PROGRAMMARE IN ASSEMBLER SU PROCESSORI X86.

Dopo svariati anni di sperimentazione e apprendimento del linguaggio assembly ho finalmente deciso di pubblicare una pagina web su questo argomento. In modo che tutti gli appassionati e tutti i neofiti della programmazione possano trarne profitto.

Non vi serviranno grosse risorse hardware o software per addentrarvi nei meandri di questo linguaggio a basso livello per eccellenza. Sara sufficiente aprire un finestra dos da windows 9X, Me, NT, 2000 o XP (Funziona tutto perfettamente anche su window 3.x e su tutte le vecchie versioni del dos) e poi digitare un comando che ormai conoscono in pochi: il mitico DUBUG.EXE del dos e quindi premere ENTER. Il gioco è fatto!

Analizziamo in prima istanza i comandi principali di DEBUG:

q : esce

h : somma e sottrae in esadecimale

r : visualizza i registri del processore: AX, BX, CX, DX, ecc...

r ax : modifica il registro ax

e : introduce una funzione od un valore in codice macchina ad un determinato indirizzo.

e 100 : modifica l'indirizzo esadecimale 100h.

t : esegue un istruzione singola. PS: assicurarsi che l'indirizzo IP sia sempre a 100h.

g : esegue più di un'istruzione alla volta e si ferma all'indirizzo specificato (di solito si parte da 100h).

g 102 : si ferma l'esecuzione all'indirizzo 102h.

u : lista un programma partendo da un determinato indirizzo per un determinato numero di linee.

u 100 : visualizza il codice dall'indirizzo 100h in poi.

a : permette di introdurre le funzioni in linguaggio assembler a partire da un determinato indirizzo.

a 100 : parte dall'indirizzo 100h la programmazione in assembler.

n : assegna un nome al file contenete il codice da salvare. (impostare su CX la lunghezza in byte del codice da salvare).

n nome.com : da al nostro codice il nominativo al file eseguibile nome.com

(attenzione però non lo salva!!!).

w : salva il programma.

d : DUMP, stampa della memoria. Si deve specificare l'indirizzo da cui partire per la visualizzazione.

d 200 : visualizza la memoria del calcolatore a partire dall' indirizzo d 200h.

Con questo elenco dovreste avere un valido manuale di riferimento sui principali comandi di DEBUG.EXE.

avete capito come funziona? Spero di si!! Facciamo una prova.

Al prompt digitiamo:

C:\ > DEBUG.EXE ----------> premiamo il tasto 'enter'.

poi ....

appare:

- --------> trattino.

digitiamo ad esempio:

- h 3 2 -------> premiamo il tasto 'enter'.

e quindi ci compare su schermo:

0005 0001

questo è il risultato sia dell'addizione (5) e della sottrazione (1). PS: ricordatevi che sono numeri esadecimali!!!

I numeri trattati da debug sono esadecimali come dicevo. Nel caso di questo esempio FFFFh è il max numero raggiungibile e corrisponde a -1 e non a 65536 in decimale.

tutto ok.

ora digitiamo il tasto 'q' per uscire da DEBUG.EXE:

-q --------> poi premiamo il tasto 'enter'.

ora siamo tornati al prompt dei comandi!!!

c:\>

tutto qui: siamo fuori da DEBUG.EXE.

Vi schematizzo velocemente la differenza tra byte(8 bit) e word(16 bit):

```
   8 bit   +   8 bit
11111111 11111111
 byte        byte
|-----------------------|
  word a 16 bit
```

In poche parole 8 bit fanno 1 byte e 2 byte fanno 1 word, che quindi è a 16 bit!
Come vedremo più avanti un registro è composto da una word perciò è a 16 bit.

FUNZIONI MATEMATICHE:

Proviamo ad impartire delle semplici operazioni matematiche al nostro processore di addizione, sottrazione, moltiplicazione e divisione.

Ovviamente la prima è l'addizione: in linguaggio assembly il comando è ADD (codice macchina 01h d8h)

Inseriamo con il comando 'r ax' il numero 4 ad esempio nel registro ax. Poi con il comando 'r bx' inseriamo il numero 5 ad esempio.

Ora con il comando 'e 100' impartiamo il comando di addizione al calcolatore all'indirizzo 100h e ci verrà visualizzato un simile messaggio:

0f79:0100 00._ --------> inseriamo qui '01' e poi premiamo il tasto di 'spaziatura'

allora avremo:

0f79:0100 00.01 00._ ---------> inseriamo qui 'd8' e poi il tasto 'invio'

Così abbiamo impartito il comando di addizione al calcolatore.

