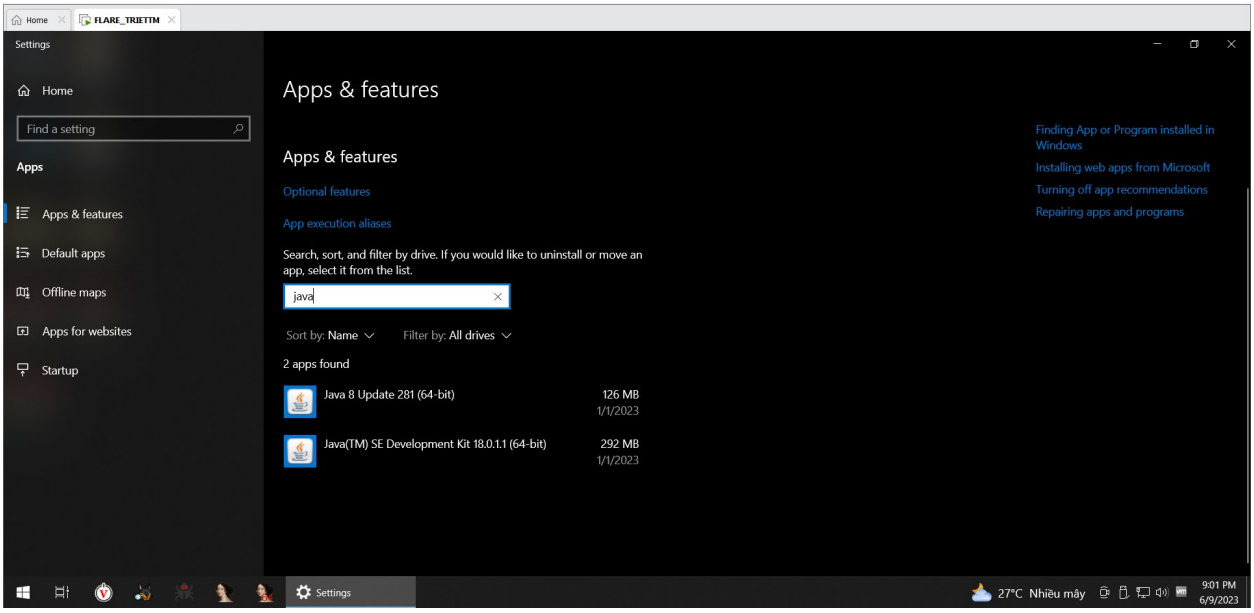
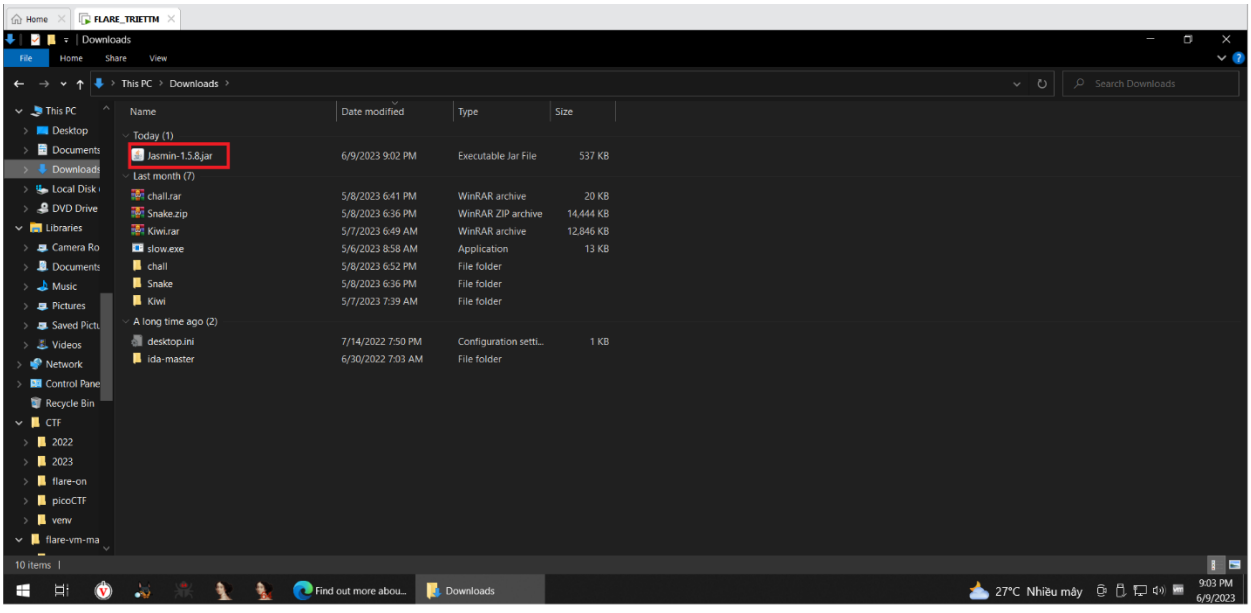
Lab 8: Using Jasmin to run x86 Assembly Code

