آزمایش 12

طاها موسوی 98243058

نیلوفر مرادی جم 97243063

گروه 2

سوالات تحلیلی:

۱. آ) صفحه بندی حافظه را توضیح دهید. (ب) آدرس خطی شامل چه بخش هایی است. (پ) صفحه بندی و قطعه بندی را مقایسه نمایید.

آ) صفحه بندی حافظه یا همان paging مکانیزمی است برای اختصاص حافظه به این صورت که آدرس خطی به آدرس حقیقی به صورت پنهان ترجمه می‌شود. در واقع برای آنکه کانفلیکتی بین حافظه برنامه ها پیش نیاید، یک فضای خطی نسبتا بزرگی که یکسان است به هر برنامه اختصاص داده می‌شود. و در این مکانیزم آدرس های مجازی به فیزیکی تبدیل می‌شود.)

آدرس خطی هم که به صورت مجازی استفاده می‌شود، توسط برنامه تولید می‌شود.

ب)

- page directory entry: مشخص می‌کند این آدرس در کدام ردیف جدول page directory است.
- page table entry: مشخص می‌کند این آدرس در کدام ردیف جدول page است.
- memory page offset address: موقعیت آدرس در page را نشان می‌دهد.

ج) این مقایسه را روی جدول زیر انجام می‌دهیم. شباهت ها را با رنگ سبز نشان داده ایم.

Feature	Paging	Segmentation
Size	4K bytes	Any size
Levels of privilege	2	4
Base address	4K-byte aligned	Any address
Dirty bit	Yes	No
Access bit	Yes	Yes
Present bit	Yes	Yes
Read/write protection	Yes	Yes

- سایز پیج: در پیجینگ این سایز ثابت و برابر 4kb است ولی در قطعه بندی هر سایزی می‌تواند باشد.
- سطوح دسترسی: این عدد برای پیجینگ برابر 2 و برای قطعه بندی برابر 4 است.
- Base address: به دلیل سایز پیج، بیس آدرس برای پیجینگ مضربی از 4kb است ولی در قطعه بندی هر آدرسی می‌تواند باشد.
- هم پیجینگ و هم قطعه بندی دارای access bit و present bit هستند. ولی فقط پیجینگ علاوه بر آن دو شامل dirty bit هم است.
- هر دو دارای read/write protection هستند.

2. آ) **یک** descriptor **در ثبات قطعه چه اطلاعاتی را به دست می‌دهد؟ (ب) برای تکامل پردازنده چه راهکاری برای چالش گلوگاه بودن پهنای باند حافظه پیشنهاد میشود؟**

آ)

- base: بیس آدرس سگمنت
- limit: طول سگمنت
- P : بیت حاضر (1 = present , 0 = not present)
- Descriptor privilege level: توصیفگر سطح امتیاز که از 0 تا 3 است.
- S: توصیفگر دسکریپتور(0 = segment descriptor, 1= code or data segment descriptor)
- TYPE: نوع سگمنت
- access bit, granularity bit
- D: سایز دیفالت عملیات (1 = 32-bit segment, 0 = 16-bit segment)

- G: بیت گارانتی (1 = Segment length is page granular (4GB in steps of 4K و 0 = Segment length is byte granular (4KB))

-0 : بیت برای سازگاری با پردازنده های آینده باید صفر باشد

- AVL: فیلد در دسترس برای کاربر یا س یستم عامل

ب) هرگاه در انتقال داده ها از RAM به CPU مشکلی وجود داشته باشد که سرعت آن را کاهش دهد و موجب کاهش عملکرد شود یعنی گلوگاهی ایجاد شده. پهنای باند حافظه برابر است با حاصل ضرب عرض رابط حافظه و نرخ انتقال. اگر بتوانیم کانال های حافظه بیشتری را اضافه کنیم، احتمالا بتوانیم مشکل گلوگاه پهنای باند را برطرف کنیم. این یعنی ظرفیت بیشتری را برای RAM قائل شویم

3. **شباهت ها و تفاوت ها بین دو فناوری** MMX **و** SSE **را شرح دهید**.

