|  |  |  |
| --- | --- | --- |
| Altomare Andrea | Lab. Sistemi esperienza n. 11 | Classe 3^A 11/03/2015 |

**Assembly 8086: Interrupt e stampa**

**INTERRUPT**

Con il termine “***Interrupt***” ci si riferisce a dei segnali di interruzione, che possono avvenire tra la fase di **execute** e di **fetch** nell’esecuzione di un programma, atti a richiamare delle **routine** per eseguire, solitamente, input o output. Essendo delle chiamate a dei sottoprogrammi (le routine, appunto), gli interrupt si devono distinguere fra loro attraverso dei codici (ad ognuno corrisponderà quindi un interrupt diverso).

Gli interrupt si differenziano ulteriormente in:

* **Interrupt *Hardware***: dati da dispositivi come schermo o tastiera;
* **Interrupt *Software***: dati dal programmatore per eseguire una certa operazione (il nome “Interrupt Software” non è comunque propriamente corretto).

In sintesi, dunque, con gli interrupt si richiamano routine del BIOS o del DOS (MS-DOS, il sistema operativo su cui si basano gli odierni sistemi Microsoft Windows) per gestire input e output. Nel BIOS sono contenute tutte le routine per la gestione dell’hardware costituente il computer. Le routine richiamate dagli interrupt prendono il nome di **ISR** (acronimo di “*Interrupt Service Routine*”) che sono sostanzialmente i servizi relativi al servizio e all’interrupt indicati nel codice. Nello specifico, gli interrupt **10h** richiamano routine del BIOS, mentre gli interrupt **21h** (tipicamente) richiamano routine del DOS.

La sintassi per richiamare un interrupt è **INT <codice>** (il codice corrisponde al tipo di interrupt richiamato, per esempio **10h**).

Dovendo interagire con il BIOS, questo, essendo contenuto nella ROM (o EP-ROM) che è più lenta rispetto alla RAM, viene caricato in quest’ultima. Nei nuovi sistemi, però, non è possibile interagire direttamente col BIOS, l’esecuzione dei programmi infatti si distingue in:

* **Protected Mode**: modo odierno: il programma, durante l’esecuzione, non può uscire dal suo spazio di lavoro. (Non gli è possibile quindi andare ad agire direttamente sul BIOS);
* **Real Mode**: modo utilizzato nei sistemi più vecchi: il programma può lavorare dove vuole, anche fuori dal suo spazio di lavoro. (Modo usato fino a Windows 98).

**STACK**

Quando si richiamano questi sottoprogrammi si interrompe il flusso di esecuzione del programma vero e proprio per passare ad altre istruzioni contenute nella RAM, alcune informazioni riguardanti il programma richiamante il sottoprogramma vengono quindi salvate in memoria, precisamente nello **Stack**. Queste informazioni sono i valori di:

* Registro dei FLAG;
* CS (*Code Segment*);
* IP (*Instruction Pointer*).

Una volta terminata l’esecuzione del sottoprogramma quest’ultimo ritorna il controllo al programma chiamante, “riprendendo” quindi le informazioni salvate in precedenza nello stack.

Lo stack è una memoria con politica **LIFO** (acronimo di “*Last In First Out*”): in questo modo (come si evince dal nome) l’ultimo dato ad entrare è anche l’ultimo ad uscire. In Assembly, l’istruzione per inserire un dato nello stack è **PUSH** (sintassi: **PUSH <dato>**, in questo caso il <dato> è quindi il sorgente) mentre quella per estrarre è **POP** (sintassi: **POP <dato>**, in questo caso invece il <dato> è la destinazione). Bisogna tenere conto del fatto che le istruzioni PUSH e POP lavorano con registri e variabili a 16 bit, non è per cui possibile eseguire un’istruzione di inserimento, per esempio, con il registro **al** (ad 8 bit), mentre con **ax** (essendo a 16 bit) è corretto. Il registro contenente gli indirizzi che puntano alle locazioni di memoria dello stack è **SP** (acronimo di “*Stack Pointer*”).

