

## USO DEV C++: DEBUGGING

### Codice

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int var = 3;
    int varA = 0;
    int varB = 0;

    if(var == 3)
        varA = var;
    var++;
    varB = var;

    system ("PAUSE");
    return 0;
}
```

```
test.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int var = 3;
    int varA = 0;
    int varB = 0;

    if(var == 3)
        varA = var;
    var++;
    varB = var;

    system ("PAUSE");
    return 0;
}
```

### Primo step

The screenshot shows the Dev-C++ 4.9.9.2 IDE with the debugger active. The main window displays the C++ code with the line `int var = 3;` highlighted. The 'Progetto' pane on the left shows the variable values: `var = 1999412788`, `varA = 2686792`, `varB = 2`, and `$eax = 0x0`. The 'Finestra CPU' window shows the assembly code for the `main` function, with the `ESP` register highlighted in red. The 'Registri' pane on the right shows the `EAX` register set to `0x0`.

Le variabili contengono valori casuali

L'accumulatore (registro EAX) è inizializzato a zero

## Secondo step

Ora **var** contiene il valore 3

The screenshot shows the Dev-C++ IDE with the following components:

- Project Explorer:** Shows a project named "[\*] test.c" with a list of variables: `var = 3`, `varA = 2686792`, `varB = 2`, and various system registers like `$eax = 0x0`, `$es = 0x2b`, etc.
- Code Editor:** Displays the C source code for `main()`. The lines `int var = 3;` and `int varA = 0;` are highlighted in yellow. The code includes `<stdio.h>` and `<stdlib.h>`, and contains a loop that increments `var` until it reaches 3, then prints "PAUSE" and returns 0.
- Finestra CPU (CPU Window):** Shows the assembly code for the `main` function. The current instruction is `0x4012c1 <main+49>: mov dword ptr [ebp], eax`, which corresponds to the assignment `varA = var;` in the C code.
- Sintassi Assembler:** Shows the Intel syntax selected.
- Registri:** A list of registers with their current values: `EAX: 0x0`, `EBX:`, `ECX:`, `EDX:`, `ESI:`, `EDI:`, `EBP:`, `ESP:`, `EIP:`, `CS:`, `DS:`, `SS:`, `ES:`.

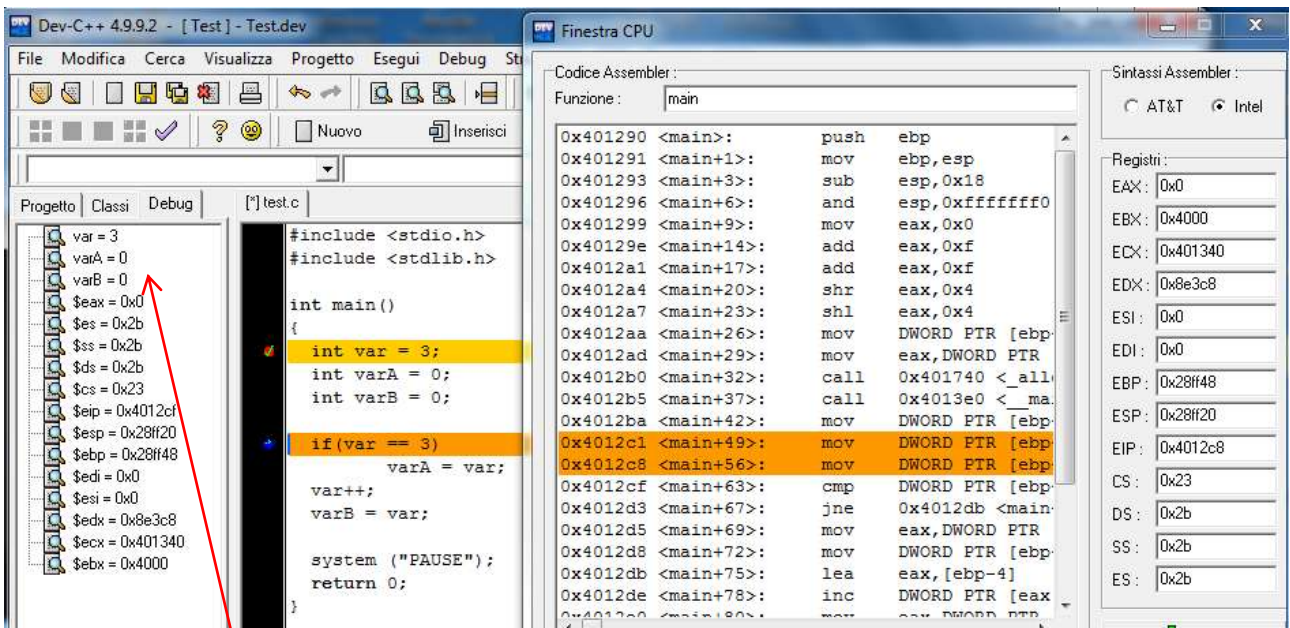
## Terzo step

Ora **varA** contiene il valore 0 e vengono visualizzate le variazioni di contenuto degli altri registri

The screenshot shows the Dev-C++ IDE with the following components:

- Project Explorer:** Shows the same project "[\*] test.c" with updated variable values: `var = 3`, `varA = 0`, `varB = 2`, and various system registers like `$eax = 0x0`, `$es = 0x2b`, etc.