تفاوت:

- MMX دستورات کمتری نسبت به SSE دارد.
- دستورات SSE بر روی اعداد اعشاری و دستورات MMX بر روی اعداد صحیح محاسبات انجام می-دهند.
- دستورات SSE از رجیستر ها جداگانه ولی دستورات MMX از رجیستر مشترک برایب عملیات هایشان استفاده می‌کنند.
- دستورات SSE می‌توانند بر روی 4 عدد اعشاری 32 بیتی و دستورات MMX می‌توانند بر روی 2 عدد صحیح 32 بیتی یا 4 عدد 16 بیتی تا 8 عدد 8 بیتی عملیات انجام دهند.

شباهت:

- از سری دستورات simd محسوب می‌شوند.
- کارایی واحد FPU به صورت قابل ملاحضه ای کاهش می یابد در هر دوی آن ها.
- هر دو دارای رجیستر ها mm0 تا mm7 هستند.
- هر دو دستورات برای پردازش های ماتریسی استفاده می‌شوند.

رفرنس های سوالات تحلیلی:

- کلاس درس و اسلاید های درسی

- https://www.studytonight.com/operating-system/paging-in-operating-systems
- https://www.geeksforgeeks.org/paging-in-operating-system/
- https://www.apica.io/5-common-performance-bottlenecks/
- https://www.techspot.com/article/2166-mmx-sse-avx-explained/

دستور کار:

در این آزمایش ما قصد داشتیم به وسیله کتابخانه conio.h در c++، کدی به زبان cpp بزنیم که دارای بلاکی از کد اسمبلی باشد.

```cpp
#include <conio.h>
#include <stdio.h>

#define VEC_SIZE 50
#define STEP_SIZE 0.2

float a_vec[VEC_SIZE];
float b_vec[VEC_SIZE];
float y_vec[VEC_SIZE];

float zi1[VEC_SIZE];    //y^2 - a
float zi2[VEC_SIZE];    //y^2 + b
float zo1[VEC_SIZE];    //y^2 + (a+b)*y + ab
float zo2[4 * VEC_SIZE];//y^2 + (a+b)*y + ab
```

سایز وکتور و طول گام در اول برنامه و طبق صورت دستور کار مشخص شده اند.

در ادامه سه وکتور a و b و y را بصورت گلوبال تعریف کردیم که مقادیر اولیه را نگه میدارند.

خروجی های برنامه نیز در وکتورهای zi1, zi2, zo1, zo2 همانطور که درصورت سوال گفته شده ذخیره میشوند و به صورت گلوبال تعریف شده اند.

```cpp
int main() {
    printf("a = ");
    float a = cin_float();
    printf("\nb = ");
    float b = cin_float();
    printf("\n");

    create_vectors(a, b);

    calculate_zi_vectors(y_vec, a_vec, b_vec); //y^2 - a and y^2 + b

    calculate_equation_vector(y_vec); // y^2 + (a+b)*y + ab

    calculate_equation_vectors_SSE(y_vec); // y^2 + (a+b)*y + ab using SSE

    for (int i = 0; i < VEC_SIZE; i++) {
        printf("---------------------- y = %f ----------------------\n", y_vec[i]);
        printf("zi1 = %f\n", zi1[i]);
        printf("zi2 = %f\n", zi2[i]);
        printf("zo1 = %f\n", zo1[i]);
        printf("zo2 = %f\n", zo2[i]);
    }

    return 0;
}
```

با توجه به ترتیب مراحل در تابع main هرکدام را توضیح میدهیم.

```
float cin_float() {
    int c = '0';
    int neg;

    c = _getche();

    neg = (c == '-') ? -1 : 1;
    c = (c == '+' || c == '-') ? '0' : c;

    float int_part = 0;
    while (c >= '0' && c <= '9') {
        int_part = int_part * 10 + (c - '0');
        c = _getche();
    }

    //user is done entering number
    if (c != '.') return neg * int_part;

    //user entered a '.' meaning they will enter the fractional part now
    c = '0';
    int pow = 1;
    float fraction_part = 0;
    while (c >= '0' && c <= '9') {
        fraction_part = fraction_part + ((c - '0') / (float)pow);
        pow *= 10;
        c = _getche();
    }

    return neg * (int_part + fraction_part);
}
```

این تابع یک مقدار float را به صورت کارکتر کارکتر از ورودی میگیرد و به مقدار float متناظرش cast کرده و return میکند. هر عدد float میتواند از یک قسمت صحیح و یک قسمت اعشاری تشکیل شود.

