Guida all'utilizzo dell'assembly in Visual Studio .NET

Questo tutorial spiega come utilizzare il codice assembly in un progetto Visual Studio .NET. Illustra l'aggiunta di codice assembly a un progetto contenente una funzione main() in C e il debug di un progetto con assembly.

Creazione di un progetto C/Assembly

Il progetto che svilupperemo in questo tutorial sarà costituito da una funzione main() scritta in C. Questa funzione chiamerà una funzione assembly denominata clear() . Poiché Visual Studio non riconosce il codice assembly, sarà necessario indicare a Visual Studio quale programma chiamare per compilarlo. In questa sezione, illustreremo i passaggi fondamentali per creare un progetto, aggiungere codice assembly, specificare le istruzioni di compilazione personalizzata e compilare il progetto.

Passaggio 1: creare un progetto

Crea un progetto standard di Visual Studio .NET 2003 C++. Dovresti ripetere le stesse operazioni che hai fatto per creare le soluzioni: crea una nuova soluzione vuota, un progetto Visual C++, e scorri verso il basso per selezionare un'applicazione console Win32. Nelle impostazioni dell'applicazione, assicurati di impostarla come progetto vuoto.

Aggiungi un file sorgente C con una funzione main() . Per questo tutorial, creeremo una funzione main() che chiamerà una funzione assembler denominata clear() di tipo void e non richiede parametri. Definiremo clear() utilizzando codice assembly in un file separato chiamato clear.asm . Poiché si trova in un file separato, clear() dovrà essere dichiarata all'inizio del file. La nostra funzione main() sarà simile a questa:

```
esterno vuoto chiaro();

 int principale() {
    chiaro();
    restituisci 1;
 }
```

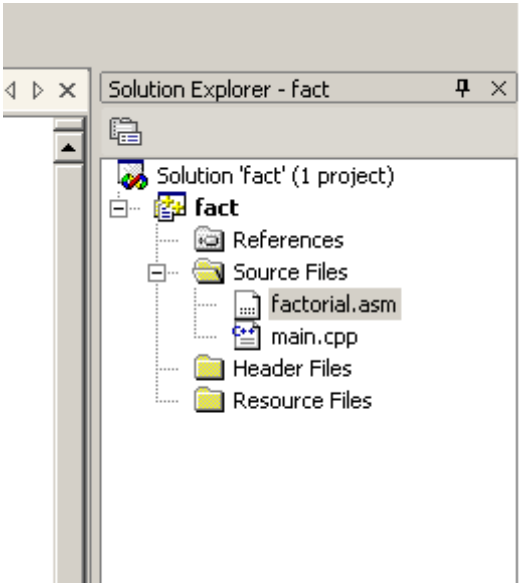
Nota sul compilatore C++: se si utilizza un compilatore C++, i nomi delle funzioni verranno alterati. Per evitare ciò, utilizzare

```
 esterno "C" {
     vuoto chiaro();
  }
```

(Grazie a Gabriel Zabusek per questa nota.)

Passaggio 2: aggiungere il codice di assemblaggio

È possibile aggiungere i file creati (sia .c che .asm) ai file sorgente facendo clic con il pulsante destro del mouse sulla cartella dei file sorgente in Esplora soluzioni e selezionando Aggiungi elemento esistente. Dopo aver aggiunto entrambi i file ai file sorgente, Esplora soluzioni dovrebbe apparire come segue:



Aggiungi il file contenente il codice sorgente dell'assembly al progetto. Se non è ancora stato creato, puoi farlo selezionando FileView nella finestra Progetto, facendo clic con il pulsante destro del mouse sul nome del progetto e selezionando "Aggiungi file al progetto...". Quando viene visualizzata la finestra di dialogo, digita il nome con cui desideri salvare il file del codice assembly (nel nostro caso, clear.asm). VS ti avviserà che il file non esiste e ti chiederà se desideri comunque creare un riferimento ad esso nel progetto. Seleziona Sì. Espandi l'elenco ad albero nella finestra del progetto fino a visualizzare il nome del file assembly (clear.asm). Fai doppio clic sul nome del file. VS ti chiederà se desideri creare un nuovo file con quel nome. Seleziona Sì. Un nuovo file verrà creato e aperto nell'editor.