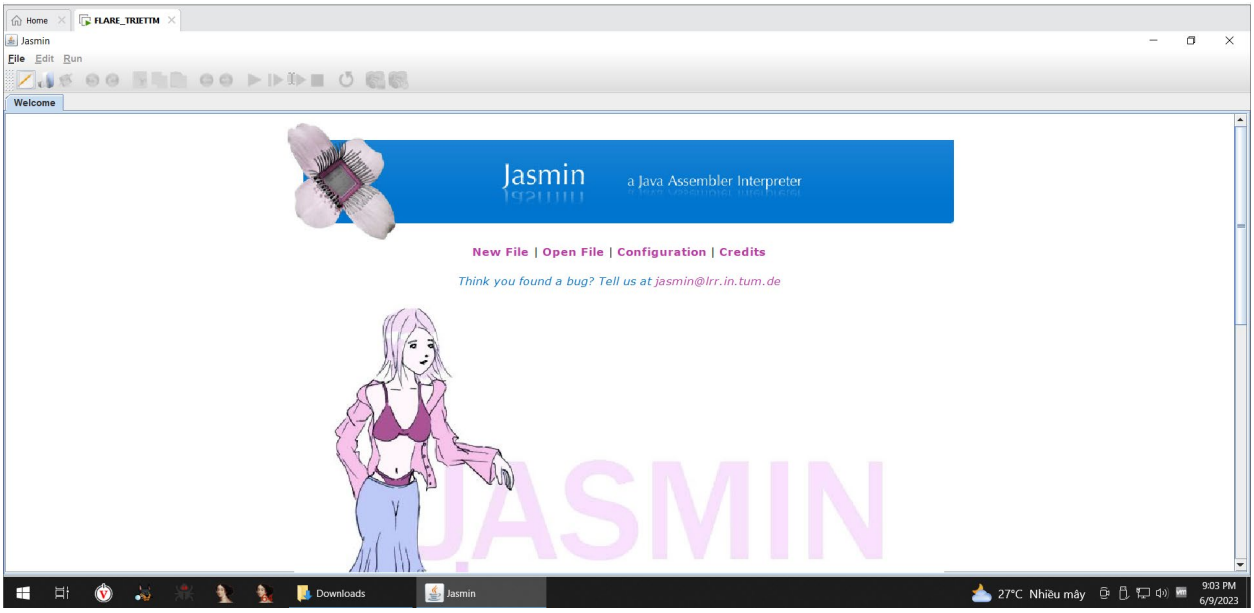
Install Java

Em đã cài đặt Java sẵn trên máy của mình để có thể chạy công cụ Jasmin.



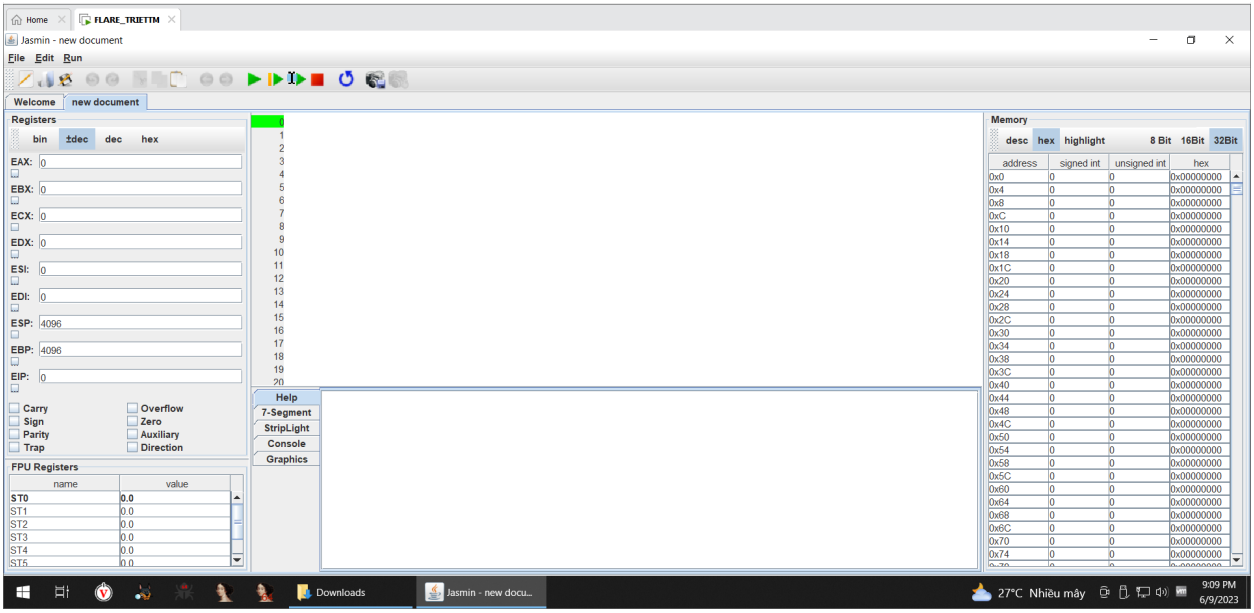
Download Jasmin





Download Jasmin successfully.

Understanding the Jasmin Window



Registers

- Data used during processing is stored in the registers EAX, EBX, ECX, and EDX.
- The ESP (Extended Stack Pointer) contains the address of the top of the Stack.
- The EIP (Extended Instruction Pointer) contains the address of the the next instruction to be processed.

Flags

- These one-bit values that are used for branching. For example the JZ instruction will jump if the Zero flag is 1 (set), and the JNZ instruction will jump if the Zero flag is 0 (cleared).

Code

- This is where you type in commands, such as mov eax,4

Help

- Help messages appear here.

Memory

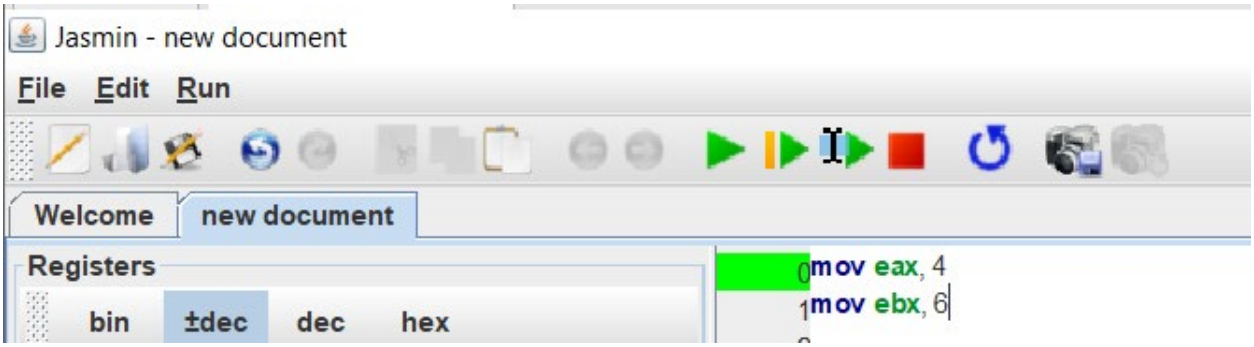
- This processor has 0x1000 = 4096 bytes of RAM, which is not enough to run complete modern programs, but plenty for running little assembly programs for learning purposes.
- With the Memory pane scrolled to the top, as shown in the image above, you see memory that the program will use to store data during processing.
- Scroll this pane to the bottom to see the Stack, which starts at address 0x1000 and grows downward.