در این تابع ما ابتدا منفی یا مثبت بودن را تشخیص می‌دهیم. سپس قسمت صحیح را می‌خوانیم و در ادامه در صورت موجود بودن قسمت اعشاری آن را خوانده و به قسمت صحیح اضافه می‌کنیم. همانطور که مشخص است تشخیص وجود بخش اعشاری به وسیله کارکتر "." میباشد.

در تابع main دو مقدار a و b به وسیله این تابع از ورودی خوانده میشوند.

```
//input a and b, give value to a_vec and b_vec and y_vec
void create_vectors(float a, float b) {
    float start = 0;
    for (int i = 0; i < VEC_SIZE; i++, start += STEP_SIZE) {
        a_vec[i] = a;
        b_vec[i] = b;
        y_vec[i] = start;
    }
}
```

این تابع مقادیر a و b را درون وکتوری ذخیره میکند و همچنین در وکتور y_vec، به ترتیب از 0 شروع کرده و اعداد را به اندازه step_size اضافه میکند.

در پایان مقادیر در تابع y_vec بصورت 0 و 0.2 و 0.4 و 0.6 و 0.8 و ... میشوند.

```
//input y[] and a[] and b[], give value to zi1 and zi2
void calculate_zi_vectors(float y[], float a[], float b[]) {
    for (int i = 0; i < VEC_SIZE; i++) {
        zi1[i] = (y[i] * y[i]) - a[i];
        zi2[i] = (y[i] * y[i]) + b[i];
    }
}
```

وکتورهای zi با توجه به معادله بیان شده در صورت سوال پر میشوند.

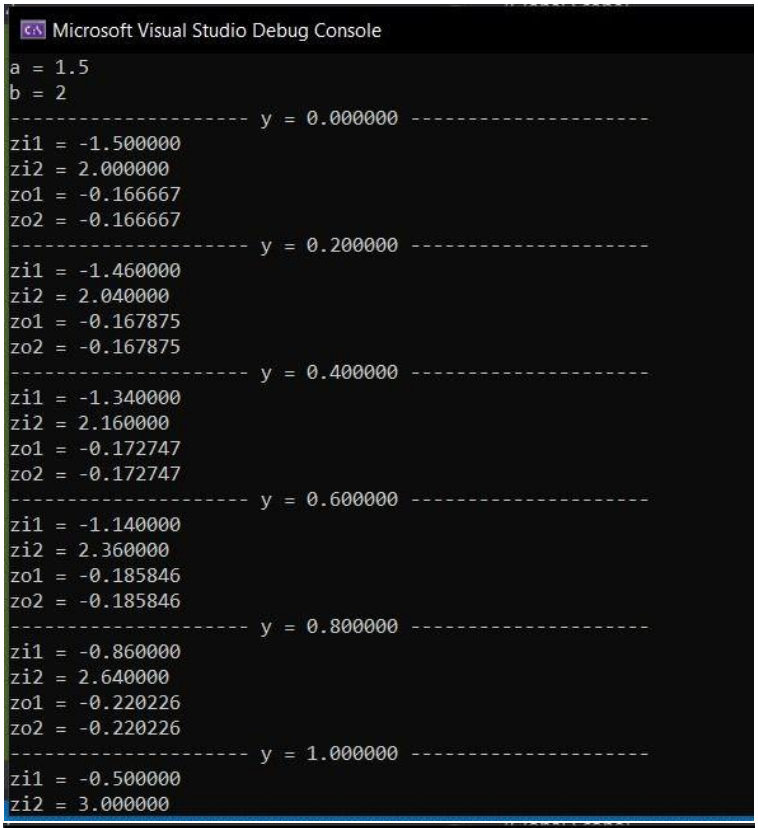
```
//input y[], give value to zo1
void calculate_equation_vector(float y[]) {
    for (int i = 0; i < VEC_SIZE; i++) {
        zo1[i] = 1 / (2 * zi1[i] * zi2[i]);
    }
}
```

