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.

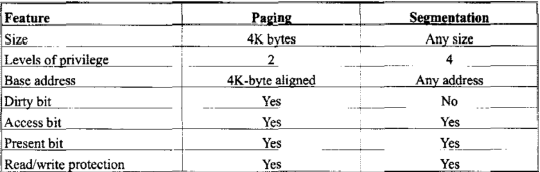
ب)

- 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.



* 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.

.**2آ( 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 ایحاد شده. پهنای باند حافظه برابر است با حاصل ضرب عرض رابط حافطه 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

.**3شباهت 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 می­دهند.
* 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.

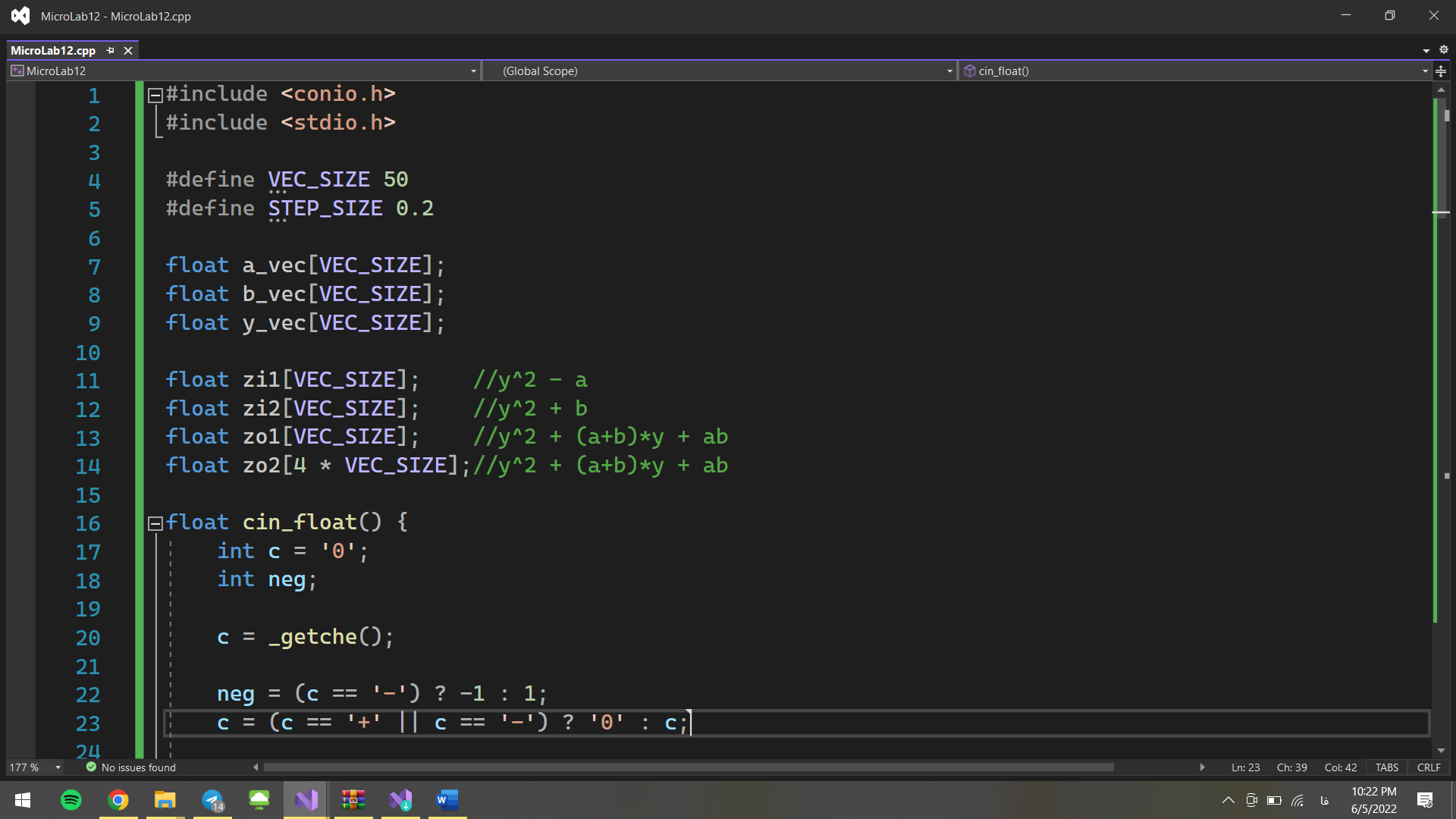
**رفرنس های سوالات تحلیلی**:

- 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:**

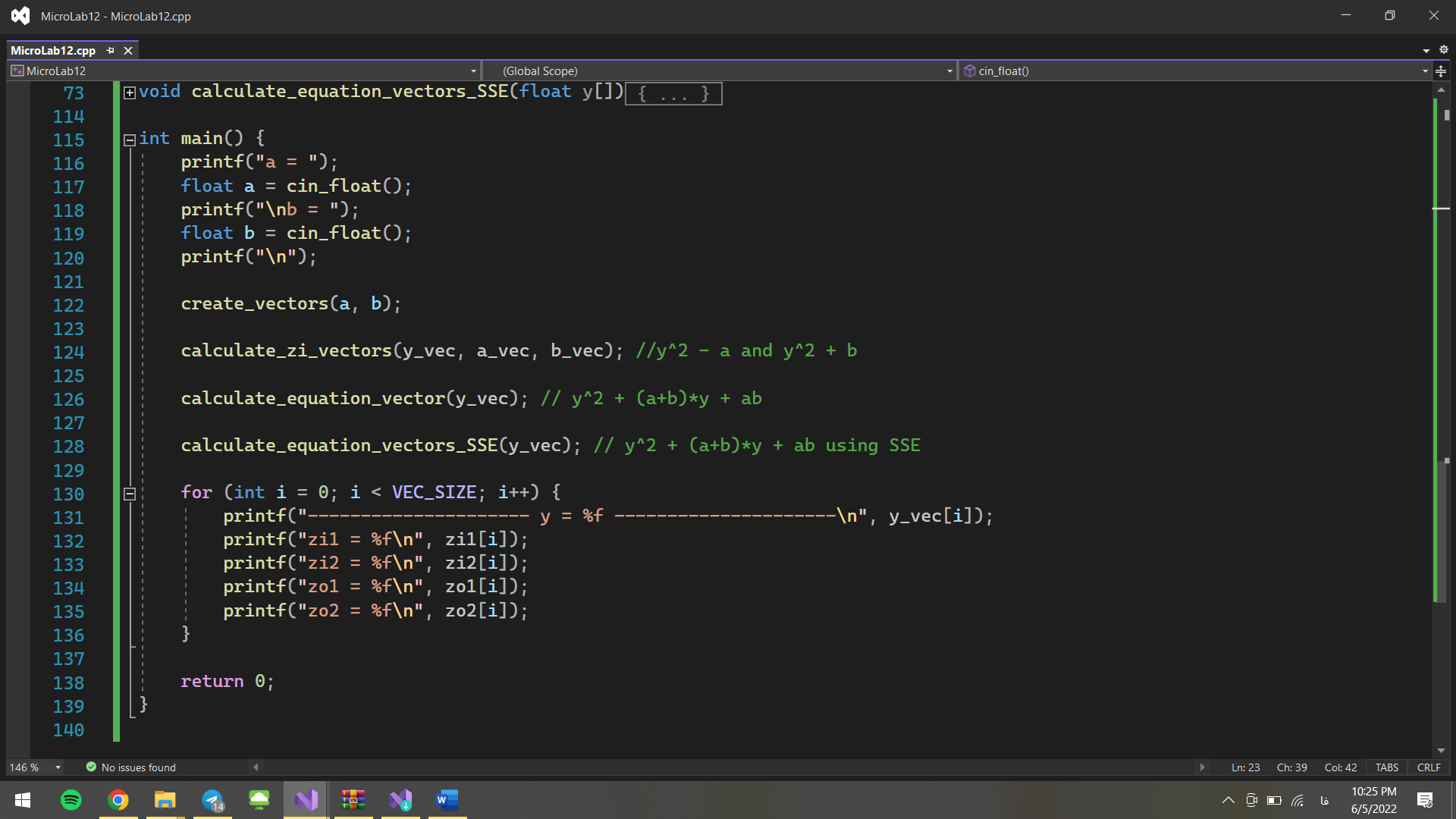
در این آزمایش ما قصد داشتیم به وسیله کتابخانه conio.h در c++، کدی به زبان cpp بزنیم که دارای بلاکی از کد اسمبلی باشد.