In una memoria monosegmentata (come quella riprodotta nell’emulazione dei programmi COM dall’emu8086) i vari “segmenti” occupano un certo spazio, se gli indirizzi crescono andando dal basso verso l’alto (prendendo come riferimento questi due versi), lo stack andrà dall’alto verso il basso: in senso opposto, quindi, alle altre parti di memoria rappresentanti gli altri “segmenti”.

**SERVIZI**

Poiché a differente servizio impostato uno stesso interrupt esegue operazioni diverse, per eseguire correttamente un interrupt a seconda di ciò che si vuole realizzare bisogna “settare” il servizio.

Sintassi: **MOV AH,<servizio>**

AH: è il registro nel quale bisogna impostare il servizio. <servizo>: è, appunto, il servizio.

**STAMPA (OUTPUT)**

Per eseguire la stampa sullo schermo bisogna richiamare una routine che gestisce, ovviamente, lo schermo collegato al computer, per cui prima di eseguire l’interrupt bisogna impostare il giusto servizio (ovviamente nel registro AH, come detto in precedenza).

**0Ah**

Con il servizio 0Ah si “setta” la “**Modalità Testo**”, ovvero sarà possibile dare in output un certo carattere (corrispondente al codice ASCII contenuto nel registro sorgente) alla posizione del cursore. Questo servizio non sposta però il cursore durante la stampa ed ha la caratteristica di stampare lo stesso carattere tante volte quanto vale il registro **CX**. Ad esempio, il codice

MOV CX,10

MOV AH,0Ah

MOV AL,’A’

INT 10h

Stamperà a schermo il carattere ‘A’ per dieci volte consecutive.

**0Eh**

Con questo servizio si imposta la “**Modalità *Teletype***” (Modalità Telescrivente), ovvero non solo stamperà un carattere a schermo ma sposterà automaticamente anche il cursore, permettendo quindi (per esempio) la stampa di una sequenza di caratteri senza doversi preoccupare di spostare manualmente il cursore ogni volta.

**02h**

Il servizio 02h permette di gestire la posizione del cursore sullo schermo. Si setta, come di consueto, il servizio su AH, poi si impostano su DH e DL rispettivamente la riga e la colonna ove dovrà essere spostato il cursore, infine si richiama l’interrupt 10h. DX è infatti il registro che gestisce la “tabella” per la stampa sullo schermo.

Un esempio di quanto scritto sopra può essere quello in cui bisogna andare a capo, per farlo infatti non si può utilizzare il carattere ASCII corrispondente al tasto “*ENTER*” (comunemente usato, appunto, per andare a capo) ma bisogna avvalersi del servizio qui descritto.

MOV AH,02h

INC DH

MOV DL,0

INT 10h

**PROBLEMA**

Stampare correttamente un numero a più cifre sullo schermo (in questo caso si è utilizzato un numero grande al massimo 255, Byte).