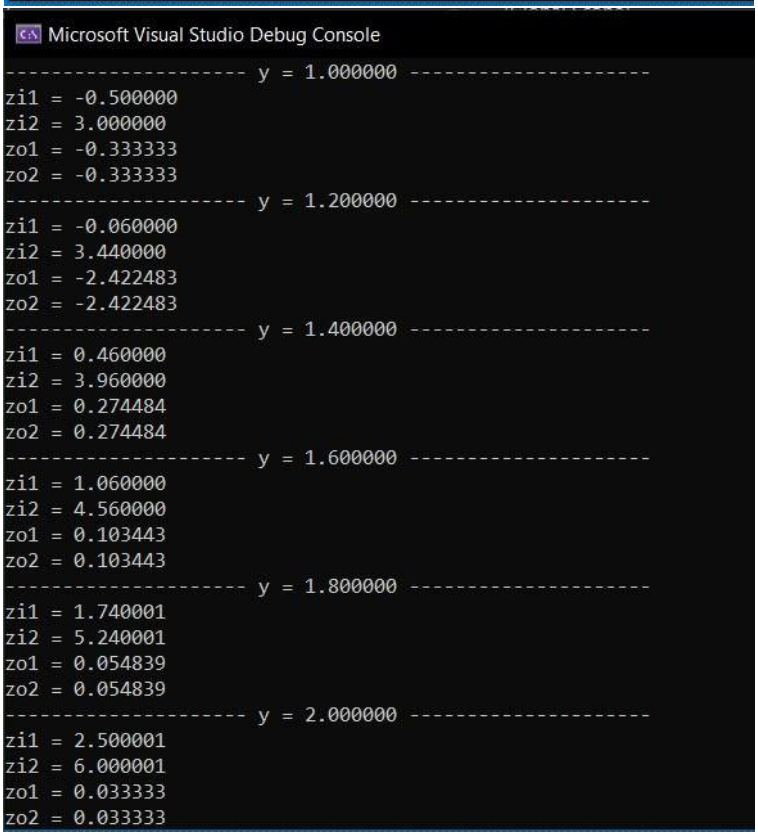
وکتور zo با توجه به معادله بیان شده در صورت سوال برابر ضرب دو مقدار zi1 و zi2 بوده و پر میشود.

