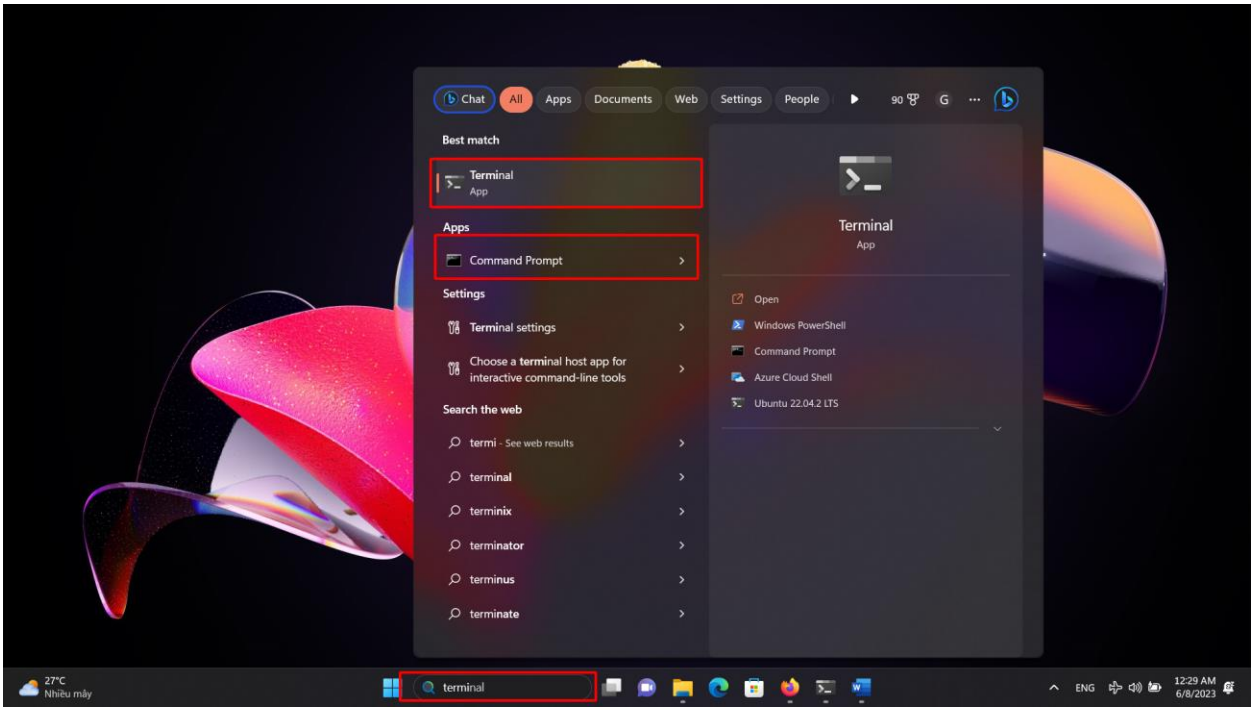
Using Jasmin to run x86 Assembly Code (15 points)

Install Java

Để cần lab này, chúng ta cần sử dụng tới java, để kiểm tra rằng máy mình đã có java chưa, thì chúng ta chỉ cần bật **CMD** hoặc **Terminal** lên bằng cách vào search và gõ **CMD** hoặc **Terminal**

Trong trường hợp này em sẽ dung terminal vì tính tiện lợi của nó , vì có thể thực hiện nhiều câu lệnh giống trong môi trường linux

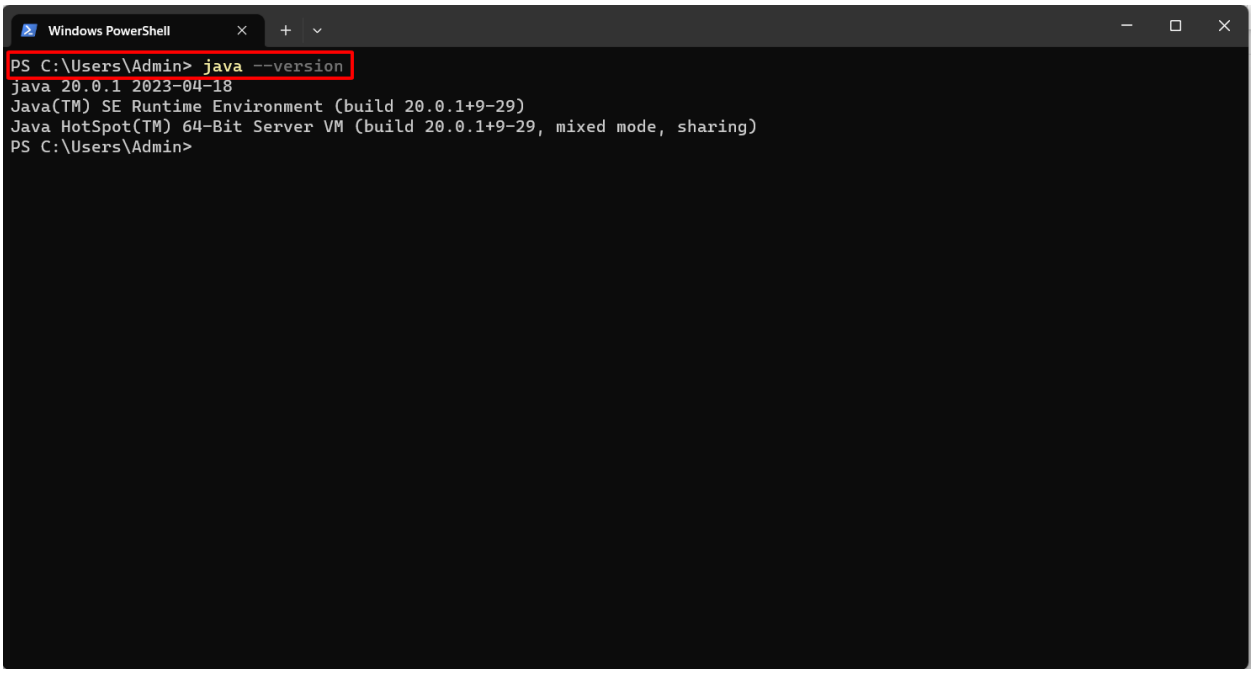
Nếu chưa có Terminal, ta có thể tải thông qua **Microsoft Store,** chỉ cần đơn giản tải search Terminal và phần việc còn lại để cho máy tự cài đặt



