Laboratorium programowania niskopoziomowego

LAB 14 – Operacje przy użyciu instrukcji wektorowych AVX.

O P I S

Forma zajęć: przykładowe zadania do analizy

Student na zajęciach powinien:

A. przekopiować kod i ewentualnie uzupełnić go odpowiednimi otoczkami C++
B. opowiedzieć co się dzieje w kodzie i jak go rozumie
C. odpowiedzieć na pytania prowadzącego zajęcia

WIELOMIANY

Zadanie 1: obliczanie wielomianu postaci: $ax^3 + bx^2 + cx + d$ dla wartości typu double

```
double wielomian_v0(double a, double b, double c, double d, double x)
{
    return a*x*x*x + b*x*x + c*x + d;
}

double wielomian_v1(double a, double b, double c, double d, double x)
{
    double wynik = 0;
    __asm
    {
        vmovsd    xmm0, a;
        vmovsd    xmm1, b;
        vmovsd    xmm2, c;
        vmovsd    xmm3, d;
        vmovsd    xmm4, x;
        vmulsd    xmm0, xmm0, xmm4; ax
        vmulsd    xmm2, xmm2, xmm4; cx
        vmulsd    xmm4, xmm4, xmm4; xx
        vmulsd    xmm0, xmm0, xmm4; axxx
        vaddsd    xmm0, xmm0, xmm3; axxx + d
        vmulsd    xmm1, xmm1, xmm4; bxx
        vaddsd    xmm0, xmm0, xmm2; axxx + cx + d
        vaddsd    xmm0, xmm0, xmm1; axxx + bxx + cx + d
        vmovsd    wynik, xmm0;
    }
    return wynik;
}

double wielomian_v2(double a, double b, double c, double d, double x)
{
    double wynik = 0;
    __asm
    {
        vmovsd    xmm0, a;
        vmovsd    xmm1, b;
        vmovsd    xmm2, c;
        vmovsd    xmm3, d;
        vmovsd    xmm4, x;
        vmulsd    xmm0, xmm0, xmm4; ax
        vfmadd231sd    xmm3, xmm2, xmm4; cx + d
        vmulsd    xmm4, xmm4, xmm4; xx
        vfmadd132sd    xmm0, xmm3, xmm4; axxx + cx + d
```

```
        vfmadd231sd     xmm0, xmm1, xmm4; axxx + bxx + cx + d
        vmovsd     wynik, xmm0;
    }
    return wynik;
}

double wielomian_v3(double a, double b, double c, double d, double x)
{
    double wynik = 0;
    __asm
    {
        vmovsd     xmm0, a;
        vmovsd     xmm1, b;
        vmovsd     xmm2, c;
        vmovsd     xmm3, d;
        vmovsd     xmm4, x;
        vfmadd132sd     xmm0, xmm1, xmm4; ax + b
        vfmadd231sd     xmm3, xmm2, xmm4; cx + d
        vmulsd     xmm4, xmm4, xmm4; xx
        vfmadd132sd     xmm0, xmm3, xmm4; (ax + b)xx + cx + d
        vmovsd     wynik, xmm0;
    }
    return wynik;
}

double wielomian_v4(double a, double b, double c, double d, double x)
{
    double wynik = 0;
    __asm
    {
        vmovsd     xmm0, x;
        vmovsd     xmm1, a;
        vmovsd     xmm2, b;
        vmovsd     xmm3, c;
        vmovsd     xmm4, d;
        vfmadd213sd     xmm1, xmm0, xmm2; ax + b
        vfmadd213sd     xmm3, xmm0, xmm4; cx + d
        vmulsd     xmm0, xmm0, xmm0; xx
        vfmadd213sd     xmm0, xmm1, xmm3; (ax + b)xx + cx + d
        vmovsd     wynik, xmm0;
    }
    return wynik;
}

double wielomian_v5(double a, double b, double c, double d, double x)
{
    double wynik = 0;
    __asm
    {
        vmovsd     xmm0, x;
        vmovsd     xmm1, a;
        vmovsd     xmm2, b;
        vmovsd     xmm3, c;
        vmovsd     xmm4, d;
        vfmadd213sd     xmm1, xmm0, xmm2; ax + b
        vfmadd213sd     xmm1, xmm0, xmm3; (ax + b)x + c
        vfmadd213sd     xmm0, xmm1, xmm4; ((ax + b)x + c)c + d
        vmovsd     wynik, xmm0;
    }
    return wynik;
}
```

