

به نام خدا

گزارش کار آزمایشگاه ریزپردازنده

آزمایش ۱۱

مدرس : مهندس بی طالبی

تارا برقیان

مهرشاد سعادت‌نی نیا

نیم سال اول ۱۴۰۰-۰۱



## فهرست

۳	..... مقدمه :
۳	..... توضیح کد:

## مقدمه :

در این تمرین با ترکیب برنامه نویسی زبان اسمبلی ۸۰۸۶ و C++ برای اجرای دستورات SIMD آشنا شدیم.

## توضیح کد:

ابتدا یک تابع برای تشخیص علامت داریم detectSign()

سپس برای ورودی گرفتن از کاربر از کتابخانه conio استفاده کردیم. برای کامل خواندن اعداد و تشخیص علامت و اعشار آن را کاراکتر به کاراکتر خواندیم.

```
float getInput()
{
    // sign = 1 [number is positive]
    // sign = -1 [number is negative]
    int sign = 1;
    int ch = '0';
    int zero = '0';
    float inputNumber = 0;
    int place = 1;

    ch = _getche();

    // determine sign of input number
    sign = detSign(ch);
    if (sign != 0) {
        ch = zero;
    }
    else {
        sign = 1;
    }

    // get integer part of input number
    while (ch >= '0' && ch <= '9')
    {
        inputNumber = (inputNumber * 10) + (ch - zero);
        ch = _getche();
    }

    if (ch == '.') {
        // get fraction
        ch = zero;
        while (ch >= '0' && ch <= '9')
        {
            inputNumber += ((ch - zero) / (float)place);
            place *= 10;
            ch = _getche();
        }
        return (float)sign * inputNumber;
    }
    else {
        return (float)sign * inputNumber;
    }
}
```

```
6
7 int detSign(int ch) {
8     if (ch == '-')
9     {
10         return -1;
11     }
12     else if (ch == '+')
13     {
14         return 1;
15     }
16     return 0;
17 }
```

سپس با کمک زبان `cpp` مقدار های `x+a` و `x+b` را حساب کرده و حاصل ضربشان را هم محاسبه میکنیم

و سپس به زبان اسمبلی بر میگردیم و مجدد ضرب دو وکتور را با اسمبلی حاسبه میکنیم. ( با کمک دستور `mulps` )

چون طبق توضیحات منابع در پیوست اینگونه عملیات مورد نظر `packed` خواهد بود.

توضیحات دستورات `SSE` بطور مفصل در داکيومنت `0x86` در سایت `oracle` موجودست.

کد زیر قسمت محاسبه ی حاصل ضرب به کمک اسمبلی است که با استفاده از `_asm block` می توانیم آنرا داخل کد `c++`

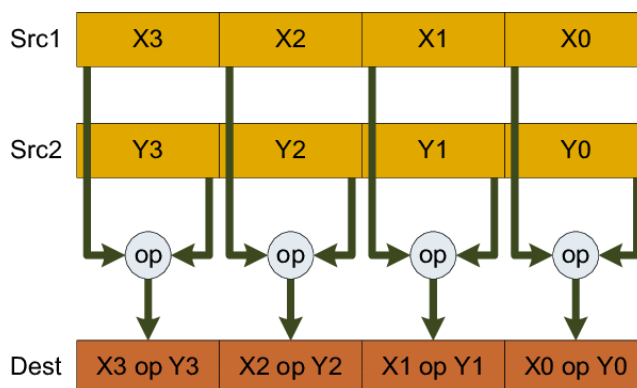
بنویسیم.

```
_asm
{
    mov ecx, 0
    MUL_LOOP:
    movaps xmm2, oword ptr vi1Copy[ecx]
    movaps xmm0, oword ptr vi2Copy[ecx]
    mulps xmm2, xmm0
    movaps oword ptr vo2Copy[ecx], xmm2
    add ecx, 16
    cmp ecx, 400
    jnz MUL_LOOP
}
```

بعنوان مثال به کمک دستورات `movaps` می توانیم ۴ مقدار ممیز شناور را از مموری در رجیستر های `xmm` قرار دهیم و به کمک آن عملوند های خود را آماده کنیم سپس به کمک دستور `mulps` می توانیم این دو رجیستر را به صورت `packed` در هم ضرب کنیم.

سپس مقدار اندیس ها را افزایش می دهیم تا برای چهار مقدار بعدی محاسبه صورت گیرد و به همین ترتیب در یک حلقه مقادیر را محاسبه می کنیم.

روند منطقی این عملیات به کمک شکل زیر بیان میشود.



نتایج بدست آمده (۲۰ سطر اول) از وارد کردن دو عدد ۱۰ و ۵ را در زیر مشاهده می کنید.

```
Select Microsoft Visual Studio Debug Console
5
number A: 5.000000
10
number B: 10.000000

row 1 :
vi1 : 5.000000 | vi2 : 10.000000 | vo1 : 50.000000 | vo2 : 50.000000

row 2 :
vi1 : 5.100000 | vi2 : 10.100000 | vo1 : 51.510002 | vo2 : 51.510002

row 3 :
vi1 : 5.200000 | vi2 : 10.200000 | vo1 : 53.039997 | vo2 : 53.039997

row 4 :
vi1 : 5.300000 | vi2 : 10.300000 | vo1 : 54.590004 | vo2 : 54.590004

row 5 :
vi1 : 5.400000 | vi2 : 10.400000 | vo1 : 56.160000 | vo2 : 56.160000

row 6 :
vi1 : 5.500000 | vi2 : 10.500000 | vo1 : 57.750000 | vo2 : 57.750000

row 7 :
vi1 : 5.600000 | vi2 : 10.600000 | vo1 : 59.360001 | vo2 : 59.360001

row 8 :
vi1 : 5.700000 | vi2 : 10.700000 | vo1 : 60.989998 | vo2 : 60.989998

row 9 :
vi1 : 5.800000 | vi2 : 10.800000 | vo1 : 62.640003 | vo2 : 62.640003

row 10 :
vi1 : 5.900000 | vi2 : 10.900000 | vo1 : 64.309998 | vo2 : 64.309998

row 11 :
vi1 : 6.000000 | vi2 : 11.000000 | vo1 : 66.000000 | vo2 : 66.000000

row 12 :
vi1 : 6.100000 | vi2 : 11.100000 | vo1 : 67.710007 | vo2 : 67.710007

row 13 :
vi1 : 6.200000 | vi2 : 11.200000 | vo1 : 69.440002 | vo2 : 69.440002

row 14 :
vi1 : 6.300000 | vi2 : 11.300000 | vo1 : 71.190002 | vo2 : 71.190002

row 15 :
vi1 : 6.400000 | vi2 : 11.400001 | vo1 : 72.960007 | vo2 : 72.960007

row 16 :
vi1 : 6.500000 | vi2 : 11.500000 | vo1 : 74.750000 | vo2 : 74.750000

row 17 :
vi1 : 6.600000 | vi2 : 11.600000 | vo1 : 76.560005 | vo2 : 76.560005

row 18 :
vi1 : 6.700000 | vi2 : 11.700001 | vo1 : 78.390007 | vo2 : 78.390007

row 19 :
vi1 : 6.800000 | vi2 : 11.800000 | vo1 : 80.240005 | vo2 : 80.240005

row 20 :
vi1 : 6.900001 | vi2 : 11.900001 | vo1 : 82.110008 | vo2 : 82.110008

row 21 :
```