|  |  |  |
| --- | --- | --- |
| Altomare Andrea | Lab. Sistemi esperienza n. 13 | Classe 3^A 6/05/2015 |

**Assembly 8086: Procedure**

**PROCEDURE**

In Assembly un sottoprogramma è chiamato “**procedura**”. Come nei linguaggi ad alto livello (dove i sottoprogrammi vengono chiamati anche *metodi* o *funzioni*) in Assembly si utilizzano le procedure per una migliore organizzazione del codice e soprattutto quando bisogna far ricorso a delle determinate istruzioni più volte (per esempio, per l’acquisizione di un numero o l’output di qualcosa a schermo).

Quando si vuole definire una procedura ed il relativo codice, la sintassi è la seguente:

<NOME\_PROCEDURA> PROC

---------------

---------------

---------------

---------------

RET

<NOME\_PROCEDURA> ENDP

Innanzitutto va chiarito che le procedure si possono definire SOLO dopo l’istruzione **RET** del programma principale, se lo si fa prima si avranno degli errori. Inizialmente va messo il nome della procedura che si vuol creare, di seguito la parola chiave “**PROC**” che indica che ciò che si sta creando è proprio una procedura. Dopodiché si scrivono le istruzioni che costituiranno la procedura, le quali termineranno con l’istruzione **RET** il quale ritornerà il controllo al programma chiamante. Infine si riscrive il nome della procedura e di seguito la parola chiave “**ENDP**” ad indicare la fine di una procedura (quest’ultimo passaggio con ENDP non è necessario né obbligatorio per tutti gli assemblatori).

Per chiamare una procedura in un programma si usa l’istruzione **CALL <NOME\_PROCEDURA>**.

**Procedure con passaggio di parametri per valore**

Solitamente, quando si lavora con un linguaggio ad alto livello, in una funzione c’è il bisogno dell’uso di parametri (propriamente detti “**argomenti**”, quando si passano ad una funzione vengono automaticamente salvati nello **stack**); in Assembly questi si gestiscono utilizzando appropriatamente i **registri**. Per esempio se si ha una variabile “n” che si vuole passare come parametro per valore, prima di chiamare la procedura si sposta il valore della variabile su un registro apposito (per esempio ax), nella procedura si svilupperanno poi tutte le operazioni col dato della variabile “n”, se poi la procedura è stata designata per essere una **funzione** (ovvero che ritorna un valore) allora semplicemente si passa il valore situato nel registro interessato nella variabile che deve assumere quello specifico dato.

**Procedure con passaggio di parametri per riferimento**

Quando si vuole passare una variabile per riferimento in un linguaggio ad alto livello (per esempio il C) si utilizza un’apposita variabile detta “puntatore” che contiene l’indirizzo della variabile a cui si riferisce, operando con il puntatore ed eseguendo operazioni su di esso si va dunque ad agire in memoria sulla variabile passata. Nel linguaggio Assembly, similmente, si sposta nel registro usato nella procedura l’offset della variabile da passare per riferimento. Se per esempio di vuole operare con un vettore “V”, ne si sposta l’offset su un registro (per esempio di) con l’istruzione **MOV DI,OFFSET V**. Nella procedura dunque qualsiasi operazione effettuata sul contenuto dell’indirizzo di memoria puntato dal registro DI si ripercuoterà automaticamente sul vettore V. Per operare sul contenuto di un indirizzo puntato dal registro e non sull’indirizzo stesso (dato che normali operazioni come *ADD DI,8* equivalgono alla modifica dell’indirizzo di memoria puntato da DI e non alla modifica del contenuto puntato in memoria) quando si esegue un qualsiasi comando, come un’addizione, il registro che punta alla variabile si mette tra parentesi quadre “**[]**” indicando quindi di eseguire l’istruzione non direttamente sul contenuto di DI (ovvero l’indirizzo della variabile) ma sul contenuto **puntato**, sintassi: **ADD [DI],5**.