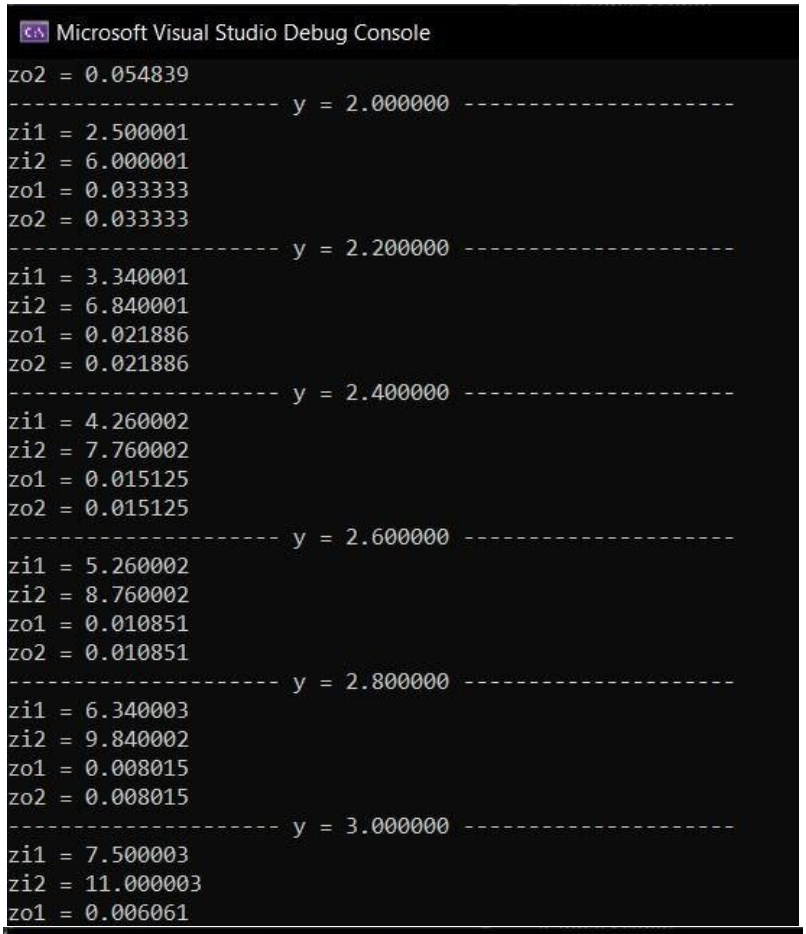
```
void calculate_equation_vectors_SSE(float y[]) {
    // align address to a 16-byte boundary
    __declspec(align(16)) float temp_zi1[VEC_SIZE];
    __declspec(align(16)) float temp_zi2[VEC_SIZE];
    __declspec(align(16)) float one_vec[VEC_SIZE];
    __declspec(align(16)) float two_vec[VEC_SIZE];
    __declspec(align(16)) float zo2_temp[4 * VEC_SIZE];

    for (int i = 0; i < VEC_SIZE; i++) {
        temp_zi1[i] = zi1[i];
        temp_zi2[i] = zi2[i];
        one_vec[i] = 1;
        two_vec[i] = 2;
    }

    __asm {

        mov ecx, 0

        LBL:
            movaps xmm0, oword ptr temp_zi1[ecx]    // load pointer to the "ecx"th value in temp_zi1
            movaps xmm1, oword ptr temp_zi2[ecx]    // load pointer to the "ecx"th value in temp_zi2
            movaps xmm2, oword ptr one_vec[ecx]     // load pointer to the "ecx"th value in one_vec
            movaps xmm3, oword ptr two_vec[ecx]     // load pointer to the "ecx"th value in two_vec
            movaps xmm4, xmm0                       // load 4 float values from temp_zi1
            movaps xmm5, xmm1                       // load 4 float values from temp_zi2
            movaps xmm6, xmm2                       // load 4 float values from one_vec
            movaps xmm7, xmm3                       // load 4 float values from two_vec
            mulps xmm4, xmm5                        // xmm4 = xmm4 * xmm5 --> zi1 = zi1 * zi2
            mulps xmm4, xmm7                        // xmm4 = xmm4 * xmm7 --> zi1 = zi1 * 2 --> (zi1 = 2 * zi1 * zi2)
            divps xmm6, xmm4                        // xmm6 = xmm6 / xmm4 --> one_vec = one_vec / zi1 --> one_vec = 1 / (2 * zi1 * zi2)
            movaps oword ptr zo2_temp[ecx], xmm6    // store in zo2_temp
            add ecx, 16                             // ecx = ecx + 16 : to help get the pointer to the next value in the vectors
            cmp ecx, 400                            // in each loop we multiply 4 values => we need 100/4 = 25 iterations => 25*16 = 400
            jnz LBL
    }

    for (int i = 0; i < VEC_SIZE; i++) {
        zo2[i] = zo2_temp[i];
    }
}
```