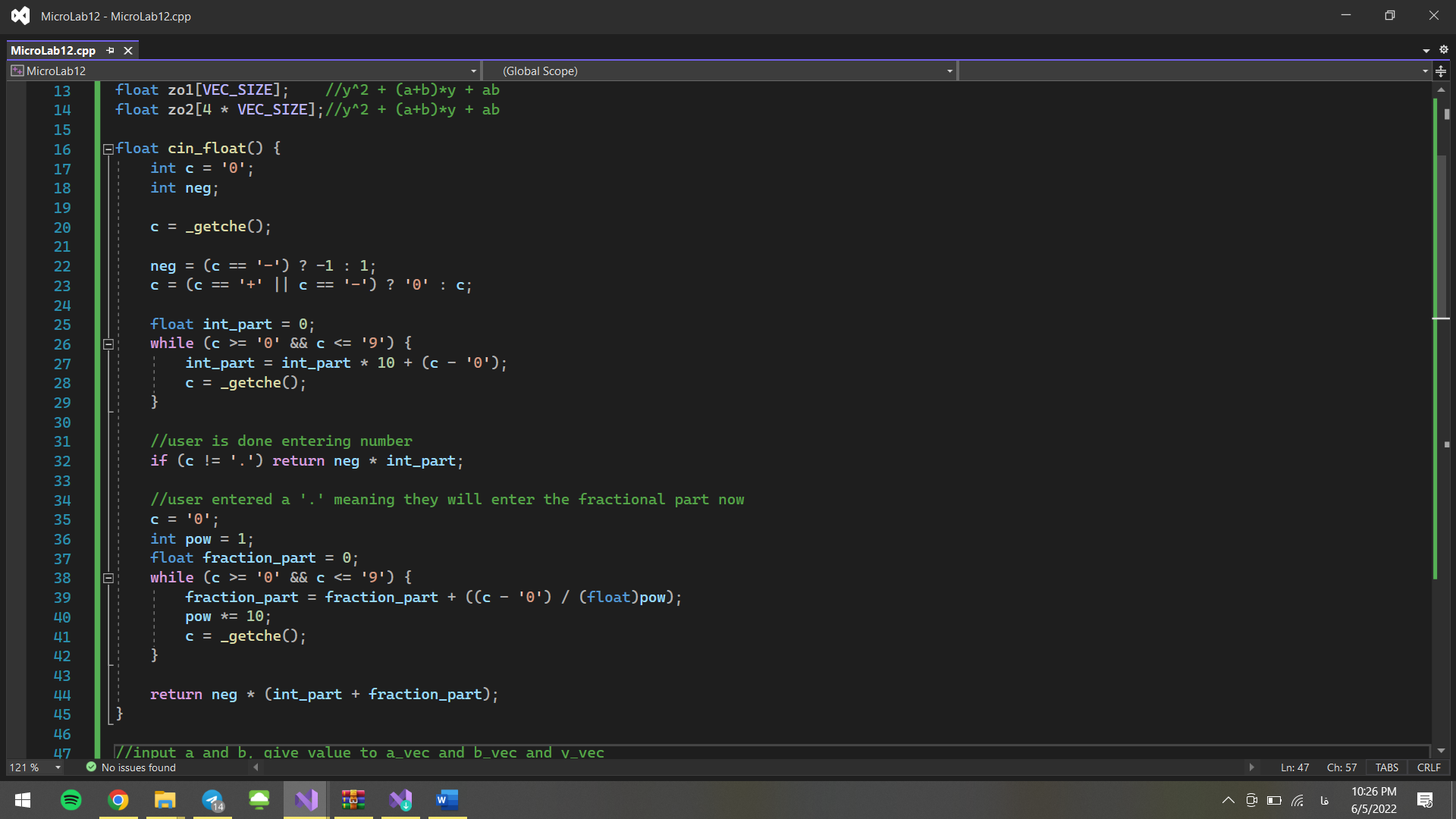
سایز وکتور و طول گام در اول برنامه و طبق صورت دستور کار مشخص شده اند.

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.