Dobbiamo però farci dare il risultato a questo punto.

Perciò con il comando 't' eseguiamo questa singola istruzione (ammesso che il registro ip sia settato a 100h)

Avremo il numero 0009 come risultato nel registro ax.

Tutto questo procedimento equivale al comando in assembly : 'add ax,bx' (risultato in ax).

Vi ricordo che per continuare con gli esempi è necessario riportare ogni volta il registro ip a 100h in questo modo:

- r ip ---------> premino il tasto enter.

quindi inseriamo:

IP 0102

: 100 --------> premiamo il tasto enter.

quindi digitiamo 'r' e premiamo enter per verificare se il registro ip è a 100h, così comparirà all'incirca questo:

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000 DS=14A0 ES=14A0 SS=14A0 CS=14A0 IP=0100 NV UP EI PL NZ AC PE NC 14A0:0100 29D8 SUB AX,BX

Ora passiamo all' operazione di sottrazione (sub in assembly).

Vi dico subito che tutti i passaggi sono identici all' addizione tranne che per il codice macchina da inserire.

Il codice macchina da inserire per la sottrazione è : 29h d8h.

Corrispondente al comando assembly: sub ax,bx (gli indirizzi coinvolti sono sempre ax e bx con risultato in ax).

Ora passiamo all' operazione di moltiplicazione (mul in assembly).

Vi dico subito che tutti i passaggi sono identici all' addizione tranne per il codice macchina da inserire.

Il codice macchina da inserire per la moltiplicazione è : f7h e3h.

Corrispondente al comando assembly: mul bx (qui sono coinvolti il registro bx che è sempre moltiplicato per ax è il risultato è dato dalla composizione del registro dx con ax (valore a 32bit)).

Ora passiamo all' operazione di divisione (div in assembly).

Vi dico subito che tutti i passaggi sono identici all' addizione tranne per il codice macchina da inserire.

Il codice macchina da inserire per la divisione è : f7h f3h.

Corrispondente al comando assembly: div bx (qui sono coinvolti il registro ax che è sempre diviso per bx, il risultato è nel registro ax con il resto inserito nel registro dx.

Esaminiamo i registri per capire come sono composti.

REGISTRI:

- ax (16 bit) = ah(la 'h' sta per high) a 8 bit + al (la 'l' sta per low) a 8 bit
- bx (16 bit) = bh(la 'h' sta per high) a 8 bit + bl (la 'l' sta per low) a 8 bit
- cx (16 bit) = ch(la 'h' sta per high) a 8 bit + cl (la 'l' sta per low) a 8 bit
- dx (16 bit) = dh(la 'h' sta per high) a 8 bit + dl (la 'l' sta per low) a 8 bit
- e così via .....

Questo schema sui registri servirà in seguito perché a volte sarà necessario manipolare solo una parte del registro.

Fino ad ora vi faccio notare che abbiamo operato in codice macchina che è ancora più a basso livello del linguaggio assembler!!

Vediamo come possiamo fare, in pratica, per stampare su schermo un carattere usando sempre dei codici macchina inseriti nei registri del nostro processore.

Con il comando 'r ax' inseriamo 0200 che sarebbe 02h in ah. ---> diciamo al processore che c'è da stampare un carattere su schermo.

Poi in dl inseriamo 41 ad esempio(da 00h a ffh), che indica il carattere ascii 'A', quindi in dx inseriamo 0041 --- > Con questo passaggio abbiamo scelto il carattere 'A'.

Fatto questo dobbiamo comunicare al nostro microprocessore che stiamo usando una routine legata insieme al sistema operativo e non del bios. Quindi con il comando 'e 100' inseriamo in codice 'cd' e '21'. Il che vale a dire in assembler a 'int 21'. -----> int sta per interrupt.

Di seguito chiudiamo il programma inserendo il codice esadecimanle 'cd' seguito da '20' con il comando 'e 102'.

Ed eseguiamo il tutto con 'g 104'. (Si usa 104 e non 102 perchè si devono racchiudere tutte le istruzioni siccome la 102 viene ancora eseguita mentre la 104 è esclusa!).

Così viene visualizzata la 'A', subito l'interlinea sotto 'g 102' e poi lo stato dei registri.

PS: assicuriamoci sempre che l'indirizzo ip sia uguale a 100h.