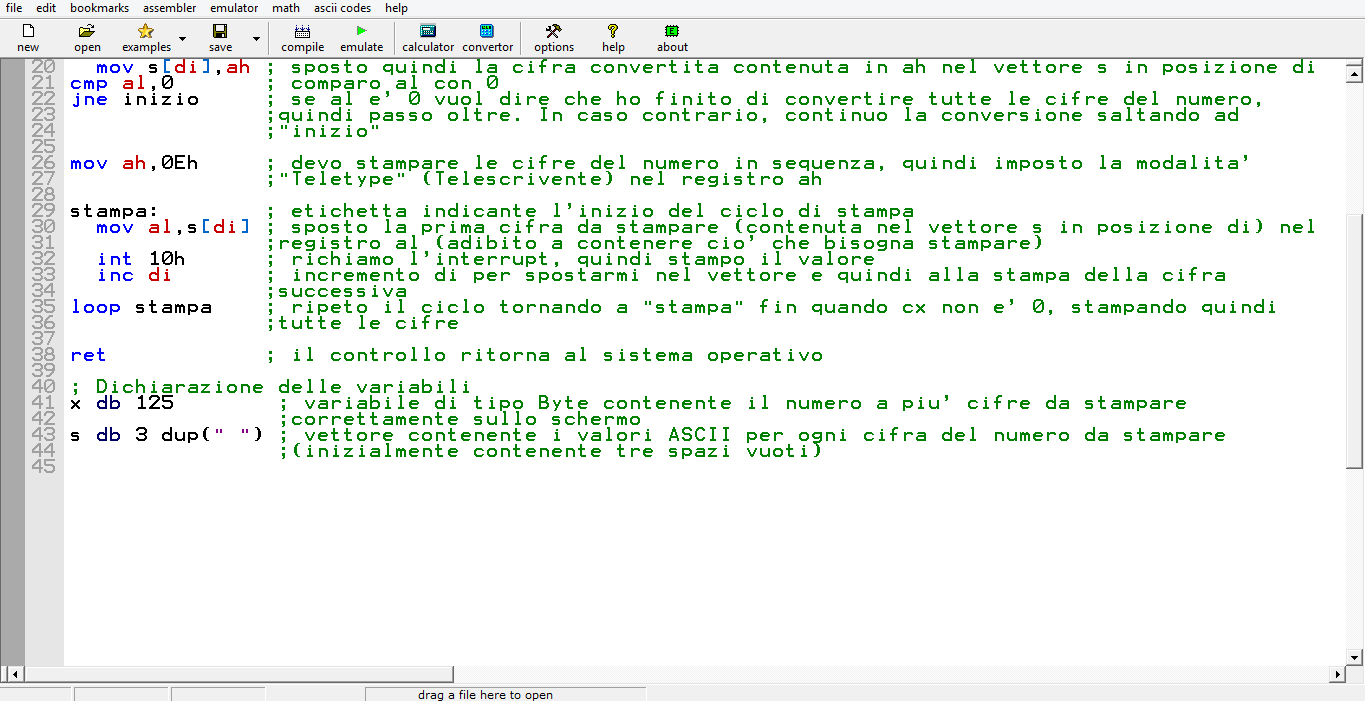
Il numero (si è utilizzato 125) è contenuto nella variabile Byte x (*x db 125*), per contenere i caratteri corrispondenti ad ogni cifra si è usato un vettore inizializzato con tre spazi ( *s db 3 dup(" ")* ).