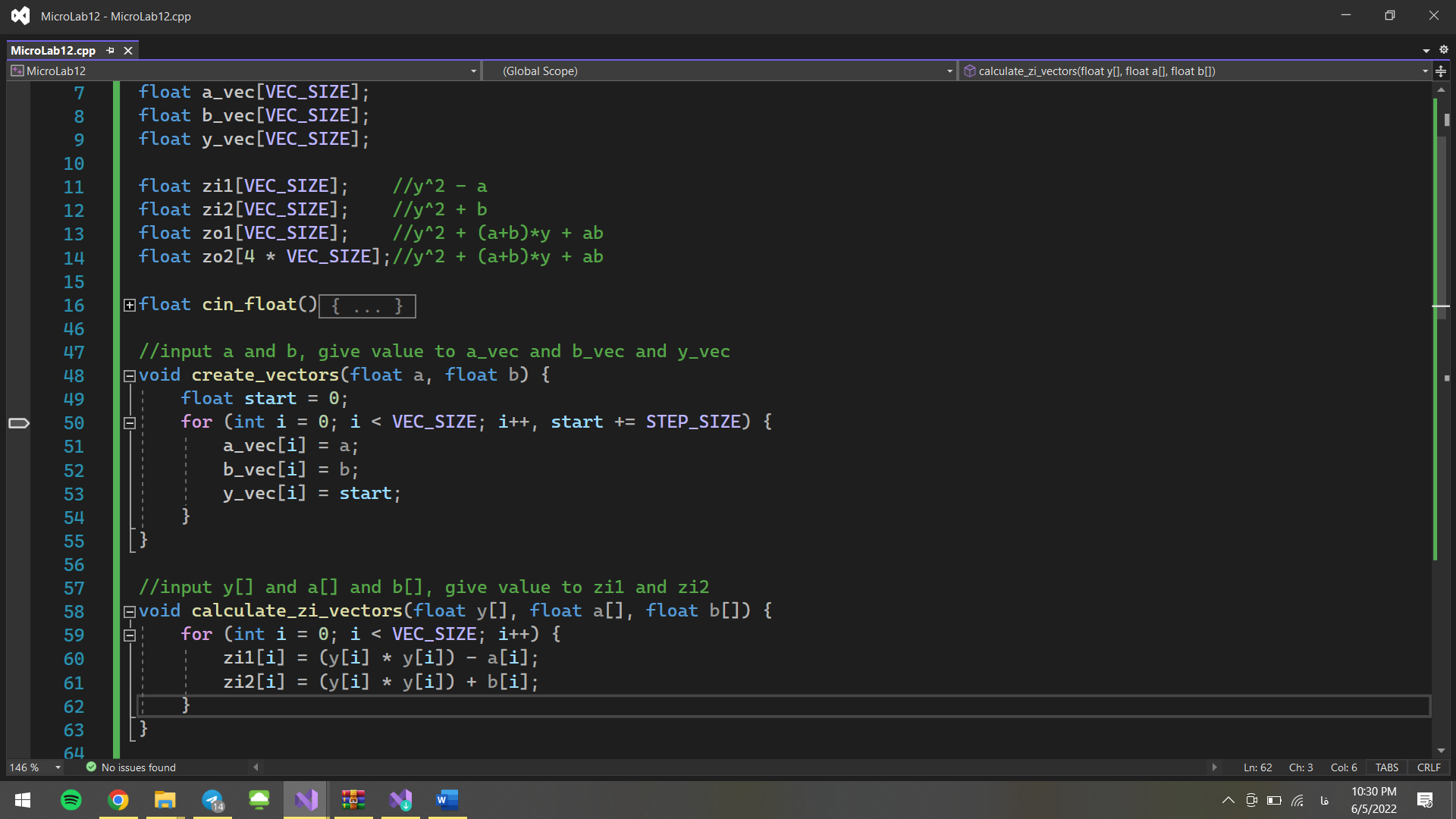
با توجه به ترتیب مراحل در تابع main هرکدام را توضیح میدهیم.



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.

