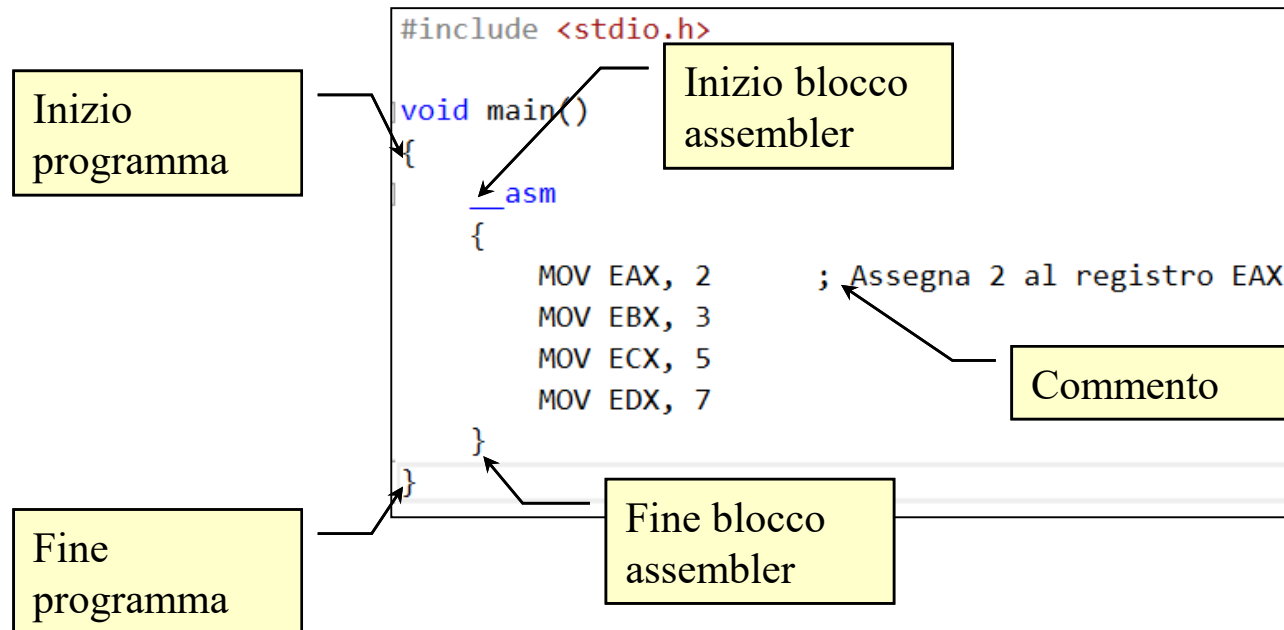


1 – Programma C con assembler inline

Creare un nuovo progetto contenente il seguente programma:



- Le parentesi graffe esterne indicano l'inizio e la fine del programma C, quelle interne, precedute dalla parola chiave “`__asm`”, delimitano il blocco di assembler inline.
- Il carattere punto e virgola indica l'inizio di un commento all'interno del blocco assembler.

2 – Debug di istruzioni assembler

1. Inserire un breakpoint nella prima riga assembler
2. Compilare ed eseguire il programma in “Debug mode” (F5)
3. Aprire la finestra “Registers” selezionando “Windows→Registers” dal menù Debug (oppure Alt+F5)
4. Avanzare di una istruzione alla volta con “Step Over” (F10)
5. Osservare l’effetto delle istruzioni assembler sui registri
6. Scegliere “Continue” per lasciare eseguire il programma fino alla fine

```
1  #include <stdio.h>
2
3  void main()
4  {
5      __asm
6      {
7          MOV EAX, 2
8          MOV EBX, 3
9          MOV ECX, 5
10         MOV EDX, 7
11     }
12 }
```

Registers

EAX = CCCCCCCC EBX = 00220000 ECX = 00000000
EDX = 00CD8580 ESI = 00CD1046 EDI = 0053FCE0
EIP = 00CD169E ESP = 0053FC14 EBP = 0053FCE0
EFL = 00000200

Autos Locals Registers Threads Modules Watch 1

main.c - PrimoProgetto (Global Scope)