Sau đó ta sẽ khởi chạy câu lệnh

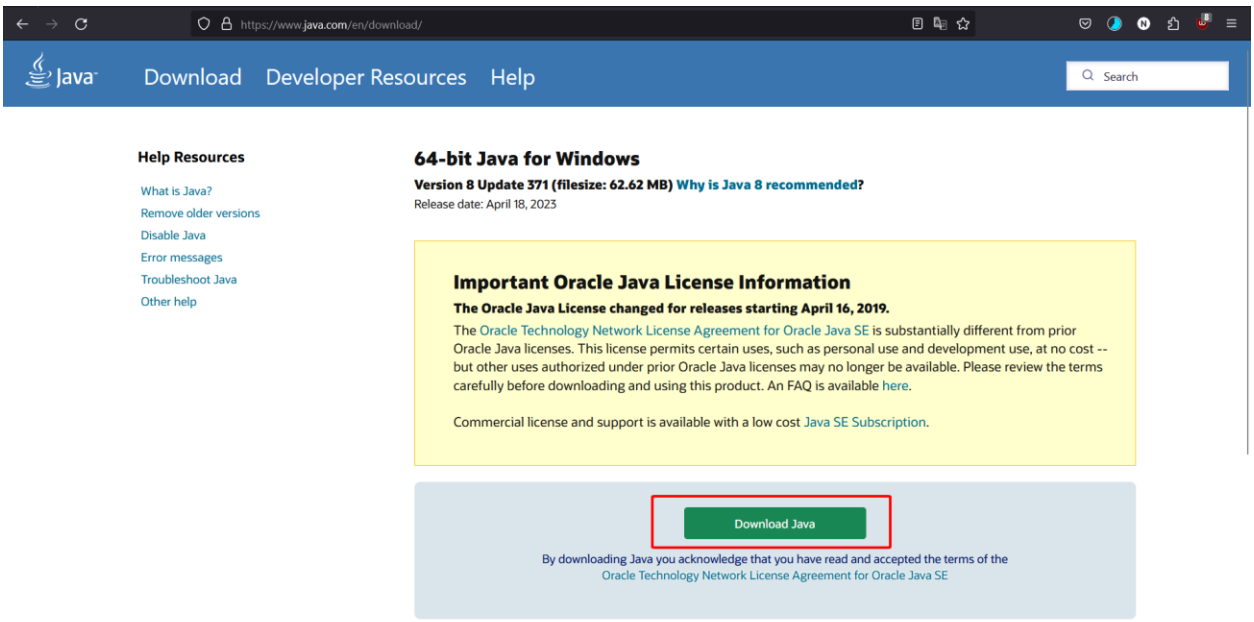
- Java –version

Câu lệnh trên giúp chúng ta kiểm tra xem bên trong máy của chúng ta đã cài các gói hỗ trợ của Java chưa

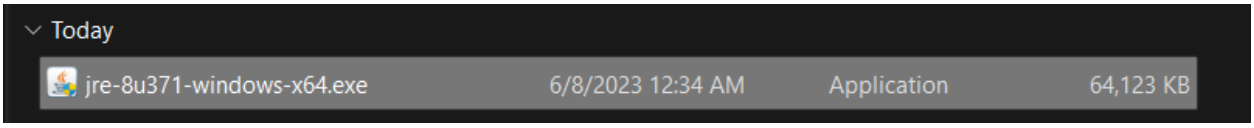


Như ta đã thấy hình trên thì máy tính này đã cài đặt Java.

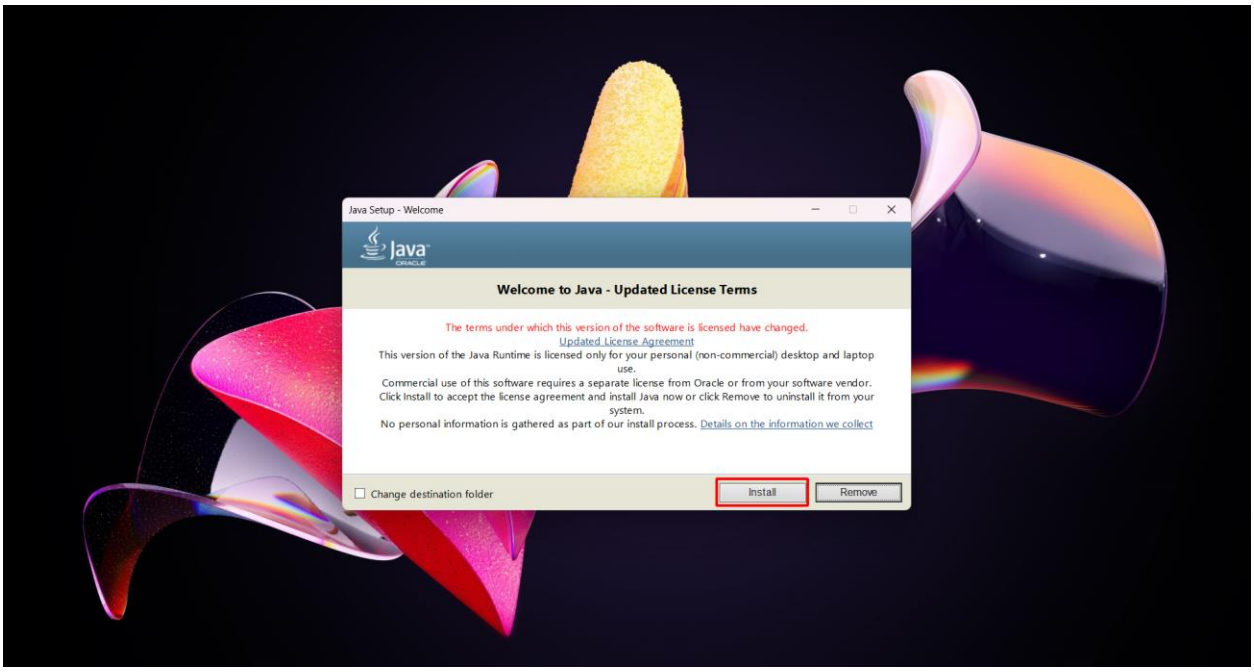
Trong trường hợp máy chưa có phần mềm Java thì chúng ta sẽ cần cài đặt thông qua đường link hướng dẫn sau: https://www.java.com/en/download/