TRANSPONOWANIE

Zadanie 2: Transponowanie macierzy dynamicznej o rozmiarze 4 x 4 elementy DOUBLE/INT6 w systemie x84

```
void Transponuj4x4(double **tab)
{
    __asm
    {
    //  ; transponuje macierz tab o domyślnym rozmiarze 4x4
    //  ; elementy qword lub double/INT64 lub zespolone single
            push   esi;
            mov    esi, tab
            mov    eax, [esi]
            mov    ecx, [esi + 8]
            mov    edx, [esi + 12]
            mov    esi, [esi + 4]

            vmovdqu ymm0, ymmword ptr[eax]
            vmovdqu ymm1, ymmword ptr[ecx]
            vperm2i128 ymm2, ymm0, ymm1, 20h
            vperm2i128 ymm4, ymm0, ymm1, 31h

            vmovdqu ymm0, ymmword ptr[eesi]
            vmovdqu ymm1, ymmword ptr[edx]
            vperm2i128 ymm3, ymm0, ymm1, 20h
            vperm2i128 ymm5, ymm0, ymm1, 31h

            vpunpcklqdq ymm0, ymm2, ymm3
            vpunpckhqdq ymm1, ymm2, ymm3
            vpunpcklqdq ymm2, ymm4, ymm5
            vpunpckhqdq ymm3, ymm4, ymm5

            vmovdqu  ymmword ptr[eax], ymm0
            vmovdqu  ymmword ptr[eesi], ymm1
            vmovdqu  ymmword ptr[ecx], ymm2
            vmovdqu  ymmword ptr[edx], ymm3
            pop esi;
    }
}
```

Zadanie 3: Transponowanie macierzy dynamicznej o rozmiarze 8 x 8 elementy FLOAT/INT w systemie x64

```
MTD8x8 proc uses rbx
local    y8:ymmword,y9:ymmword,y10:ymmword,y11:ymmword
;transponuje macierz dynamiczną tab 8x8
;elementy dword lub single (int/float)

    vmovdqu    y8, ymm8
    vmovdqu    y9, ymm9
    vmovdqu    y10, ymm10
    vmovdqu    y11, ymm11
```

```
    mov     r8,[rcx+32]
    mov     r9,[rcx+40]
    mov     r10,[rcx+48]
    mov     r11,[rcx+56]
    mov     rax,[rcx]
    mov     rbx,[rcx+8]
    mov     rdx,[rcx+24]
    mov     rcx,[rcx+16]

    vmovdqu ymm0, ymmword ptr[rax]
    vmovdqu ymm1, ymmword ptr[r8]
    vperm2i128 ymm2, ymm0, ymm1, 20h
    vperm2i128 ymm8, ymm0, ymm1, 31h

    vmovdqu ymm0, ymmword ptr[rbx]
    vmovdqu ymm1, ymmword ptr[r9]
    vperm2i128 ymm3, ymm0, ymm1, 20h
    vperm2i128 ymm9, ymm0, ymm1, 31h

    vmovdqu ymm0, ymmword ptr[rcx]
    vmovdqu ymm1, ymmword ptr[r10]
    vperm2i128 ymm4, ymm0, ymm1, 20h
    vperm2i128 ymm10, ymm0, ymm1, 31h

    vmovdqu ymm0, ymmword ptr[rdx]
    vmovdqu ymm1, ymmword ptr[r11]
    vperm2i128 ymm5, ymm0, ymm1, 20h
    vperm2i128 ymm11, ymm0, ymm1, 31h

; »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»

    vpunpcklqdq ymm0, ymm2, ymm4
    vpunpcklqdq ymm1, ymm3, ymm5
    vpunpckhqdq ymm2, ymm2, ymm4
    vpunpckhqdq ymm3, ymm3, ymm5

    vpunpcklqdq ymm4, ymm8, ymm10
    vpunpcklqdq ymm5, ymm9, ymm11
    vpunpckhqdq ymm8, ymm8, ymm10
    vpunpckhqdq ymm9, ymm9, ymm11

; »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»

    vpunpckldq ymm10, ymm8, ymm9
    vpunpckhdq ymm11, ymm8, ymm9
    vpunpckldq ymm8, ymm4, ymm5
    vpunpckhdq ymm9, ymm4, ymm5
    vpunpckldq ymm4, ymm2, ymm3
    vpunpckhdq ymm5, ymm2, ymm3
    vpunpckldq ymm2, ymm0, ymm1
    vpunpckhdq ymm3, ymm0, ymm1

    vpunpcklqdq ymm0, ymm2, ymm3
    vpunpckhqdq ymm1, ymm2, ymm3
    vpunpcklqdq ymm2, ymm4, ymm5
    vpunpckhqdq ymm3, ymm4, ymm5
    vpunpcklqdq ymm4, ymm8, ymm9
    vpunpckhqdq ymm5, ymm8, ymm9
    vpunpcklqdq ymm8, ymm10, ymm11
    vpunpckhqdq ymm9, ymm10, ymm11
```

```
    vmovdqu  ymmword ptr[rax], ymm0
    vmovdqu  ymmword ptr[rbx], ymm1
    vmovdqu  ymmword ptr[rcx], ymm2
    vmovdqu  ymmword ptr[rdx], ymm3
    vmovdqu  ymmword ptr[r8], ymm4
    vmovdqu  ymmword ptr[r9], ymm5
    vmovdqu  ymmword ptr[r10], ymm8
    vmovdqu  ymmword ptr[r11], ymm9

    vmovdqu    ymm8, y8
    vmovdqu    ymm9, y9
    vmovdqu    ymm10, y10
    vmovdqu    ymm11, y11
    ret
MTD8x8 endp
```