Inserisci il codice assembly. In questo tutorial, cancelleremo i registri EAX ed EBX. Per farlo, useremo questo codice:

```
.586; Processore di destinazione. Istruzioni per l'uso per macchine di classe Pentium
.MODEL FLAT, C; Utilizza il modello di memoria flat. Utilizza le convenzioni di chiamata C
.STACK ;Definisce un segmento di stack di 1 KB (non richiesto per questo esempio)
.DATA; Crea un segmento di dati vicino. Le variabili locali vengono dichiarate dopo
                ;questa direttiva (non richiesta per questo esempio)
.CODE ;Indica l'inizio di un segmento di codice.

PROC chiaro
  xor eax, eax
```

xor ebx, ebx
Giusto
cancella ENDP
FINE

Passaggio 3: imposta comandi di compilazione personalizzati

Ora forniamo i comandi che VS utilizzerà per compilare il codice assembly. VS non compila il codice sorgente dell'assembly. Deve chiamare un programma esterno denominato ml.exe per eseguire la compilazione. La riga di comando deve essere aggiunta alle opzioni di compilazione personalizzata delle impostazioni del progetto. Per farlo, fare clic con il pulsante destro del mouse sul nome del file assembly (nell'immagine seguente, factorial.asm) nella finestra Progetto. Selezionare Proprietà... dal menu a comparsa.

Selezionare la cartella Custom Build Step e aggiornare i campi a) Riga di comando e b) Output:

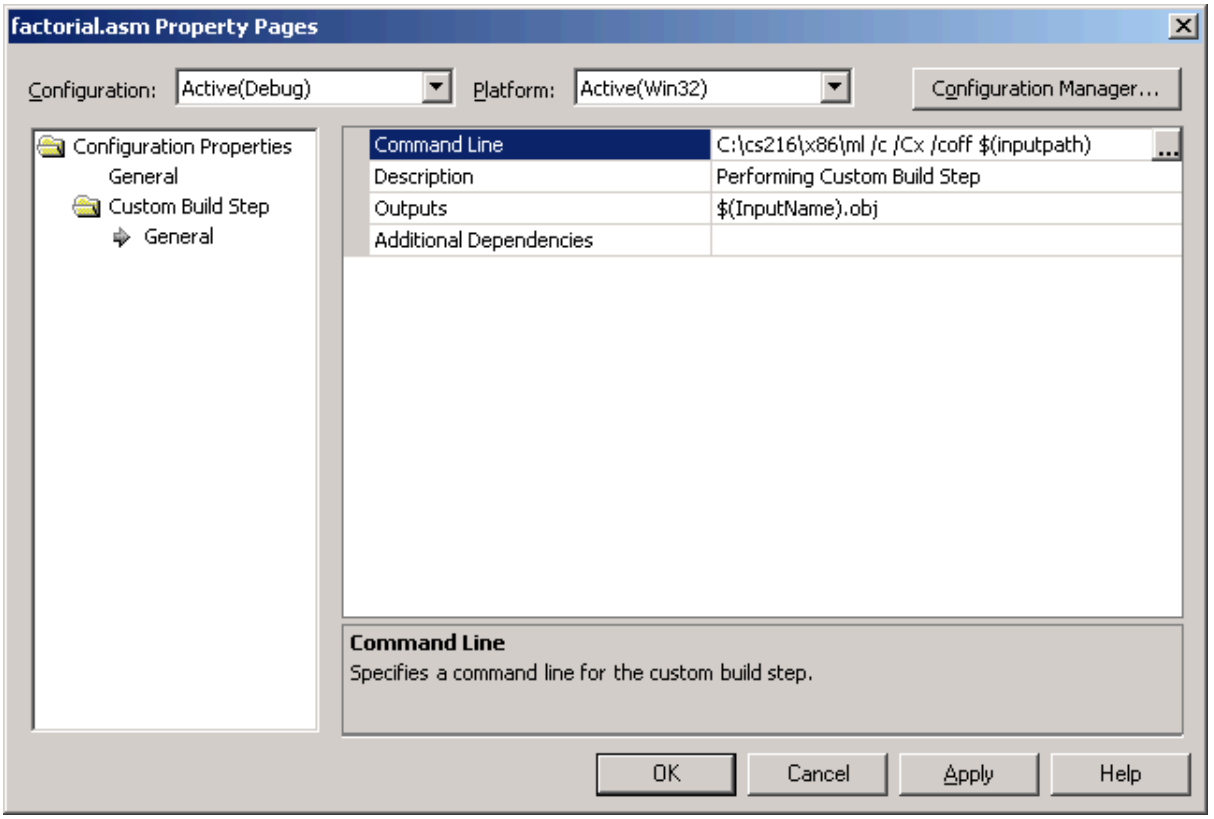
Campo Riga di comando . Questo è il comando effettivo che verrà eseguito per compilare il file factorial.asm . Richiede l'esecuzione di ml.exe (l'assembler) sul file di input. È necessario assicurarsi che la prima parte del comando *corrisponda* effettivamente alla directory in cui è stato scaricato il file ml.exe (non è necessario aggiungere l' estensione .exe). Il seguente comando funzionerà se il file ml.exe si trova nella directory C:\cs216\x86 :