Sau khi download xong, chúng ta chỉ cần chạy file đã download xuống. Trong trường hoợ này nó nàm ở trong htuw mục Download của user:



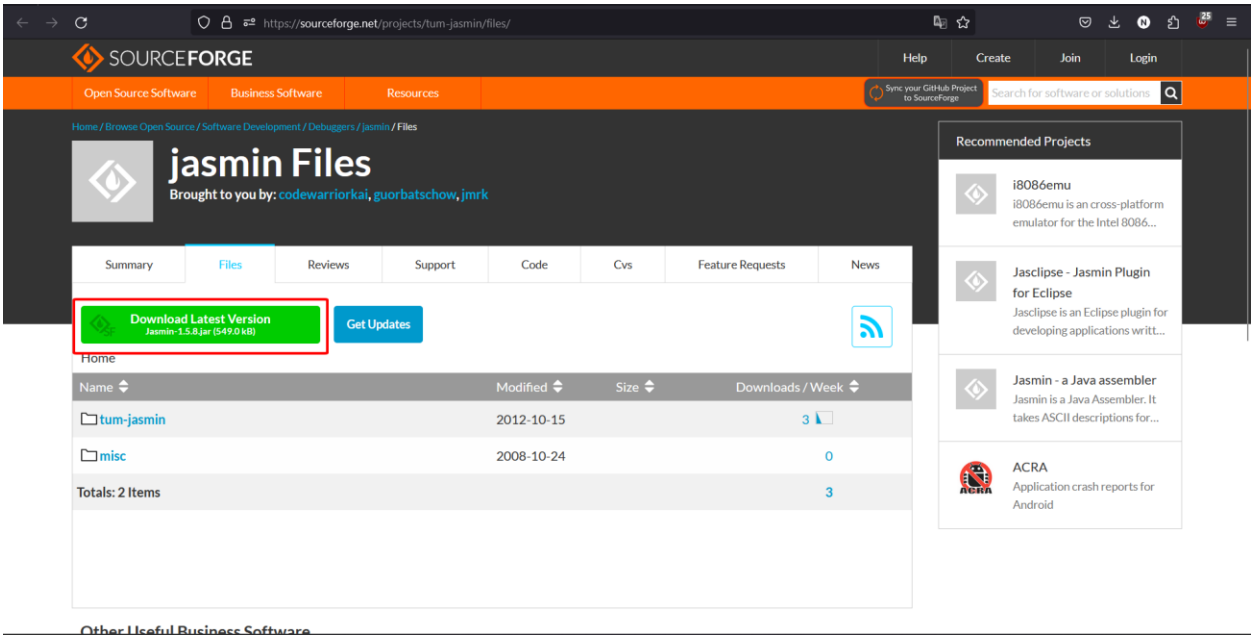
Sau khi nhấn cài đặt, chúng ta chỉ cần việc nhấn **Install** và để cho quá trình cài đặt JRE được tiến hnafh một cách tự động



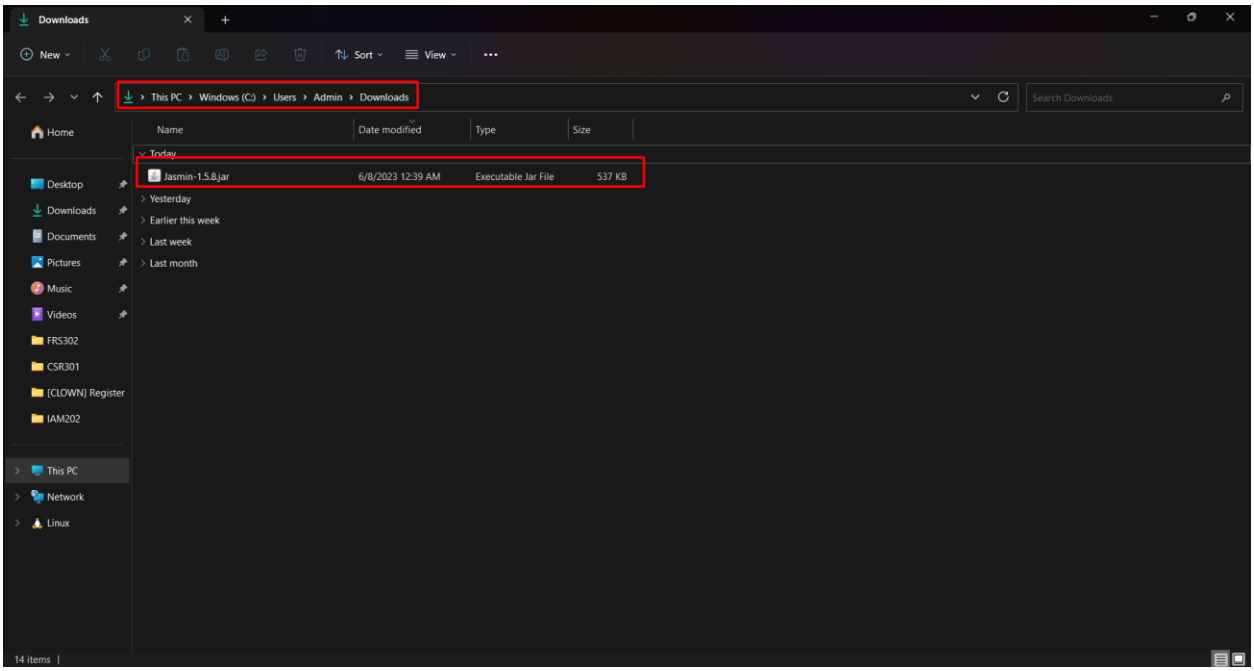
Download Jasmin

Để tải Jasmin, chúng ta sẽ vào link sau đây: https://sourceforge.net/projects/tum-jasmin/files/

Sau đó thì nhấn chọn **"Download latest version"** như hình dưới đây:



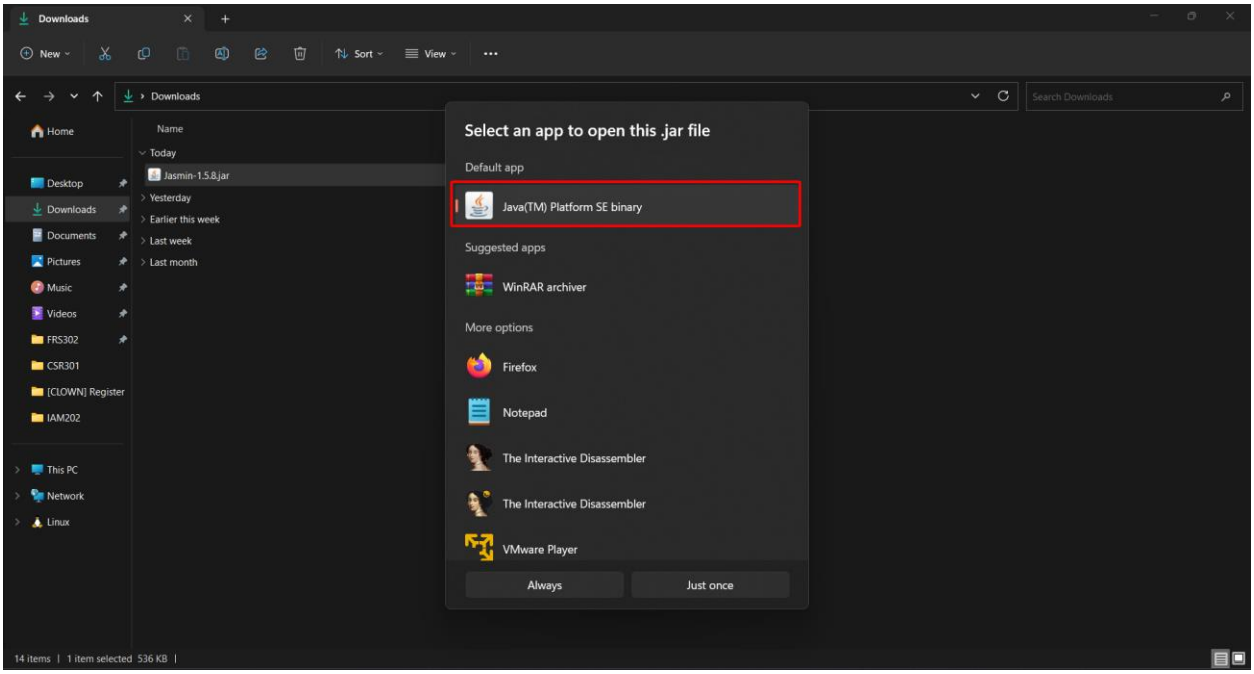
Sau khi Download, chúng ta sẽ đi vào nơi mà chứa file này. Trong trường hợp này là trong folder **Downloads.**



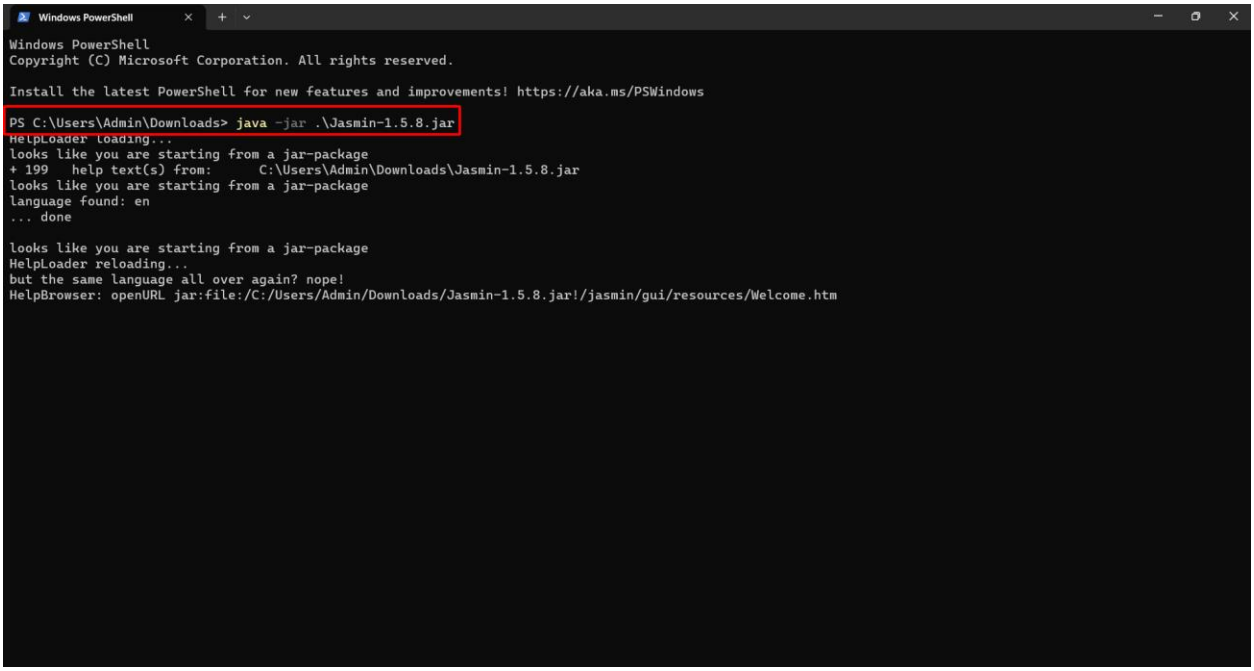
Understanding the Jasmin Window

Vì đây là file jar, một file java executable file, vì thế nếu chúng ta muốn mở thì sẽ có hai cách.

