|  |  |  |
| --- | --- | --- |
| Altomare Andrea | Lab. Sistemi esperienza n. 12 | Classe 3^A 1/04/2015 |

**Assembly 8086: Interrupt DOS e input**

**INTERRUPT DOS**

Con gli interrupt DOS si richiamano routine (servizi) caricate in RAM come funzionalità base del sistema operativo. A differenza di quelli BIOS, richiamabili tramite il codice 10h, quelli **DOS** si richiamano con il codice **21h** (sintassi: **INT 21h**).

**INPUT**

Gli input si potrebbero compiere anche con i servizi BIOS, ma non è molto conveniente da un punto di vista della comodità, quindi si utilizzano i servizi DOS.

**Input di un dato: 1h**

L’input di un dato si esegue richiamando il servizio **1h** tramite l’interrupt 21h (DOS).

Tramite il codice

MOV AH,1h ; “Settaggio” del servizio su AH

INT 21h ; Chiamata all’ISR tramite l’interrupt DOS 21h

Si potrà dunque effettuare la lettura di un dato in input. Questo servizio opera implicitamente col registro **AX**: il dato immesso da tastiera verrà infatti messo in **AL**. La particolarità di questo servizio è che, dopo aver acquisito il dato, esegue “l’***echo***”, ovvero lo dà in output (automaticamente) per mostrarlo a schermo all’utente che lo digita.

**Input di un numero**

Quando si acquisisce qualcosa da tastiera, in **AL** si ottiene il relativo **codice ASCII**. Per cui, per acquisire correttamente un numero da tastiera, quando viene digitata una cifra bisogna sottrarle **48** (che è il codice ASCII corrispondente al numero **0**), ottenendo quindi la cifra effettiva con cui poter operare matematicamente.

**Input di un numero a più cifre**

Per dare in input un numero a più cifre, oltre alla problematica dell’inserimento delle singole cifre (spiegata nel paragrafo precedente), c’è anche quello della corretta memorizzazione in una variabile o in un registro. Innanzitutto stabilire quale registro o variabile utilizzare come **accumulatore** per le cifre che verranno immesse e quindi come “contenitore” del numero completo. L’utilizzo di un registro come accumulatore è in realtà preferibile rispetto a quello di una variabile dato che migliorerà l’efficienza del programma risparmiando accessi in memoria. La procedura per poter acquisire in input un numero a più cifre consiste nell’ottenere da tastiera la cifra (la quale sarà contenuta in **al**), sottrarvi **48** (per i motivi spiegati in precedenza), e addizionarlo al numero contenuto nel registro o variabile accumulatore; prima di aggiungerlo però si moltiplica il numero totale per **10** dato che il valore di ogni cifra sarà “incrementata” di una posizione: se una cifra aveva il valore di una decina adesso ha quello di una centinaia, se era una centinaia adesso diventa una migliaia, e così via… Se, per esempio, si vuole immettere il numero 76, si inserisce il 7, il contenuto dell’accumulatore (che all’inizio è **0**) verrà moltiplicato per **10**, quindi diventa 0, aggiungendo 7 si otterrà quindi il valore 7 come totale, poi si digita 6: il totale (7) viene moltiplicato per 10 diventando 70, poi vi si aggiunge il 6, ottenendo correttamente il numero a più cifre **76**. Si termina di acquisire cifre da tastiera (e quindi tutto il procedimento per “creare” il numero totale) quando l’utente prende il tasto “**ENTER**” (comunemente chiamato “*INVIO*”) dato che è ormai convenzione comune di terminare un input alla pressione di questo tasto.

**Servizio DOS output stringa: 9h**

Il servizio DOS **9h** permette di dare in output un’intera stringa (vettore di caratteri) fino a quando non “incontra” il carattere del simbolo del dollaro ‘**$**’. Per dare in output una stringa con questo servizio bisogna semplicemente mettere nel registro **DX** (utilizzato implicitamente dal servizio) l’indirizzo della locazione di memoria della stessa (**offset**), le operazioni del servizio si sposteranno in automatico nel vettore per stampare tutti i caratteri. Infine si richiama l’ISR tramite l’interrupt DOS **21h**.

Esempio:

ORG 100h

MOV AH,9h ; “setto” il servizio 9h in AH

MOV DX,OFFSET MSG ; sposto l’offset del vettore msg in DX

INT 21h ; richiamo l’ISR tramite l’interrupt DOS 21h

RET

MSG DB “Ciao, Mondo!$” ; vettore di caratteri (stringa) contenente il messaggio

Questo codice produrrà a schermo la frase “Ciao, Mondo!”.

**Servizio 09h (BIOS)**

Differentemente dal servizio 9h del DOS, **09h** del BIOS imposta la modalità “**testo**” con **gestione dei colori** in output. Questo servizio funziona esattamente come 0Ah, solo che in più si ha anche la possibilità di gestire i colori di ciò che viene dato in output tramite il registro **BL**. Nel registro BL si “setta” infatti il colore: partendo da sinistra (e quindi da quelli più significativi) con i primi quattro bit si imposta il colore dello **sfondo**, con gli ultimi quattro (quelli meno significativi) si imposta invece il colore del **testo**. I colori vanno da *0000* (**nero**) a *1111* (**bianco**).

Con il codice

MOV AH,09h ; “setto” il servizio su AH

MOV CX,1 ; imposto su CX il numero di volte da stampare la lettera

MOV BL,11110000b ; “setto” il colore in BL

MOV AL,’A’ ; su AL metto ciò che bisognerà stampare

INT 10h ; richiamo l’ISR tramite l’interrupt BIOS 10h

Si otterrà a schermo una lettera ‘A’ stampata una volta, testo nero su sfondo bianco.

**PROBLEMA**

Acquisire un numero a più cifre (per un massimo di **16 bit** consentiti dal microprocessore 8086).

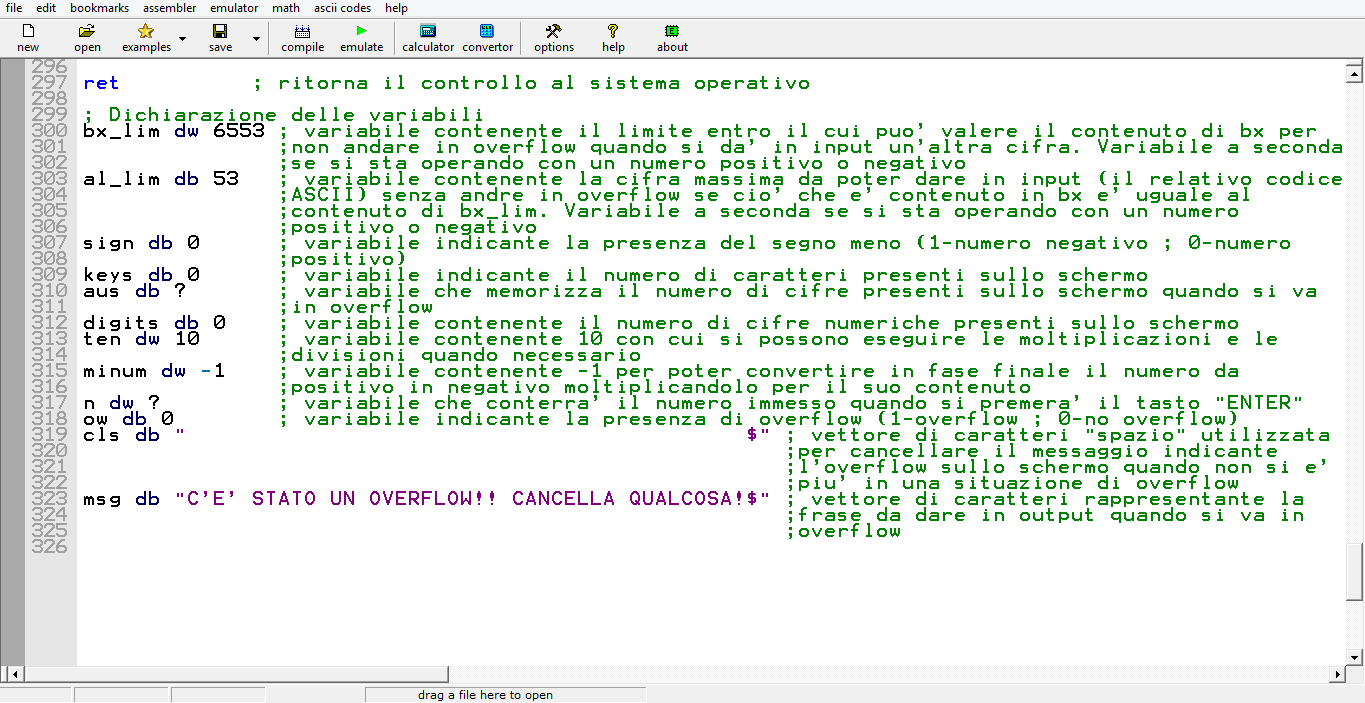
* Input di un numero a più cifre (obbligatorio)
* Dare un messaggio d’avviso quando si va in overflow (opzionale)
* Controllo sull’immissione di lettere ed altri caratteri speciali (opzionale)
* Cancellazione dei caratteri se si preme il tasto “BACKSPACE” (opzionale)
* Considerazione del segno (aggiunto da me per completezza)

L’acquisizione del numero deve essere fermata alla pressione del tasto “ENTER”. Il numero dovrà essere memorizzato infine in una variabile apposita (**n**). A seconda degli altri contenuti conseguiranno poi altre variabili.