BATTIAMO IL TUTTO IN DEBUG:

```
-r ax                    -----> premiamo il tasto 'invio'
AX 0000
:0200                    -----> premiamo il tasto 'invio'
-r dx                    -----> premiamo il tasto 'invio'
DX 0000
:0041                    -----> premiamo il tasto 'invio'
-e 100                   -----> premiamo il tasto 'invio'
```

14A0:0100 15.cd 9C.21 -----> premiamo il tasto di 'spazio' dopo il primo numero e poi 'invio'
-e 102 -----> premiamo il tasto 'invio'
14A0:0102 18.cd 00.20 -----> premiamo il tasto di 'spazio' dopo il primo numero e poi 'invio'
-
-g 104 -----> premiamo il tasto 'invio'
A -----> questo è il risultato.
L'esecuzione del programma è terminata normalmente
-

Il gioco è fatto!! questa è la nostra prima semplice applicazione in codice macchina!

Ora se vogliamo digitare un'equivalente listato in assembly e sufficiente digitare 'a 100' e premere il tasto 'invio'.

Puntualizzo subito il significato in un comando frequentissimo nell' assembler : 'mov x, y' . Questo comando in poche parole assegna un valore a un registro (es: ax) e lo sposta in esso.

PROGRAMMA CHE STAMPA UN CARATTERE IN ASSEMBLY CON L'USO DEL COMANDO 'MOV'.

-a 100 ----> premiamo il tasto 'invio' e digitiamo i seguenti comandi assembly.
14A0:0100 mov ah,02 -----> premiamo il tasto 'invio'
14A0:0102 mov dl,41 -----> premiamo il tasto 'invio'
14A0:0104 int 21 -----> premiamo il tasto 'invio'
14A0:0106 int 20 -----> premiamo il tasto 'invio'

Avrete sicuramente notato che all'inizio della riga compare il numero 0100, 0102, 0104, 0106; bene questi sono gli indirizzi di memoria in cui abbiamo scritto il nostro programmino in assembly. Quindi si parte dall indirizzo 100h e si arriva fino all'indirizzo 106h. Detto questo per eseguirlo bisogna far eseguire le istruzioni fino all'indirizzo 108h perchè vengano eseguite tutte le linee con il comando: 'g 108' e premere il tasto 'enter'.

Come segue:

-g 108 ----> premiamo il tasto 'enter'
A ----> questo è il risultato
L'esecuzione del programma è terminata normalmente

ORA VEDIAMO COME SI FA A SALVARE QUESTO PICCOLO PROGRAMMINO.

Prima di tutto dobbiamo verificare quanti byte il programma occupa in memoria e per vederlo facciamo così:

-u 100 ----> lista il programma dall' indirizzo 100h e ne visualizza la dimensione
14A0:0100 B402 MOV AH,02
14A0:0102 B22A MOV DL,41
14A0:0104 CD21 INT 21
14A0:0106 CD20 INT 20
14A0:0108 0000 ADD [BX+SI],AL ----> questa linea non fa parte del nostro programma ma rappresenta della memoria 'sporca' del calcolatore.

Con questo frammento di listato constatiamo che la memoria è occupata da 100h a 106h: quindi saranno 8 byte perché (100h = 2 byte) + (102h = 2 byte) + (104h = 2 byte) + (106 = 2 byte) così avremo 2+2+2+2 = 8 byte complessivi!

Fatto questo diamo il nome al nostro file, diciamo: 'A.COM', la dimensione in byte e poi salviamo.

-n a.com ----> nome del file e premiamo il tasto 'invio'
-r cx ----> inseriamo la dimensione del programma nel registro cx e premiamo 'invio'
CX 0000
:0008 ----> il registro cx contiene 8 byte e premiamo 'invio'
-r bx ----> inseriamo la dimensione del programma in bx se è molto grosso.
BX 0000
:0 ----> in questo caso inseriamo '0' perche il programma e piccolino
-w ----> con il comando 'w' diciamo a debug di salvare tutto
Scrittura di 00008 byte in corso

Puntualizzo che la dimensione in byte del programma è contenuta nell'unione dei registri bx e cx [bx:cx].

SICURAMENTE LA DOMANDA CHE SORGE SPONTANEA E: "COME CAVOLO SI FA A STAMPARE PIU' CARATTERI INSIEME "?

Per stampare una stringa di caratteri proviamo prima il metodo più macchinoso. Ovvero lavoriamo sui codici macchina e sui codici ascii.

-e 200 ----> premere 'invio'.
14A0:0200 00.48 00.65 00.6c 00.6c 00.6f 00.2c 00.20 00.44 ----> codici ascii dei caratteri: premere 'spazio' ogni codice
14A0:0208 00.4f 00.53 00.20 00.68 00.65 00.72 00.65 00.2e
14A0:0210 00.24 ---> carattere ascii terminatore '$'.