C:\cs216\x86\ml /c /Cx /coff $(percorso input)

(Nota che se il nome della directory include spazi, è necessario racchiuderlo tra virgolette.)

Il campo Output . Questo indica il nome del file creato dal passaggio di build. In questo caso sarà il nome del file di input con estensione .obj .

Dopo aver impostato questi due campi, fai clic su OK e chiudi la finestra di dialogo: dovresti essere pronto per iniziare la compilazione.



Fase 4 - Compilazione e collegamento

Il progetto può ora essere compilato, collegato ed eseguito come qualsiasi altro progetto VS. Premere F7 per compilare il progetto e F5 per eseguire il programma.

Debug di un progetto C/Assembly

Il debug di un programma in assembly è un po' più complesso rispetto al debug di un programma in C puro. In particolare, il comando Step-Into (F11) non consente di accedere a un modulo assembly. Per aggirare questa limitazione, dobbiamo utilizzare la finestra di disassemblaggio. Per eseguire il debug del codice assembly, dovremo interrompere il programma con un breakpoint, aprire la finestra di disassemblaggio per visualizzare il codice assembly, aprire le finestre dei registri o della memoria e utilizzare F11 per scorrere il codice nella finestra di disassemblaggio.

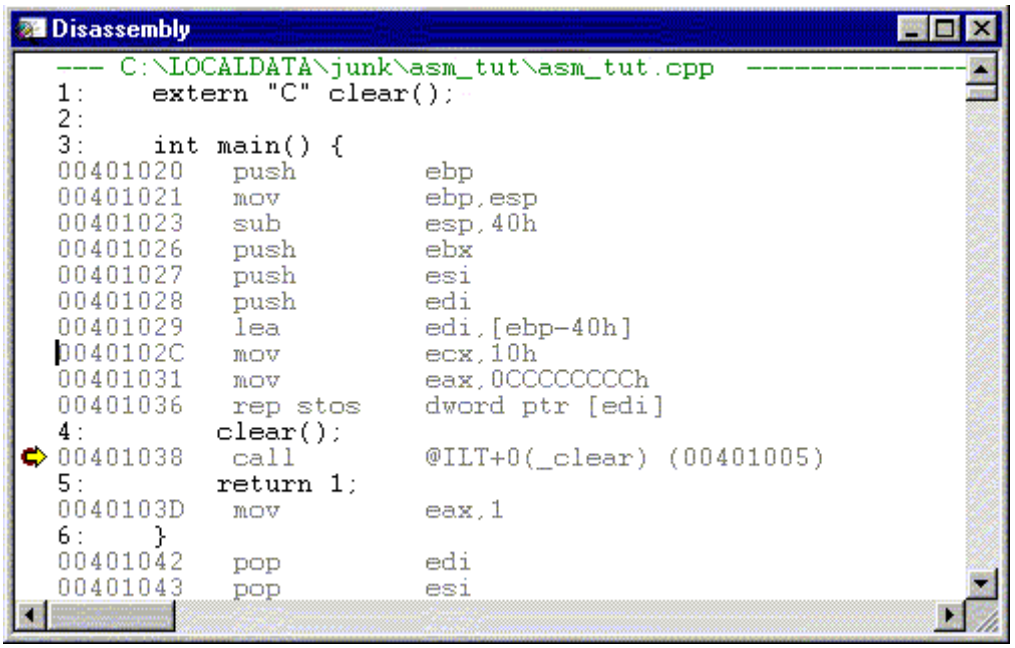
Passaggio 1: interrompere il programma

