższym kodzie programu

```
int if_3(int a,int b)
{
        int x=0;
        if (a>b)
                x = b*b - a;
        return x;
}
```

3 Zadanie 2

Proszę napisać wstawkę asemblerową realizującą warunek logiczny taki, jak w poniższym kodzie programu

```
int if_4(int a,int b)
{
        int x;
        if (a>=b)
                x = a*a + b;
        else
                x = a*a - b;
        return x;
}
```

4 Konstrukcja switch ... case

Proszę napisać wstawkę asemblerową realizującą instrukcje **switch** ... **case** tak, jak w poniższym kodzie programu

```
int case_1(int x, int op)
{
        int wynik;
        switch (op)
        {
        case 3: wynik = x +7;
                break;
        case 5: wynik = x * x;
                break;
        default:
                wynik =0;
        }
        return wynik;
}
```

Rozwiązanie:

```
int case_1_asm(int x, int op)
{
        int wynik;
__asm {
        mov ecx, op;     // op
        mov eax, x;              // x
        cmp ecx,3;               // case 3:
        je      case3;
        cmp ecx,5;               // case 5:
        je      case5;
        jmp case_default;        //jeżeli nic nie pasuje to default
case3:
        add eax, 7;
        jmp case_nic;    //jeżeli było break
case5:
        mul eax;
        jmp case_nic;    //jeżeli było break
case_default:
        mov eax, 0;
case_nic:
        mov wynik, eax;
}
}
```

5 Zadanie 3

Proszę napisać wstawkę asemblerową realizującą instrukcje **switch** ... **case** tak, jak w poniższym kodzie programu

```
int case_2(int x, int op)
{
        int wynik;
        switch (op)
        {
        case 0: wynik = x * x;
                break;
        case 1: wynik = x * (x-3);

        case 2: wynik = x * (x - 5);
                break;
        default:
                        wynik = x;
                break;
        }
        return wynik;
}
```

6 Organizowanie obiegu pętli

Proszę napisać wstawkę asemblerową realizujący pętlę **for** tak, jak w poniższym kodzie programu. Pętle należy zaimplementować przy wykorzystaniu instrukcji skoku **jnz** oraz **loop**.

```
int for_example(int N)
{
        int suma=0;
        for(int i=0;i<N;i++)
        {
                suma+=i;
        }
        return suma;
}
```

Rozwiązanie 1 zawierające pełną implementację pętli:

```
int for_example_asm1(int N)
{
int suma;
__asm
{
        mov ecx, N;
        xor eax,eax;
        xor ebx,ebx;   //int i=0;

skok:
        cmp ebx,ecx;   // i<N
        jae koniec;   // to koniec pętli
        //fragment pętli
        add eax,ebx;
        inc ebx;
        jmp skok;
koniec:

        mov suma,eax;
}
        return suma;
}
```

Opis instrukcji:

- **cmp** - wykonuje odejmowanie (przeznaczenie - źródło), zależnie od wyniku ustawia odpowiednie flagi, z których korzystają później instrukcje skoku,
- **jae** - skocz (**j**ump) jeżeli większe (**a**bove) lub równe (**e**qual),
- **jmp** - skok bezwarunkowy.

Rozwiązanie 2 zawierające najprostszą implementację pętli przy wykorzystaniu instrukcji **jnz** (skocz jeżeli nie zero(**n**ot **z**ero)):

```
__asm
{
        xor eax,eax;
        mov ecx, N;
        dec ecx;

skok1:
        add eax,ecx;
        dec ecx;
        jnz skok1;

        mov suma,eax;
}
        return suma;
}
```

Rozwiązanie 3 zawierające najprostszą implementację pętli przy wykorzystaniu instrukcji **loop**:

```
__asm
{
        xor eax,eax;
        mov ecx, N;
        dec ecx;

skok1:
        add eax,ecx;
        loop skok1;

        mov suma,eax;
}
```

Pytania problemowe:

- Jaką konstrukcję pętli w języku C++ przypomina najprostsza postać pętli w asemblerze?
- Czy postać pętli for jest łatwa do przepisania na dosłowny kod asemblera?

7 Zadanie 4

Proszę napisać wstawkę asemblerową realizującą pętlę **for** taką, jak w poniższym kodzie programu. Należy napisać dwie implementacje przy wykorzystaniu instrukcji skoku **jnz** oraz **loop**.

```
int silnia_for(int N)
{
        int silnia=1;
        for(int i=1;i<=N;i++)
        {
                silnia = silnia *i;
        }
        return silnia;
}
```

8 Zadanie 5

Proszę napisać wstawkę asemblerową realizującą pętlę **while** taką, jak w poniższym kodzie programu.

```
int while_example(int N)
{
        int wynik;
        int i=1;
        while(i<=2*N)
        {
                wynik+=i;
        }
        return wynik;
}
```