Using mov Instructions

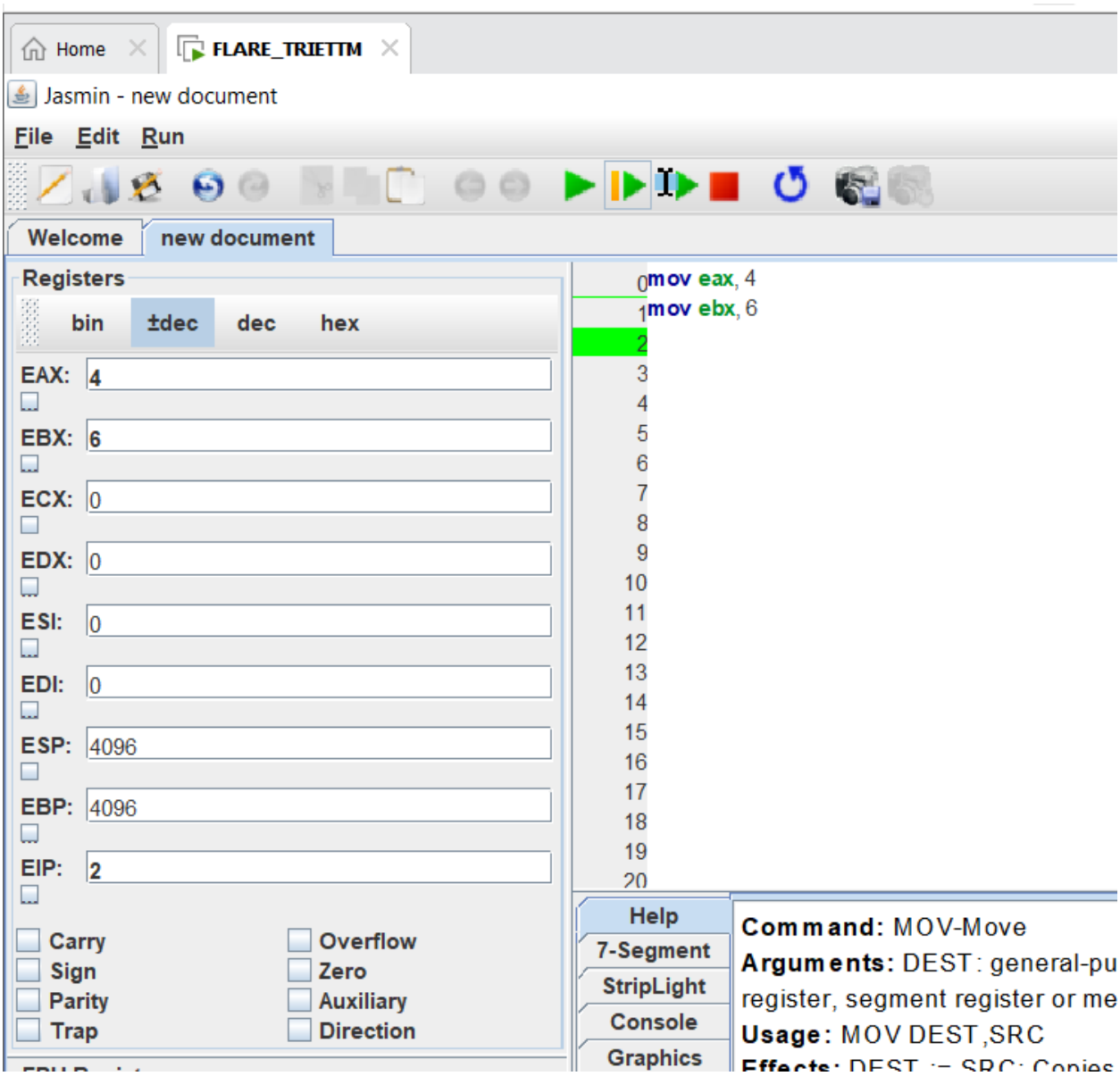
Ta gõ vào phần code của Jasmin đoạn code sau

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

Câu lệnh đầu tiên gán giá trị số 4 cho thanh ghi eax và gán giá trị số 6 cho thanh ghi ebx.



Các nút trên thanh công cụ của Jasmin bao gồm Run, Step và Reset



Sau khi chạy xong chương trình, ta có thể thấy giá trị của các thanh ghi được hiển thị ở khung của sổ bên tay trái.

EAX: 4

EBX: 6

EIP: 2

ESP và EBP: 4096

Ở phần code được viết, ta có thể thấy dòng số 2 được bôi sáng lên, đây là dấu hiệu cho thấy chương trình đang chạy tới vị trí này.

Storing Results in Memory

Thêm một số dòng vào chương trình để nó trông giống như thế này:

```
mov eax, 4
mov ebx, 6
mov [eax], ebx
mov ecx, eax
add ecx, ebx
mov [eax+4], ecx
```