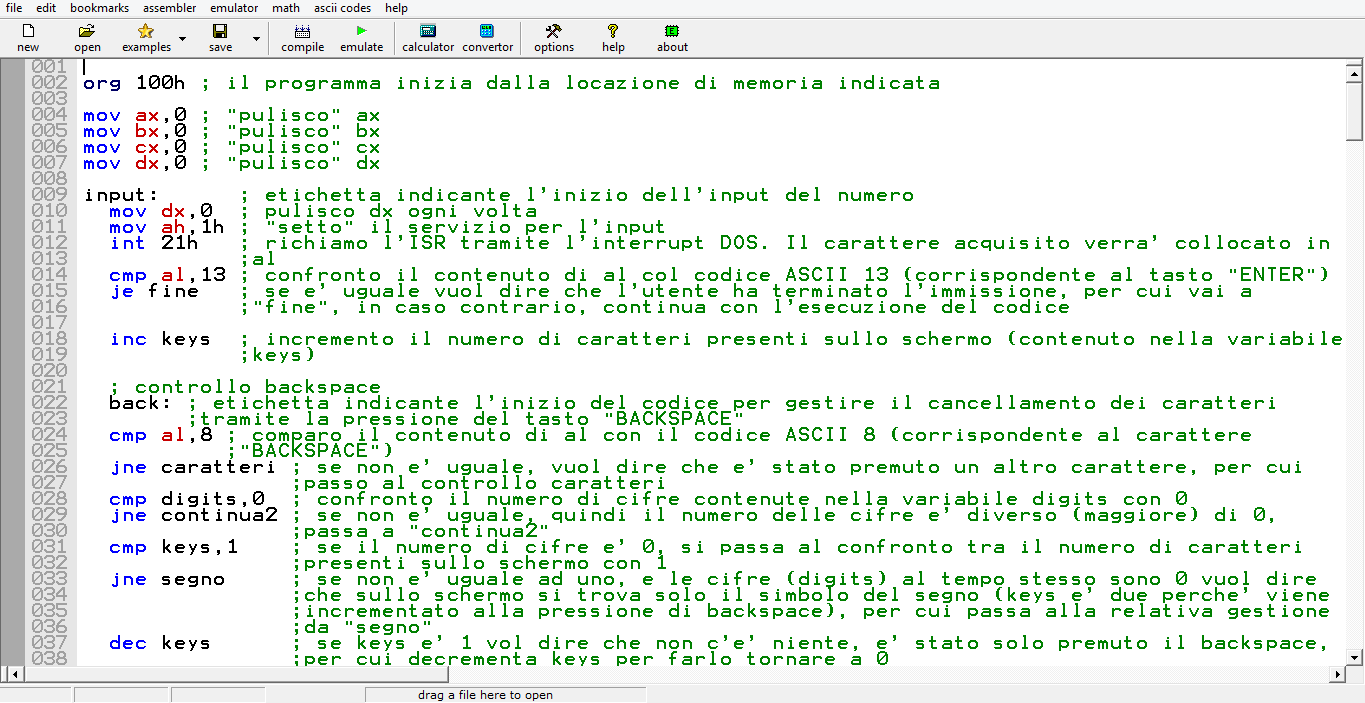
**REALIZZAZIONE DEL PROGRAMMA**

Innanzi tutto si è posto il problema dell’input di un numero a più cifre (risolto utilizzando il procedimento descritto sopra nell’apposito paragrafo). Poi, dovendo considerare l’overflow mi sarebbe bastato considerare il **Carry Flag**, oppure quando **DX** si “*sporcava*” al momento della moltiplicazione: in tutti e due i casi sarebbe stato un chiaro segno di un overflow del numero, ma così facendo non avrei potuto dare la possibilità all’utente di cancellare le cifre immesse fino a tornare nella situazione senza overflow. Ho così utilizzato la variabile “**ow**” per registrare la presenza di un eventuale overflow. Per capire quando il numero totale sarebbe andato in overflow ho confrontato (prima di compiere l’operazione di addizione dell’ultima cifra al numero totale) il numero totale con **6553**: se è maggiore, l’aggiunta di una qualsiasi cifra comporterà un overflow (65541 è infatti overflow); se è minore, l’aggiunta di una qualsiasi cifra non potrà comportare un overflow (65529 NON è overflow); se è uguale, l’overflow dipenderà da ciò che immette l’utente, per cui confronto l’ultima cifra data in input con il carattere ‘**5**’, se è minore o uguale allora non è overflow (16 bit consentono l’immissione di un numero senza considerazione del segno fino a 65535), se è maggiore allora è overflow. Per i numeri col segno (quindi considerati come negativi) vale lo stesso ragionamento, solo che il limite che per i numeri positivi senza considerare il segno è **6553**, il limite per i negativi è **3276**, mentre il carattere massimo immettibile per non incorrere in un overflow nel caso in cui si ha un totale uguale a 3276, è ‘**7**’ (16 bit consentono l’immissione di un numero con segno fino a 32767 [-32767 dato che si sta parlando di **numeri negativi**]). Per ovviare a ciò mi sono quindi servito di due variabili: “**bx\_lim**” (indicante il limite entro il quale deve stare il numero totale nel registro accumulatore) e “**al\_lim**” (indicante il carattere “massimo” immettibile). Quando si va in overflow do in output un messaggio, memorizzato come vettore di caratteri nella variabile “**msg**”. Per cancellare il messaggio quando si cancellano le cifre che hanno determinato l’overflow mi sono servito di un altro vettore di caratteri contenente solo spazi, memorizzati nella variabile “**cls**”. Si va in overflow anche quando si immettono più di cinque cifre dato che i numeri massimi (per 16 bit) sia per i numeri positivi senza considerare il segno che quelli su cui si considera il segno (in questo caso negativi) sono costituiti da un massimo di cinque cifre. Ho inserito un controllo per capire se effettivamente si sia andato in overflow quando si hanno più di 5 cifre dato che l’utente potrebbe immettere cinque zeri e poi il numero, impedendo quindi l’output dell’avviso di overflow quando non realmente necessario. Quando si preme “ENTER” concludendo l’immissione, il numero totale viene memorizzato nella variabile “**n**”. Quando si sta immettendo un numero negativo, alla fine, per la corretta memorizzazione, si deve moltiplicare per **-1** che è memorizzato nella variabile “**minum**”. Quando si eseguono le operazioni sopra descritte per acquisire il numero con più cifre, bisogna moltiplicare il numero totale per **10** che è memorizzato nella variabile “**ten**”. Il numero delle cifre immesse è memorizzato nella variabile “**digits**”, mentre il numero di caratteri nella variabile “**keys**”. Quando si vuole immettere un numero col segno viene settato a **1** l’apposita variabile “**sign**”. Per poter controllare la presenza di overflow quando si cancellano delle cifre, registro il numero di cifre presenti quando si va in overflow nella variabile “**aus**” così che quando il numero di cifre presenti, quando se ne cancellano alcune, diventano meno di quelle di “aus” vuol dire che non si è più in overflow, per cui si procede alla cancellazione del messaggio.

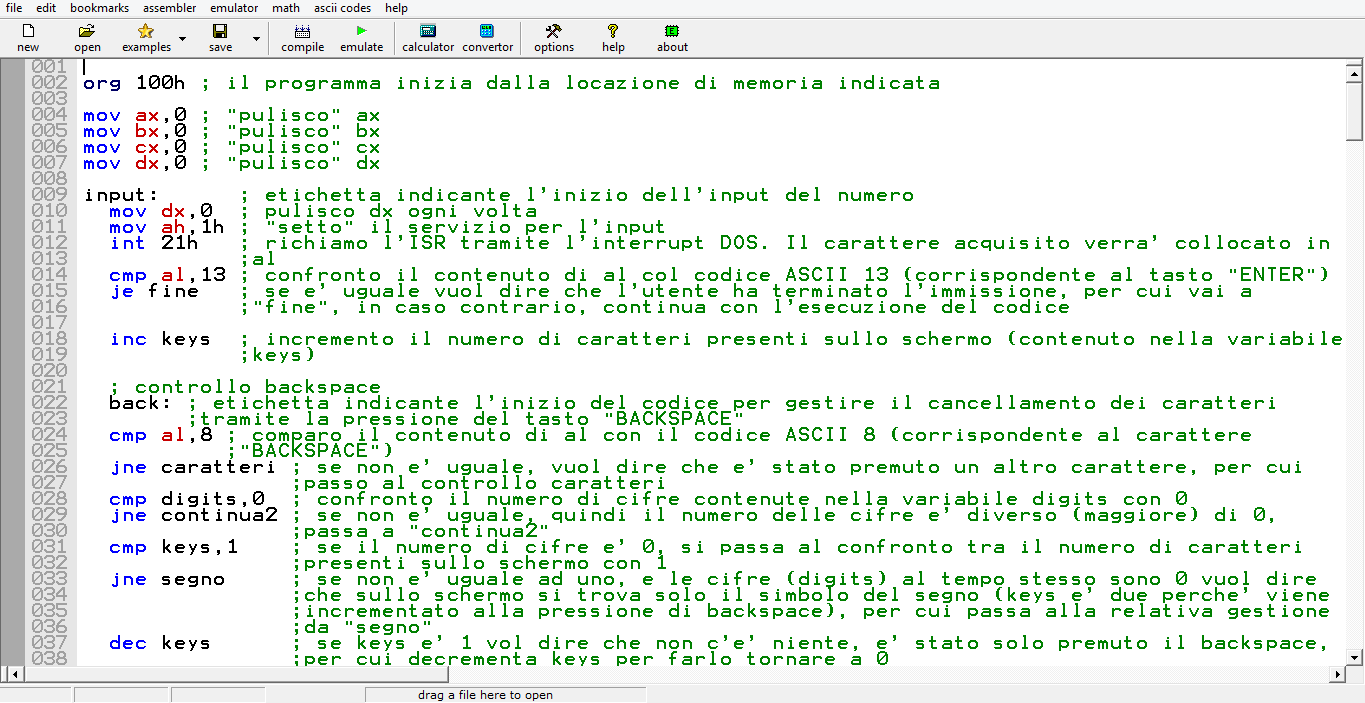
La dichiarazione delle variabili:



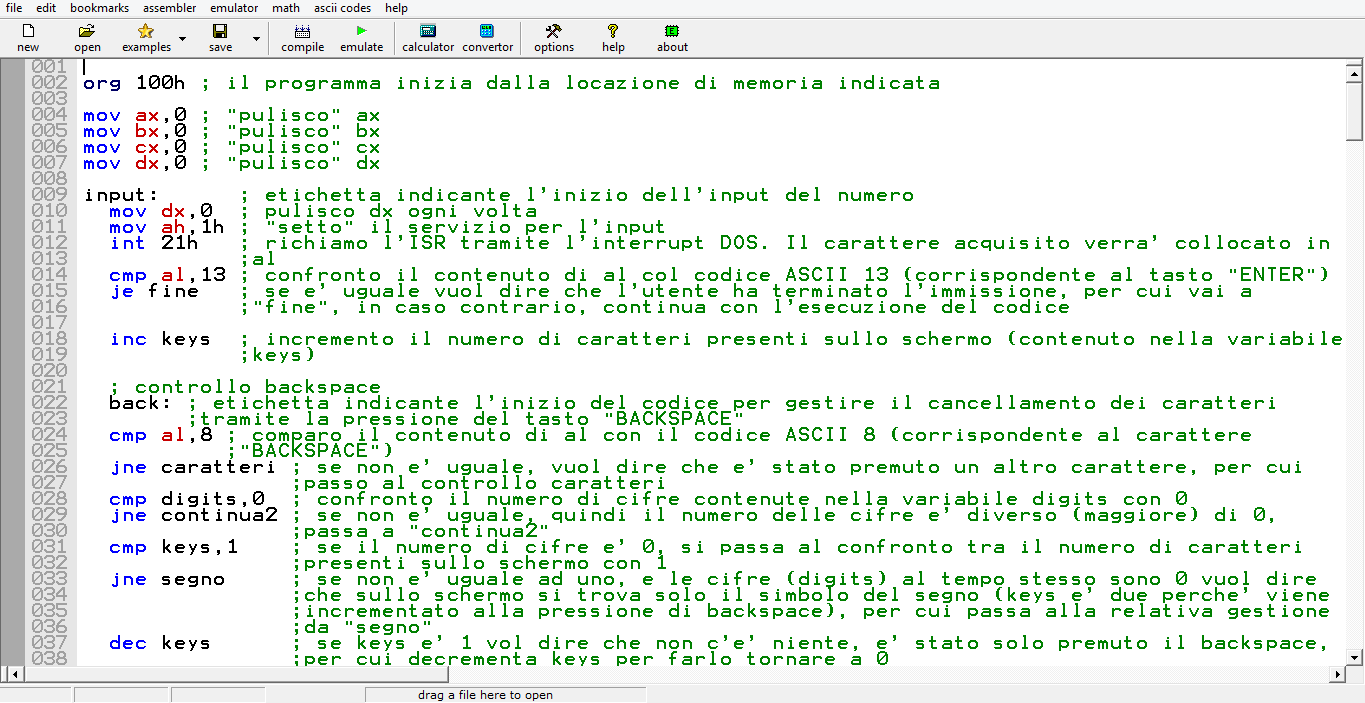
Dopo l’**org 100h** pulisco i registri.



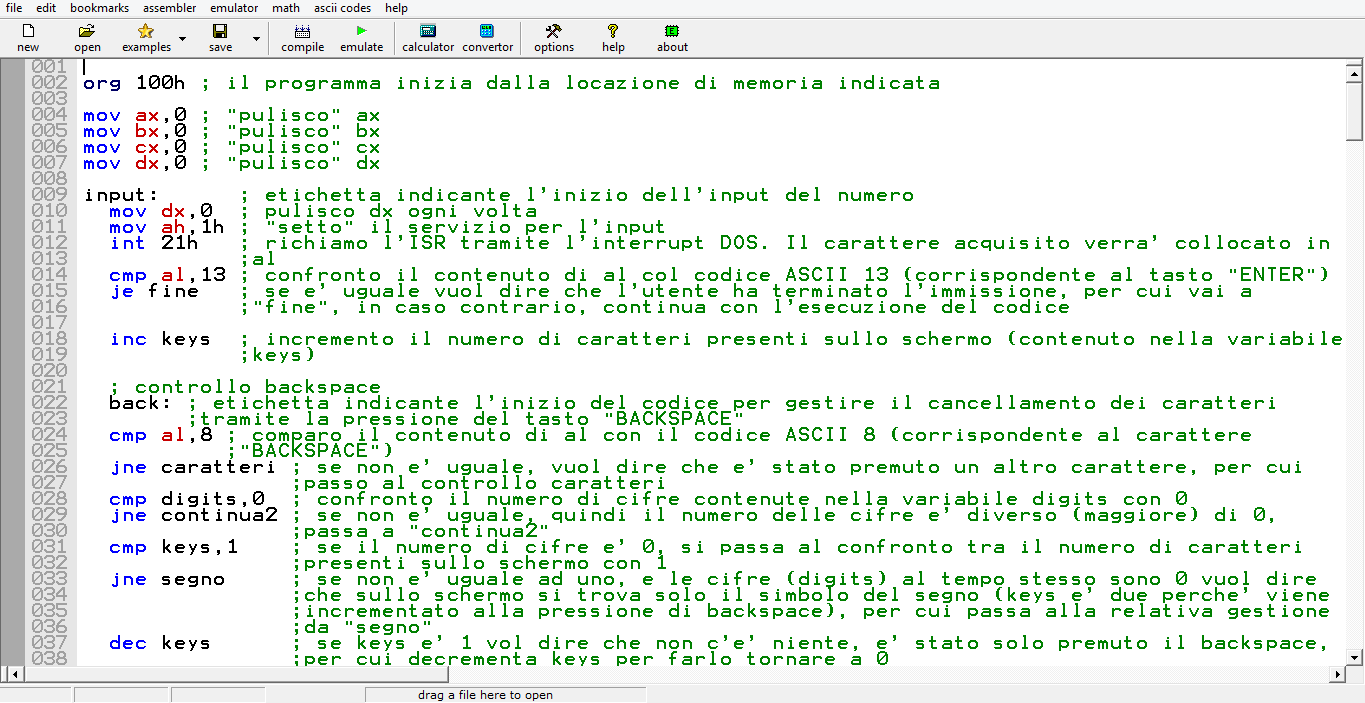
Inizio quindi l’input del numero impostando l’apposito servizio e richiamando l’**ISR** con l’interrupt DOS **21h**.