PS: il terminatore non verrà visualizzato, indica solo al computer la fine della frase.

Proviamo a visualizzare la stringa che abbiamo appena composto così come si presenta in memoria:

-d 200 -----> premiamo 'invio'.
14A0:0200 48 65 6C 6C 6F 2C 20 44-4F 53 20 68 65 72 65 2E Hello, DOS here. ---> stringa appena scritta in ram.
14A0:0210 24 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 $.....
14A0:0220 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
14A0:0230 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
14A0:0240 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
14A0:0250 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
14A0:0260 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
14A0:0270 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....

Questo insieme di numeri e caratteri è una porzione di memoria del nostro computer dove abbiamo inserito la stringa di testo. Detto anche dump mirror.

E' il caso di digitare il codice che permette la visualizzazione della stringa perché il programma faccia qualcosa.

-a 100 ---> premere 'invio'
14A0:0100 mov ah,09 ---> questa funzione stampa una stringa di caratteri su schermo
14A0:0102 mov dx,200 ----> carica in dx la stringa introdotta all'indirizzo 200h
14A0:0105 int 21 ---> esegue il tutto con l'ausilio dell' interrupt 21 del dos.
14A0:0107 int 20 ----> termina il programma e restituisce il controllo al sistema operativo.
14A0:0109
-

Per eseguire il programma digitiamo:

-g 109 ---> comando di esecuzione e premiamo 'enter'
Hello, DOS here. ----> risultato: stringa stampata!
L'esecuzione del programma è terminata normalmente ----> fine esecuzione.

PS: ricordarsi di settare l'indirizzo IP a 100h con 'r ip' !!!

Questo programma per essere salvato deve essere inserita la dimensione in cx partendo da 100h fino a 212h perché devono essere compresi anche i byte di memoria dove abbiamo inserito la stringa!

Per venire a conoscenza dei byte basta eseguire il comando 'h 212 100' e il risultato sarà 112 da inserire in cx.

VEDIAMO IN QUESTO NUOVO ESEMPIO COME STAMPARE PIU' STRINGHE USANDO IL COMANDO 'DB'

Quindi digitiamo:

-a 100 ---> introduciamo i comandi e premiamo 'enter' a partire da 100h
14A0:0100 mov ah,09
14A0:0102 mov dx,0200
14A0:0105 int 21 --> stampa la stringa contenuta in 200h
14A0:0107 mov ah,02
14A0:0109 mov dl,0a
14A0:010B int 21 ---> va a capo con il carattere ascii '0a' (equivalente a 10 in decimale)
14A0:010D mov ah,02
14A0:010F mov dl,0d
14A0:0111 int 21 ---> inizio riga con il carattere ascii '0d' (equivalente a 13 in decimale)
14A0:0113 mov ah,09
14A0:0115 mov dx,0230
14A0:0118 int 21 ---> stampa la seconda stringa contenuta in 230h
14A0:011A int 20 ---> restituisce il controllo al sistema operativo.
14A0:011C

quindi aggiungiamo le stringhe ....

-a 200 -----> introduciamo i comandi delle stringhe a 200h
14A0:0200 db 'Prima stringa stampata!!, evviva!!$' ----> con 'db' inseriamo la prima stringa in memoria
14A0:0230 db 'Ecco qua è stampata la seconda stringa!$' ---> inseriamo la seconda stringa a 230h con 'db'
14A0:0258 ---> indirizzo di fine programma per il salvataggio.

Così se proviamo a digitare 'd 200' vedremo le due stringhe in memoria!

-d 200 ----> DUMP della memoria a partire da 200h.
14A0:0200 50 72 69 6D 61 20 73 74-72 69 6E 67 61 20 73 74 Prima stringa st
14A0:0210 61 6D 70 61 74 61 21 21-2C 20 65 76 76 69 76 61 ampata!!, evviva
14A0:0220 21 21 24 20 20 20 20 20-20 20 20 20 20 20 20 20 !!$
14A0:0230 45 63 63 6F 20 71 75 61-20 8A 20 73 74 61 6D 70 Ecco qua . stamp
14A0:0240 61 74 61 20 6C 61 20 73-65 63 6F 6E 64 61 20 73 ata la seconda s
14A0:0250 74 72 69 6E 67 61 21 24-00 00 00 00 00 00 00 00 tringa!$.....
14A0:0260 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
14A0:0270 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....

PS: IL CARATTERE '$' (24 IN CODICE ASCII ESADECIMALE) INDICA IL TERMINATORE (FINE STRINGA)

Ora eseguiamo il programma:

-g 11c ---> esegue (ip deve essere a 100h)

Prima stringa stampata!!, evviva!! --> risultato prima stringa
Ecco qua è stampata la seconda stringa! ---> risultato seconda stringa
L'esecuzione del programma è terminata normalmente

ESAMINIAMO IL COMANDO 'LOOP'

Questo comando ha la stessa funzione del comando 'for.....next' del BASIC perciò ripete il salto per 'n' volte, quante ne sono specificate in 'CX'.

DIGITIAMO:

```
-a 100                          ----> premiamo 'enter'
14A0:0100 mov cx,0003           -----> ripete il ciclo per 3 volte (cx = 3 esa)
14A0:0103 mov dl,02
14A0:0105 mov ah,02
14A0:0107 int 21                ----> stampa un carattere ascii
14A0:0109 loop 0103               ----> ritorna all' indirizzo 103h.
14A0:010B int 20                ----> chiude il programma.
14A0:010D
```

RISULTATO:

```
-g 10d          ----> esegue il programma.
        ----> Stampati tre caratteri.
L'esecuzione del programma è terminata normalmente
```

Più semplice di così!!

DA QUI IN AVANTI VI ESPONGO SOLO PIU' I LISTATI E I COMMENTI PRINCIPALI, ALTRIMENTI QUI NON FINISCO PIU' E POI VA A FINIRE CHE MI PASSA LA VOGLIA DI CONTINUARE QUESTO TUTORIAL!! TANTO ORMAI DOVRESTE AVER CAPITO LE COSE PRINCIPALI.

VEDIAMO LA PROCEDURA 'CALL ind ..... RET' SIMILE A 'GOTO' NEL BASIC.

DIGITIAMO:

-a 100 -----> inseriamo i comandi
14A0:0100 mov dl,41 ------> carattere ascii 'A'
14A0:0102 mov cx,000a ------> ripete 10 volte il ciclo
14A0:0105 call 0200 -----> salta all'indirizzo 200h
14A0:0108 loop 105 -----> ripete l'indirizzo 105h
14A0:010A int 20 -----> termina in programma
14A0:010C
-a 200 ----> inseriamo i comandi dall'indirizzo 200h in avanti
14A0:0200 mov ah,02
14A0:0202 int 21 ----> stampa un carattere
14A0:0204 inc dl ----> incrementa 'dl' di 1
14A0:0206 ret -----> ritorna all'indirizzo 108h subito dopo 'call'
14A0:0207
-g 10c ----> esegue il programma
ABCDEFGHIJ ----> risultato: le prime dieci lettere dell'alfabeto.
L'esecuzione del programma è terminata normalmente

capito tutto!!!

CONDIZIONE VERO FALSO COMANDO 'J...' COME 'IF.....THEN' IN ASSEMBLY.

DIGITIAMO:

```
-A 100                          ----> inseriamo i comandi
14A0:0100 mov ax,0003           ----> valore in ax
14A0:0103 mov bx,0003           ---> valore in bx
14A0:0106 cmp ax,bx             ----> compara i valori presenti in ax e bx
14A0:0108 jz 0113               ----> se ax e bx sono UGUALI
14A0:010A mov ah,09
14A0:010C mov dx,0200
14A0:010F int 21                ----> risposta se ax e bx NON sono uguali
14A0:0111 int 20                -----> termina il programma
14A0:0113 mov ah,09
14A0:0115 mov dx,0230
14A0:0118 int 21                -----> risposta se ax è uguale a bx
14A0:011A int 20
14A0:011C
```

```
-a200                           ----> inseriamo i comandi dall'indirizzo 200h in
avanti
14A0:0200 db'sbagliato$'        ------> stringa se è diverso
14A0:020A
-a230                           -----> inseriamo i comandi dall'indirizzo 200h in
avanti
14A0:0230 db'esatto$'           ----> stringa se è uguale
14A0:0237
```

```
-g 11c                          ----> esatto
esatto                          ----> risultato: esatto perché ax e bx sono uguali
(proviamo a modificare ax con 4)
L'esecuzione del programma è terminata normalmente
```

altre opzioni:

- ja -> salta se ax è maggiore di bx
- jae -> salta se ax è maggiore o uguale a bx
- jb -> salta se ax è minore di bx
- jbe -> salta se ax è minore o uguale a bx
- jnz -> salta se ax è diverso da bx

PS: il comando 'J..' tiene conto del registro 'ZERO-FLAG' se è a '0' o a '1'.