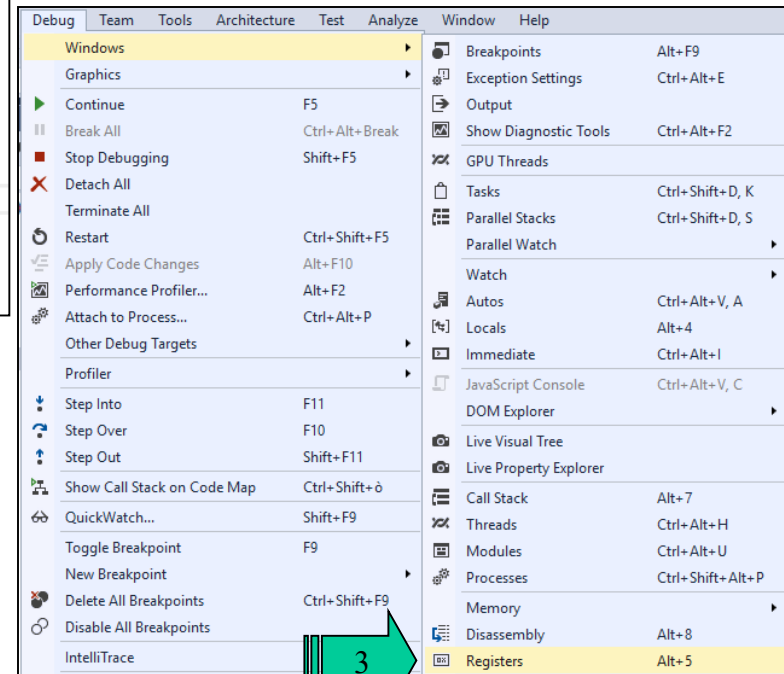
```
1  #include <stdio.h>
2
3  void main()
4  {
5      __asm
6      {
7          MOV EAX, 2
8          MOV EBX, 3
9          MOV ECX, 5
10         MOV EDX, 7
11     }
12 }
```

Registers

EAX = 00000002 EBX = 00000003 ECX = 00000000
EDX = 00CD8580 ESI = 00CD1046 EDI = 0053FCE0
EIP = 00CD16A8 ESP = 0053FC14 EBP = 0053FCE0
EFL = 00000202

Autos Locals Registers Threads Modules Watch 1

• I valori dei registri modificati dall’ultima istruzione sono evidenziati in rosso



3 – Esercizi

- Debug di istruzioni assembler

- Modificare il programma precedente per verificare il funzionamento delle seguenti istruzioni assembler (reg=registro, val=valore costante):

MOV reg,val

MOV reg,reg

ADD reg,val

ADD reg,reg

AND reg,val

AND reg,reg

INC reg

DEC reg

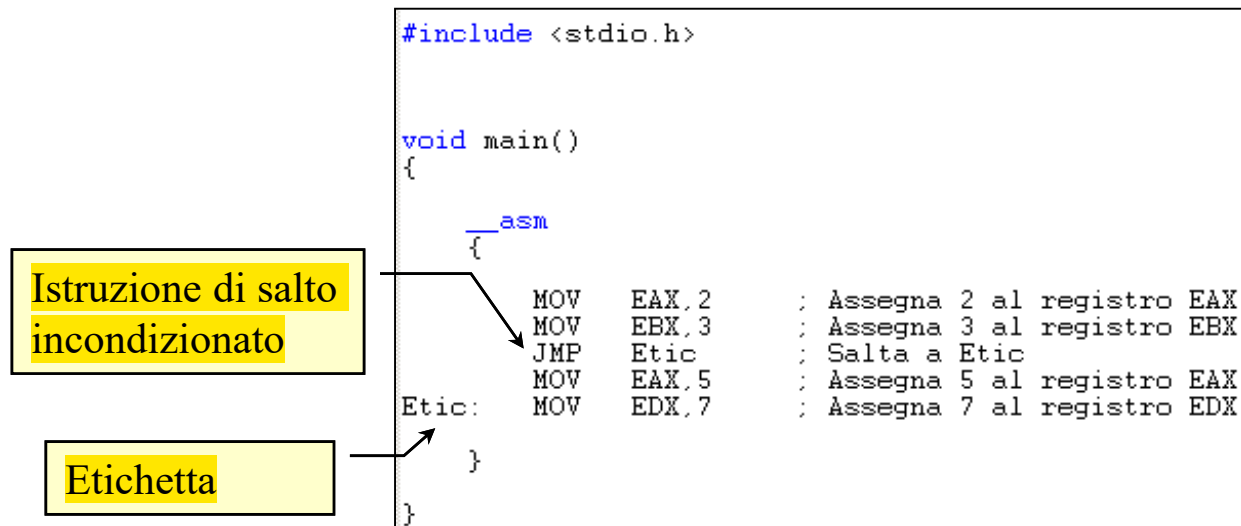
- Verificare il comportamento di altre istruzioni che eseguono operazioni su registri, ad esempio MUL, XOR, DIV, SHR, SHL (consultare la documentazione per la sintassi)