**Problemi con l’utilizzo dello stack**

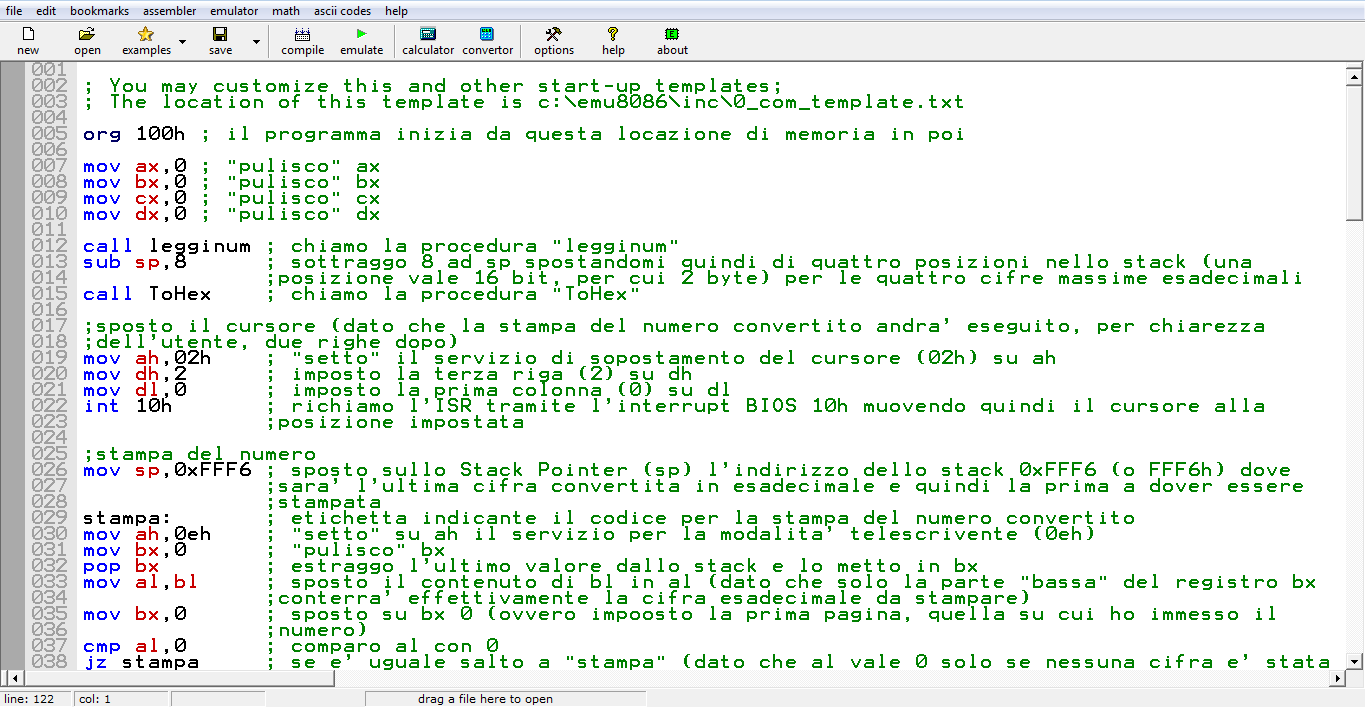
Quando si opera con le procedure va ricordato che alla chiamata di un sottoprogramma alcune informazioni vengono salvate nello **stack**, ragion per cui se nella procedura si vuole usare lo stack bisogna fare attenzione poiché se al ritorno del controllo al programma chiamante non si riporta lo Stack Pointer (**SP**) al giusto indirizzo, si rischia di ripristinare informazioni errate e quindi di avere malfunzionamenti.

**PROBLEMA**

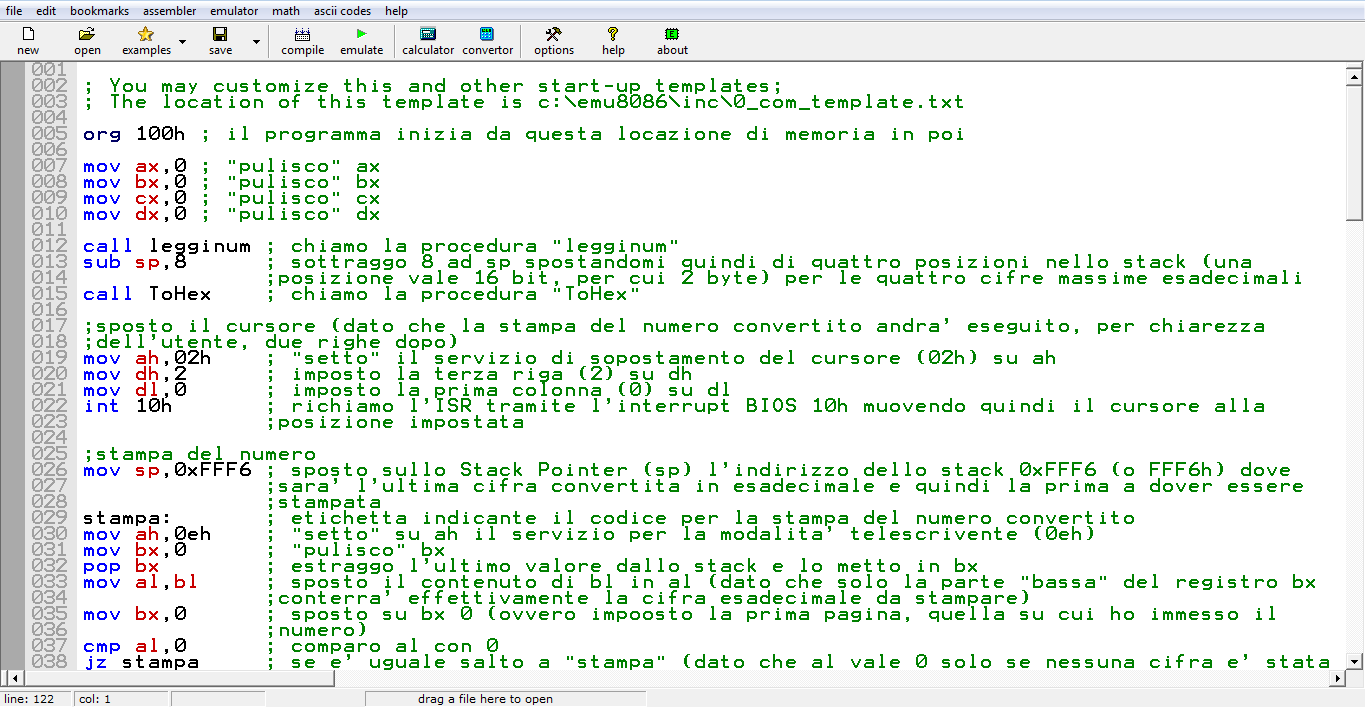
Realizzare un programma con una **procedura** “*legginum*” per l’input di un numero che dovrà essere collocato sul registro **ax**. Realizzare quindi una procedura che converta in esadecimale il numero immesso dall’utente. Il numero esadecimale deve essere rappresentato da una stringa di caratteri ed essere salvato in un vettore, oppure essere collocato lettera per lettera nello stack, va poi **stampato** a schermo.

**REALIZZAZIONE DEL PROGRAMMA**

Dopo l’**org 100h** ho “*pulito*” i registri general purpose.