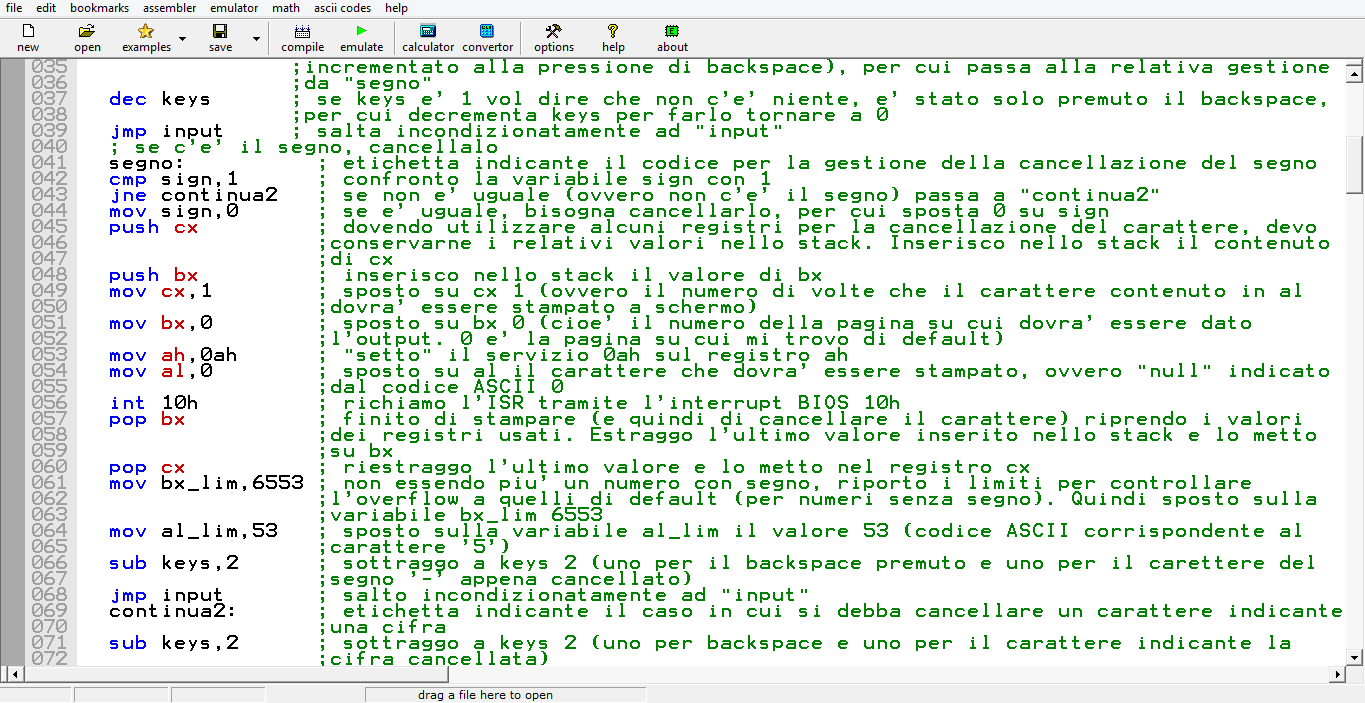


Confronto ciò che si è acquisito con il codice ASCII **13** (corrispondente al tasto “ENTER”), se è uguale si termina l’input andando a “*fine*”, altrimenti si incrementa il numero di caratteri.

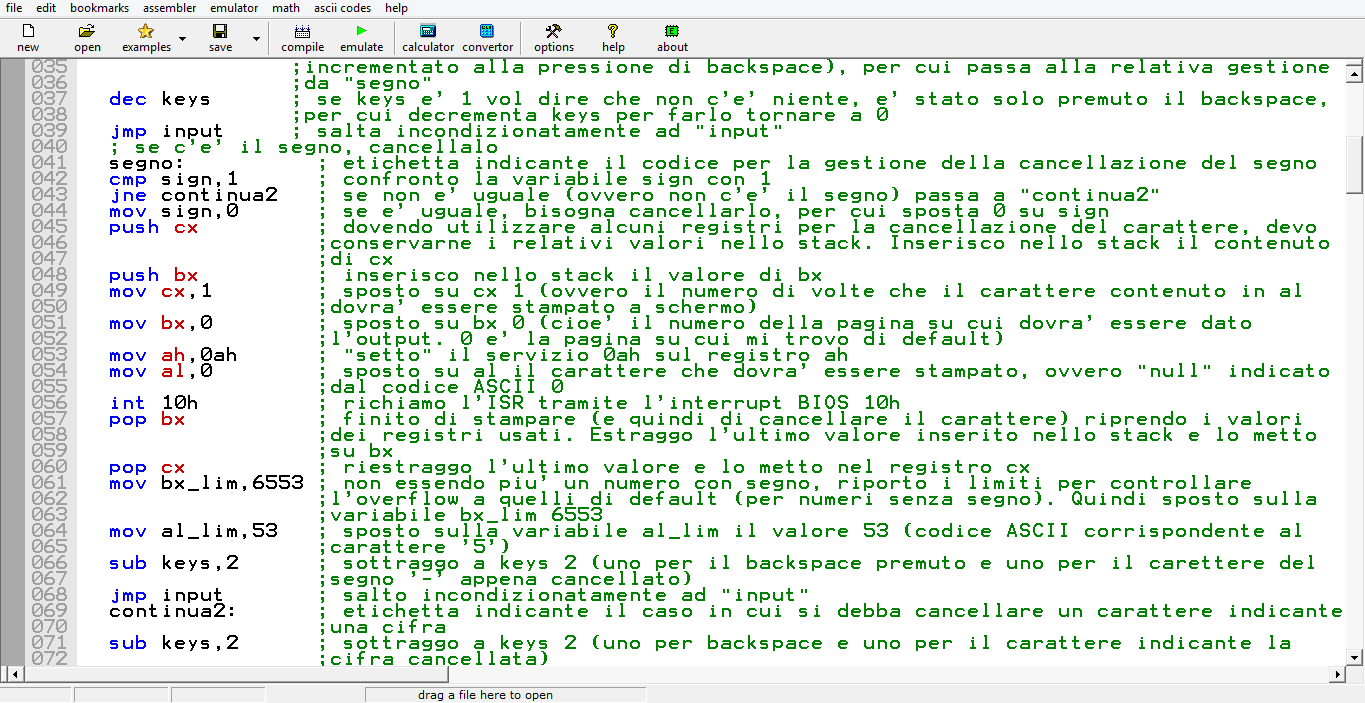


Confronto ciò che è stato immesso con **8** (codice ASCII che identifica il tasto “BACKSPACE” che si usa per cancellare qualcosa), se non è uguale passo a “caratteri” per trattare il dato immesso come tale, altrimenti inizio il codice per trattare la cancellazione di un carattere. Se le cifre non sono 0 (per cui ci sono cifre) allora passo a “*continua2*” altrimenti confronto **keys** con **1**: se è uguale vuol dire che a schermo non c’è niente (dato che l’unico carattere per cui keys è 1 è proprio il tasto “BACKSPACE” appena premuto), quindi semplicemente decremento keys e ritorno incondizionatamente ad input. Se keys non è 1 (ma le cifre sono 0 dato che ho controllato prima) vuol dire che a schermo c’è il carattere rappresentante il segno meno, per cui passo a “segno” per trattarlo.