- Code Editor:** Displays the C source code for `main()`. The lines `int var = 3;`, `int varA = 0;`, and `int varB = 0;` are highlighted in yellow. The code includes `<stdio.h>` and `<stdlib.h>`, and contains a loop that increments `var` until it reaches 3, then prints "PAUSE" and returns 0.
- Finestra CPU (CPU Window):** Shows the assembly code for the `main` function. The current instruction is `0x4012c1 <main+49>: mov dword ptr [ebp], eax`, which corresponds to the assignment `varA = var;` in the C code.
- Sintassi Assembler:** Shows the Intel syntax selected.
- Registri:** A list of registers with their current values: `EAX: 0x0`, `EBX: 0x4000`, `ECX: 0x401340`, `EDX: 0x8e3c8`, `ESI: 0x0`, `EDI: 0x0`, `EBP: 0x28ff48`, `ESP: 0x28ff20`, `EIP: 0x4012c1`, `CS: 0x23`, `DS: 0x2b`, `SS: 0x2b`, `ES: 0x2b`.

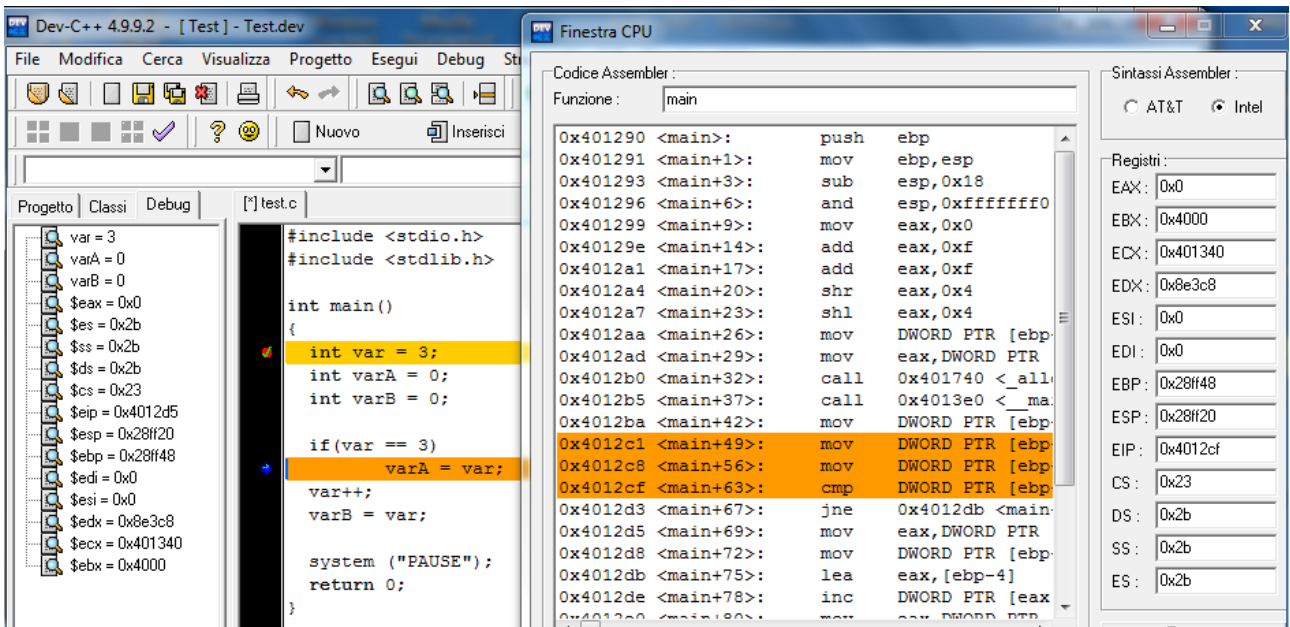
### Quarto step (quarta e quinta istruzione)



Anche le variabili **varA** e **varB** vengono inizializzate

### Quinto step

### Confronto tra contenuto di **var** ed il valore 3

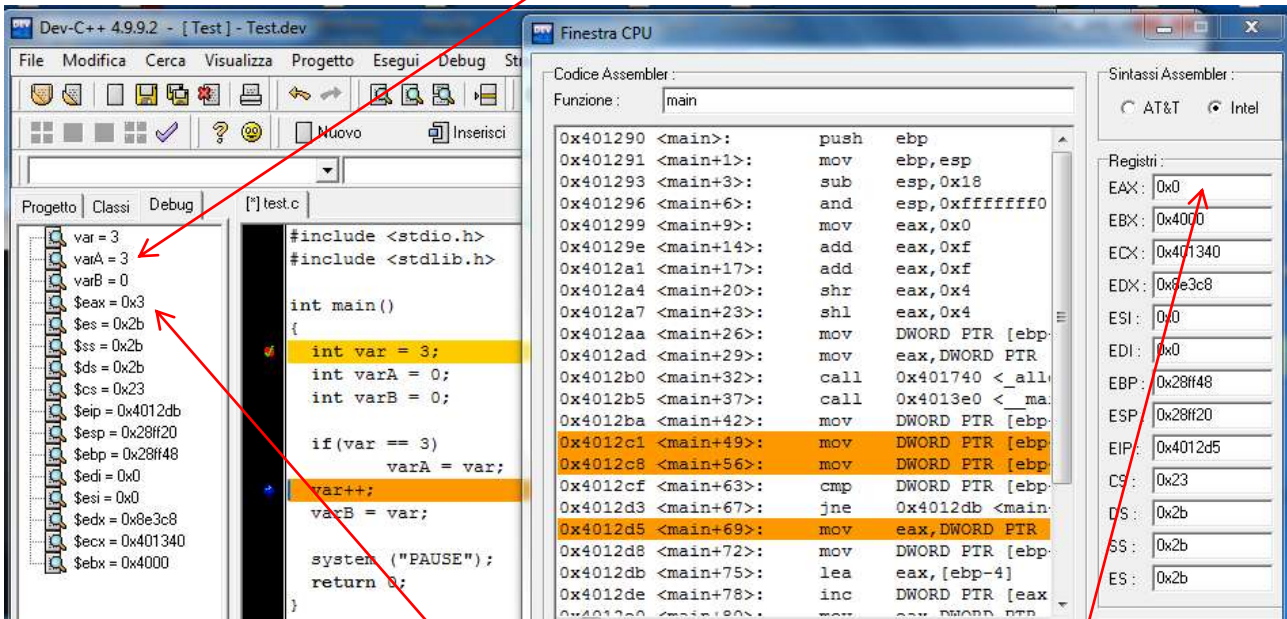


Essendo vera la condizione, non viene eseguito il salto ma si *punta* l'istruzione per copiare il valore di **var** in **varA**