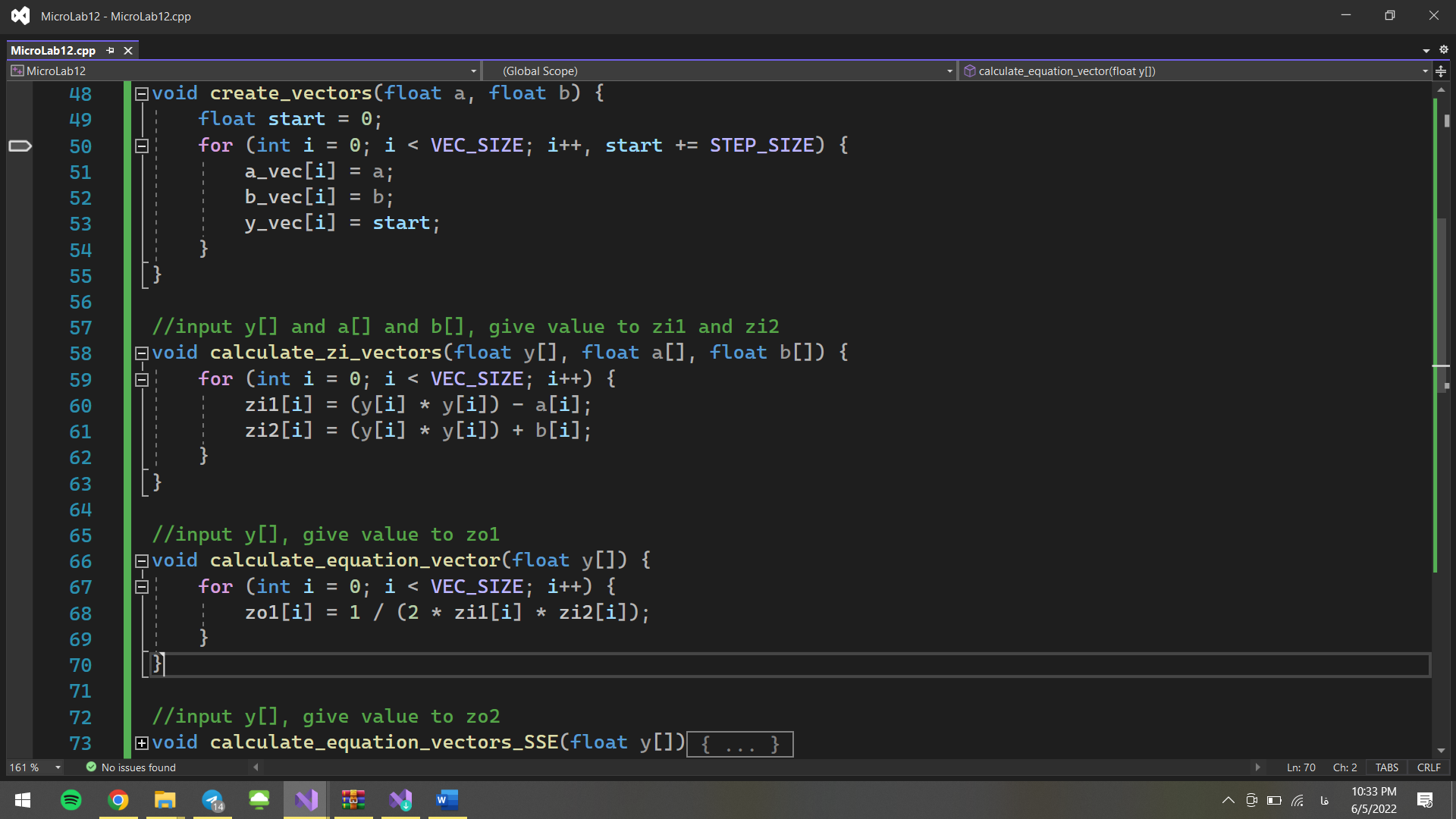
در این تابع ما ابتدا منفی یا مثبت بودن را تشخیص می‌دهیم. سپس قسمت صحیح را می‌خوانیم و در ادامه در صورت موجود بودن قسمت اعشاری آن را خوانده و به قسمت صحیح اضافه می‌کنیم. همانطور که مشخص است تشخیص وجود بخش اعشاری به وسیله کارکتر "." میباشد.