- Cách 1: Double click vào bên trong File → Open With Java Binary Executable (nếu là lần đầu tiên chạy file jar trên m



- Cách 2: Sử dụng termial tại Folder Downloads → sử dụng câu lệnh **java -jar** <tên file> Trong trường hợp này là **java -jar Jasmin-1.5.8.jar**



Trong giao diện của Jasmine, chúng ta sẽ tạo file mới bằng cách sử dụng tổ hợp phím **"Ctrl + N"** hoặc trên thanh tool bar → **chọn File → New**

Đây là tổng quan của chương trình khi tạo một file mới, bao gồm những thông tin được biểu diễn bên hình: thanh ghi, cờ hiệu, nơi chứa assembly code, bộ nhớ lưu trữ,…

Thanh ghi

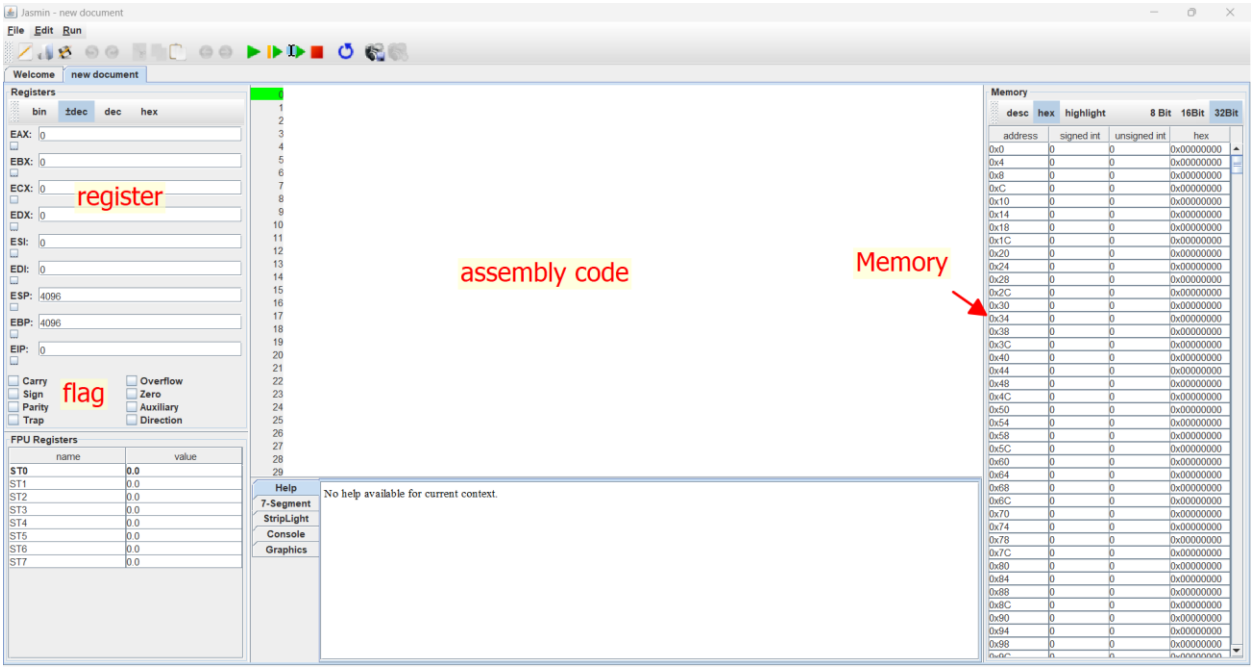
- thanh ghi lưu trữ như eax, ebx, ecx, edx, …
- ESP là thanh ghi chứ địa chỉ của trên đỉnh stakc

- EIP là instructrion pointer chứa địa chỉ kế tiếp được gọi trên stack

Các flag hay còn được gọi là status register

- flag Zero (ZF), được thiết lập khi kết quả của một hoạt động là 0,
- flag Sign (SF), được thiết lập khi kết quả của một hoạt động là âm.
- Carry flag (CF), được thiết lập khi một hoạt động tạo ra một carry từ bit cao nhất
- Overflow flag (OF), được thiết lập khi một hoạt động tạo ra một tràn sang bit dấu.

Assembly code: đơn giản chỉ là nơi hiển thị của code assembly



Using mov Instructions

Lệnh mov bên trong assembly cũng giống như là một lệnh gán biến vào bên trong thanh ghi. Với câu trúc câu lệnh là:

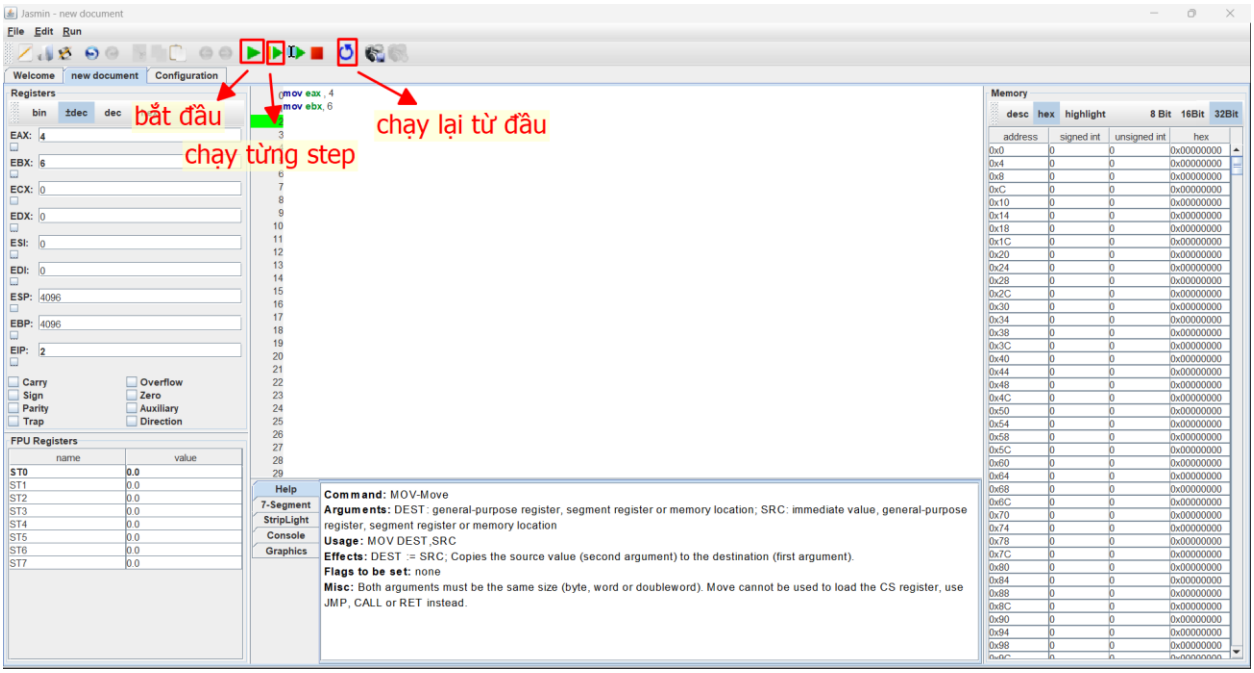
- **mov** [eax, ebx, ecx, edx], **<value>**