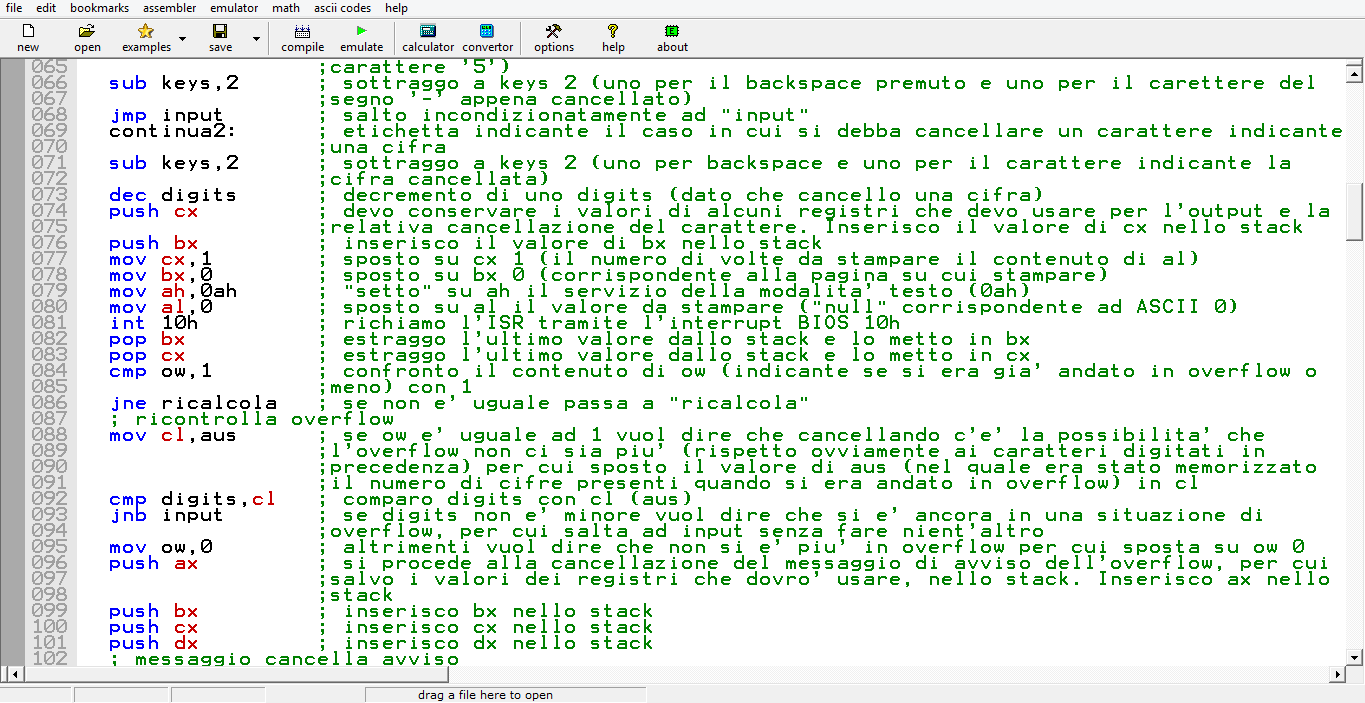




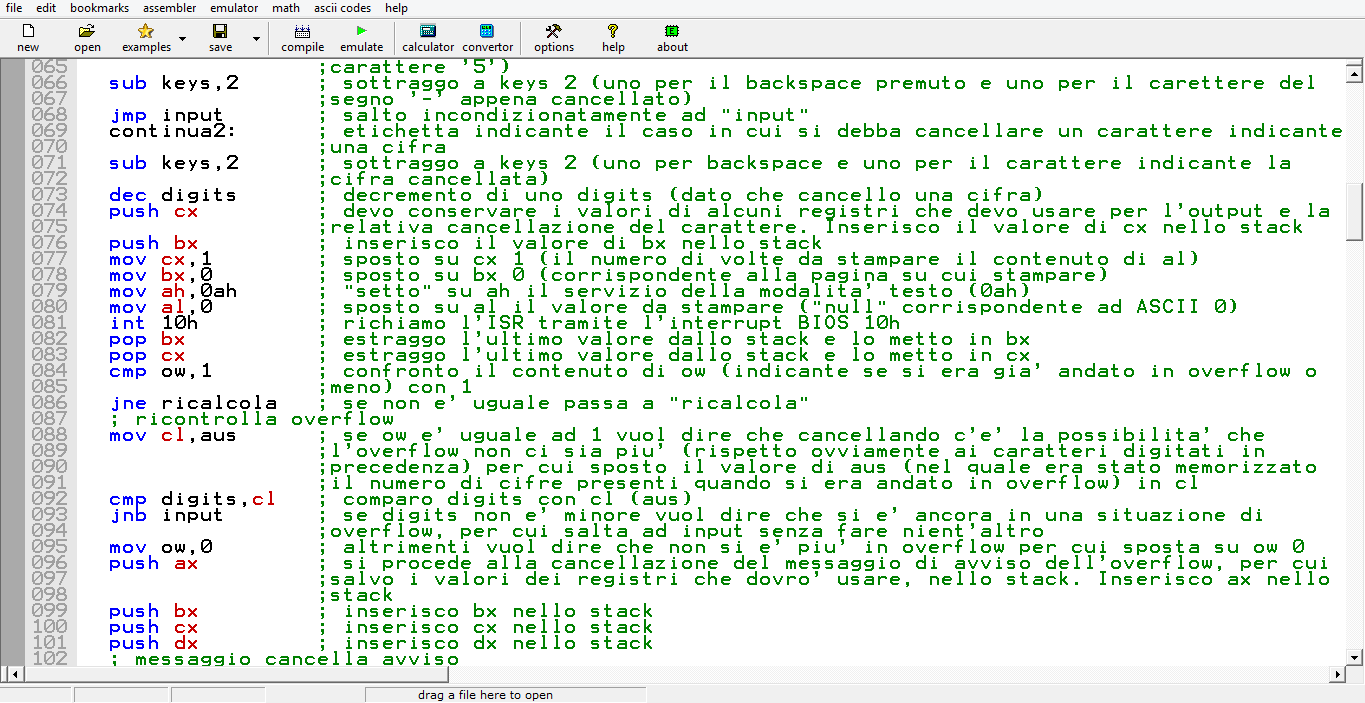
Dall’etichetta “*segno:*” tratto il carattere simboleggiante, appunto, il segno. Innanzitutto comparo **sign** con **1**, se non è uguale passo a “*continua2*”, altrimenti (dato che sto cancellando il segno) setto a **0** **sign**. Per cancellare qualcosa quando premo il tasto “BACKSPACE”, in ogni caso, setto la modalità *testo* e stampo per una volta (1 su cx) sulla pagina 0 sulla quale sto operando (0 su bx) il carattere “**null**” rappresentato dal codice ASCII **0** (su **al**), e richiamo l’**ISR** tramite l’interrupt BIOS **10h**. Prima di farlo però preservo i valori dei registri nello **stack**, e una volta finito di “cancellare” il carattere li riestraggo. Siccome non considero più il numero come negativo risposto su **bx\_lim** e **ax\_lim** i suoi valori originari. Poi sottraggo a keys **2** (uno per il “BACKSPACE” e uno per il carattere appena cancellato), infine salto incondizionatamente ad “*input*”.

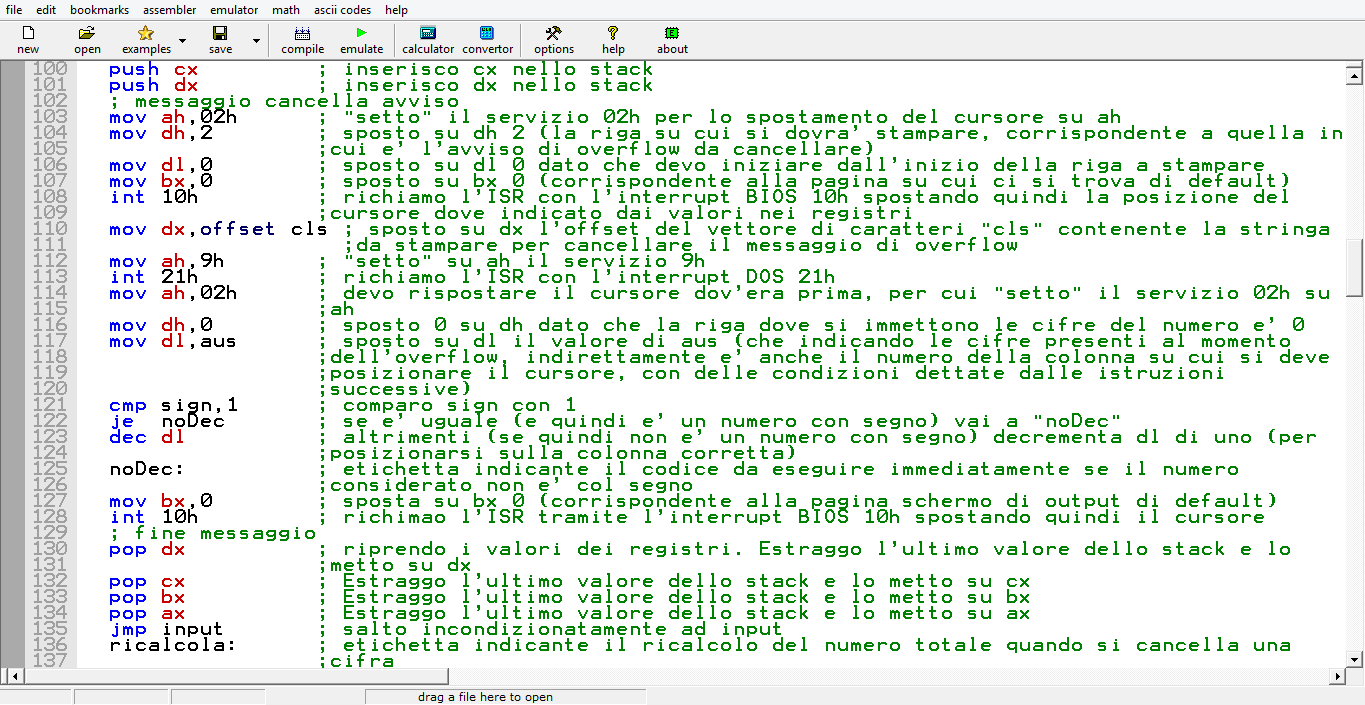


Se sono passato all’etichetta “continua2:” vuol dire che sto cancellando una cifra, per cui sottraggo 2 a keys (uno per “BACKSPACE” e uno per il carattere della cifra che si cancella), decremento digits e cancello il carattere come spiegato.

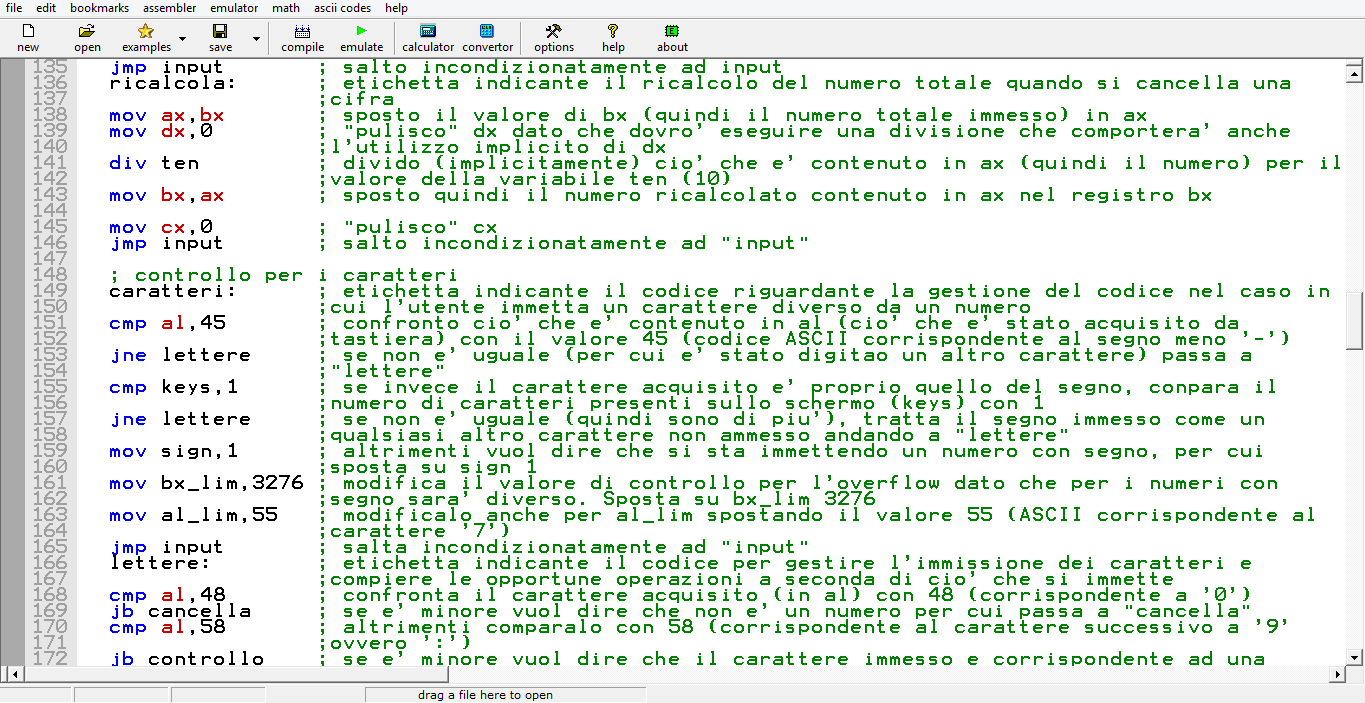


Siccome sto cancellando un carattere, se sono in overflow (per questo confronto **ow** con **1**) eseguo il codice per capire se non sono più in overflow, altrimenti passo a “ricalcola”. In caso sono in overflow, per capire se cancellando la cifra non lo sono più devo confrontare digits con aus (per i motivi scritti sopra), quindi sposto su cl aus e lo comparo con digits, se non è minore vuol dire che si è ancora in overflow, quindi torno ad “input”, altrimenti sposto su ow 0 e procedo con la cancellazione del messaggio di overflow. Sposto il cursore alla posizione dove nella quale si trova il messaggio tarmite il servizio **02h**. Stampo quindi la stringa di spazi per dare l’impressione di cancellare il messaggio, mi servo quindi del servizio **9h** del DOS e del vettore “cls”. Finito di cancellare il messaggio, risposto il cursore dove si trovano le cifre tramite il servizio BIOS **02h**, la riga è la 0 e la colonna è **aus** dato che è proprio dopo l’ultima cifra (le colonne come le righe partono da 0) se si tratta di un numero con segno, altrimenti se è un numero senza segno, lo decremento. Per fare quest’ultima operazione comparo sign con 1, se è uguale (e quindi si tratta di un numero negativo) passo a “*noDec*”, altrimenti decremento dl. Da “*noDec:*” imposto la pagina su cui stampare a 0 e richiamo l’**ISR** con l’interrupt BIOS **10h**. Infine riprendo i valori dallo stack (messi prima per conservare i valori dei registri da utilizzare) e salto incondizionatamente ad “*input*”.