## Sesto step

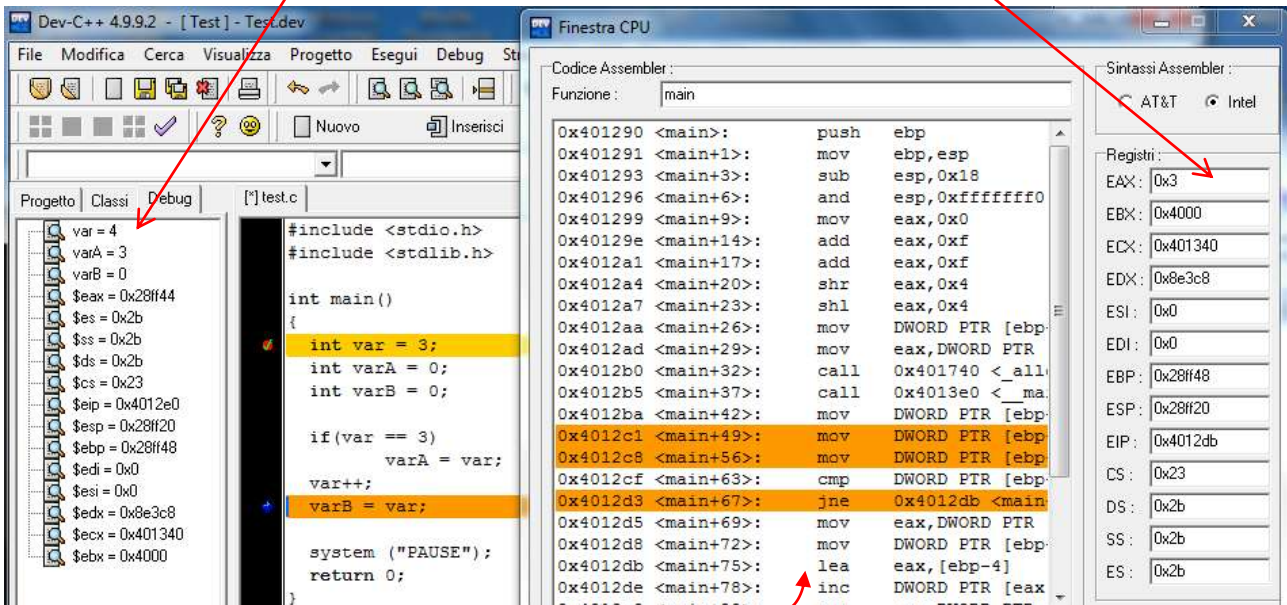
Ora **varA** contiene lo stesso valore di **var** cioè 3



**Nb:** nell'accumulatore è memorizzato tale valore(3) ma nella finestra CPU sarà visualizzato allo step successivo

## Settimo step

Il contenuto di **var** è **incrementato** interessando il contenuto dell'accumulatore (valore prima dell'incremento)

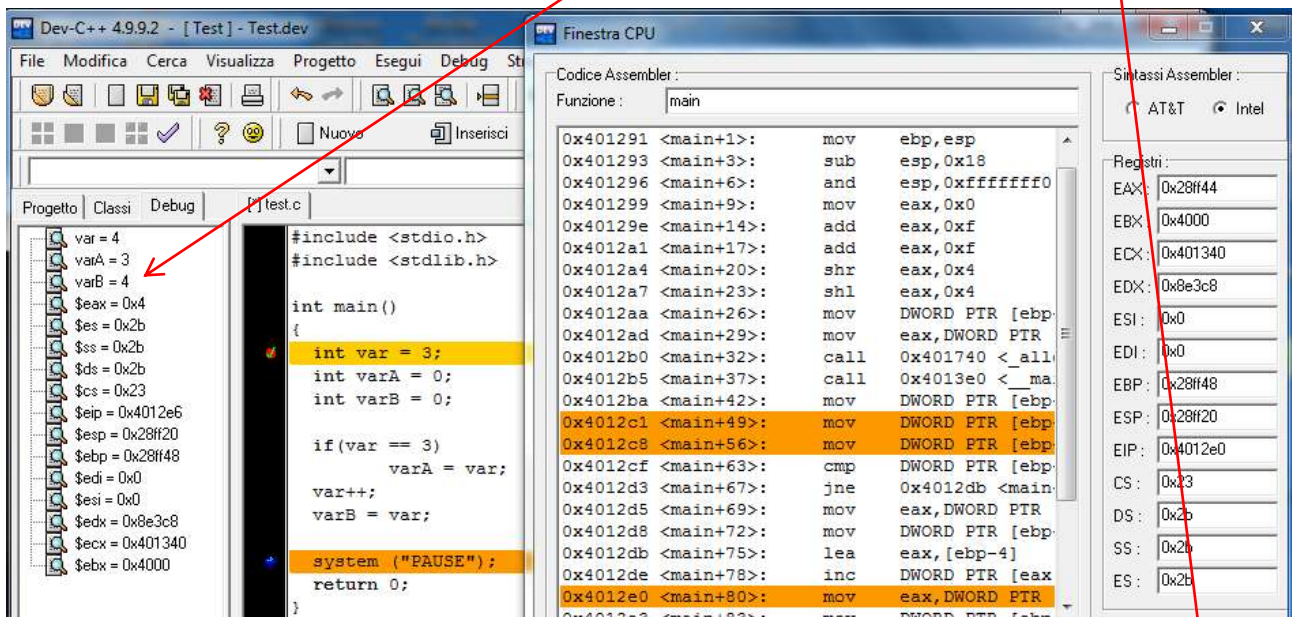


**LEA Load Effective Address**

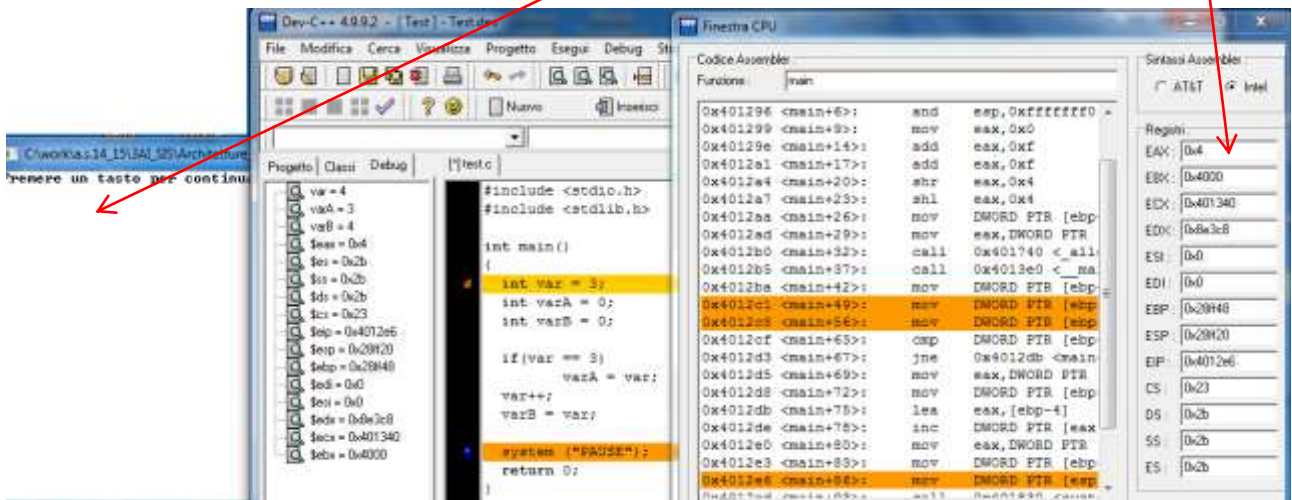
**Nb:** nell'accumulatore è memorizzato nuovo valore (indirizzo ebp-4) ma nella finestra CPU sarà visualizzato