mov eax, 4: Gán giá trị số 4 vào thanh ghi eax

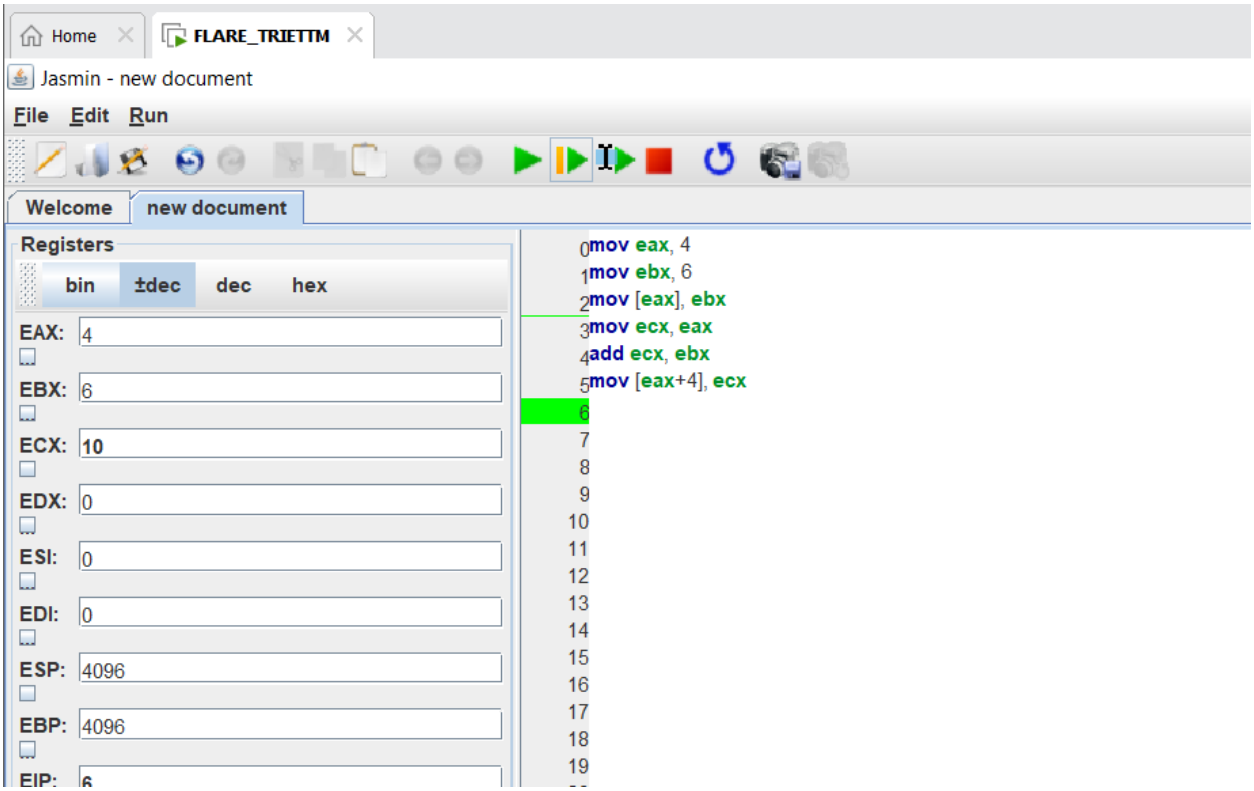
mov ebx, 6: Gán giá trị số 6 vào thanh ghi ebx

mov [eax], ebx: Đưa giá trị của ebx lúc này là 6 vào vị trí tại vùng nhớ được trỏ đến bởi thanh ghi eax(lúc này eax đang có giá trị là 4 nên địa chỉ này là 4 luôn)

mov ecx, eax: Đưa giá trị lúc này của eax là 4 vào thanh ghi ecx

add ecx, ebx: Cộng giá trị thanh ghi ebx lúc này là 6 và giá trị thanh ghi ecx lúc này là 4 lại với nhau kết quả ra 10 rồi lưu kết quả này vào trong thanh ghi ecx. Cuối cùng ecx amng giá trị 10 còn ebx vẫn mang giá trị là 6.

mov [eax+4], ecx: Đưa giá trị của ecx lúc này là 10 vào vùng nhớ được trỏ tới bởi eax + 4 là 8. Vậy ta gán giá trị 10 vào vùng nhớ 8.

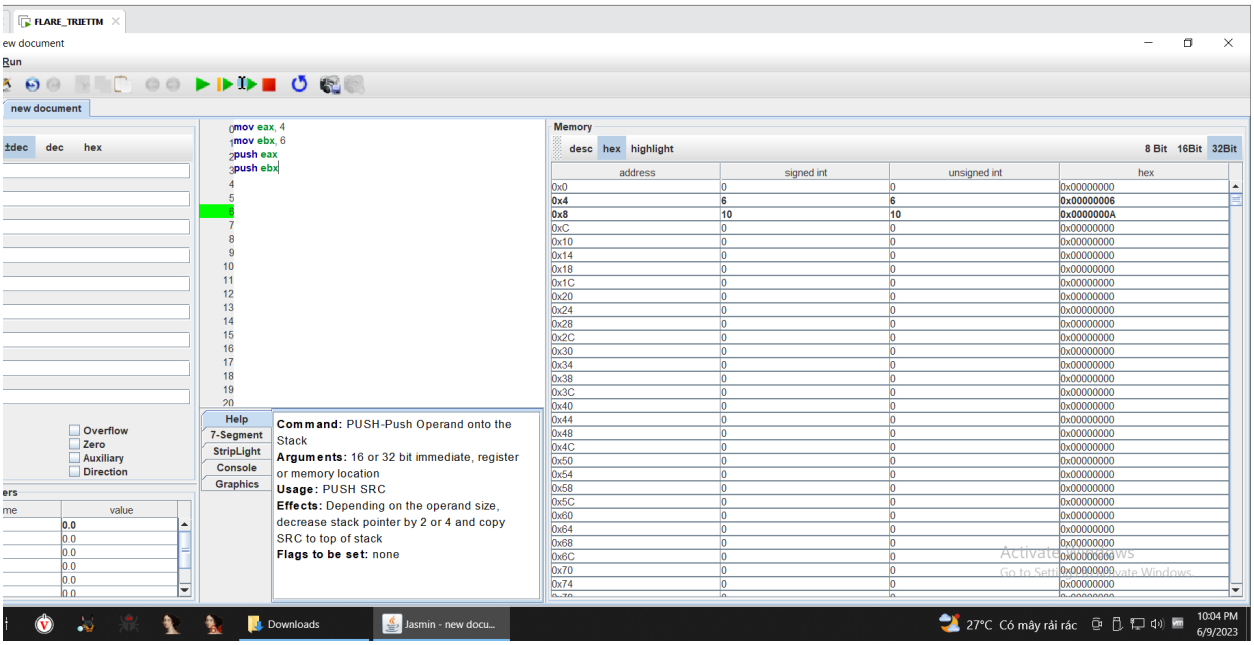


Sau khi kết thúc chương trình kết quả như hình trên.