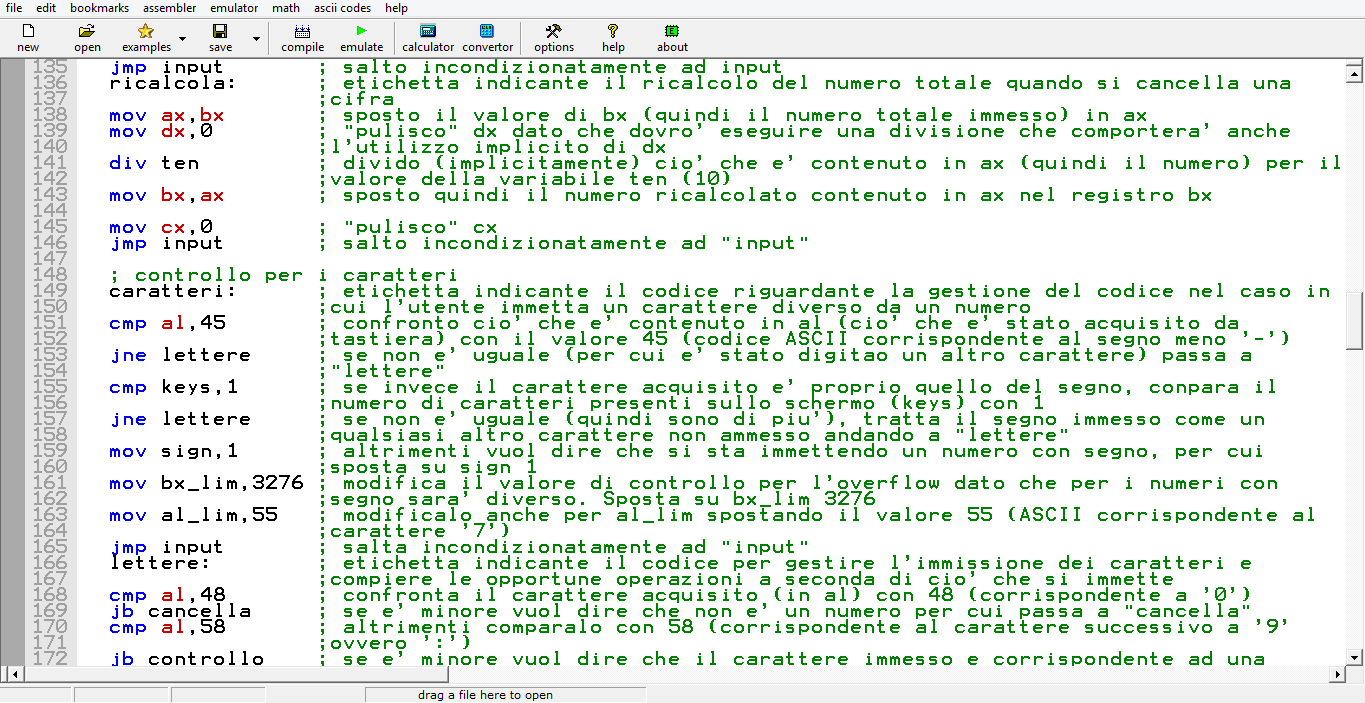




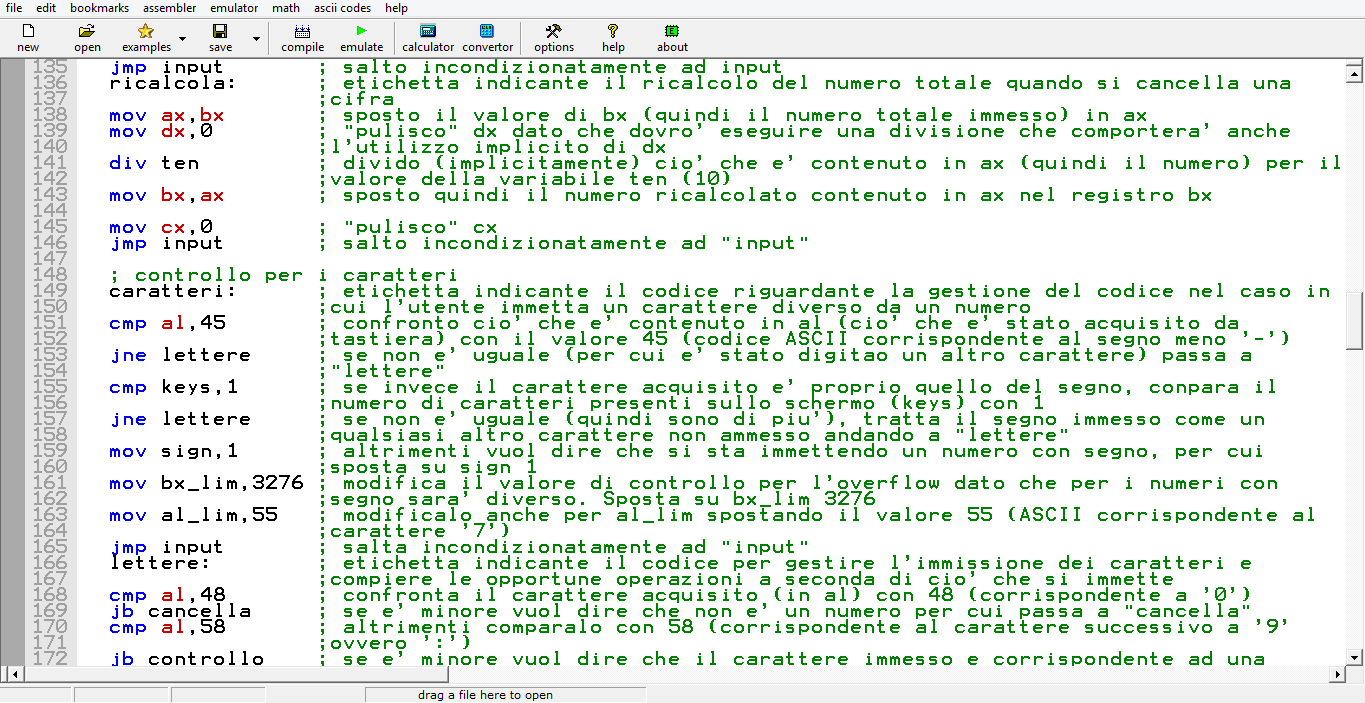
Se non si era in una situazione di overflow allora dopo il confronto si sarà passati all’etichetta “*ricalcola:*” dopo la cui ci sarà il codice per ricalcolare il numero totale quando si cancella una cifra. Semplicemente si esegue il procedimento inverso a quello descritto sopra per l’acquisizione di un numero a più cifre: si divide il numero per 10. Sposto quindi il numero totale contenuto in **bx** (che utilizzo come registro accumulatore) in ax e lo divido per **10** (nella variabile **ten**), e sposto il risultato (contenuto sempre in **ax**) in **bx**. Pulisco cx e salto incondizionatamente ad “*input*”.

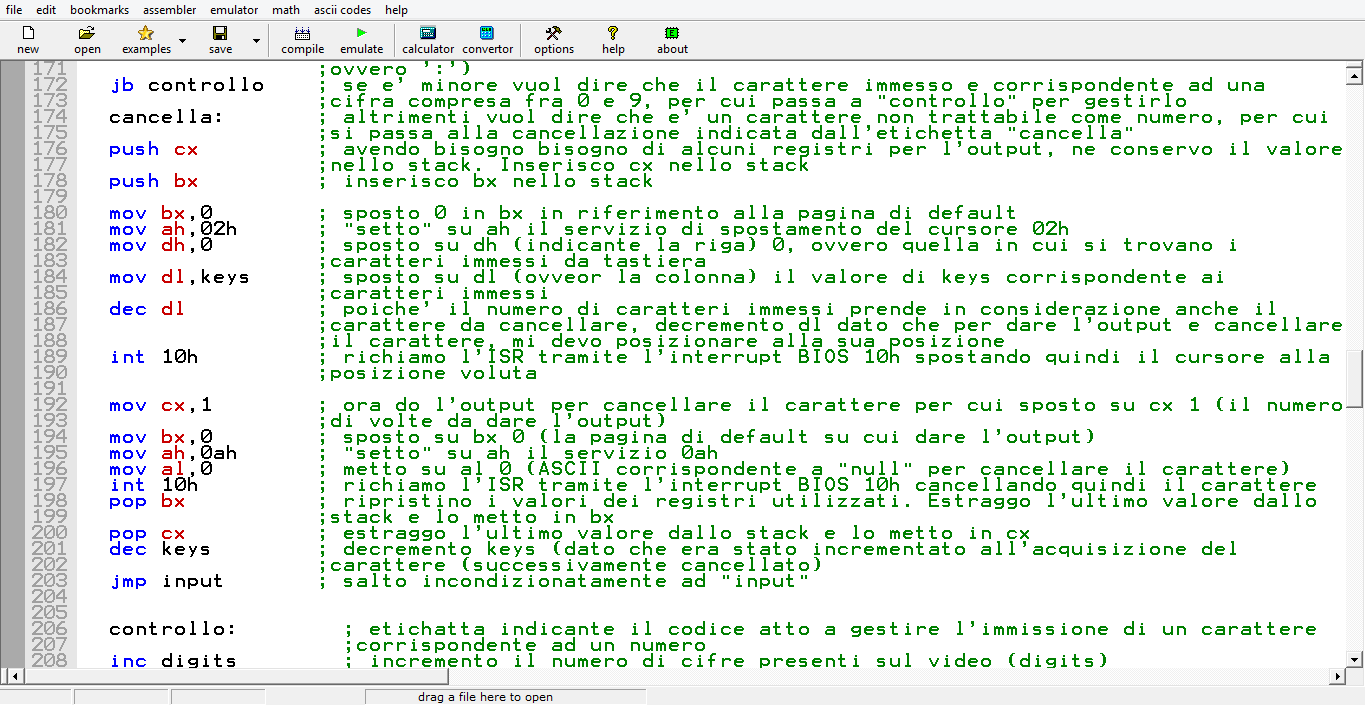


Dall’etichetta “caratteri:” inizia il codice per gestire, appunto, i caratteri non corrispondenti a “BACKSPACE” o “ENTER”. Innanzitutto confronto **al** con **45** (codice ASCII corrispondente al simbolo del segno meno ‘**-**‘), se non è uguale passo a “*lettere*”, altrimenti confronto **keys** con **1**, se non è uguale vuol dire che il segno c’è già, per cui passo a “*lettere*” per trattare il carattere come un qualsiasi altro carattere speciale o lettera che non è una cifra, altrimenti vuol dire che l’utente sta cercando di immettere un numero negativo per cui setto **sign** a **1** e modifico **bx\_lim** e **al\_lim** per portarli ai valori utili per i numeri con segno. Ritorno infine ad “*input*”.

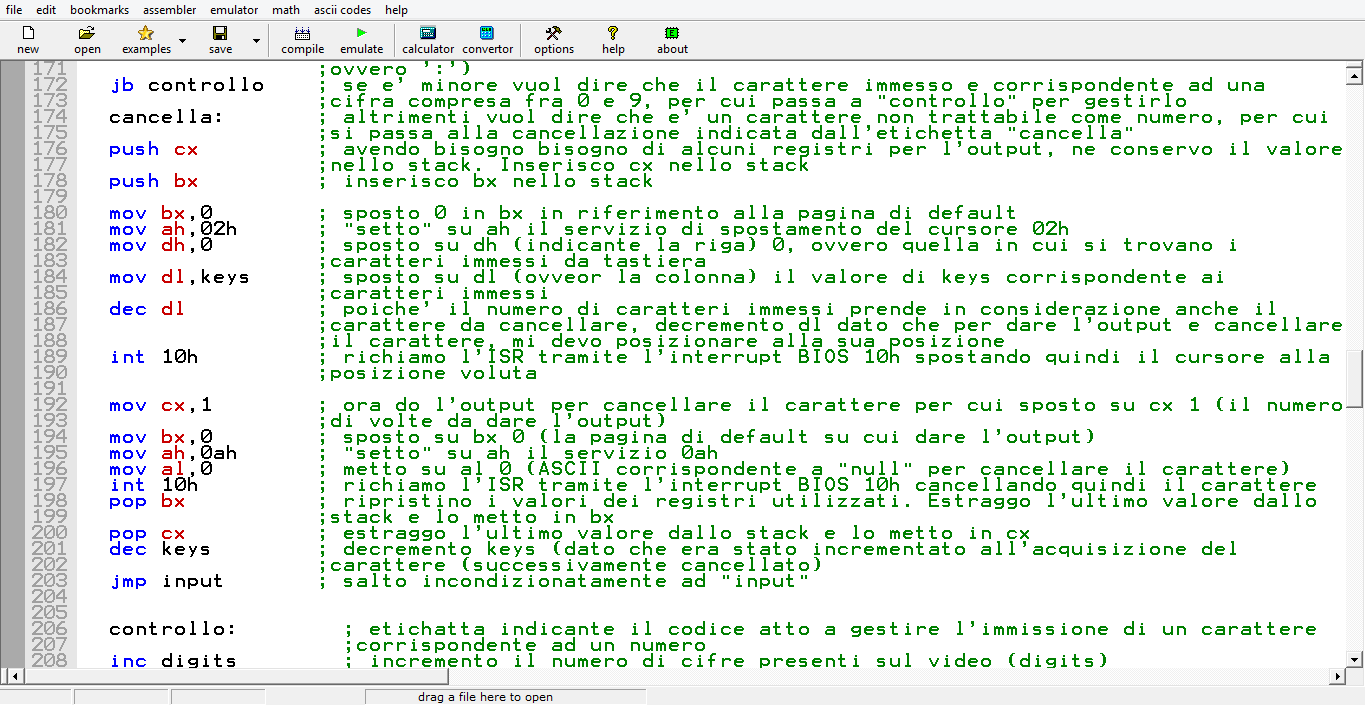


Dall’etichetta “*lettere:*” eseguo il codice per capire se il carattere immesso è o meno una cifra. Confronto dunque al con **48** (codice ASCII corrispondente a **0**), se è minore vuol dire che è un carattere speciale perciò passo a “cancella”, altrimenti lo confronto con **58** (codice ASCII corrispondente al carattere dei “due punti” ‘**:**’ il quale viene subito dopo il carattere della cifra **9**), se è minore vuol dire che il carattere che si sta analizzando è una cifra, per cui va trattata come tale e si passa a “*controllo*”, altrimenti si esegue il codice di “*cancella*”.