Inserisci un breakpoint sulla riga di codice che richiama la routine assembly. Esegui il programma principale premendo F5 (Go). Verrà eseguito normalmente e si interromperà quando raggiunge il breakpoint.

Passaggio 2: aprire la finestra di smontaggio

Apri la finestra Disassembly selezionando Debug->Finestre->Disassembly dal menu Debug (queste opzioni di menu probabilmente non appariranno finché

non avrai eseguito il programma). Puoi anche visualizzare il disassembly facendo clic con il pulsante destro del mouse sul codice C e selezionando " Vai a Disassembly" . Apparirà una nuova finestra simile a questa:

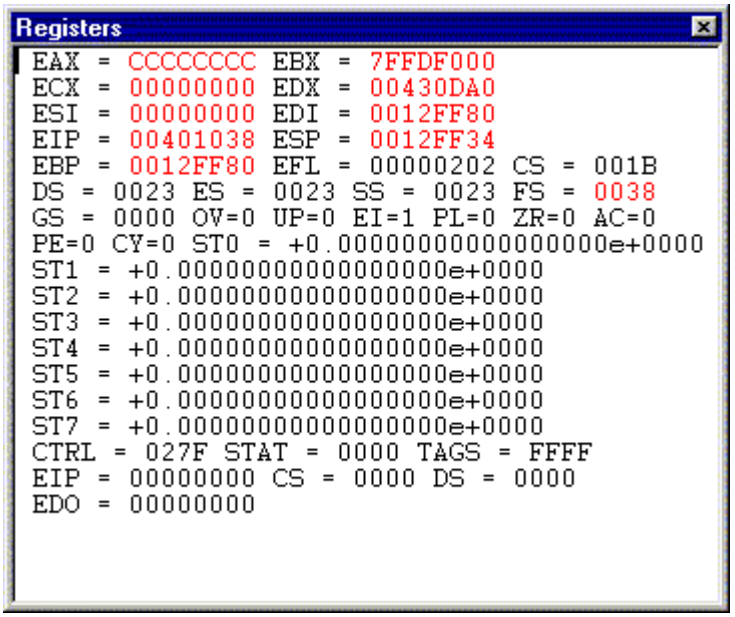


La finestra Disassembly mostra le istruzioni di assemblaggio del file oggetto. Il codice C effettivamente scritto è elencato in nero. Il codice disassemblato è elencato in grigio dopo la corrispondente istruzione C. Queste sono le istruzioni di assemblaggio effettive che verranno eseguite durante l'esecuzione del programma. La freccia gialla, che indica la prossima istruzione da eseguire, è presente in questa finestra. La freccia, tuttavia, non punta all'istruzione C clear(); , ma piuttosto all'istruzione di assemblaggio 00401038 call @ILT+0(_clear) (00401005) .

La finestra Disassembly consente di procedere nel codice un'istruzione di assembly alla volta.