I flag sono indicati con il comando 'r':

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=14A0 ES=14A0 SS=14A0 CS=14A0 IP=0100 NV UP EI PL NZ NA PO NC
14A0:0100 B80300 MOV AX,0003
```

Se noi eseguiamo il programma fino all'indirizzo 11ah in modo da non eseguire il comando int 20 avremo i flag e tutti gl'altri registri non azzerati:

-g 11a
esatto
AX=0924 BX=0003 CX=0000 DX=0230 SP=FFEE BP=0000 SI=0000 DI=0000
DS=14A0 ES=14A0 SS=14A0 CS=14A0 IP=011A NV UP EI PL ZR NA PE NC
[redacted]
14A0:011A CD20 INT 20

Il discorso dei flag è abbastanza complicato a voi vi basti sapere questo e tutto funzionerà benissimo.

COME SI FA A LEGGERE UN CARATTERE DA TASTIERA?!?

digitiamo:

-a 100 --> inseriamo i comandi
14A0:0100 mov ah, 01
14A0:0102 int 21 --> routine di lettura di un singolo carattere da tastiera

ps: il carattere inserito viene memorizzato nel registro AL come codice ascii in formato esadecimale

COME SI FA A LEGGERE UNA STRINGA DA TASTIERA?!?

-a 100 --> inseriamo i comandi
14A0:0100 mov ah,0a ---> codice di lettura stringa
14A0:0102 mov dx,0200 ---> indirizzo in cui viene memorizzata la stringa
14A0:0105 int 21 --> funzione dos

-a 200 --> inseriamo i comandi all'ind. 200h
14A0:0200 db 5,'_ _ _ _ _ _ $' --> spazio riservato alla lunghezza della stringa di 5 byte compreso il tasto di 'INVIO'. Il carattere '$' indica il terminatore. I trattini in basso '_' in realtà sono 6 spazi vuoti per evitare che una stringa troppo lunga cancelli il terminatore.

PROGRAMMA CHE LEGGE UNA STRINGA DA TASTIERA E LA VISUALIZZA SU SCHERMO A CAPO

digitiamo:

-a 100 --> inserisce i comandi
14A0:0100 mov ah,0a
14A0:0102 mov dx,200
14A0:0105 int 21 --> legge la stringa di 5 byte da tastiera e la memorizza all'ind. 200h
14A0:0107 mov ah,02
14A0:0109 mov dl,0a
14A0:010B int 21
14A0:010D mov ah,02
14A0:010F mov dl,0a
14A0:0111 int 21 --> ritorno a capo
14A0:0113 mov ah,09
14A0:0115 mov dx,202
14A0:0118 int 21 --> visualizza la stringa partendo da 202h anziché 200h per evitare i primi 2 byte inutili
14A0:011A int 20 --> uscita dal programma
14A0:011C
-a 200
14A0:0200 db 5,'_ _ _ _ _ $' --> permette una stringa di 5caratteri caratteri compreso l'invio. Necessari 6 byte più il terminatore.
14A0:0207

CREARE UN FILE SU DISCO

Il programma che stiamo per scrivere crea un file su disco completamente vuoto assegnando solamente il nome il drive e se il file è in formato normale, di sola lettura, non visibile, ecc....

digitiamo:

-a 100 ---> inseriamo i comandi
14A0:0100 mov ah,3c --> crea il file
14A0:0102 mov cx,0 --> determina il tipo di file: normale
14A0:0105 mov dx,200 --> indirizzo in cui va a cercare il nome e il drive del file
14A0:0108 int 21
14A0:010A int 20
14A0:010C
-a 200
14A0:0200 db 'c:\dati.dat',0 --> stringa del nome file. lo '0' indica la fine della stringa
14A0:020B

Questo prog. scrive un file vuoto in c:\ con il nome di dati.dat di tipo normale

Tipo file determinato da 'mov cx,X' :

- 0h --> normale
- 1h --> sola lettura
- 2h --> non visibile
- 4h --> file di sistema
- 8h --> Etichetta del disco
- 10h --> sotto directory
- 20h --> file di archivio

SALVARE DEI DATI SU DI UN FILE

Vi ricordo che il file deve essere stato creato in precedenza su disco con il listato descritto poco sopra ad esempio. Altrimenti non funziona nulla!

digitiamo:

```
-a 100                          --> inseriamo i comandi
14A0:0100 mov ah,3d             --> apre il file
14A0:0102 mov al,01             --> file di sola scrittura
14A0:0104 mov dx,200            --> nome del file
14A0:0107 int 21
14A0:0109 mov di,ax             --> salva di 'di' il descrittore del file
14A0:010B mov ah,42
14A0:010D mov al,02
14A0:010F mov bx,di
14A0:0111 mov cx,0000
14A0:0114 mov dx,0000
14A0:0117 int 21                --> si posiziona alla fine del file per evitare che
vengano sovrascritti i dati precedenti
14A0:0119 mov ah,40             --> scrive il file
14A0:011B mov bx,di             --> mette la descrizione in bx
14A0:011D mov cx,80             --> numero max di byte da salvare è 80.
14A0:0120 mov dx,300            --> caratteri da scrivere all' ind. 300h.
14A0:0123 int 21
14A0:0125 mov ah,3e             --> chiusura file
14A0:0127 mov bx,di             --> mette la descrizione in bx
14A0:0129 int 21
14A0:012B int 20                --> ritorno al dos
```

```
-a 200
14A0:0200 db 'c:\dati.dat',0    --> percorso e nome file.
14A0:020C
-a 300
14A0:0300 db 'ciao belli!'      --> stringa da salvare (in questo caso non deve
superare gli 80 byte)
```

All'ind. 102h il comando 'mov al,01' indica che il file è di sola scrittura ma possono esserci anche altri parametri:

- al=0 --> sola lettura
- al=1 --> sola scrittura
- al=2 --> sia lettura che scrittura
- Oltre a questi parametri esistono anche quelli per la condivisione di rete dalla ver. dos 3.0 in avanti, a cui non faro riferimento.

LETTURA DI UN FILE DA DISCO

Digitiamo:

```
-a 100
14A0:0100 mov al,00            --> file di sola lettura
14A0:0102 mov ah,3d            --> parametro per l'apertura
14A0:0104 mov dx,200           --> indirizzo dove c'è il percorso e il nome del file
14A0:0107 int 21
14A0:0109 mov di,ax            --> memorizza il descrittore del file in di
14A0:010B mov ah,3f            --> parametro per la lettura
14A0:010D mov bx,di            --> mette il descrittore del file in bx
14A0:010F mov cx,0080          --> numero di byte dal leggere
14A0:0112 mov dx,0300          --> indirizzo dove viene memorizzato il contenuto
del file
14A0:0115 int 21
14A0:0117 mov ah,3e            --> parametro per la chiusura del file
14A0:0119 mov bx,di            --> descrittore del file in bx
14A0:011B int 21
14A0:011D int 20
14A0:011F
-a 200
14A0:0200 db 'c:\dati.dat',0       --> percorso e nome del file
14A0:020C
```

Il risultato è visualizzabile all' indirizzo 300h con il comando 'd 300'

Così:

```
-d 300
14A0:0300 63 69 61 6F 20 62 65 6C-6C 69 21 00 00 00 00 00 ciao belli!.....
14A0:0310 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
14A0:0320 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
14A0:0330 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
14A0:0340 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
14A0:0350 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
14A0:0360 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
14A0:0370 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

PS: IL DESCRITTORE RAPPRESENTA LA TIPOLOGIA DI FILE E IN QUESTI ESEMPI VIENE MEMORIZZATO NEL REGISTRO 'DI'. SI UTILIZZA IN DIVERSI MOMENTI DELLA PROCEDURA DI SALVATAGGIO DATI E ANCHE IN QUELLA DI LETTURA DATI.

L'ULTIMA COSA SUI FILE E' CHE:

SALVATAGGIO:

- ESISTE LA CREAZIONE DEL FILE.
- ESISTE L'APERTURA DEL FILE.
- ESISTE IL SALVATAGGIO DEI DATI SUL FILE.

- ED ESISTE LA CHIUSURA DEL FILE.

LETTURA:

- ESISTE L'APERTURA DEL FILE ESISTENTE.
- ESISTE LA LETTURA DEL FILE DOPO CHE E' STATO APERTO.
- ED ESISTE LA CHIUSURA DEL FILE.

PROGRAMMA CHE CANCELLA LO SCHERMO IN MODALITA' TESTO

Digitiamo:

```
1673:0100 B90000 MOV CX,0000        --> 'ch = riga in alto a sinistra' + 'cl= colonna in alto a destra'
1673:0103 BA4F18 MOV DX,184F        --> 'dh = riga in basso a destra' + 'dl = colonna in basso a destra'
1673:0106 B707 MOV BH,07            --> attributo di finestra normale
1673:0108 B406 MOV AH,06            --> aggiornamento finestra verso l'alto
1673:010A B000 MOV AL,00            --> numero di linee (00 = intero schermo)
1673:010C CD10 INT 10               --> interruzione bios per lo schermo
1673:010E CD20 INT 20               --> uscita al dos.
```

Questo programma imposta il cursore per saltare tutte le colonne e le linee dello schermo, ripulendo così tutto quanto dal vostro monitor.