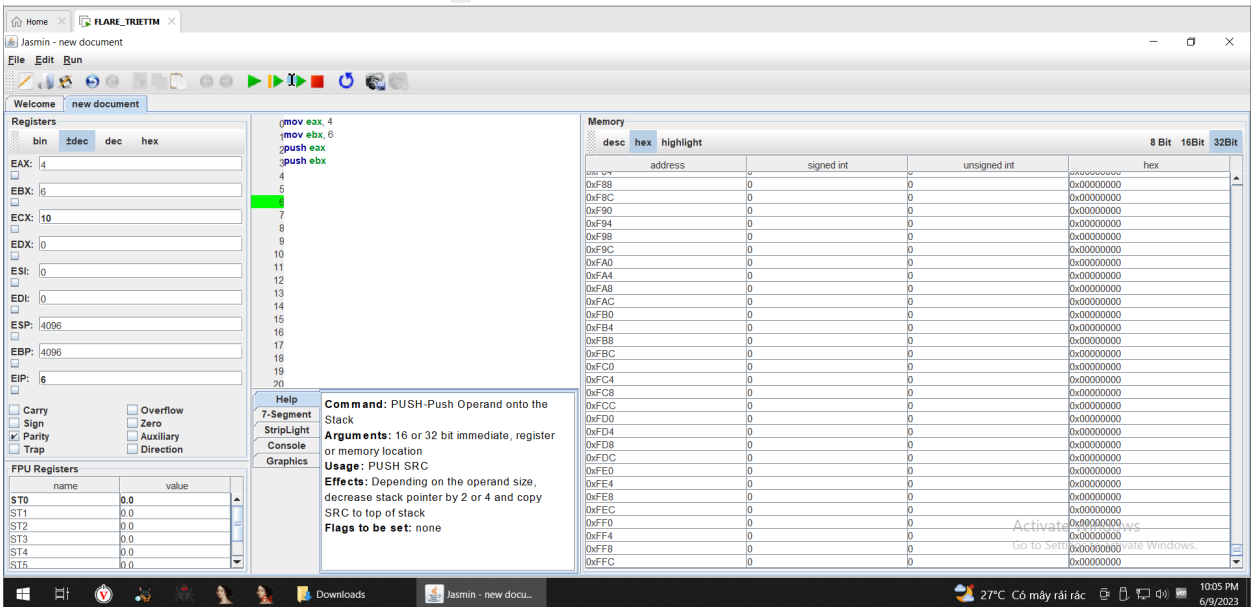
Using the Stack

Ta sẽ điền vào Jasmin một đoạn chương trình như sau đây để tìm hiểu về cách hoạt động cảu stack

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

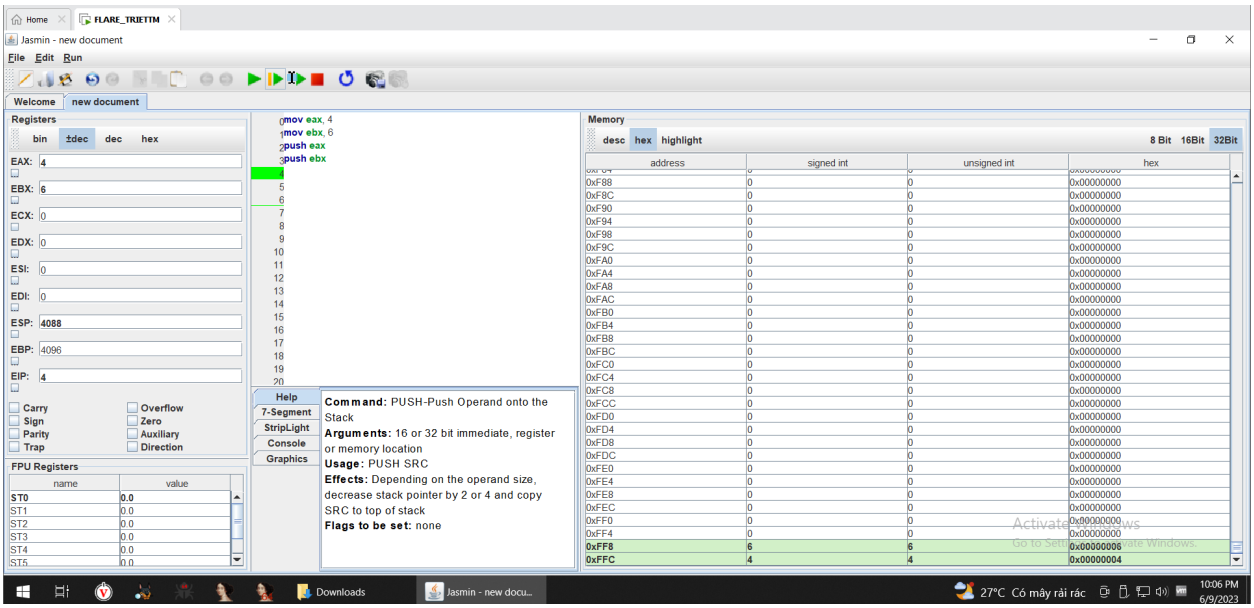


Trước khi chạy chương trình, ta có thể thấy EBP và ESP có giá trị là 4096 hay 0x1000 ở dạng số thập lục phân



Ở đây ta có thể thấy độ lớn tối đa của stack cũng là 0x1000

Understanding Push



Lệnh asm "push" trong ngôn ngữ lập trình assembly được sử dụng để đẩy (push) một giá trị lên đỉnh của ngăn xếp (stack). Ngăn xếp là một vùng nhớ được sử dụng để lưu trữ dữ liệu tạm thời trong quá trình thực thi chương trình.

Cú pháp của lệnh "push" thường có dạng:

```
push <giá_trị>
```

Trong đó, <giá_trị> có thể là một giá trị hằng số hoặc một địa chỉ bộ nhớ.

Khi lệnh "push" được thực thi, giá trị của <giá_trị> sẽ được ghi vào ô nhớ trên đỉnh của ngăn xếp và đỉnh của ngăn xếp sẽ được di chuyển lên trên. Nghĩa là, giá trị cuối cùng được đẩy vào ngăn xếp sẽ trở thành đỉnh mới của ngăn xếp.

Lệnh "push" thường được sử dụng trong việc lưu trữ các tham số hàm, địa chỉ trở về (return address) của các hàm, và các giá trị tạm thời trong quá trình thực thi chương trình.

Ví dụ, nếu ta có đoạn mã assembly sau:

```
push 10
```

Đoạn mã này sẽ đẩy giá trị 10 lên đỉnh của ngăn xếp. Sau khi thực thi lệnh này, đỉnh của ngăn xếp sẽ chứa giá trị 10.