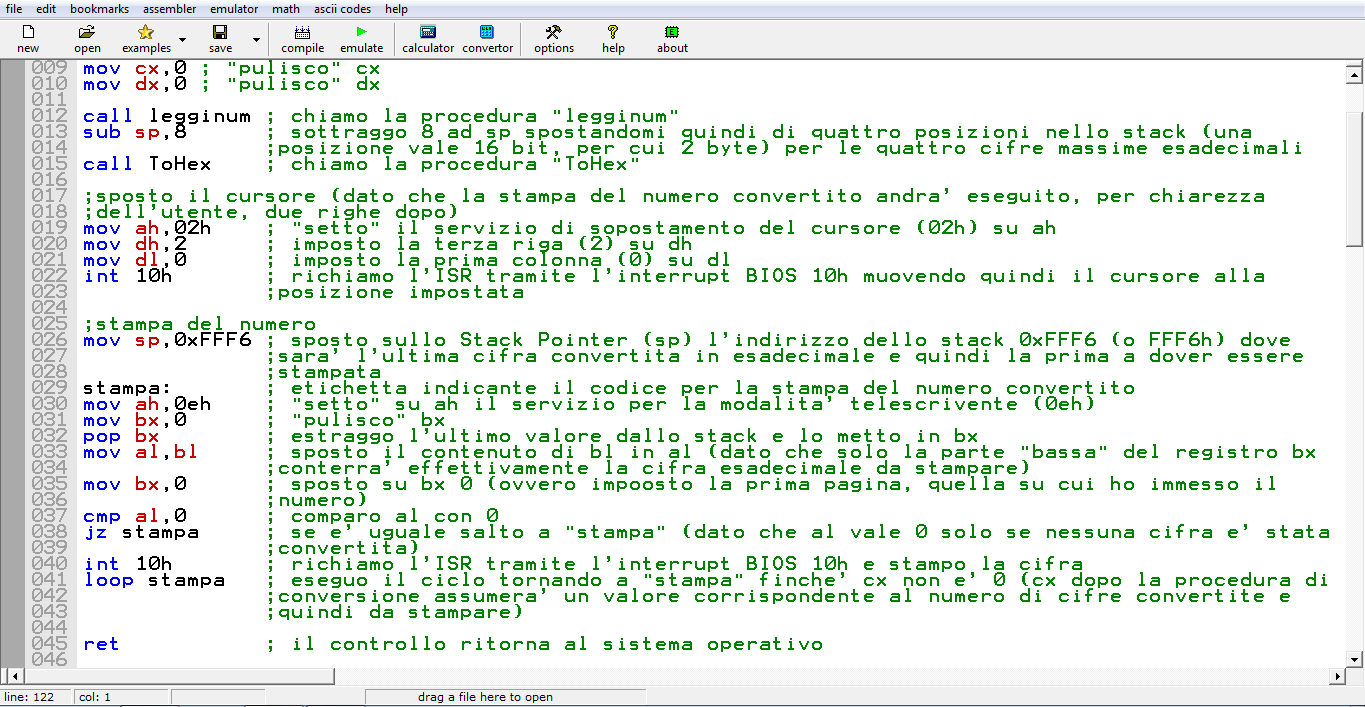


Chiamo quindi la procedura “*legginum*”, dopodiché sottraggo ad **SP 8** per spostarmi nello **stack** di **4** posizioni (una posizione dello stack equivale a **16 bit** e quindi **2 Byte**) corrispondenti a tutte le quattro cifre possibili del numero esadecimale in cui convertire quello in base dieci. Chiamo la procedura “*ToHex*” per convertire il numero. Terminata l’esecuzione dell’ultima procedura procedo alla stampa del numero convertito, sposto quindi il cursore nella terza riga per permettere all’utente di vedere comunque il numero immesso: setto il servizio **02h** in **ah**, imposto su **dh** la riga numero **2** e su **dl** la colonna numero **0**, richiamo poi l’**ISR** tramite l’**interrupt BIOS 10h**.