**REALIZZAZIONE DEL PROGRAMMA**

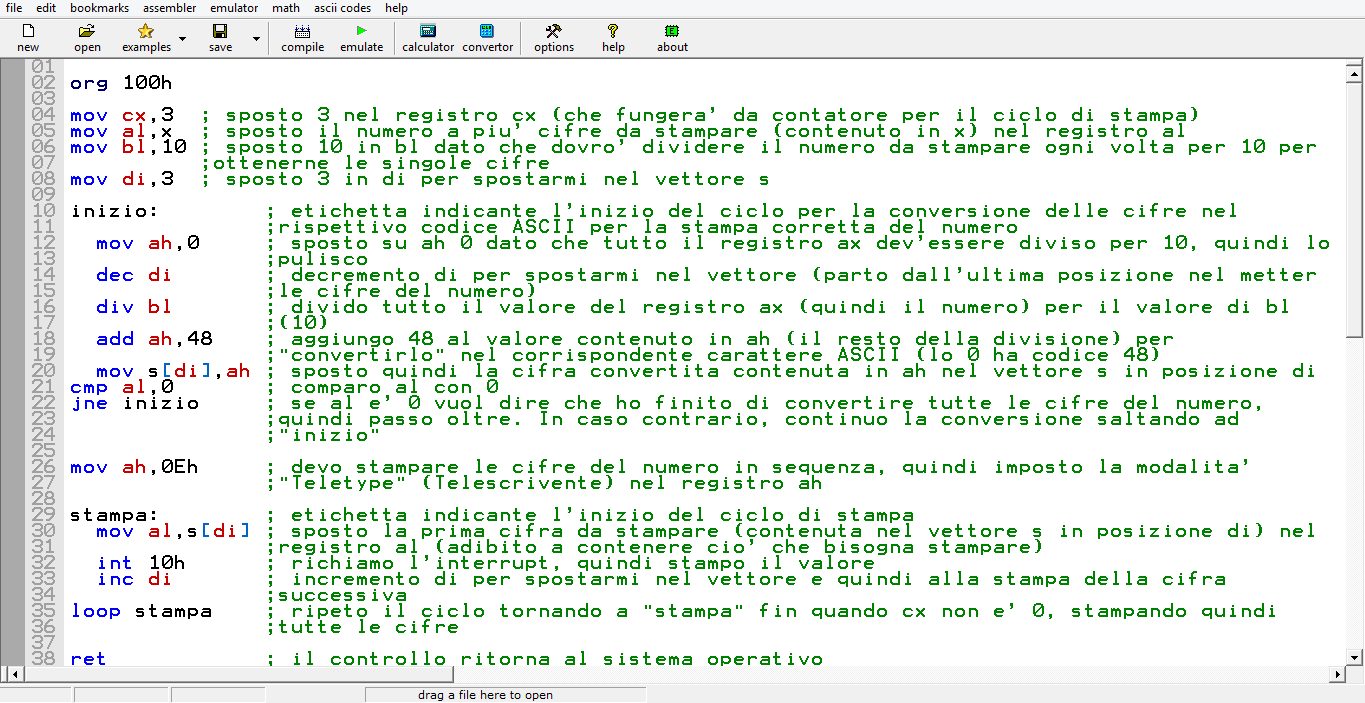
Il problema che si pone principalmente e implicitamente nella stampa di un numero è quello che, al contrario delle lettere che possono essere stampate senza alcun tipo di operazione specifica o dettagliata di conversione, i numeri vanno convertiti nel corrispondente carattere ASCII. Per compiere questa “conversione” da numero a carattere corrispondente si somma un certo valore per stampare il codice ASCII corretto, altrimenti verrà dato in output il carattere corrispondente al codice ASCII del numero selezionato. Il valore da sommare è **48** dato che è il codice corrispondente al carattere ‘**0**’, per cui sommando, per esempio, 48 a 5 (5+48) si ottiene il codice corrispondente al carattere ‘5’ che si vuole stampare.

Un numero a più cifre però non ha un carattere corrispondente dato che in questo modo ne servirebbero infiniti, per questo ci si può avvalere della stampa di una sola cifra per volta (da ‘0’ a ‘9’) ciclicamente. Bisogna quindi “scomporre” il numero da stampare nelle sue cifre costituenti, in questo caso il numero 125 dovrà essere scomposto nelle cifre e quindi nei corrispondenti caratteri ‘1’ ‘2’ ‘5’. Per compiere questa scomposizione si divide semplicemente ogni volta il numero per **10** cosicché il resto sia sempre l’ultima cifra a destra (ottenendo quindi la “scomposizione”).