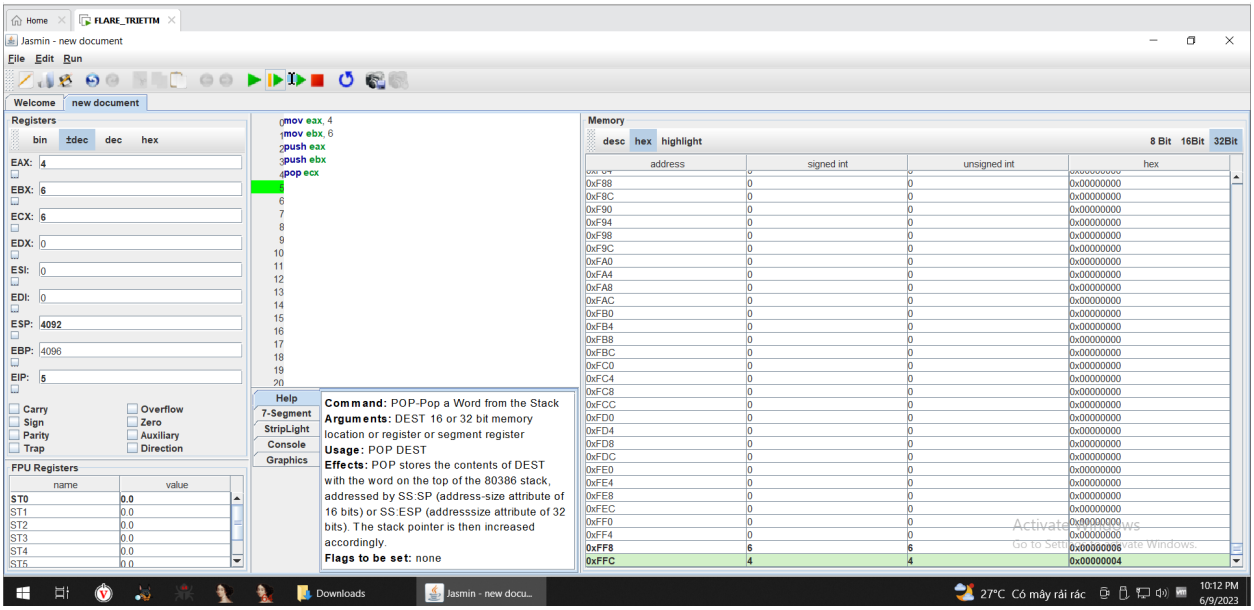
Hay trong chính trường hợp đoạn code trên, ta sẽ push 2 giá trị chauws bên trong 2 thanh ghi eax và ebx lên stack.

Understanding Pop

Thêm một đoạn pop nữa vào code nó sẽ trông như thế này:

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

```
push ebx
pop ecx
```



Lệnh asm "pop" trong ngôn ngữ lập trình assembly được sử dụng để lấy (pop) giá trị từ đỉnh của ngăn xếp (stack). Ngăn xếp là một vùng nhớ được sử dụng để lưu trữ dữ liệu tạm thời trong quá trình thực thi chương trình.

Cú pháp của lệnh "pop" thường có dạng:

```
pop <đích>
```

Trong đó, <đích> có thể là một thanh ghi hoặc một vị trí bộ nhớ để lưu trữ giá trị được lấy từ đỉnh của ngăn xếp.

Khi lệnh "pop" được thực thi, giá trị từ đỉnh của ngăn xếp sẽ được lấy ra và gán vào <đích>. Sau đó, đỉnh của ngăn xếp sẽ được di chuyển xuống (giảm địa chỉ) để trỏ tới phần tử tiếp theo trong ngăn xếp

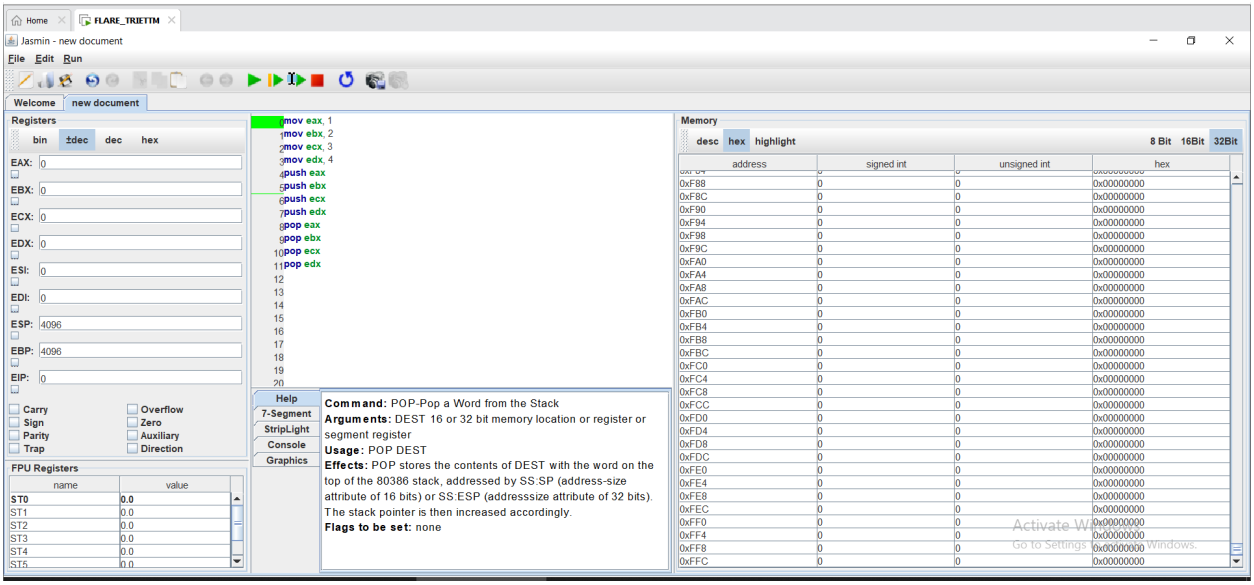
Lệnh "pop" thường được sử dụng để lấy giá trị từ ngăn xếp sau khi giá trị đó đã được đẩy vào bằng lệnh "push". Nó thường được sử dụng để lấy các tham số hàm, giá trị trả về từ hàm hoặc các giá trị tạm thời trong quá trình thực thi chương trình.