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

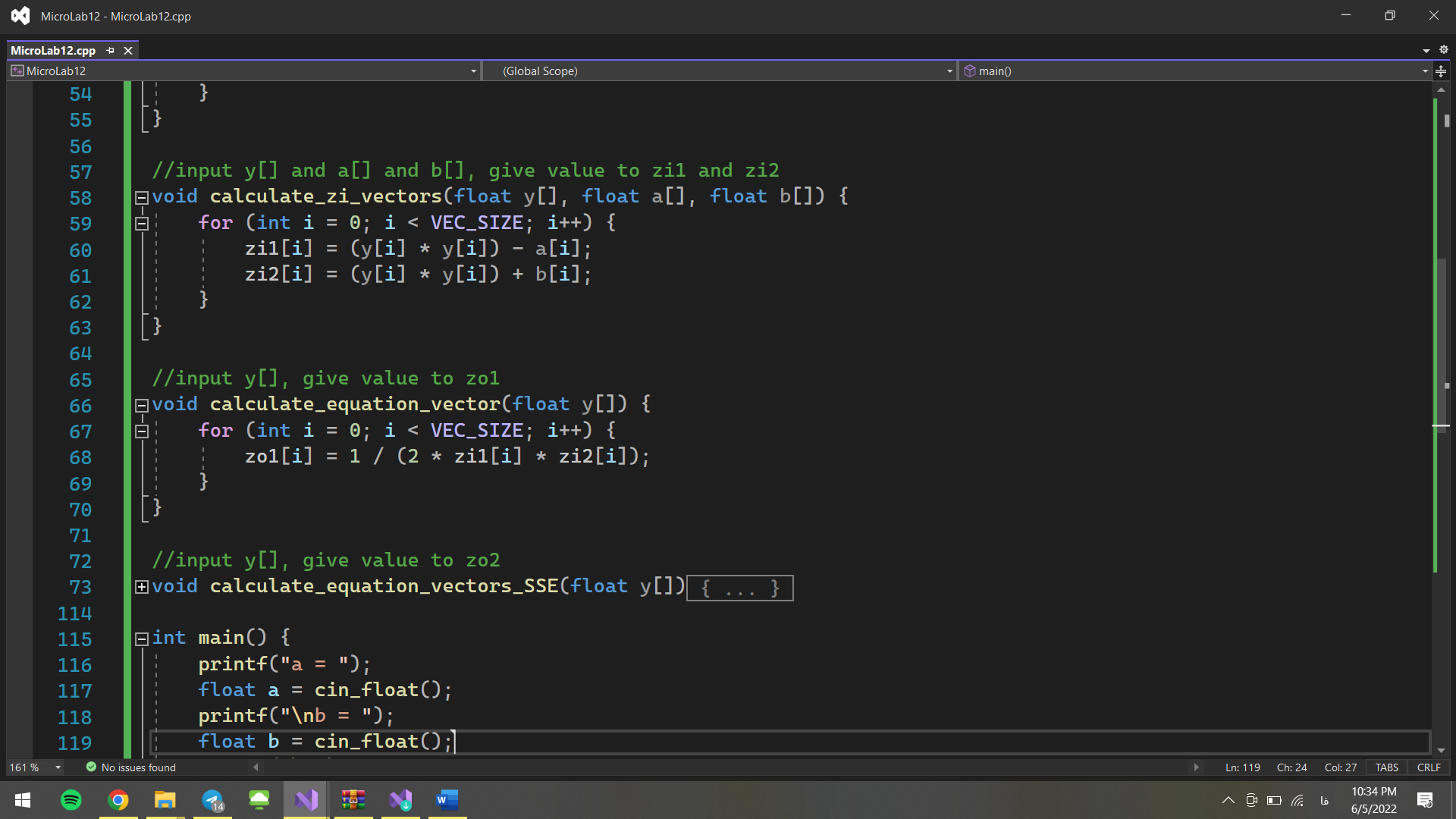


این تابع مقادیر a و b را درون وکتوری ذخیره میکند و همچنین در وکتور y\_vec، به ترتیب از 0 شروع کرده و اعداد را به اندازه step\_size اضافه میکند.