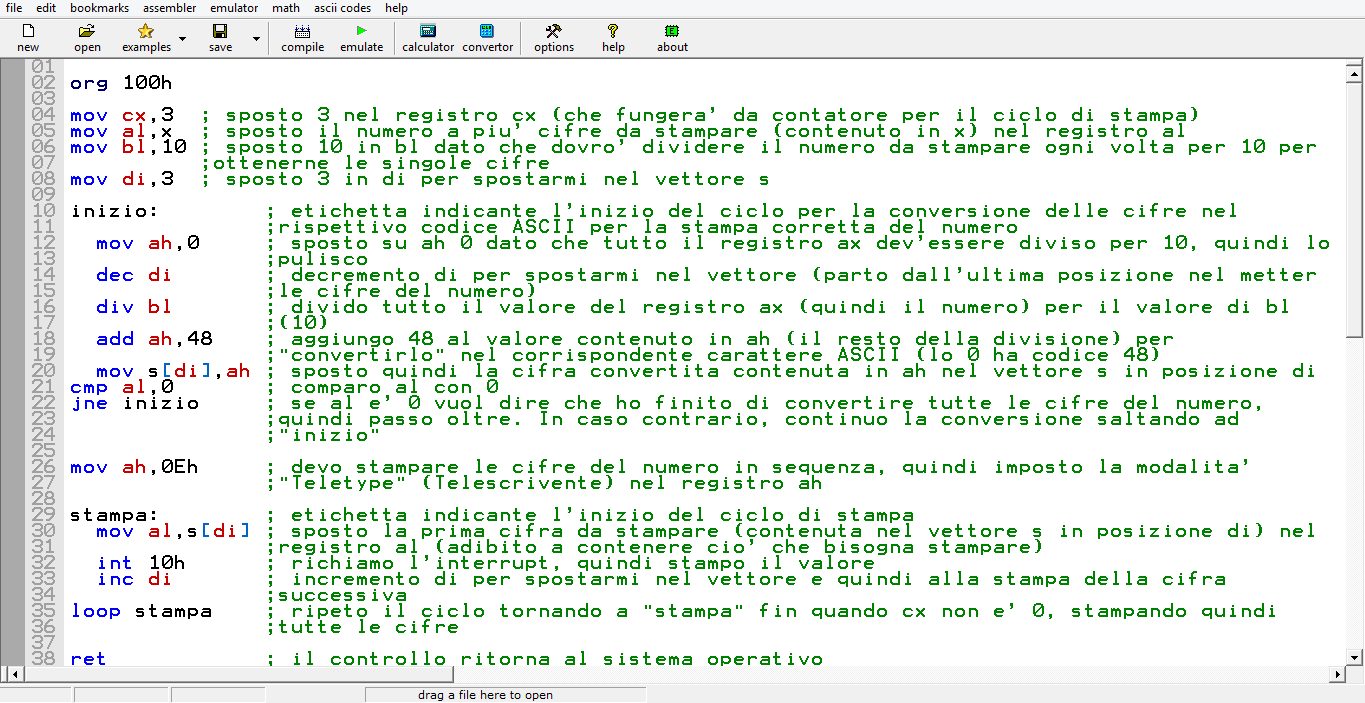
Innanzitutto ho dichiarato le variabili, una contenente il numero da stampare e un vettore dove andranno i codici ASCII corrispondenti alle cifre del numero.



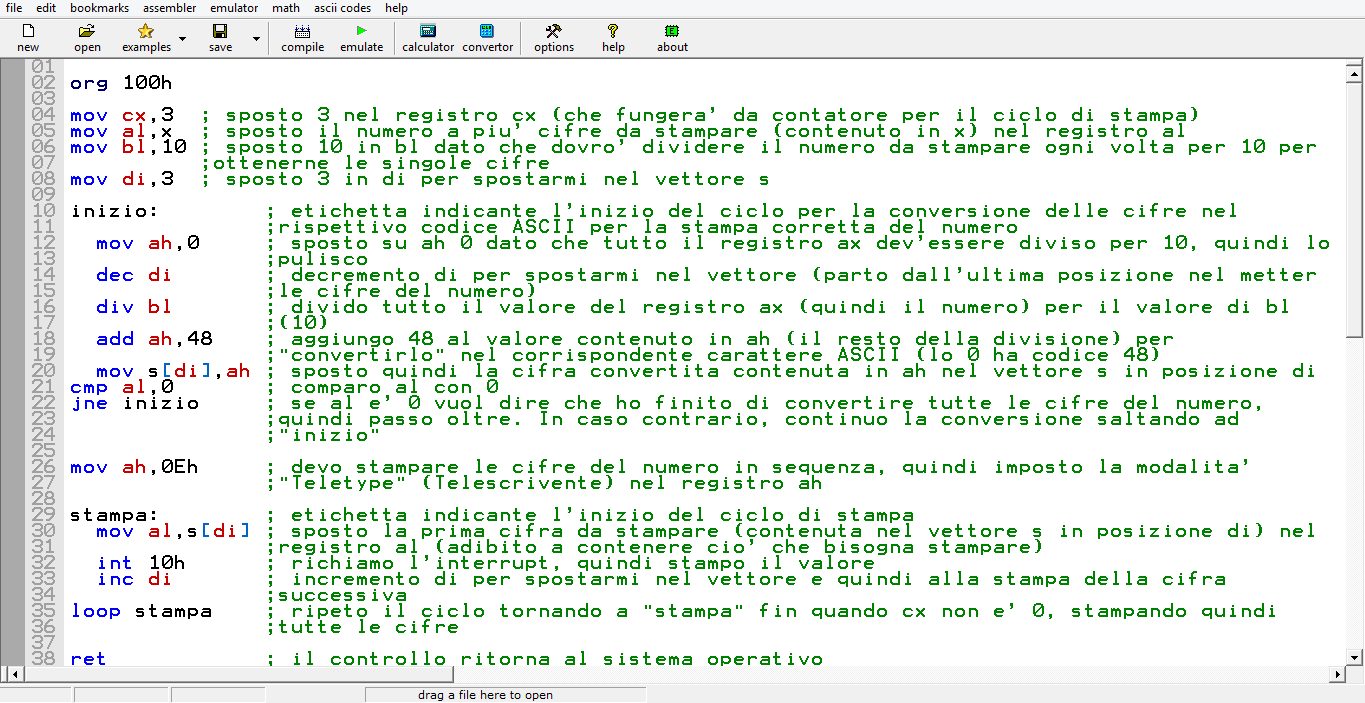
Sono passato quindi alla scrittura vera e propria del codice del programma. Dopo l’**org 100h** ho spostato il valore 3 nel registro contatore CX dato che il ciclo di stampa dovrà essere ripetuto 3 volte in quanto il numero massimo di cifre che possono costituire un numero rientrante in 8 bit (un Byte) ovvero 255 è 3. Poi sposto il numero da stampare (contenuto nella variabile **x**) nel registro **al** per poterci lavorare. Sposto quindi **10** in **bl** per poterlo utilizzare in seguito nelle divisioni. Dovendomi spostare in un vettore di 3 spazi sposto **3** nel registro indice **DI** (in realtà, dato che il vettore contiene 3 spazi, il valore massimo dovrebbe essere **2**, il problema in questo caso però non sussiste dato che il decremento di DI avverrà prima di usarlo con il vettore dentro il ciclo).