ILOCZYN SKALARNY

Zadanie 3: Iloczyn skalarny dwóch wektorów typu DOUBLE o długości N

```
double iloczynVektor(double* tab1, double *tab2, int n)
{
    double sum = 0;
    double zerod = 0;
    __asm {

            push esi
            push edi

            vbroadcastsd    ymm0, zerod;
            mov     ecx, n;
            mov     esi, tab1;
            mov     edi, tab2;
            shl     ecx, 3
            p2:
            sub     ecx, 32
            vmovupd ymm1, ymmword ptr[esi + ecx]
            vmulpd  ymm1, ymm1, ymmword ptr[edi + ecx]
            vaddpd  ymm0, ymm0, ymm1
            jnz     p2
            vperm2f128 ymm1, ymm0, ymm0, 1; lo 1 : = hi 0
            vaddpd   ymm0, ymm0, ymm1
            vpermilpd ymm1, ymm0, 1
            vaddpd    ymm0, ymm0, ymm1
            vmovsd    sum, xmm0;

            pop edi
            pop esi
    }
    return sum;
}
```

```
double iloczynVektor2(double* tab1, double *tab2, int n)
{
    double sum = 0;
    double zerod = 0.0;
    __asm {
            push esi
            push edi
            vbroadcastsd    ymm0, zerod;
            vmovupd ymm2, ymm0
            vmovupd ymm4, ymm0
            vmovupd ymm6, ymm0

            mov     ecx, n;
            mov     esi, tab1;
            mov     edi, tab2;
            shl     ecx, 3
            p1:
            sub     ecx, 128
            vmovupd ymm7, ymmword ptr[esi + ecx]
            vmovupd ymm1, ymmword ptr[edi + ecx]
            vmulpd  ymm7, ymm7, ymm1
            vaddpd  ymm6, ymm6, ymm7
            vmovupd ymm5, ymmword ptr[esi + ecx + 32]
            vmovupd ymm1, ymmword ptr[edi + ecx + 32]
            vmulpd  ymm5, ymm5, ymm1
            vaddpd  ymm4, ymm4, ymm5
            vmovupd ymm3, ymmword ptr[esi + ecx + 64]
            vmovupd ymm1, ymmword ptr[edi + ecx + 64]
            vmulpd  ymm3, ymm3, ymm1
            vaddpd  ymm2, ymm2, ymm3
            vmovupd ymm1, ymmword ptr[esi + ecx + 96]
            vmovupd ymm7, ymmword ptr[edi + ecx + 96]
            vmulpd  ymm1, ymm1, ymm7
            vaddpd  ymm0, ymm0, ymm1

            jnz     p1
            vaddpd   ymm0, ymm0, ymm2
            vaddpd   ymm4, ymm4, ymm6
            vaddpd   ymm0, ymm0, ymm4
            vperm2f128 ymm1, ymm0, ymm0, 1; lo 1 : = hi 0
            vaddpd   ymm0, ymm0, ymm1

            vpermilpd ymm1, ymm0, 1
            vaddpd    ymm0, ymm0, ymm1
            vmovsd    sum, xmm0

            pop edi
            pop esi
    }
    return sum;
}
```

Zadanie 4: Iloczyn macierzy kwadratowych(NxN) o dowolnym rozmiarze wielokrotności liczby 2 typu double na rejestrach 128 bitowcyh XMM (SSE)

```
void iloczynMatrixXMM(double **tab1, double **tab2, double **tab3, int rozmiar)
{
    double zerod = 0.0;
    __asm
    {
            push   edi
            push   esi
            push   ebx

            mov    eax, rozmiar
            mov    esi, tab2
            i:
            mov    edi, tab1
            mov    edi, [edi + 4 * eax - 4]
            mov    edx, rozmiar
            j:
            movddup    xmm0, zerod;
            mov    ecx, rozmiar
            mov    ebx, [esi + 4 * ecx - 4]
            k:
            movddup xmm1, qword ptr[edi + 8 * ecx - 8]
            dec     ecx
            movupd  xmm2, xmmword ptr[ebx + 8 * edx - 16];
            mulpd   xmm1, xmm2;
            mov     ebx, [esi + 4 * ecx - 4]
            addpd   xmm0, xmm1
            jnz    k

            sub    edx, 2
            mov    ebx, tab3
            mov    ebx, [ebx + 4 * eax - 4]
            movupd    xmmword ptr[ebx + 8 * edx], xmm0;
            jns    j
            dec    eax
            jnz    i


            pop    ebx
            pop    esi
            pop    edi
    }
}
```