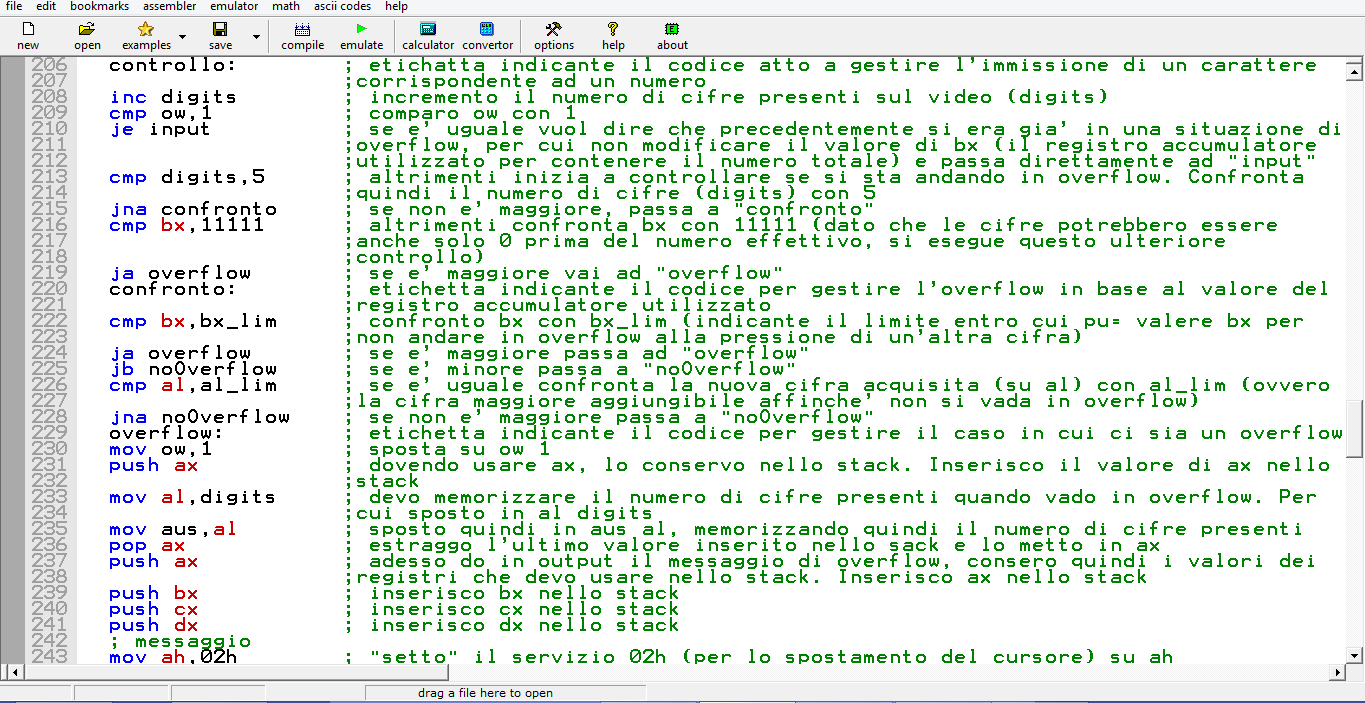




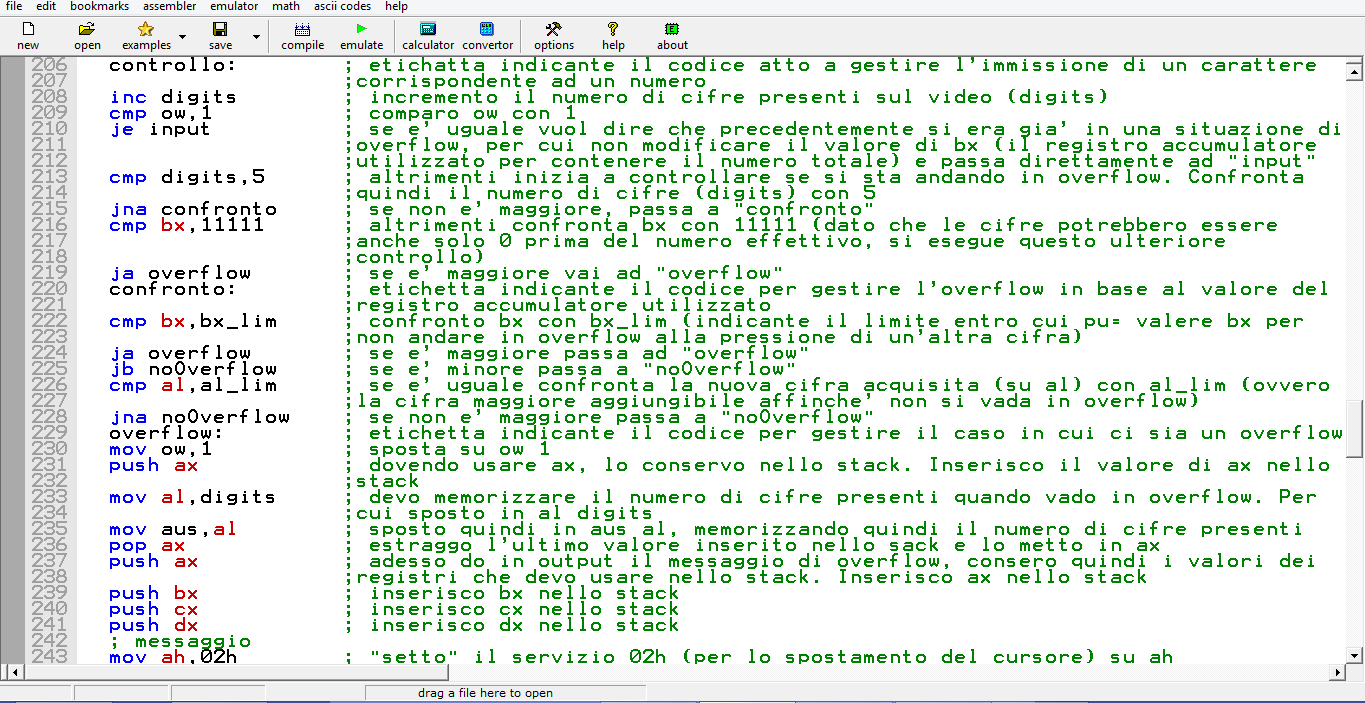
Dall’etichetta “*cancella:*” c’è il codice atto a trattare la cancellazione dei caratteri speciali e delle lettere dato che non possono essere immessi in quanto si sta acquisendo un numero. Innanzitutto sposto il cursore ad una posizione indietro (dato che se per esempio si immette la lettera ‘*d*’ il cursore andrà avanti, ma per cancellarla bisognerà spostarsi indietro) quindi sulle colonne (**dl**) sposto **keys** e lo decremento, poi richiamo l’interrupt 10h e sposto il cursore. Stampo quindi come visto prima il “**null**” alla posizione del carattere da cancellare per eliminarlo dallo schermo. Tutto questo preservando i valori dei registri nello stack. Decremento **keys** (dato che all’immissione della lettera o del carattere speciale era stato comunue incrementato), e ritorno ad “*input*”.



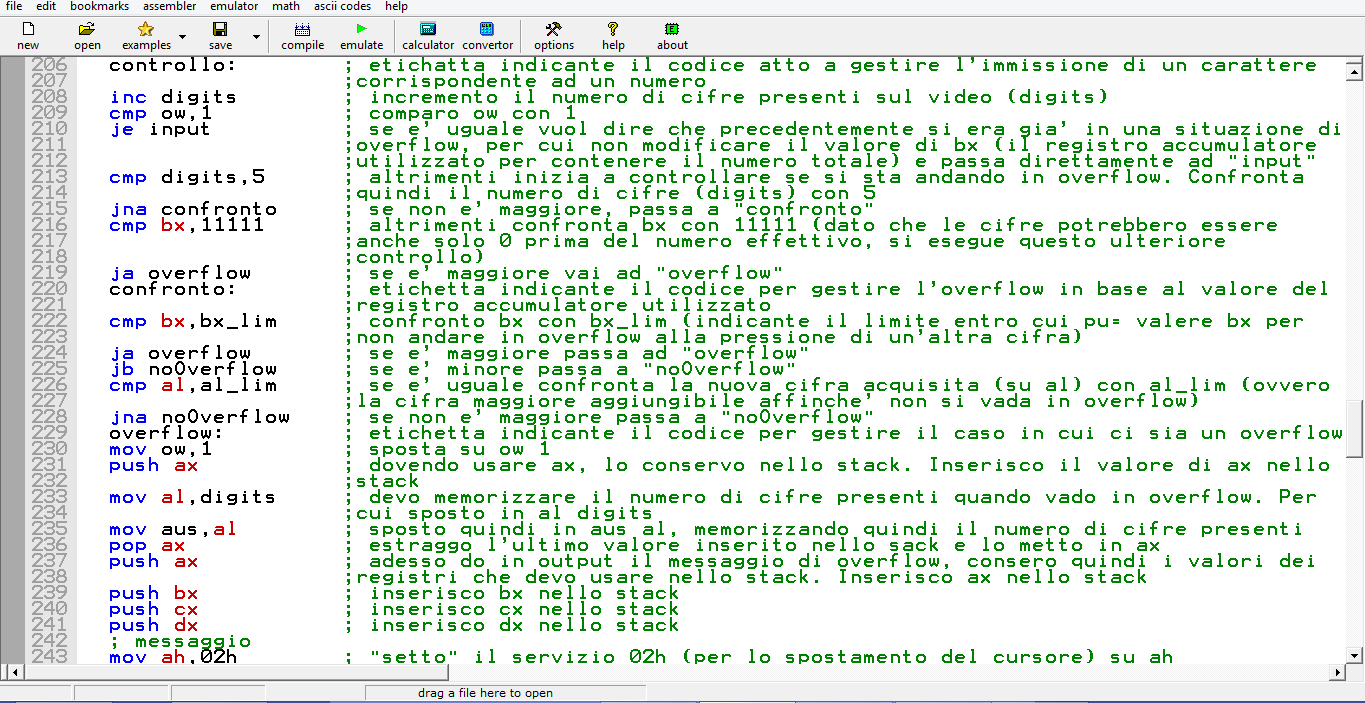
Se sono all’etichetta “**controllo:**” vuol dire che il carattere immesso è una cifra, ergo incremento **digits**, poi confronto **ow** con **1**, se è uguale vuol dire che sono già in overflow, per cui non devo fare niente e ritorno ad “*input*”, altrimenti passo ad eseguire le operazioni per capire se si sta in overflow o meno. Innanzitutto confronto **digits** con **5**, se non è maggiore passo a confronto, altrimenti potrebbe essere overflow (ma non per forza dato che il numero di cifre potrebbe anche essere dato in parte o totalmente da zeri messi prima di tutte le altre cifre), per cui confronto **bx** con **11111** (il quale è il numero minimo che, in congruenza con un numero di cifre maggiore a **5**, se superato dà overflow), quindi se bx è maggiore vado ad “*overflow*”, se no si esegue direttamente il codice di “*confronto*” continuando l’analisi per capire se si sta andando in overflow.