Ví dụ, như đoạn mã asm trên:

```
pop ecx
```

Tức là lấy ecx từ đỉnh stack ra khỏi nó.

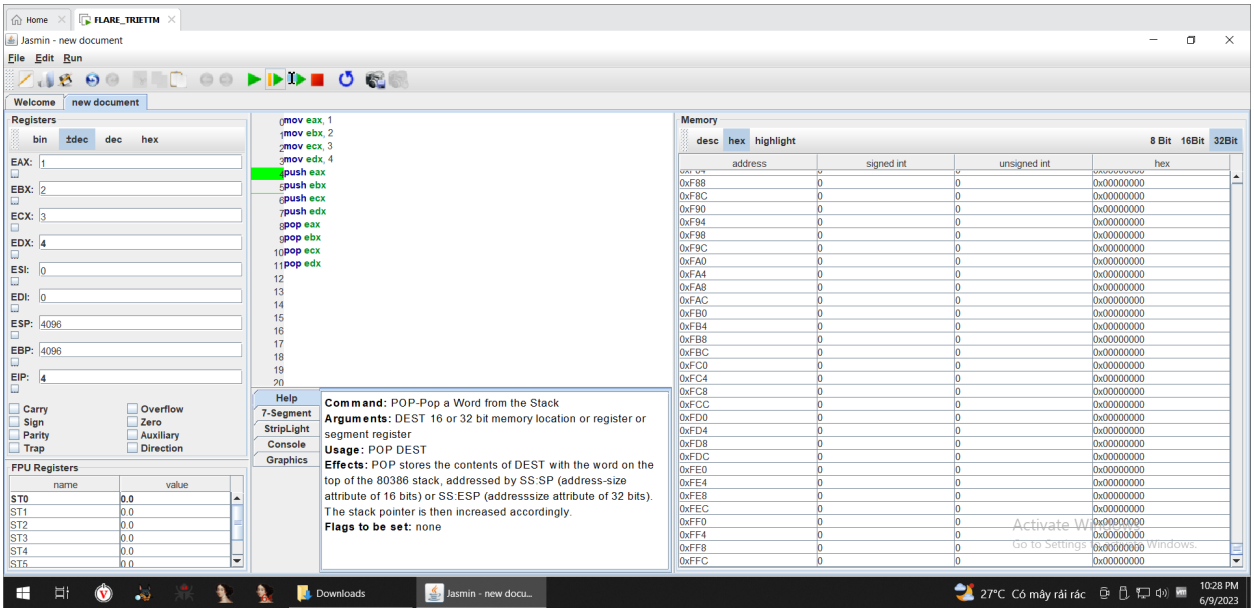
Reversing a Sequence



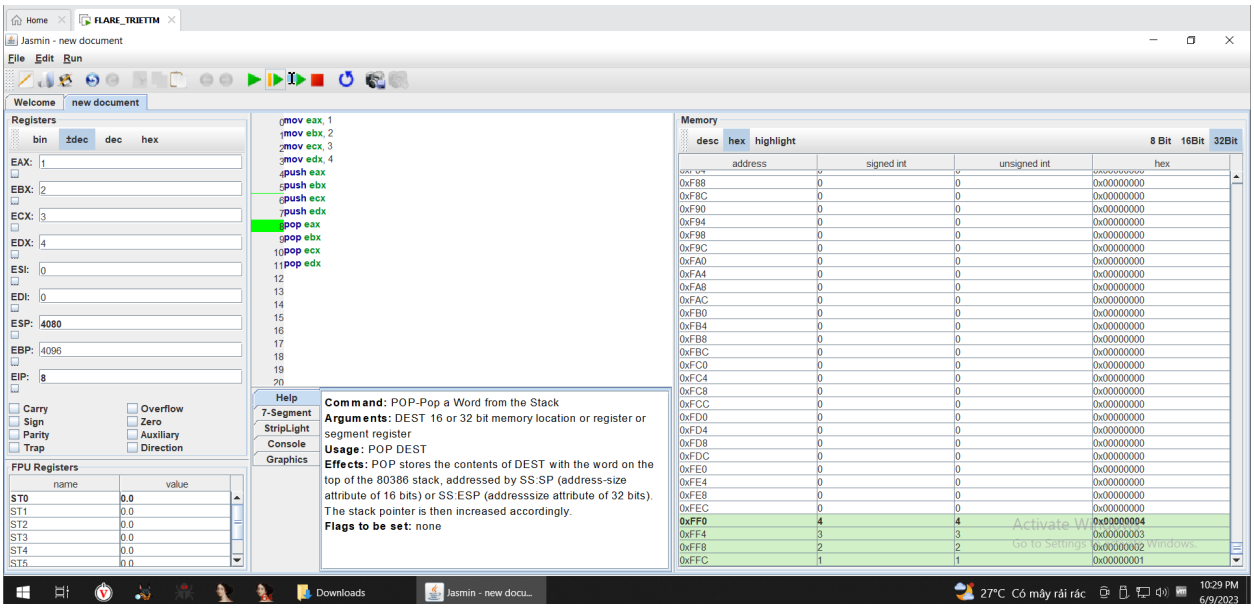
Các lệnh này tải giá trị vào bốn thanh ghi, đẩy chúng vào ngăn xếp theo thứ tự và lấy chúng ra khỏi ngăn xếp theo thứ tự.

Tuy nhiên, vì ngăn xếp là một cấu trúc FILO (First In, Last Out), điều này làm đảo ngược thứ tự các giá trị.