Trong câu lệnh này, chúng ta sẽ thực hiện :

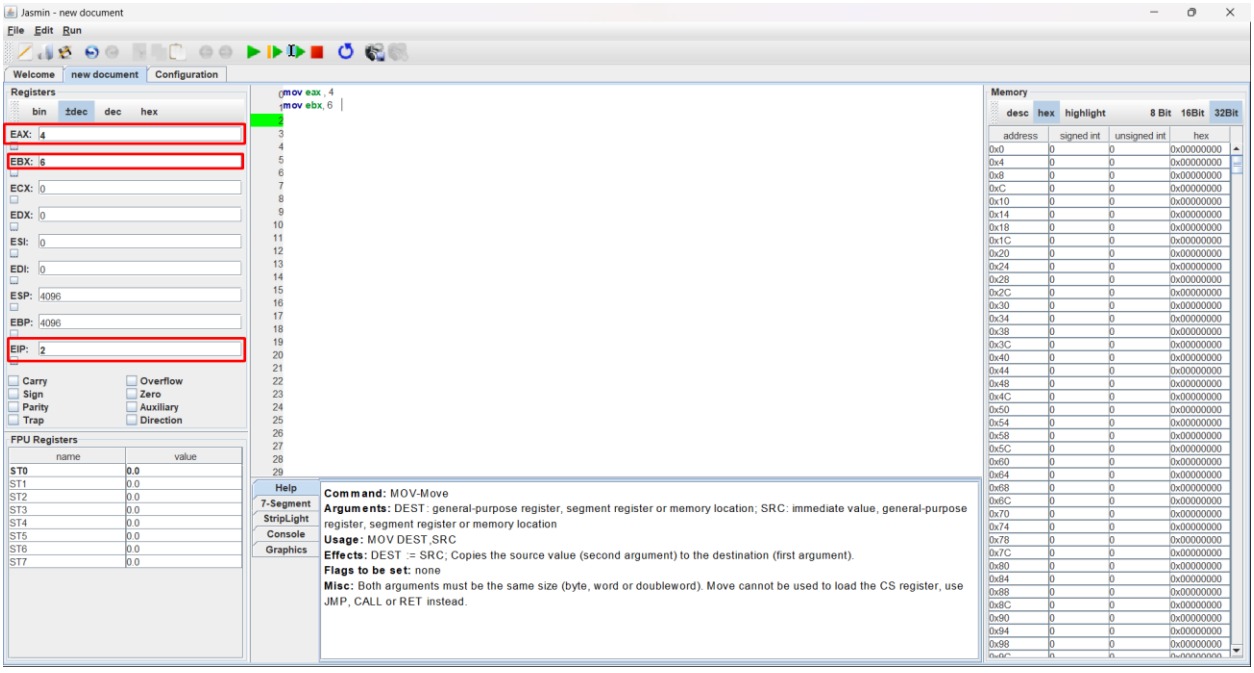
```
mov eax , 4
mov ebx, 6
```

Câu lệnh trên có nghĩa là gán cho eax giá trị = 4 và ebx có giá trị = 6

Sau khi viết chương trình xong, chúng ta sẽ bắt đầu thực thii chương trình bằng những button có sẵn bên trong chương trình jasmin



Khi bắt đầu chạy chương trình, chúng ta sẽ để ý giá trị bên thanh ghi, ta sẽ thấy rằng giống như chúng ta đã dự đoán ben đầu, eax sẽ giữ giá trị là 4, ebx sẽ giữ giá trị là 6, EIP sẽ có giá trị là 2 tại đã chạy qua hai instruction là 0 và 1



Tiếp theo chúng ta sẽ thực hiện một bài toán khó hơn và nó sẽ liên quan tới bộ nhớ

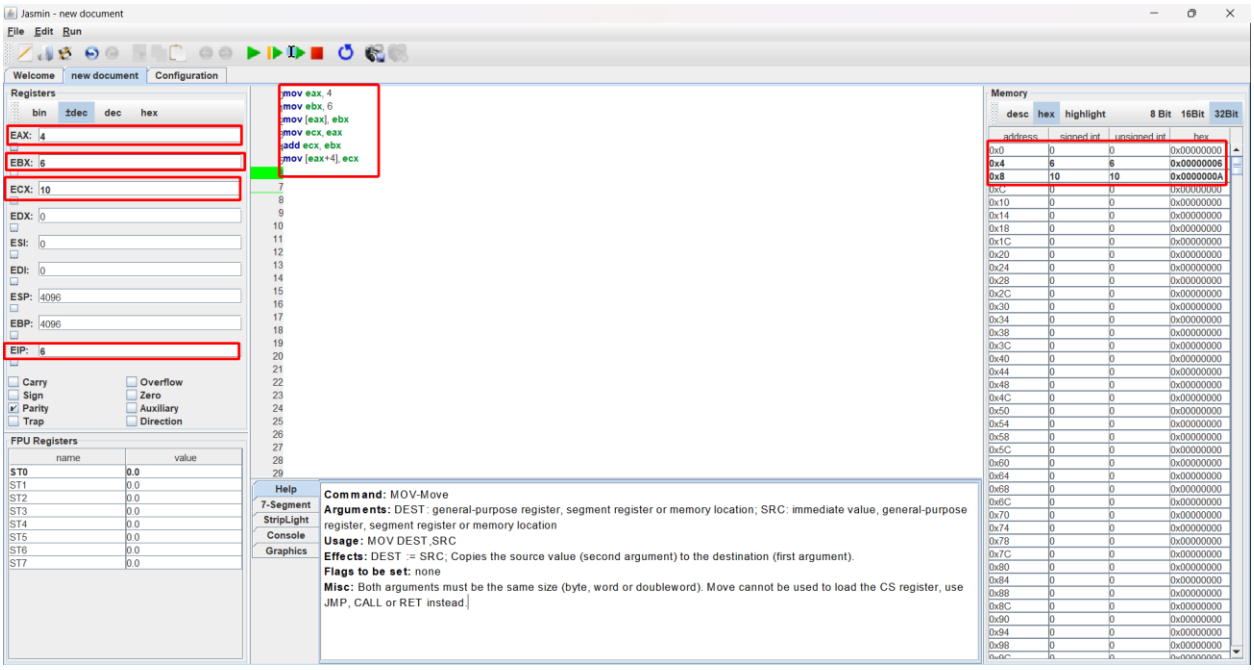
```
mov eax, 4
mov ebx, 6
mov [eax], ebx
```

```
mov ecx, eax
add ecx, ebx
mov [eax+4], ecx
```

Đến với bài toán này, ban đầu chúng ta vẫn sẽ cho **eax** và **ebx** có giá trị lần lượt là 4 và 6. Sau đó chúng ta sẽ cho giá trị của ebx gắn với trỏ tới phần memory được trỏ bởi eax (0x4). Sa đó chúng ta sẽ gán giá trị **eax** vào **ecx.** Sử dụng hàm add để **add** giá trị của **ebx** vào **ecx** là 6 + 4 = 10, nhưu vậy giá trị của **ecx** hiện tại đang là 10. Sau đó chúng ta truyền giá trị của **ecx** vào phần con trỏ của **eax** + **4** byte, tức nghĩa là phần memory 0x8 đang có giá trị là 10

Nói tóm lai chúng ta sẽ có như sau:

- eax: 4
- ebx: 6
- ecx: 10
- vùng nhớ 0x4: 6
- vùng nhớ 0x8: 10



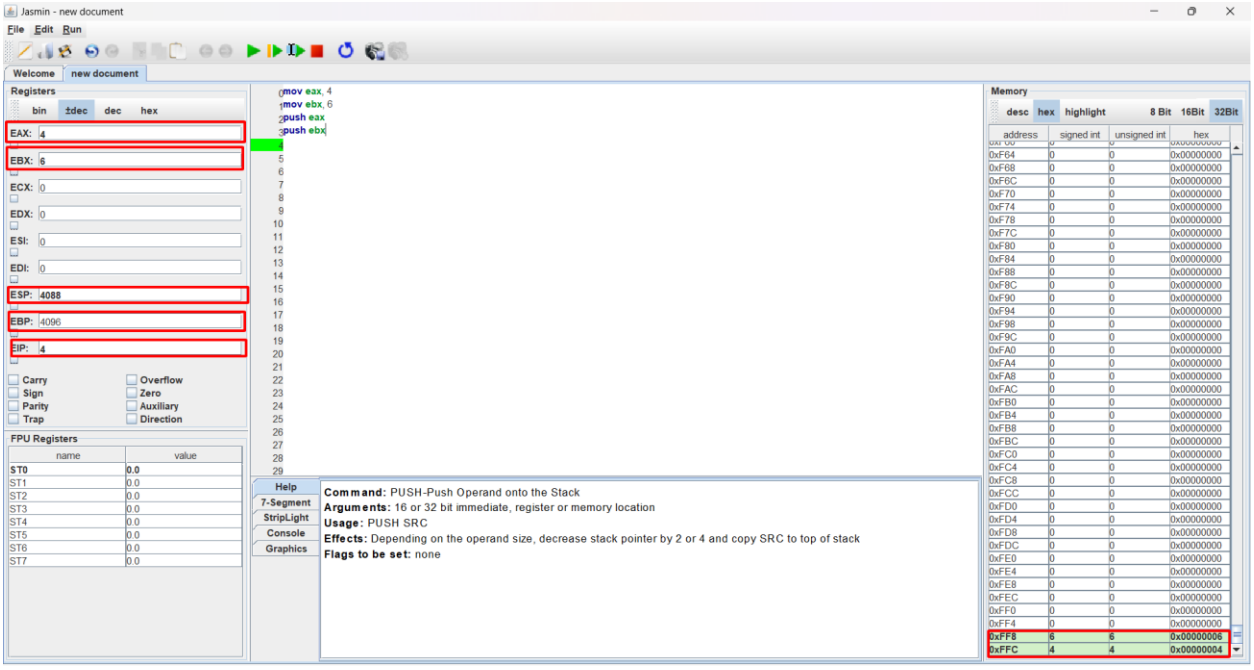
Như chúng ta đã dự đoán từ trước, những vùng này đã có những giá trị như chúng ta đã dự đoán

Using the Stack

Bây giờ chúng ta sẽ thử nghiệm với trên stack, trong stack, ebp sẽ có giá trị cao nhất, trong trường hợp này la 4096, tại bộ nhớ chỉ cấp phát 4096 byte. Khi càng ngày càng push vào stack thì phần push sau đó sẽ luôn có địa chỉ bé hơn địa chỉ của ebp (base pointer)

```
mov eax, 4
mov ebx, 6
push eax
push ebx
```

Phân tích câu lệnh, ta thấy được rằng **eax** sẽ lưu giữ giá trị bằng 4, và **ebx** được lưu giữ giá trị là 6, sau đó ta bỏ eax vào bên trong stack, nằm ở giá trị 4096 biểu thị bên trong mã hex: 0xFFC, sau đó thêm **ebx** vào bên trong stack, stack pointer (esp) sẽ trỏ tới ebx, esp sẽ biểu thị giá trị địa chỉ là 0xFF8, vì ebp là stack base pointer nên giá trị vẫn giữ nguyên là 4096