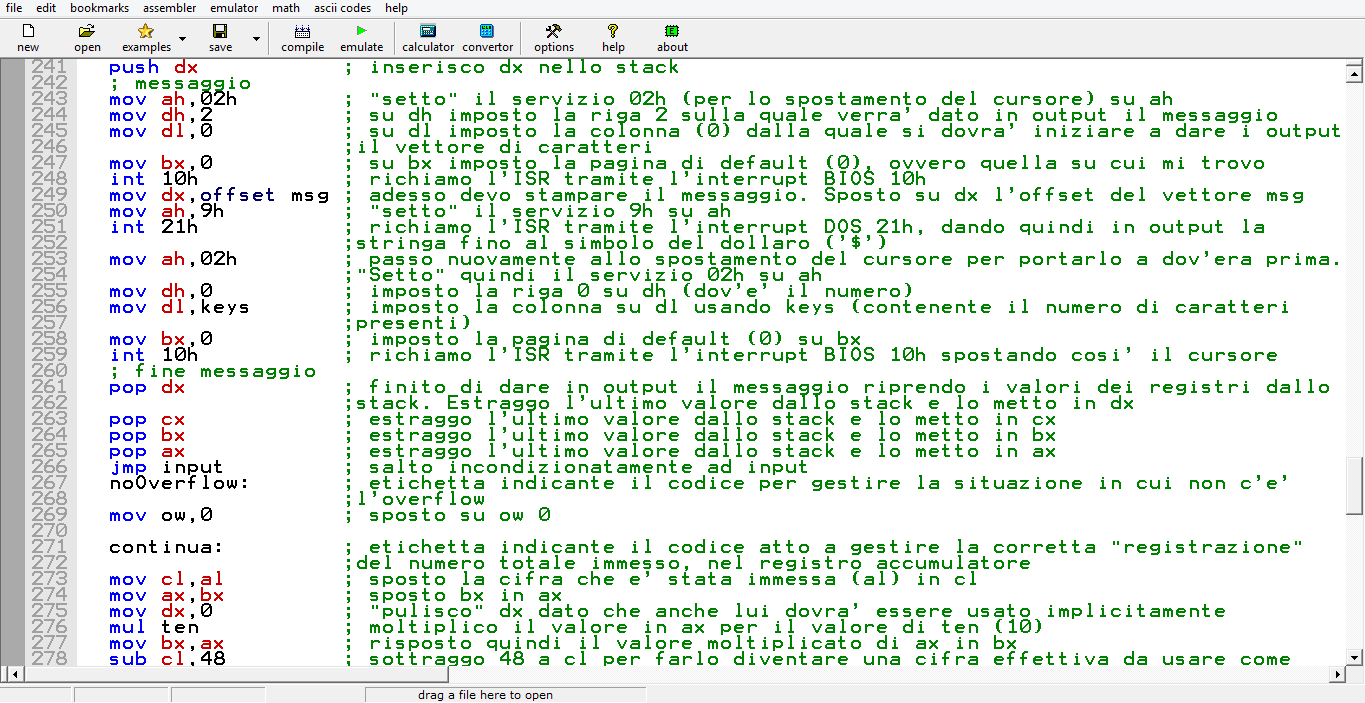


Dall’etichetta “*confronto:*” continuo con i controlli per l’overflow. Confronto **bx** con **bx\_lim**, se è maggiore passo a “*overflow*”, se è minore vado a “*noOverflow*”, se è uguale confronto ciò che l’utente ha appena immesso (in **al**) con **al\_lim**, se non è maggiore vado a “*noOverflow*”, altrimenti eseguo il codice di “*overflow*”.

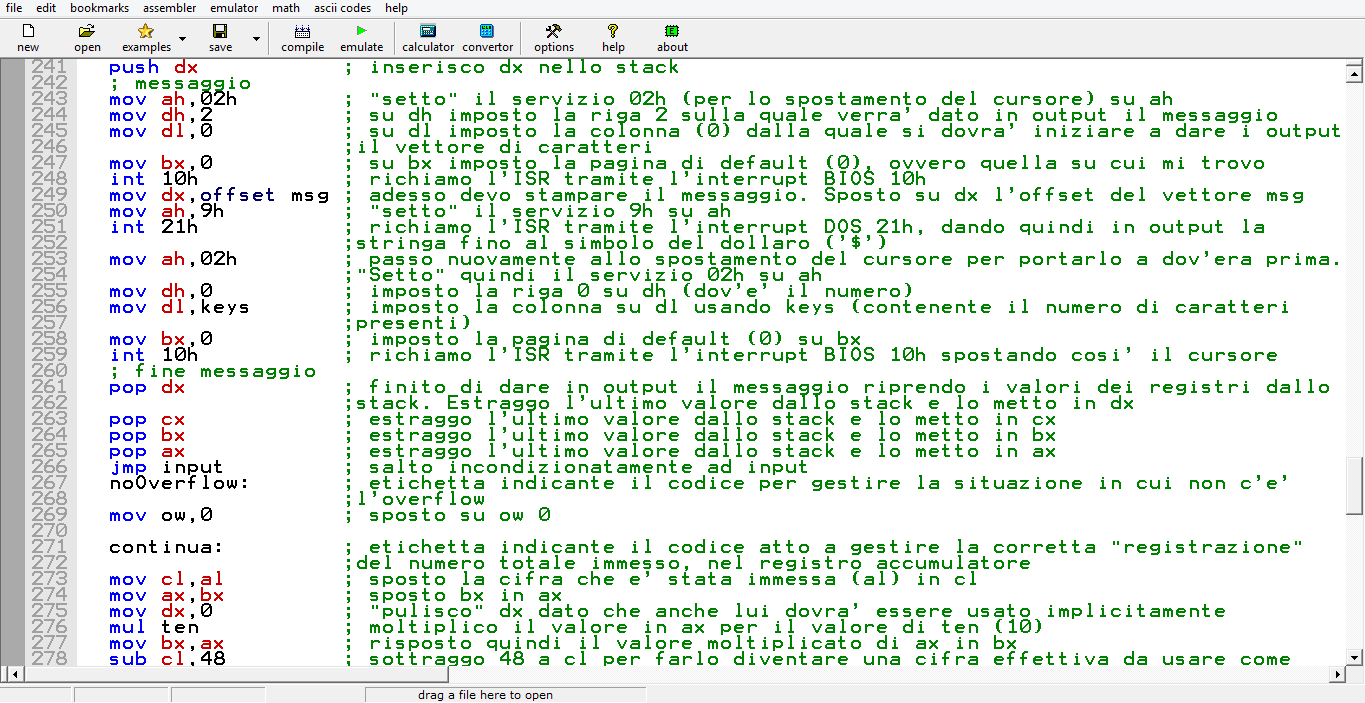


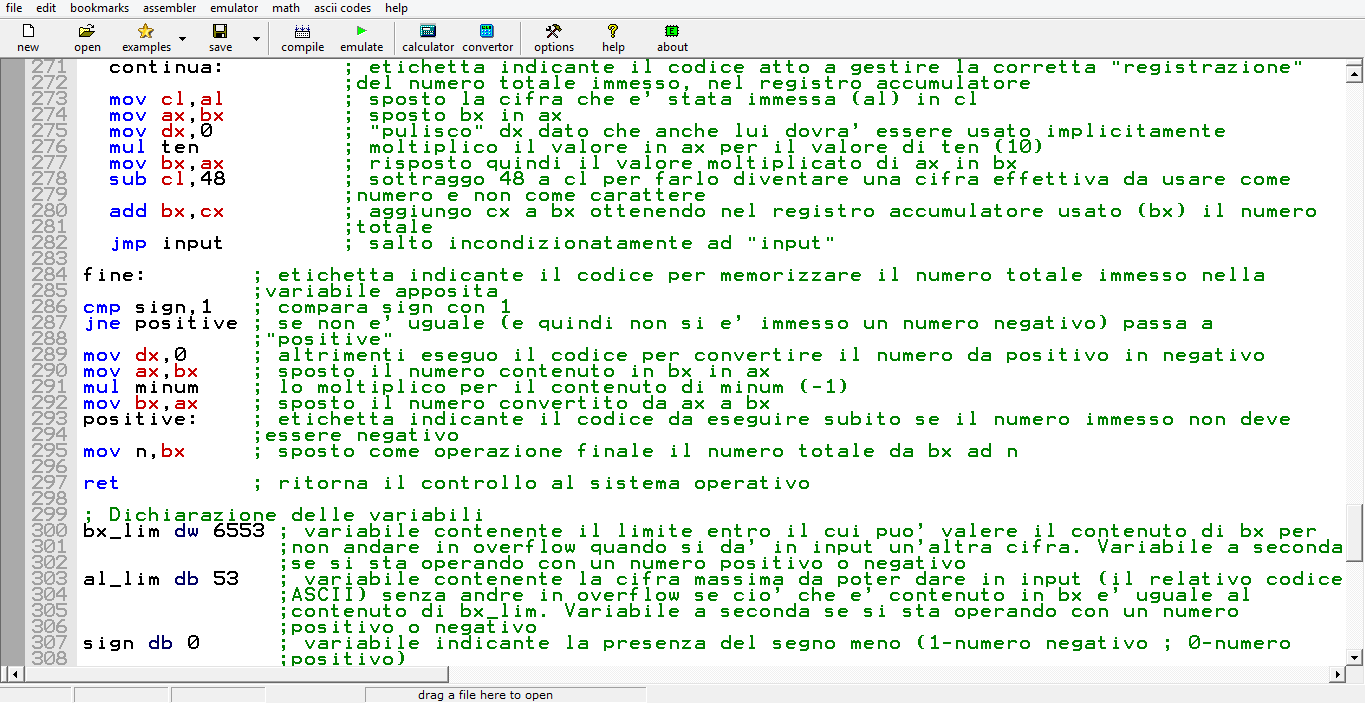
Da “*overflow:*” passo al codice per gestire la situazione di overflow. Prima di tutto setto **ow** ad **1**, poi salvo il numero delle cifre (**digits**) su **aus** (conservo il valore di ax nello stack dato che non posso scambiare direttamente due variabili, per cui devo passare per al), poi do in output il messaggio come spiegato nei paragrafi precedenti e faccio ritornare il cursore alla posizione dopo l’ultima cifra immessa dando alla colonna (dl) il valore di keys (ovvero del numero di caratteri immessi). Infine torno ad “*input*”.





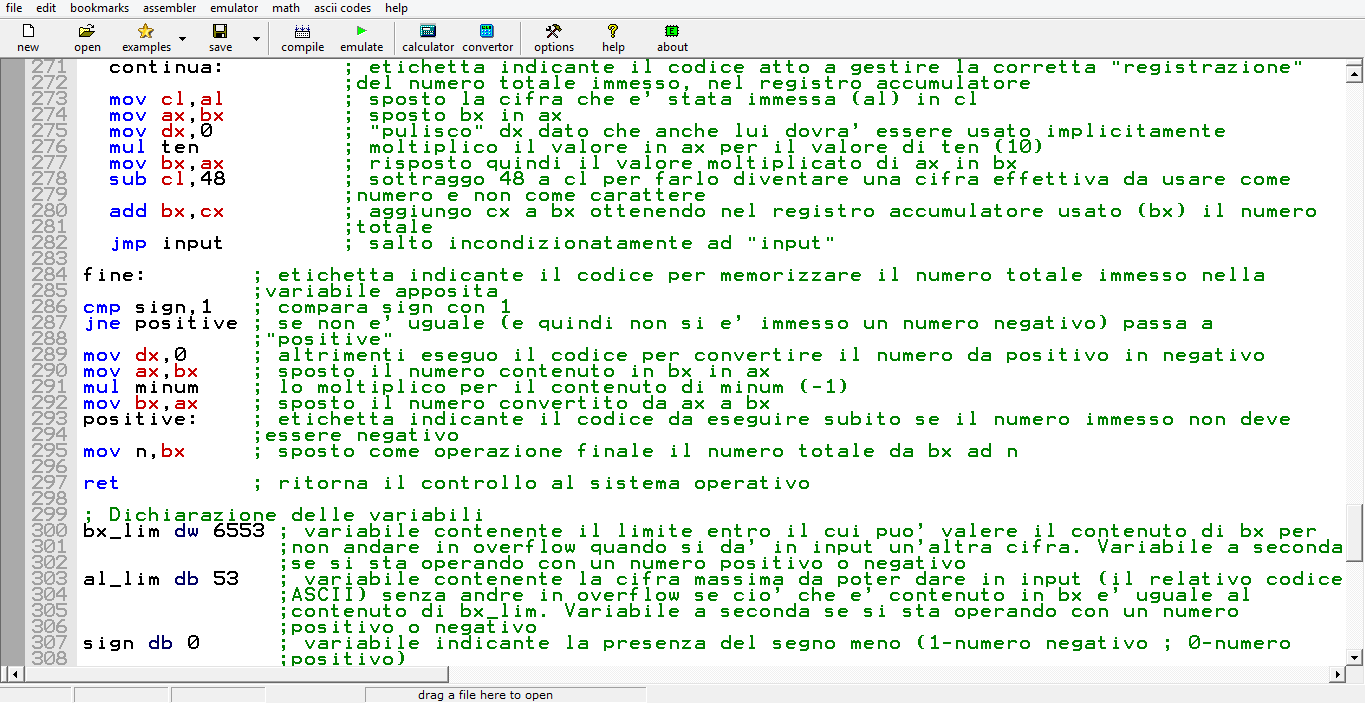
Dall’etichetta “*noOverflow:*” si esegue il codice nel caso in cui non ci sia l’overflow, passando poi direttamente a “*continua:*” da cui si esegue il codice per l’incremento e la memorizzazione corretta del numero totale come spiegato sopra secondo la cifra che l’utente ha digitato. Si sposta quindi **0** su **ow**. Poi si sposta la cifra digitata (in **al**) su **cl**, si sposta il numero totale da **bx** ad **ax** e lo si moltiplica per **10** (**ten**), si risposta quindi il numero totale da **ax** a **bx** e gli si aggiunge la nuova cifra digitata (ora in **cl**) dopo avergli sottratto **48**. Si passa infine ad “*input*”.