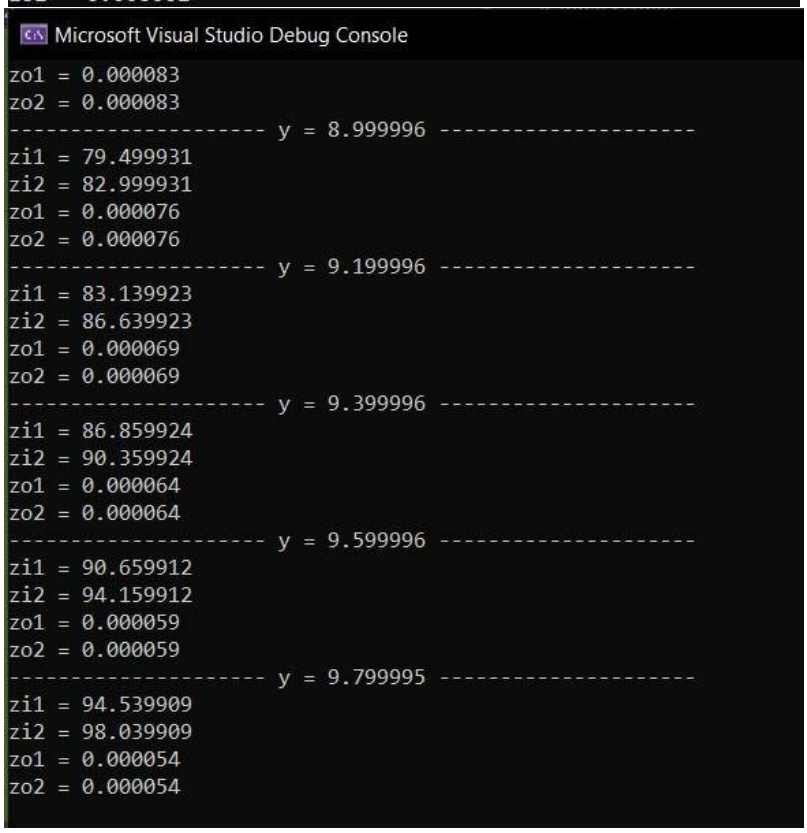
در این تابع همان عملیات تابع قبلی را با اسمبلی انجام دادیم و سپس مقادیر به دست آمده را در وکتور ZO میریزیم. توضیحات مربوط به هر خط اسمبلی جلوی آن نوشته شده.

اسکرین شات خروجی ها بعد از اجرای برنامه:






Microsoft Visual Studio Debug Console
zo2 = 0.054839
---------- y = 2.000000 ----------
zi1 = 2.500001
zi2 = 6.000001
zo1 = 0.033333
zo2 = 0.033333
---------- y = 2.200000 ----------
zi1 = 3.340001
zi2 = 6.840001
zo1 = 0.021886
zo2 = 0.021886
---------- y = 2.400000 ----------
zi1 = 4.260002
zi2 = 7.760002
zo1 = 0.015125
zo2 = 0.015125
---------- y = 2.600000 ----------
zi1 = 5.260002
zi2 = 8.760002
zo1 = 0.010851
zo2 = 0.010851
---------- y = 2.800000 ----------
zi1 = 6.340003
zi2 = 9.840002
zo1 = 0.008015
zo2 = 0.008015
---------- y = 3.000000 ----------
zi1 = 7.500003
zi2 = 11.000003
zo1 = 0.006061


Microsoft Visual Studio Debug Console
zo1 = 0.000083
zo2 = 0.000083
---------- y = 8.999996 ----------
zi1 = 79.499931
zi2 = 82.999931
zo1 = 0.000076
zo2 = 0.000076
---------- y = 9.199996 ----------
zi1 = 83.139923
zi2 = 86.639923
zo1 = 0.000069
zo2 = 0.000069
---------- y = 9.399996 ----------
zi1 = 86.859924
zi2 = 90.359924
zo1 = 0.000064
zo2 = 0.000064
---------- y = 9.599996 ----------
zi1 = 90.659912
zi2 = 94.159912
zo1 = 0.000059
zo2 = 0.000059
---------- y = 9.799995 ----------
zi1 = 94.539909
zi2 = 98.039909
zo1 = 0.000054
zo2 = 0.000054

رفرنس دستور کار:

کلاس درس و اسلاید های درسی

- https://stackoverflow.com/questions/14038754/why-i-cannot-compile-the-assembly-codes-for-x64-platform-with-vc2010

- https://docs.microsoft.com/en-us/cpp/assembler/inline/asm?view=msvc-170

- https://www.elprocus.com/8086-assembly-language-programs-explanation/#:~:text=Assembly%20Level%20Programming%208086,-The%20assembly%20programming&text=The%20microcontroller%20or%20microprocessor%20can,memory%20to%20perform%20the%20tasks.