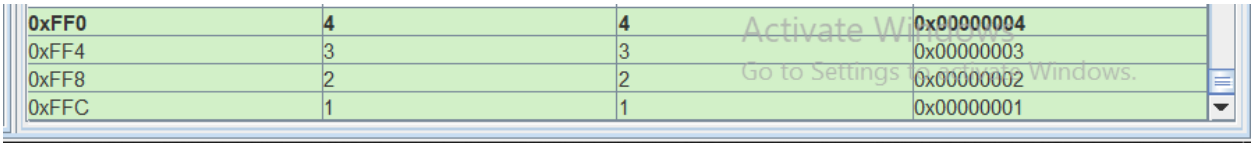
Đẩy Bước này bốn lần để chỉ thực thi bốn lệnh đầu tiên, như được thể hiện dưới đây:

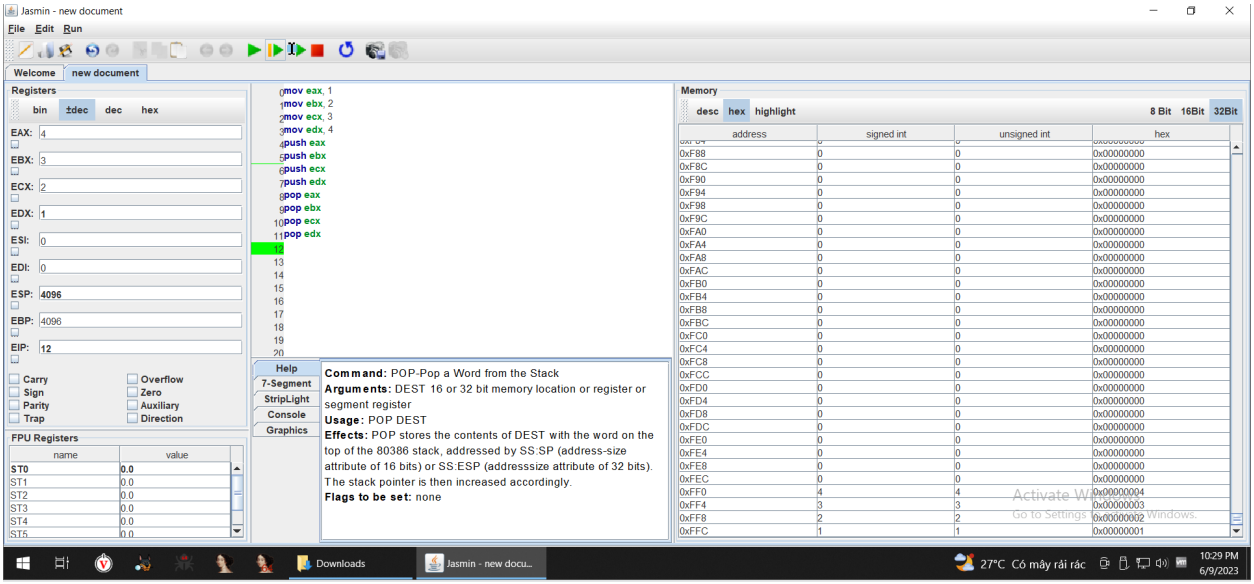


Giá trị của các thanh ghi đã được tải. Bây giờ chúng sẽ được đẩy lên stack.

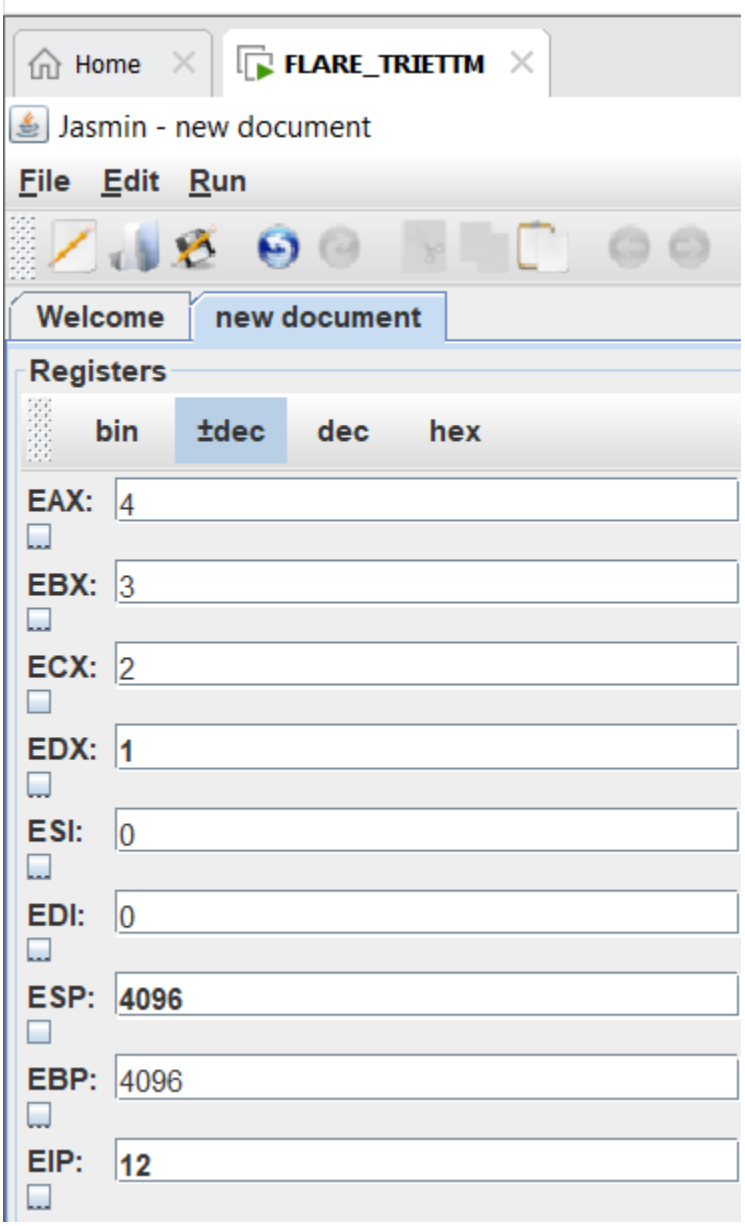


Như ta có thể thấy các giá trị đã được đẩy lên stack





Và sau khi thực hiện các lệnh pop tới các thanh ghi làm đích thì ta đã đảo ngược giá trị của các thanh ghi lại nhờ việc sử dụng stack.


Home
FLARE_TRIETTM
Jasmin - new document
File
Edit
Run
Welcome
new document
Registers
bin
±dec
dec
hex
EAX:
4
EBX:
3
ECX:
2
EDX:
1
ESI:
0
EDI:
0
ESP:
4096
EBP:
4096
EIP:
12