allo step successivo; registro BP (Base Pointer) punta ad un punto all'interno dello stack

## Ottavo step

Ora la variabile **varB** contiene il valore di **var** cioè **4** influenzando il contenuto dell'accumulatore

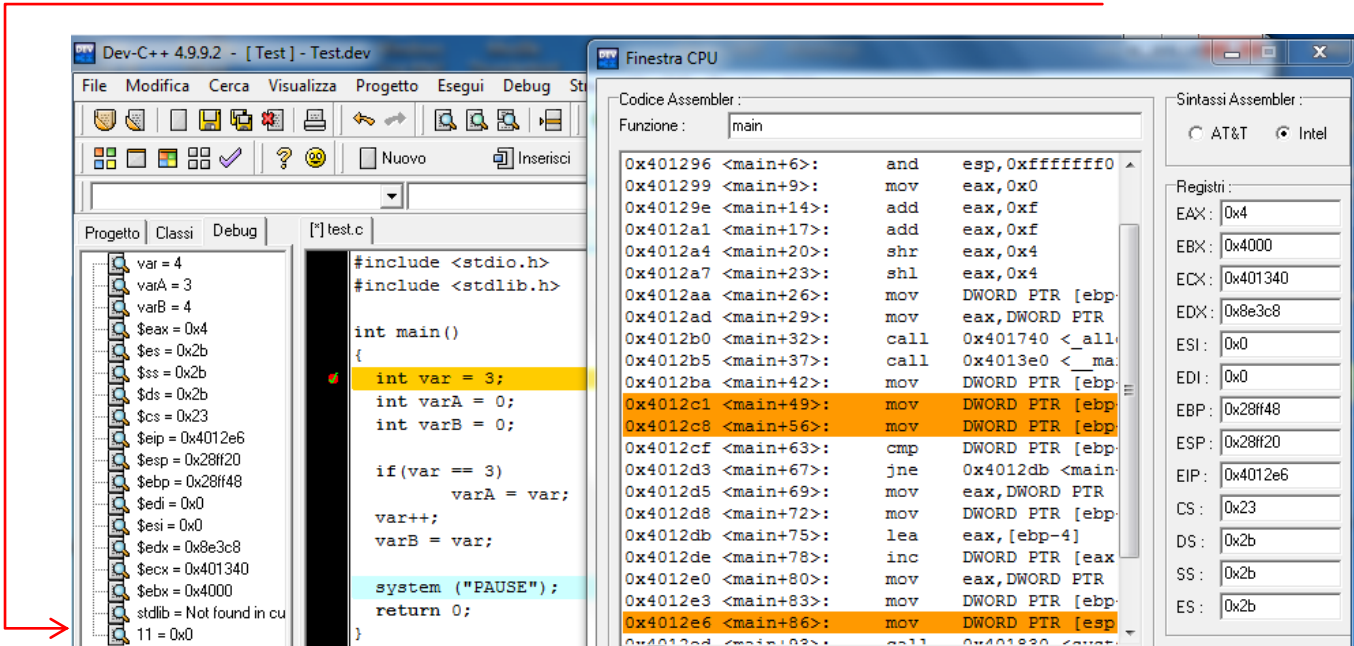


L'esecuzione della chiamata a SO apre la finestra console e attende la pressione di un tasto:

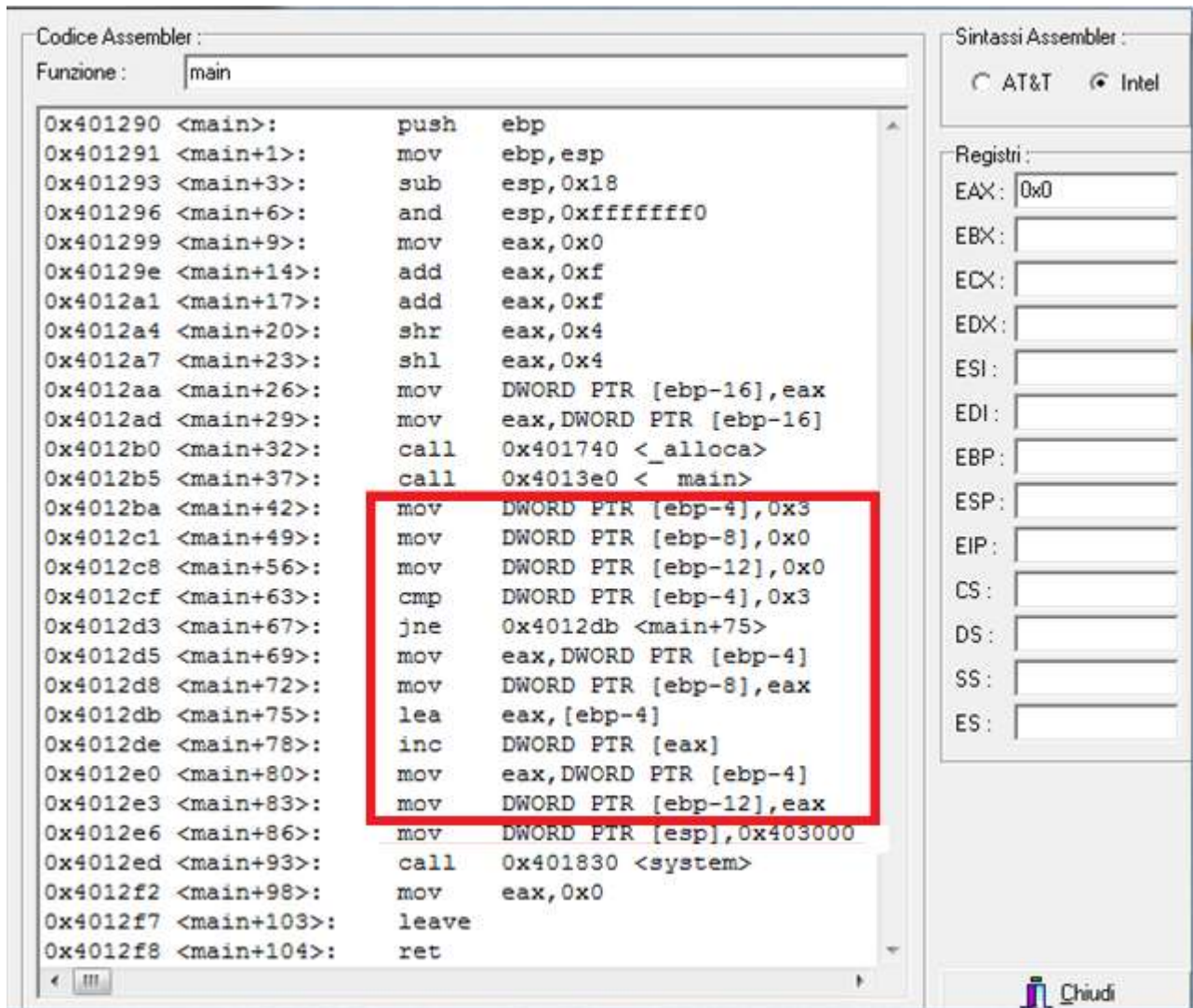




Dopo aver premuto un tasto nella finestra console, l'esecuzione termina (11 linee di codice) :



### Codice Assembly - Intel



## Simulando in JASMIN

Welcome test\_c.asm

Registers

	bin	±dec	dec	hex