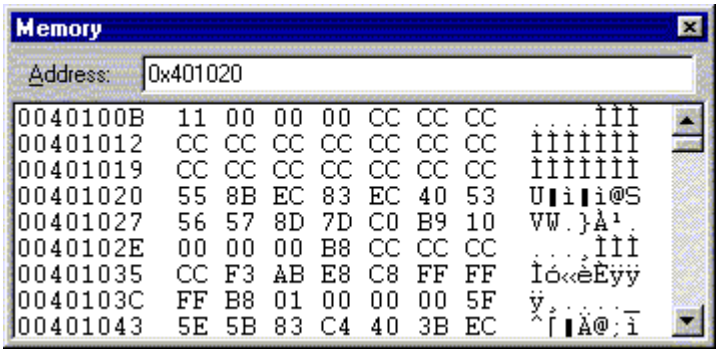
Fase 3 - Visualizzare i registri e la memoria

A questo punto, potrebbe essere opportuno aprire le finestre Registro e Memoria. Queste finestre forniscono un'istantanea del contenuto dei registri della CPU e della memoria di sistema tra un'esecuzione e l'altra delle istruzioni del programma. Aprite la finestra Registro selezionando Debug->Finestre->Registri dal menu Debug (queste opzioni di menu probabilmente non appariranno finché non avrete eseguito il programma). Dovrebbe apparire così:



È possibile visualizzare i valori dei registri EAX, EBX, ECX, EDX, ESI, EDI, ESP ed EBP, nonché alcuni altri registri e flag di stato presenti nel processore (questi registri provengono da un Pentium Pro; i registri di altri processori potrebbero essere diversi, sebbene gli 8 elencati siano presenti). È possibile fare clic con il pulsante destro del mouse nella finestra dei registri per selezionare quali registri (FP, MMX, ecc.) visualizzare.

Per esaminare la memoria, seleziona Debug->Finestre->Registri dal menu Debug (queste opzioni di menu probabilmente non appariranno finché non avrai eseguito il programma). Appare così:



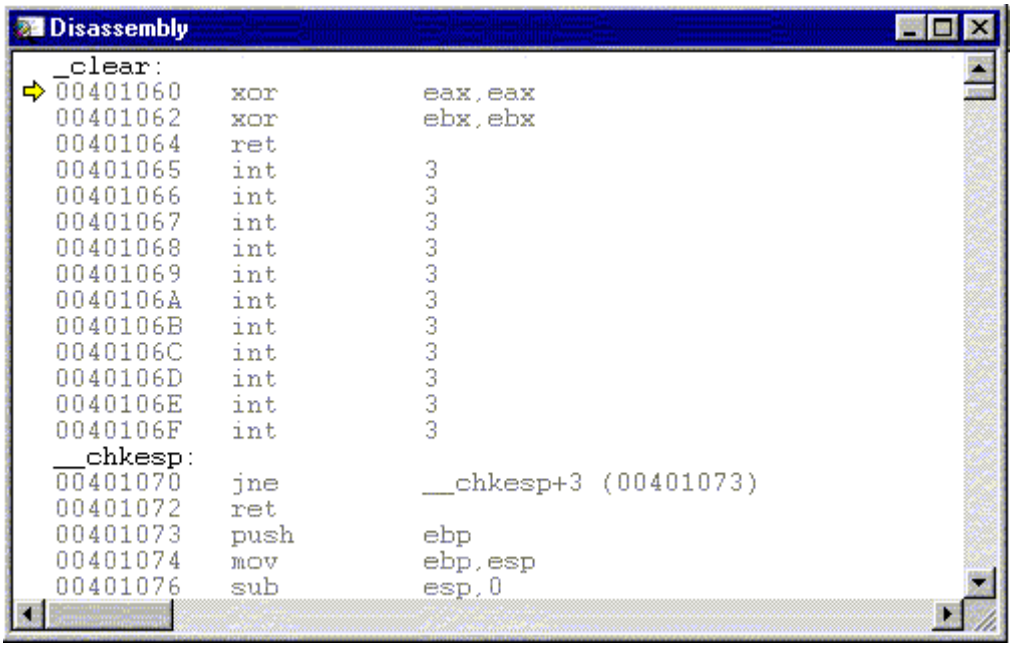
Fornisce un dump della memoria con l'indirizzo di memoria a sinistra, i valori esadecimali del contenuto della memoria a destra e la rappresentazione ASCII dei valori esadecimali a destra. Una particolare posizione di memoria può essere visualizzata digitando l'indirizzo nella casella di testo nella parte superiore della finestra. La finestra mostrata sopra mostra l'inizio del nostro programma. La prima istruzione si trova nella posizione di memoria 0x00401020 ed è 0x55. Questa è la codifica esadecimale di 'push ebp'. I sei numeri successivi sulla riga indicano le posizioni di memoria successive. In questo esempio, su ogni riga di questa finestra vengono visualizzate sette posizioni di memoria. Si noti che la finestra può essere ridimensionata per modificare il