È buona norma commentare ogni istruzione assembler (utilizzando il ';'), ad esempio:

ADD EAX,ECX ; Somma al registro EAX il contenuto del registro ECX

4 – Etichette e salti

Creare un nuovo progetto contenente il seguente programma e verificarne il funzionamento con il debugger:



- L'istruzione "MOV EAX,5" non viene eseguita, in quanto preceduta da una istruzione di salto incondizionato (JMP).
- Le etichette permettono di indicare l'istruzione a cui saltare (ogni etichetta deve terminare con il carattere ':')

5 – Salti condizionali

Trasformare il blocco assembler del programma precedente nel seguente (oppure creare un nuovo progetto) ed eseguirlo passo a passo con il debugger. Qual è il valore di EAX al termine dell'esecuzione?

```
E1:      MOV  EAX,3
        CMP  EAX,5
        JG   Fine
        INC  EAX
        JMP  E1
Fine:    SUB  EAX,5
```

- L'istruzione CMP sottrae il secondo operando al primo, modificando alcuni flag di conseguenza, senza alterare i valori dei due operandi.
- L'istruzione "JG Fine" salta all'etichetta "Fine" solamente se nella precedente operazione CMP il primo operando era maggiore del secondo, altrimenti l'esecuzione prosegue.

N.B. La finestra "Registers" mostra il valore esadecimale dell'intero registro che contiene i flag (EFL): per ricavare il valore di un singolo flag, è necessario convertire il numero in base 2 e controllare il bit corrispondente (ad esempio il flag di riporto CF è il bit meno significativo di EFL)

6 – Esercizi

- Istruzioni di salto

- Controllare nella documentazione il significato e il funzionamento delle varie istruzioni di salto JE, JL, JG, JLE, JGE, JZ, JNZ, ...; scrivere i seguenti programmi assembler, eseguirli passo a passo con il debugger analizzandone il comportamento.

```
L1:    MOV EBX,1
        MOV EAX,10
        ADD EBX,2
        DEC EAX
        JNZ L1
```

```
C1:    MOV EBX,10
        MOV EAX,0
        CMP EBX,EAX
        JE Fine
        INC EAX
        DEC EBX
        JMP C1
```

Fine:

```
Ciclo: MOV ECX,1
        MOV EAX,0
        CMP ECX,10
        JGE Fine
        ADD EAX,ECX
        INC ECX
        JMP Ciclo
Fine:  MOV EDX,EAX
```

```
Ciclo: MOV ECX,2
        MOV EAX,1
        CMP ECX,10
        JG Fine
        MUL ECX
        INC ECX
        JMP Ciclo
Fine:  MOV EDX,EAX
```


7 – Codice assembler efficiente

Chi programma in assembler cerca solitamente di massimizzare la velocità di esecuzione del programma; a tale proposito, molte operazioni vengono solitamente implementate in modo meno intuitivo ma più efficiente. La tabella seguente riporta alcuni esempi: si rifletta sul motivo di tali scelte, seguendone passo a passo l'esecuzione per verificarne il corretto funzionamento.

Operazione	Meno efficiente	Più efficiente (<u>PERCHÉ?</u>)
Incrementare (decrementare) un registro	ADD EAX,1 SUB EAX,1	INC EAX DEC EAX
Azzeramento di un registro	MOV EAX,0	XOR EAX,EAX (o SUB EAX,EAX)
Controllare se un registro è zero	CMP EAX,0 JE Etichetta	TEST EAX,EAX JZ Etichetta
Moltiplicare un numero (in AX) per 2	MOV BX,2 MUL BX	SHL AX,1
Dividere un numero (in AX) per 2	MOV BL,2 DIV BL	SHR AX,1

8 – L'istruzione LOOP

L'istruzione LOOP <etichetta> decrementa il registro ECX e salta a etichetta se il risultato è maggiore di zero:

LOOP Etichetta  DEC ECX
JNZ Etichetta

LOOP è perciò molto utilizzato in tutti i casi in cui un programma deve eseguire una certa operazione un dato numero di volte. Si consideri ad esempio il programma seguente:

Label:	MOV ECX,10 XOR EAX,EAX ADD EAX,ECX LOOP Label
--------	--------------------------------------------------------

Quanto valgono EAX e ECX al termine dell'esecuzione?