EAX:		4		
EBX:		0		
ECX:		0		
EDX:		0		
ESI:		0		
EDI:		0		
ESP:		4096		
EBP:		4096		
EIP:		11		

```

0 mov DWORD [ebp-4], 0x3
1 mov DWORD [ebp-8], 0x0
2 mov DWORD [ebp-12], 0x0
3 cmp DWORD [ebp-4], 0x3
4 jne label
5 mov eax, DWORD [ebp-4]
6 mov DWORD [ebp-8], eax
7 label: lea eax, [ebp-4]
8 inc DWORD [eax]
9 mov eax, DWORD [ebp-4]
10 mov DWORD [ebp-12], eax

```

11

Memory

desc	hex	highlight	8 Bit	16Bit	32Bit
address	signed int	unsigned int	hex		
0xFF4	4	4	0x00000004		
0xFF8	3	3	0x00000003		
0xFFC	4	4	0x00000004		

**EBP: il Registro BP (Base Pointer)** punta ad un punto all'interno dello stack  
4096 decimale corrisponde a 1000H ed EBP-4 cioè 4092 corrisponde a FFCH

Anche, allocando all'inizio della memoria

```

MOV [0x4], 03; assegnazione var = 3
MOV [0x8], 00; assegnazione varA = 0
MOV [0xC], 00; assegnazione varB = 0
CMP [0x4], 03; confronta con dato
JNZ esci
; istruzione del costrutto alternativa
MOV EAX, [0x4]
MOV [0x8], EAX; assegna ad altra variabile varA il valore di var
esci:
; proseguo programma
MOV EBX, [0x4]
INC EBX
MOV [0x4], EBX; incrementa var
MOV [0xC], EBX; assegna ad altra variabile varB il valore di var incrementato

```

Nb: Senza poter emulare le chiamate a SO