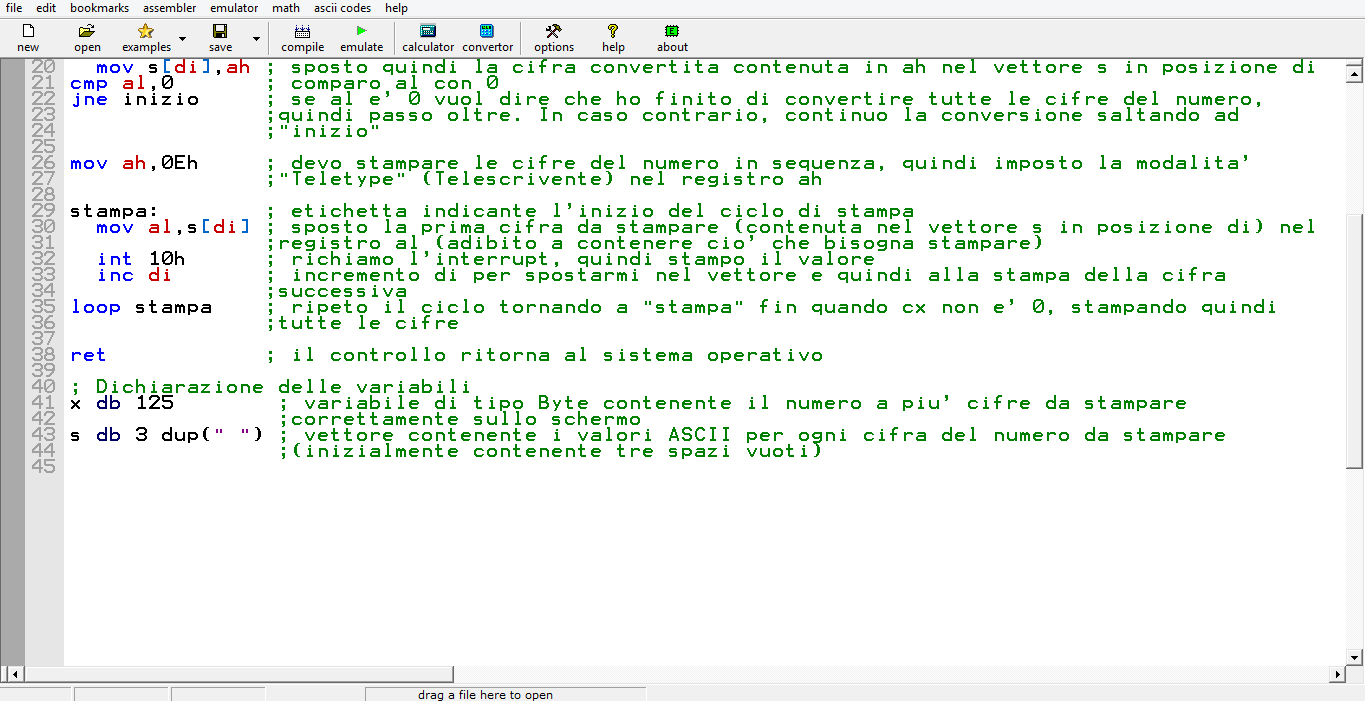
Inizio quindi il ciclo di “conversione” con l’etichetta “*inizio:*”. Ciclicamente pulisco inizialmente il registro **ah** dato che dovrò usare l’intero registro **ax** (nel cui al è contenuto il numero da scomporre) per la divisione per **10**. Decremento di per spostarmi nel vettore (inizio dal “fondo” andando al contrario per poi, nel ciclo di stampa, stampare dalla cifra più significativa, altrimenti, partendo sempre dalla posizione **0** del vettore, si stamperà il numero al contrario, in questo caso da 125 si otterrà 521 sullo schermo). Divido il numero contenuto in **al** per **10** (contenuto in **bl**) ottenendo così come resto in **ah** la cifra “scomposta” e in al le restanti cifre ancora da scomporre. “Converto” la cifra precedentemente “scomposta” (contenuta in ah) nel carattere corrispondente aggiungendovi **48**, ottenendo quindi il codice ASCII corretto. Sposto infine la cifra “convertita” (contenuta in **ah**) nel vettore **s** in posizione **di** (adibito a contenere questi caratteri).



Il ciclo di conversione va ripetuto finché **al** non contiene **0** (per cui non c’è più alcuna cifra da scomporre), quindi comparo il valore contenuto nel registro al con 0, dunque compio un salto condizionato: se è uguale a 0 vuol dire che il ciclo è terminato per cui passa all’istruzione successiva, altrimenti riesegui il ciclo saltando ad “*inizio*”.



Concluso il ciclo di “conversione” non resta altro che la stampa. Innanzitutto imposto la modalità teletype per stampare i caratteri sullo schermo e spostare automaticamente il cursore, sposto quindi **0Eh** nel registro **ah**. Inizio il ciclo di stampa con l’etichetta “*stampa:*”. Sposto il valore contenuto nel vettore **s** in posizione **di** (**s[di]**) nel registro **al** (adibito a contenere i caratteri da stampare). Richiamo quindi la routine del BIOS con l’**interrupt 10h** (**int 10h**) e stampo il carattere sullo schermo. Incremento poi **di** per spostarmi nel vettore. Il ciclo si ripeterà tornando a “*stampa*” finché **cx** non assumerà il valore di **0** con l’istruzione **loop stampa**. Infine, concluso anche il ciclo di stampa, il controllo ritorna al sistema operativo con l’istruzione **ret**.



Ho infine controllato il corretto funzionamento del programma verificando che la stampa del numero a più cifre non presentasse difetti.