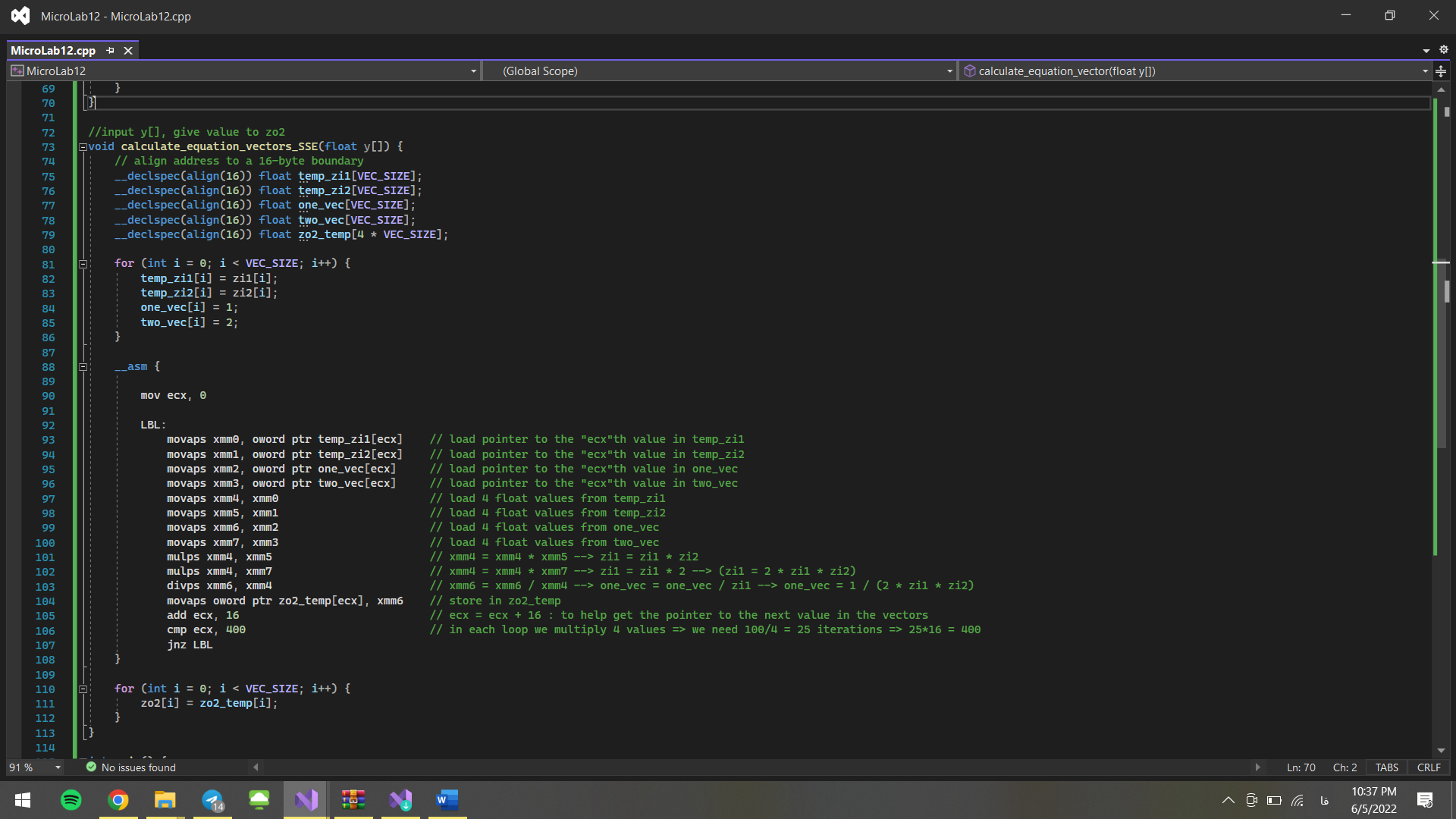
در پایان مقادیر در تابع y\_vec بصورت 0 و 0.2 و 0.4 و 0.6 و 0.8 و ... میشوند.