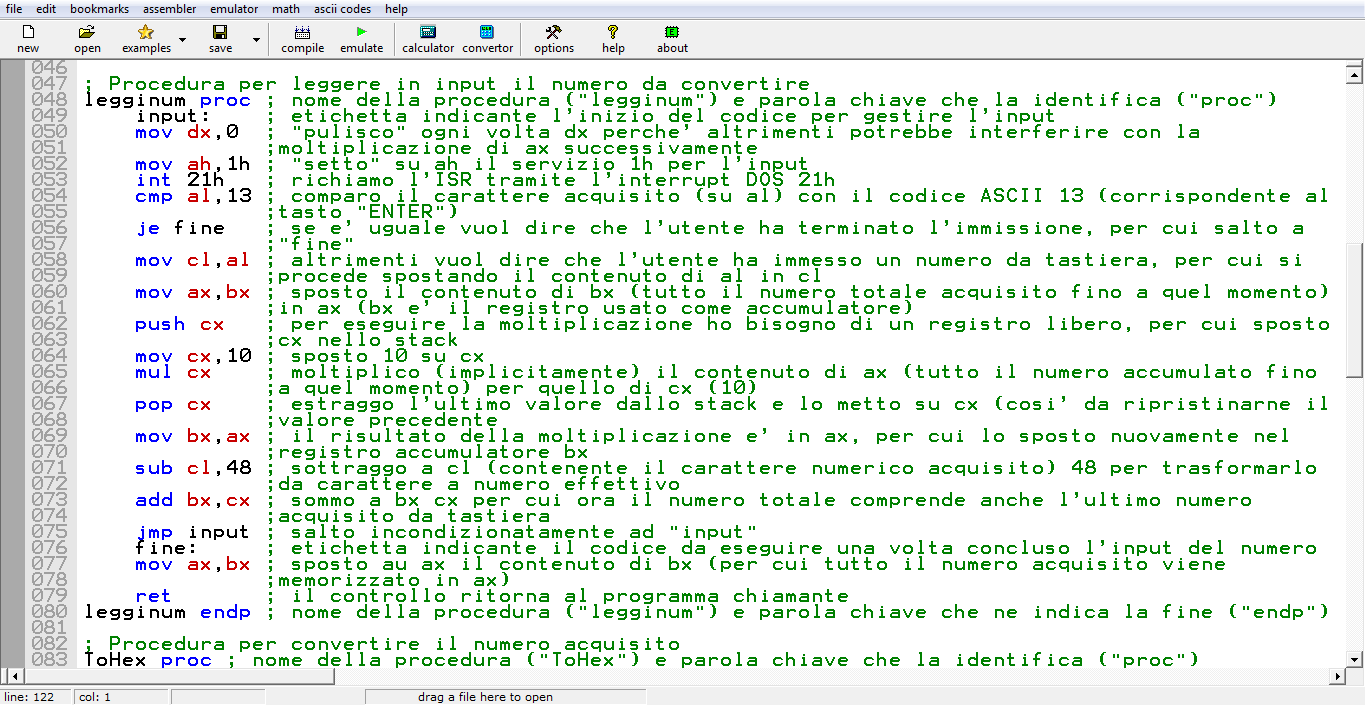


Poi sposto su **SP** (Stack Pointer) il valore esadecimale **0xFFF6** spostandomi quindi nello **stack** alla posizione in cui è stata memorizzata l’ultima cifra esadecimale convertita, e quindi quella da cui bisogna iniziare a stampare il numero esadecimale. Inizio il ciclo di stampa con l’etichetta “*stampa:*”, setto la *modalità teletype* **0eh** sul registro **ah**, “pulisco” **bx** spostandovi **0** ed estraggo l’ultimo valore dallo stack e lo metto su **bx** (quindi su bx vi è ora la cifra da stampare), sposto il contenuto di **bl** in **al** (dato che in realtà il valore ASCII della cifra occupa solo **8 bit**, ed è quindi tutto in **bl**), sposto su **bx 0** impostando la prima pagina (quella in cui l’utente ha digitato il numero da convertire) e comparo **al** con **0**, se è uguale (**zero**) allora salto direttamente a “*stampa*” poiché l’unico caso in cui una delle posizioni dello stack è vuota (**0**) (in questo caso) è quando non vi è alcuna cifra, e quindi nulla da stampare, altrimenti stampo la cifra memorizzata richiamando l’**ISR** tramite l’**interrupt BIOS 10h**, salto poi a “*stampa*” in **loop** finché **cx** non è **0** (cx in precedenza avrà assunto il valore corrispondente al numero di cifre presenti nel numero esadecimale convertito).