In questo caso i registri 'cx' e 'dx' possono essere inseriti prima della procedura per visualizzare un carattere determinandone così la posizione.

ARRIVATI A QUESTO PUNTO ABBIAMO LA CONOSCENZA PER CREARE PRATICAMENTE QUALUNQUE TIPO DI PROGRAMMA TESTUALE ESISTENTE. QUELLO CHE CI MANCA E' LA GESTIONE DELLA GRAFICA IN ASSEMBLER. BENE VEDIAMO COME PROVVEDERE!

GESTIONE DELLA MODALITA' GRAFICA TRAMITE LE FUNZIONI BIOS CON L'INTERRUZIONE 10h.

Digitiamo un programma che inizializza la modalità grafica:

```
-a 100                          --> inseriamo i comandi
14A0:0100 mov ah,0               -->   modalità di visualizzazione
14A0:0102 mov al,13              -->    modalità grafica VGA 320X200 pixel per 256 colori
14A0:0104 int 10                 --> interruzione grafica del bios
14A0:0106 int 20                 -->  ritorno al sistema operativo
14A0:0108
```

Forma più breve:

```
-a 100
14A0:0100 mov ax,0013     -->   unione di 'ah+al'
14A0:0103 int 10          -->   funzione bios
14A0:0105 int 20          -->   uscita
14A0:0107
```

Altri parametri da assegnare in 'al':

- 0 --> 40X25 bn
- 1 --> 40X25 colore
- 2 --> 80X25 bn
- 3 --> 80X25 colore
- 4 --> 320X200 cga colore
- 6 --> 640X200 cga bn
- 10 --> 640X350 ega
- 13 --> 320X200 vga 256 colori

VEDIAMO ORA COME VISUALIZZARE UN PIXEL COLORATO SU SCHERMO.

Digitiamo:

```
-a 100                          --> digitiamo
14A0:0100 mov ax,0013            --> inizializza la modalità vga
14A0:0103 int 10                 --> interruzione grafica
14A0:0105 mov ah,0c              --> permette di visualizzare uno o più pixel
14A0:0107 mov al,01              --> colore del pixel
14A0:0109 mov cx,a0              --> coordinata X
14A0:010C mov dx,54              --> coordinata Y
14A0:010F mov bx,1               --> numero di pagina
14A0:0112 int 10                 --> interruzione grafica
14A0:0114 int 20                 --> ritorno del controllo al sistema operativo
14A0:0116
-
```

Il risultato è la visualizzazione di un pixel blu sullo schermo.

AUTORE:

Dott. Bartolomeo Davide Bertinetto

www.bertinettobartolomeodavide.it

DONAZIONE PER LA SOPRAVVIVENZA DEL SITO INTERNET E LA DIFFUSIONE DEI SUOI CONTENUTI

Questo libro racchiude parte della mia conoscenza sull'assembler. Inoltre la produzione di queste pagine ha richiesto molto tempo di lavoro e passione molto intensi, per dare luogo al mio ennesimo contributo alla programmazione!

Pertanto se ritenete che il mio lavoro possa essere stato utile e in qualche modo ha cambiato la vostra vita, sarò felice di ricevere una donazione sul mio sito(www.bertinettobartolomeodavide.it), tramite l'apposito link verso PayPal. Per donare è sufficiente una qualsiasi carta di credito ricaricabile o un conto PayPal. L'importo è completamente libero e pertinente alla vostra crescita culturale nella programmazione. Ovviamente nessuno è obbligato a donare, fatelo solo se lo ritenete giusto... Grazie infinite in ogni caso!

Bartolomeo Davide Bertinetto.

Diplomato ISEF e laureato in Scienze Motorie con Specializzazione Tecnico-Sportivo presso l'Università di Torino. Proprietario di un centro fitness...

Da sempre appassionato di computer in vari settori: Programmazione in Assembler, C/C++, Visual C+, Gwbasic, Pascal e JAVA. Creazione di circuiti elettronici e programmazione di microprocessori PIC 16F84. Ancora attivo programmatore del vecchio Commodore 64.

Esperto di Lightwave 3D per la realizzazione di immagini e animazioni 3D sintetiche, con partecipazione ad alcune edizioni del PIXEL ART di Roma.

WebMaster del sito 'www.bertinettobartolomeodavide.it', dedicato alla creazione di videogiochi 2D, che conta migliaia di visite...