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

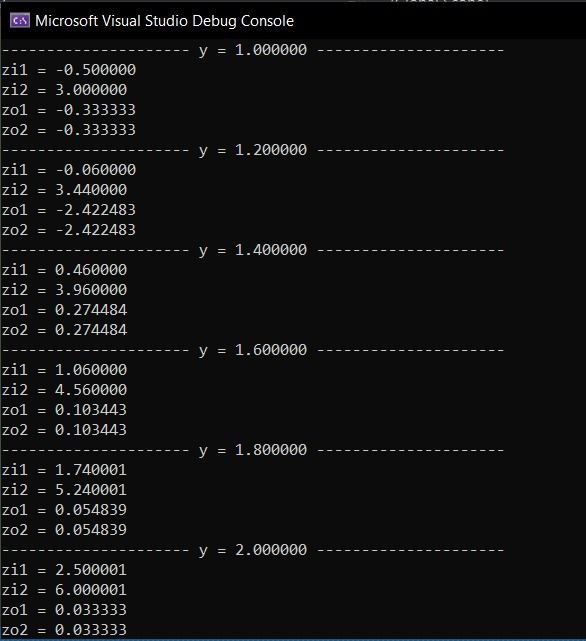
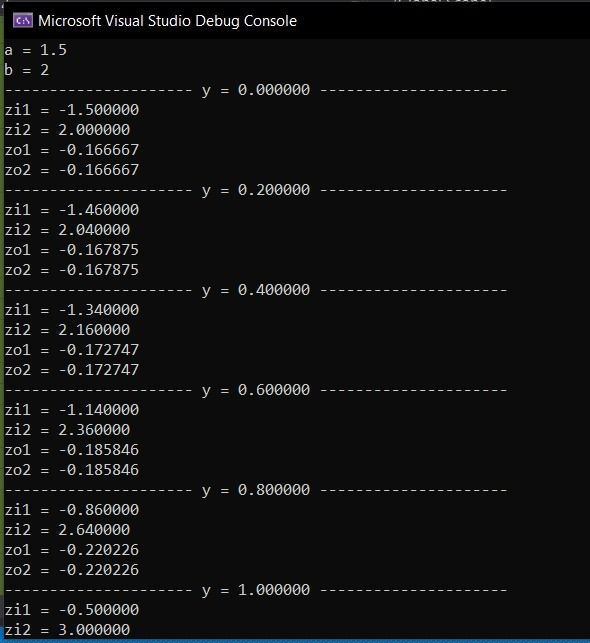


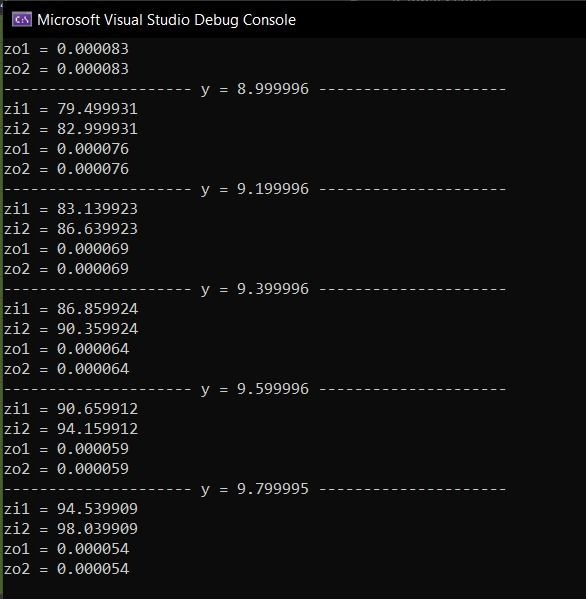
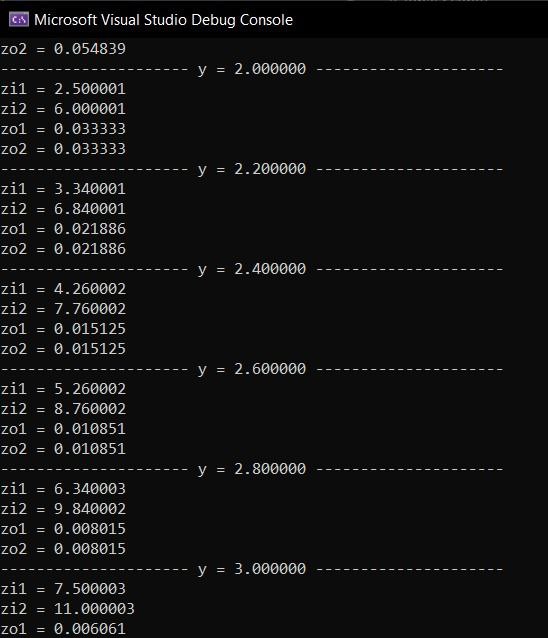
وکتور zo با توجه به معادله بیان شده در صورت سوال برابر ضرب دو مقدار zi1 و zi2 بوده و پر میشود.



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:





**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>.