numero di byte visualizzati per riga. L'ultima colonna contiene i caratteri ASCII per le posizioni di memoria. Di solito, questo dato non è corretto, a meno che non si stia visualizzando una regione di memoria contenente testo. È possibile fare clic con il pulsante destro del mouse nella finestra della memoria e regolare il modo in cui il contenuto della memoria viene raggruppato (per 1, 2, 4 o 8 byte) e visualizzato (come numeri interi con segno o senza segno, numeri in virgola mobile, esadecimali, ecc.).

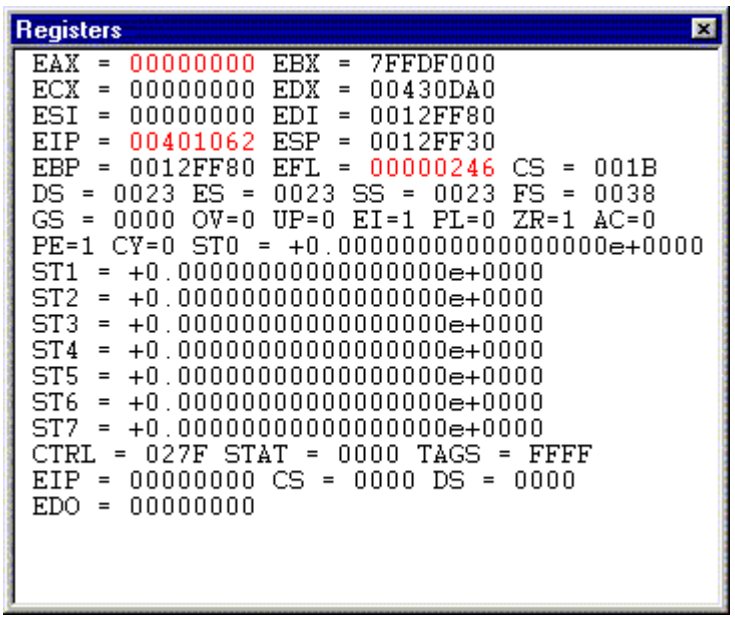
Fase 4 - Procedere con il codice

È possibile scorrere il codice passo dopo passo utilizzando F10 (Passa oltre) e F11 (Passa in). La finestra Disassembly traccia l'esecuzione delle istruzioni assembly con una freccia gialla che punta all'istruzione successiva da eseguire.

Premendo F11 una volta si esegue l'istruzione call. Ora ci troviamo di fronte a un'istruzione jmp che ci porterà all'inizio della funzione clear() . Premendo nuovamente F11 arriviamo alla prima istruzione della funzione clear() . La finestra di disassemblaggio ora appare così:



Si noti che la freccia gialla punta alla prima delle nostre due chiamate xor. La finestra Registri a questo punto non è cambiata. Premendo nuovamente il tasto F11 si esegue la prima istruzione xor , cancellando il registro EAX. La finestra Registri ora è:



Il valore EAX è ora 0x0 . Premendo nuovamente F11 si cancella il registro EBX. Premendo nuovamente F11 si esce dalla funzione clear() e si torna alla nostra istruzione C return 1; . Poiché abbiamo completato il debug della parte cruciale del nostro codice, possiamo premere F5 per andare avanti e terminare rapidamente il programma.