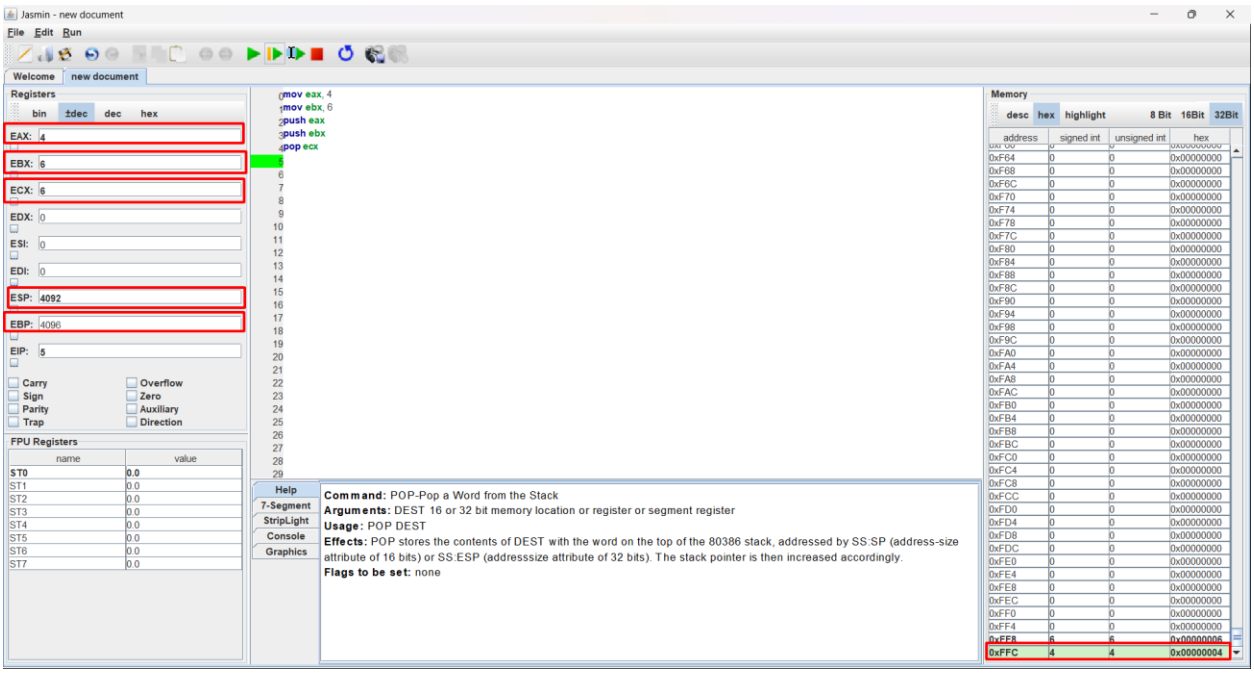
Understanding Pop

Câu lệnh pop trái ngược với push là sẽ đưa những thằng được đưa vào stack sẽ được gán vào vị trí nào đó trên thnah ghi và sau đó giá trị đó sẽ được biến mấy, thwujc hiện theo cơ chế LIFO

```
mov eax, 4
mov ebx, 6
push eax
push ebx
pop ecx
```

Thực hiện câu lệnh này, như lần trước, ta có thể thấy được câu lệnh sẽ thực thi như ban nãy đã đề cập tới. Tuy nhiên sau khi các câu lệnh được thự hiện tới instruction thứ 3, tới instruction thứ 4, câu lệnh pop đã được sử dụng. Câu lệnh này sẽ lấy giá trị trên đầu của stack hiện tại, là giá trị 6, đem giá trị số 6 vào bên trong **ecx,** và stack pointer lúc này sẽ chỉ về 4092, base pointer vẫn giữ giá trị 4096.

Kiểm tra bằng cách nhập vào assembly, ta có:



Reversing a Sequence

Bây giờ sau khi đã hiểu được cách hoạt động cơ bản của assembly, chúng ta sẽ tiến hành thử test đảo ngược số bên trong assembly bằng stack. Sử dụng các câu lệnh sau:

```
mov eax, 1
mov ebx, 2
mov ecx, 3
mov edx, 4
push eax
push ebx
push ecx
push edx
pop eax
pop ebx
pop ecx
pop edx
```

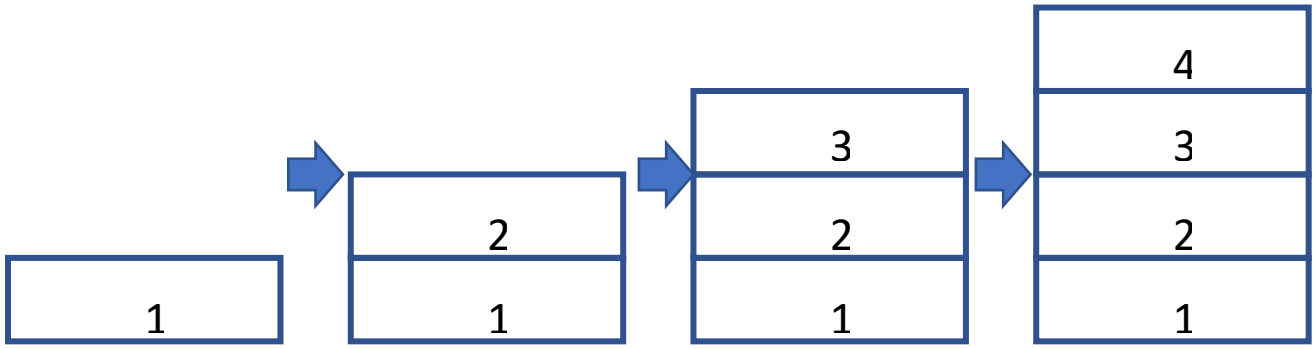
Tiến hành ban đầu, chúng ta sẽ so sánh các giá trị theo từng bước của instruction:

Instruction	Eax	Ebx	Ecx	Edx	Esp	ebp
0	1				4096	4096
1	1	2			4096	4096
2	1	2	3		4096	4096
3	1	2	3	4	4096	4096
4	1	2	3	4	4092	4096
5	1	2	3	4	4088	4096
6	1	2	3	4	4084	4096

7	1	2	3	4	4080	4096
8	4	2	3	4	4084	4096
9	4	3	3	4	4088	4096
10	4	3	2	4	4092	4096
11	4	3	2	1	4096	4096

Dựa trên bảng trên, 4 instruction đầu, ta chỉ đơn giản sử dụng lệnh mov để add giá trị lần lượt là 1,2,3,4 theo vào bên trong lần lượt các thanh ghi **eax, ebx, ecx, edx.**

Sau đó lần lượt các giá trị được đưa vào thanh ghi, được biểu diễn như sau theo 4 instructrion sau đó:



Tiếp theo sau đó, chúng sẽ pop từ từ và chuyển dần cho các giá trrij trên thanh ghi bắt đầu pop 4 cho eax, pop 3 cho ebx, pop 2 cho ecx và pop 1 cho edx