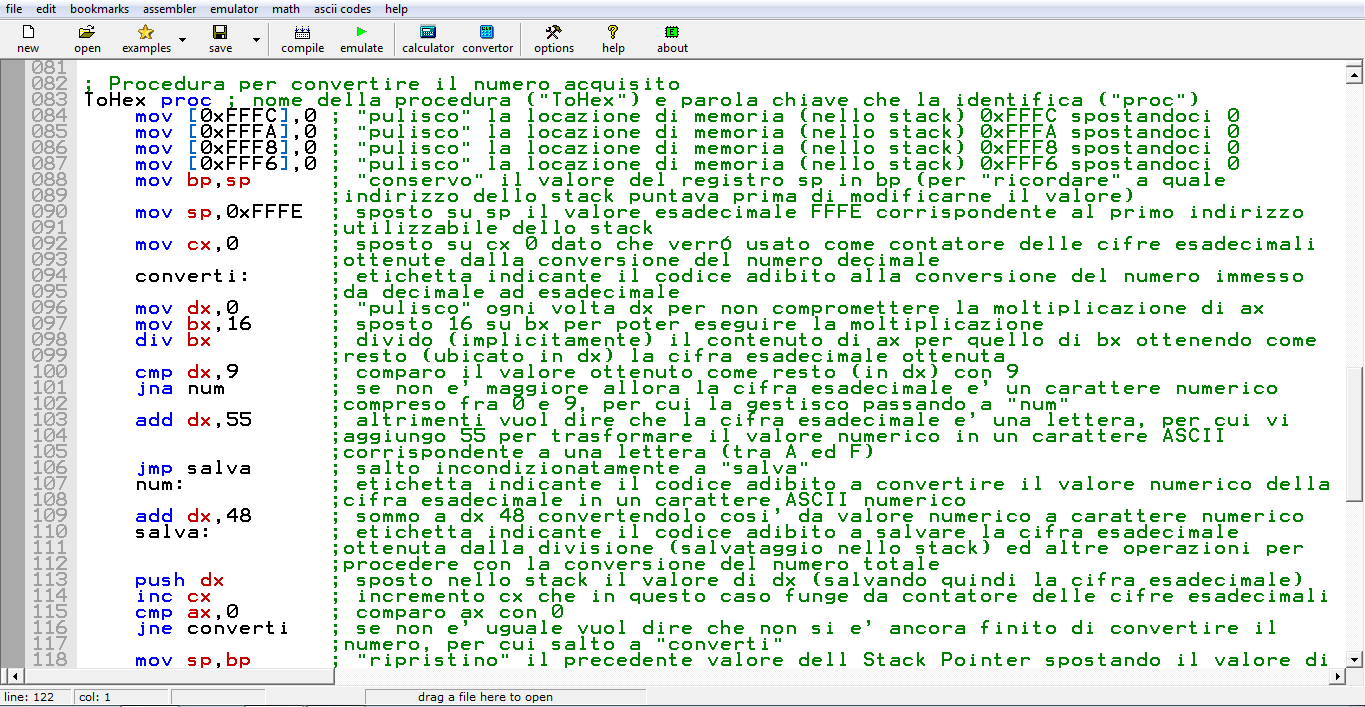
Terminata la stampa del numero, il programma ha concluso il suo lavoro. Col comando **ret** il controllo ritorna al sistema operativo.

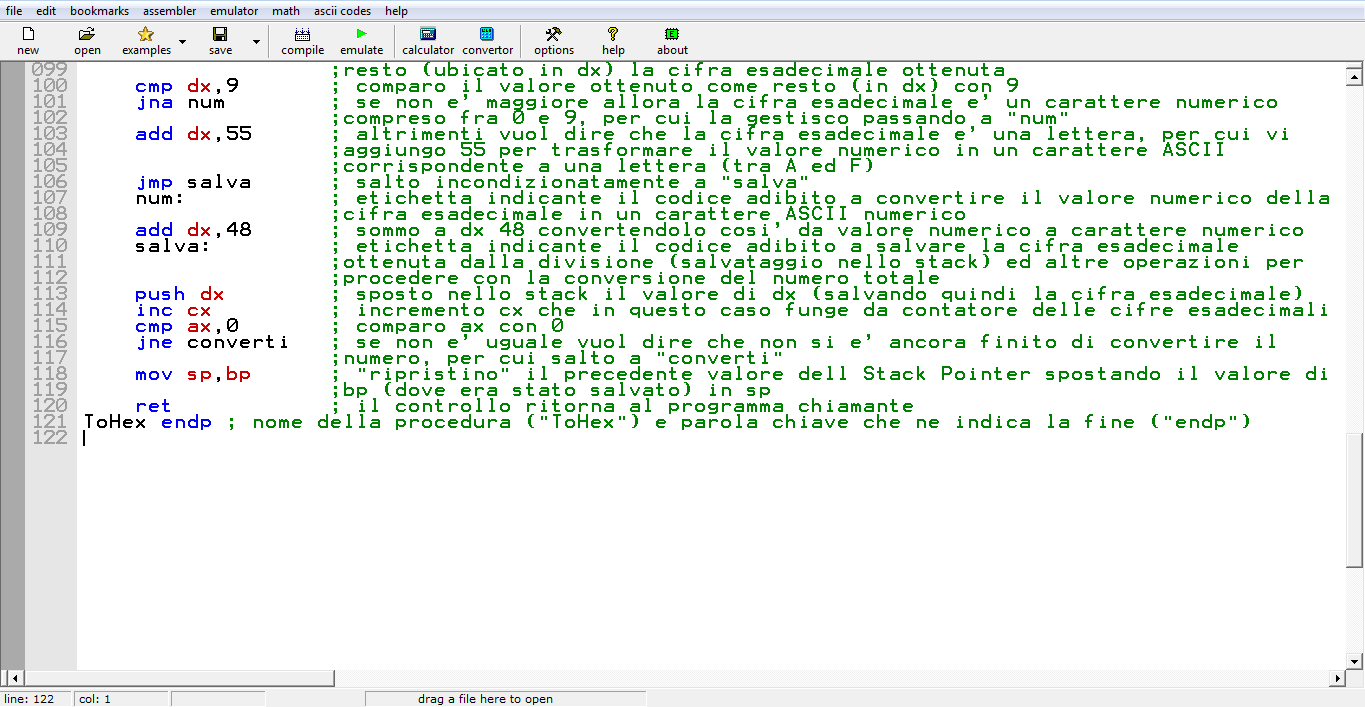


La procedura “*legginum*” ha all’interno il codice per gestire l’immissione del numero a più cifre da convertire in base esadecimale. Innanzitutto metto l’etichetta “*input:*” per indicare l’inizio del ciclo di acquisizione, poi “pulisco” ogni volta **dx** perché altrimenti potrebbe interferire con la moltiplicazione di **ax** successivamente; setto il servizio per l’input da tastiera **1h** in **ah** e richiamo l’ISR tramite l’interrupt DOS **21h**, il carattere acquisito andrà su al; comparo al con 13 (il tasto “ENTER”), se è uguale vado a “*fine*” altrimenti sposto **al** su **cl**, sposto **bx** su **ax** (in bx c’è il numero totale immesso fin’ora, è infatti usato come registro accumulatore), “conservo” **cx** nello **stack** immettendolo dato che ho bisogno di quel registro per la moltiplicazione, sposto su cx **10**, moltiplico **ax** (implicitamente) per **cx**, estraggo l’ultimo valore dallo stack e lo metto su **cx** “ripristinandone” il valore precedente, sposto il nuovo valore di **ax** in **bx**, sottraggo a **cl** **48** (per convertirlo da carattere **ASCII** rappresentante una cifra a cifra numerica effettiva), e aggiungo a **bx** il valore di **cx** ottenendo così il “nuovo” numero totale, salto poi incondizionatamente ad “*input*”. Dall’etichetta “*fine:*” si ha il codice eseguito quando l’utente preme “ENTER”: sposto **bx** su **ax** dato che il numero totale immesso deve essere contenuto in **ax** alla fine di *legginum*, in ultimo con **ret** il controllo ritorna al programma chiamante.



Con la procedura “*ToHex*” converto il numero immesso dall’utente da base dieci a base 16 (esadecimale). Innanzitutto “pulisco” le locazioni di memoria dello **stack** **0xFFFC 0xFFFA 0xFFF8 0xFFF6** per contenere le cifre del numero convertito (il massimo è **4** cifre dato che si parla di **16 bit**); poi conservo il valore di sp in bp. Sposto su sp 0xFFFE per posizionarmi nella prima posizione dello stack, sposto su cx a 0 dato che deve essere il contatore delle cifre, inizio la conversione: sposto su bx 16, pulisco dx in modo che non interferisca, divido ax per bx (16) e comparo dx con 9, se non è maggiore vuol dire che la cifra è numerica quindi salto a num e gli aggiungo 48, altrimenti gli aggiungo 55 per trasformarlo in lettera (dato che è una cifra letterale), poi la salvo nello stack, incremento cx e comparo ax con 0: se non è uguale salto a converti altrimenti ho finito e ripristino il valore si sp spostandovi il valore di bp, con ret il controllo ritorna al programma chiamante.





Ho poi verificato il corretto funzionamento del programma convertendo un numero da base dieci in esadecimale con successo.