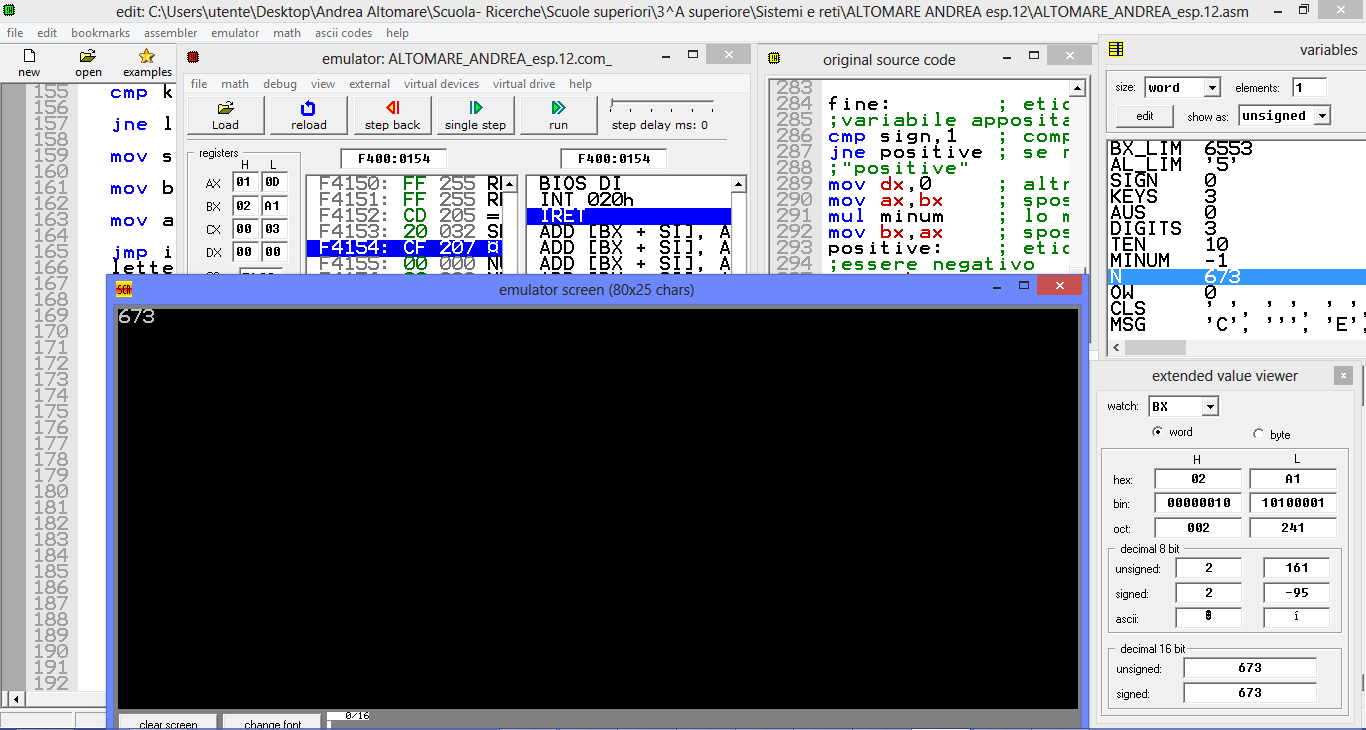


Quando l’utente preme “ENTER” terminerà l’input del numero, passando quindi all’etichetta “*fine:*”, dalla quale verrà eseguito il codice per memorizzare il numero totale digitato, nella variabile apposita.

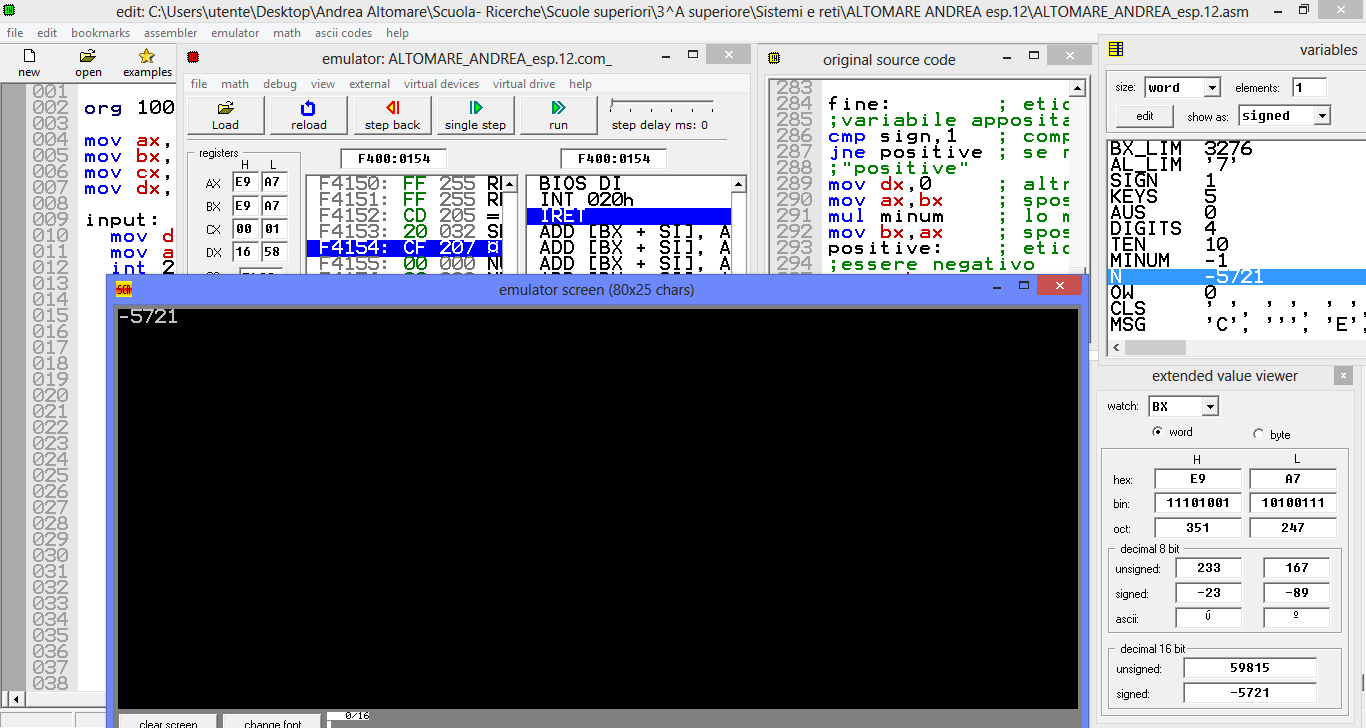
Si confronta **sign** con **1**, se non è uguale vuol dire che l’utente ha immesso un numero **positivo**, per cui passa a “*positive*”, altrimenti il numero deve essere convertito effettivamente in uno **negativo**. Si pulisce dunque il rgistro **dx** il quale viene utilizzato implicitamente nelle moltiplicazioni con numeri a 16 bit, sposto su **ax** il numero totale (contenuto in **bx**) e lo moltiplico per **-1** (**minum**), sposto il numero convertito da **ax** a **bx**, passo quindi a “*positive*”. Dall’etichetta “*positive:*” in poi si ha solo la memorizzazione del numero totale nell’apposita variabile: si sposta il contenuto di **bx** in **n**. Con **ret** il controllo ritorna al sistema operativo.



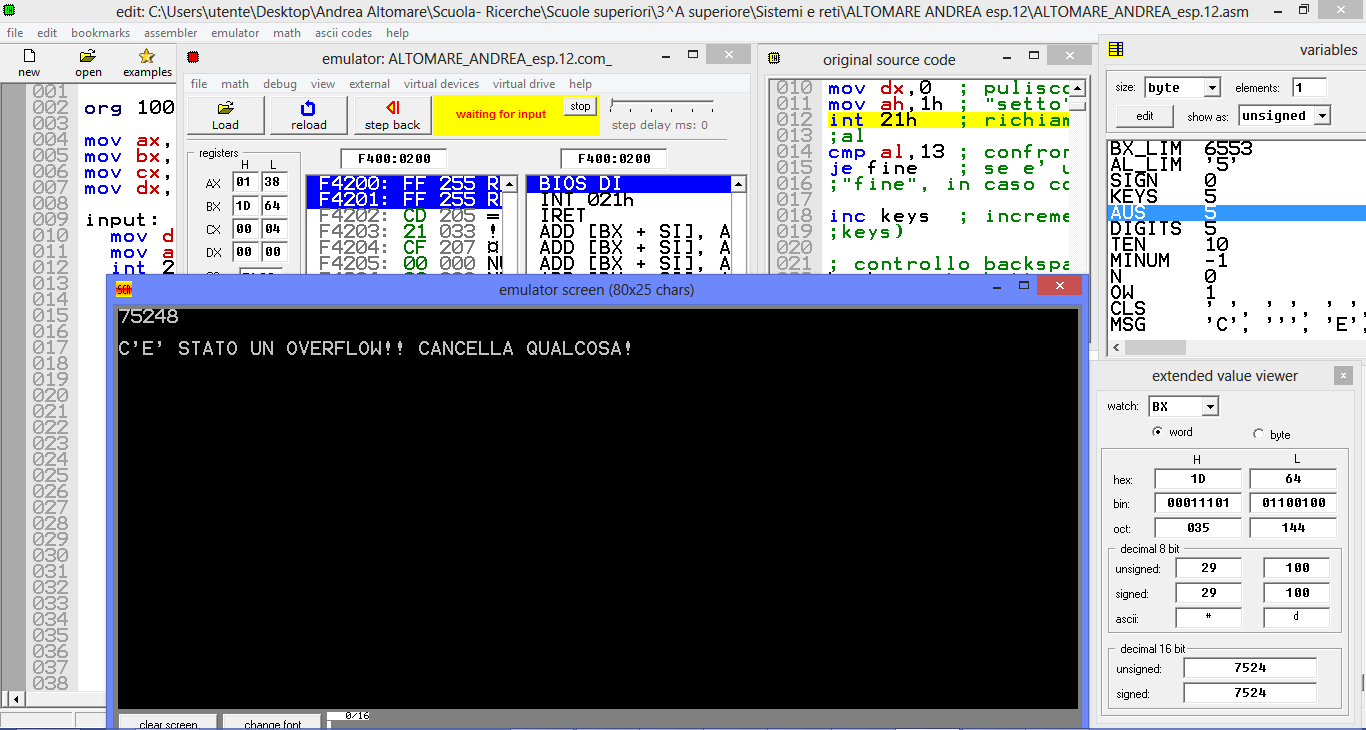
Ho infine verificato il corretto funzionamento del programma controllando l’esito delle operazioni, quando ho voluto memorizzare un **numero senza segno**:



Quando ho voluto memorizzare un **numero negativo**:



Quando sono andato in **overflow** con un **numero senza segno**:



E quando sono andato in **overflow** con un **numero negativo**:

