i trucchi dell' MS-DOS

a cura di Pierluigi Panunzi

quarta parte

I comandi esterni

Ci siamo lasciati la scorsa puntata parlando del programma «exe2bin», che in generale, per un utente «medio» del sistema operativo MS-DOS risulta un programma alquanto strano dal momento che, anche leggendo le relative spiegazioni sul manuale, non se ne riesce subito a comprendere bene il funzionamento e prima ancora l'utilità. In effetti quanto detto nel manuale risulta ben chiaro, ovviamente, solo agli addetti ai lavori, i quali conoscono perfettamente le differenze esistenti tra un programma di tipo «.COM» ed uno di tipo «.EXE»: non crediamo di sbagliare stimando ad occhio e croce nel 50% la percentuale dei cosiddetti «addetti ai lavori» rispetto agli utenti per l'appunto «medi», ai quali interessa e serve (soprattutto!) un computer funzionante con programmi funzionanti.

Questi ultimi magari sanno a malapena che esistono file di tipo «.COM» o «.EXE» e non è nemmeno detto che anche fra gli addetti ai lavori non nascano ogni tanto dei dubbi...

Ecco che dunque in questa puntata e nella successiva, prima di parlare del comando esterno «exe2bin», parleremo innanzitutto di questa non ancora ben chiara differenza tra i due tipi di file già più volte citati.

«.COM» o «.EXE»? Questo è il problema...

Dopo aver parlato per parecchie puntate di comandi interni ed esterni, ci soffermiamo un poco su di un argomento un po' più teorico e nettamente «meno da provare» sul computer: finora analizzando i comandi dell'MS-DOS in un certo senso si invitava il lettore interessato (ripetiamo «il» lettore, altrocché i venticinque lettori «manzoniani»...) a provare subito sul suo computer quanto stava leggendo.

Questa volta no, anzi il «povero» lettore deve semplicemente leggere ed elucubrare quanto diremo: meglio ancora se tiene sott'occhio le puntate iniziali della rubrica riguardante l'«Assembler 8086/88» dove troverà i dettagli di quanto qui diremo, dovendo per forza di cose trattare argomenti tipo la «segmentazione» ed i «registri di segmento» (segment register) del microprocessore 8088.

Tra parentesi la rubrica citata e la presente sono strettamente legate (n.d.r. a parte il fatto che sono redatte dalla stessa persona...) in quanto il sistema operativo MS-DOS gira in ambiente 8088 e proprio da questa puntata se ne saggerà la parentela: con questo il nostro amico lettore è avvisato.

Per poter approfondire meglio alcuni dei concetti espressi, abbiamo analizzato alcuni programmi scelti a caso (di tipo «.COM» e «.EXE»), avvalendoci dell'ausilio di due programmi molto utili: il primo, «debug.com», presente in genere nel disco di sistema, è servito come analizzatore «dinamico», dal punto di vista strettamente legato all'esecuzione, mentre il secondo è il potente programma «pctools.exe» della Central Point Software, che ci è servito come analizzatore «statico», per poter vedere i file dal punto di vista «fisico».

Con «debug» infatti possiamo vedere come funziona il programma in esame, disassemblandolo, visualizzando zone di memoria ed anche eseguendolo passo passo (ma fino ad un certo punto...); invece con il «pctools» (o un qualsiasi altro programma che permetta di leggere un file settore dopo settore) possiamo invece vedere tutti i byte (nessuno escluso) che compongono il file stesso.

Detto questo, iniziamo dunque dai file di tipo «.COM» in quanto più semplici da trattare.

Un file di tipo «.COM», dal punto di vista fisico, è esattamente l'immagine della zona di memoria in cui viene caricato per poi essere eseguito: in particolare si assume (per un vecchio retaggio proveniente dal CP/M) che l'indirizzo iniziale di caricamento e di partenza dell'esecuzione del programma stesso sia pari a 100H.

Dal momento che siamo in ambiente 8086/88, completiamo subito questa affermazione per renderla più rigorosa: il sistema operativo MS-DOS, scelto un segmento in cui allocare il programma in esame, caricherà il primo byte del file nella cella il cui offset è 0100H, il secondo byte nella cella di offset 0101H e così via, per tutta la lunghezza del file.

E qui c'è subito una limitazione importantissima: la lunghezza del file non può superare in alcun modo i 64k, come dire che il file deve poter essere contenuto all'interno del segmento prescelto dal sistema operativo: ricordiamo che stiamo parlando di segmento di codice, nel quale sono contenute le istruzioni del programma e le «costanti del programma» quali costanti numeriche vere e proprie, i messaggi, le tabelle, ecc., ma non i dati e le varia-

bili sui quali opera, che in genere si trovano nel «data segment».

Questo fatto si ritrova da parecchie parti tra i vari programmi che girano sotto MS-DOS: ad esempio il ben noto e potente Turbo Pascal è un file di tipo «.COM» e come prodotto di una compilazione di un programma scritto in Pascal permette di generare un file di codice eseguibile lungo al massimo 64k, tanto è vero che per ovviare a questo inconveniente è prevista la possibilità di generare «overlay», da richiamare ed eseguire solo nel corso dell'elaborazione.

Un altro esempio è l'arcinoto WordStar, il cui programma principale (il file «ws.com») è di lunghezza pari a 25600 byte, ai quali si aggiungono i 34k e i 46k dei due overlay più importanti che lo accompagnano.

Questo fatto dell'indirizzo iniziale pari a 100H e della lunghezza limitata a 64k in un certo senso non è di grande scomodità per il programmatore «medio», che ben difficilmente raggiungerà o peggio supererà la barriera dei fatidici 64k, che sono pur sempre tanti...

E poi ci penserà il compilatore di turno ad avvertire il programmatore.

Non a caso abbiamo citato l'esempio dell'ottimo Turbo Pascal il quale per sua esplicita natura genera file di tipo «.COM»: ciò è spiegabile in parte con la considerazione che il compilatore in esame genera innanzitutto «in memoria» una versione eseguibile del programma che stiamo compilando, versione che in seguito può essere trasferita su disco, con una semplice operazione di scrittura su file del contenuto del segmento di programma stesso.

In tal modo è proprio il codice che si trovava nel segmento di programma ad essere trasferito su disco.

Diversa è invece la tecnica nel caso più generale di compilatori di linguaggi ad alto livello, tra i quali i più famosi sono quelli della casa madre Microsoft (il macro-assembler «MASM.EXE», il compilatore Pascal «PAS.EXE», il compilatore per il Basic «BASCOM.EXE», il compilatore Fortran «FOR.EXE», ecc.), i quali senza distinzione producono file di tipo «.OBJ» che possono essere poi linkati assieme per mezzo del «LINK»: in tal modo il file ottenuto, che può essere formato da parti scritte nei più svariati linguaggi, è viceversa un file di tipo «.EXE» e perciò più generale che non un «.COM».

Senza scendere troppo nello specifico, questo accade fondamentalmente perché i moduli scritti in vari linguaggi risiedono ognuno in un segmento differente e vengono poi appunto «collegati» dal linker solo al termine. Eppoi ogni modulo è dotato del proprio segmento di dati, nonché di uno stack...

Come si vede la situazione è completamente differente che non nel caso del Turbo Pascal: infatti qui per precisa scelta dei progettisti della Borland possiamo generare un unico programma eseguibile e perciò non possiamo generare più file da linkare successivamente.

In definitiva un file di tipo «.COM» è un file completamente rilocabile nel senso che può essere caricato così com'è e ovviamente girare un qualunque segmento, senza dover essere in alcun modo alterato prima dell'esecuzione (e questa affermazione, alquanto strana, troverà la sua spiegazione fra breve): sarà sempre l'MS-DOS a decidere in quale segmento allocare il nostro file e ciò dipenderà innanzitutto dalla versione del sistema operativo stesso e poi dalla presenza di uno o più moduli «residenti» quali «command.com» aggiuntivi, spooler di stampa (vedasi la puntata precedente riguardante il comando «print»), programmi quali «SideKick», «pctools» ecc...

Detto dunque questo, vediamo ora un po' più da vicino in cosa differiscono i file di tipo «.EXE» da quelli appena visti.

I file di questo secondo tipo sono in un certo senso più generali in quanto non risentono di alcuna limitazione di ampiezza (prova ne sono i vari «pacchetti integrati», il già più volte citato «pctools» nonché gli stessi compilatori), potendo abbattere più volte barriere di 64k e poi possono, anzi devono, avere un proprio tack posto in un ben preciso «stack segment» ed inoltre non devono iniziare per forza all'indirizzo 100H, ma hanno ognuno un proprio «start address»: possono sia partire dal primo byte utile come pure dal millesimo byte, laddove, si badi bene, i primi novecentonovantanove byte, sono utilizzati come dati.

Tutto questo però si paga, anche se in maniera trasparente e cioè invisibile, a livello di memorizzazione sotto forma di file, che in questo caso non è la fedele immagine della zona di memoria in cui il programma viene poi caricato, ma contiene una parte iniziale dedicata all'indicazione di tutti i fatti salienti che il sistema operativo deve conoscere all'atto del caricamento prima, ed all'esecuzione poi.

In un certo senso un file di tipo «.EXE» possiede in testa una specie di «carta di identità» che serve all'MS-DOS per caricarlo e farlo eseguire correttamente: i file di tipo «.COM» invece non hanno tale carta di identità in quanto tutti rigorosamente «somiglianti» (partono tutti da 100H e sono al massimo di 64k).

Per capire ancora meglio quali e quante sono le differenze tra i due tipi di file pensiamo ad un programma generato dal più volte citato Turbo Pascal ed uno generato da quel potente e completo linguaggio che è l'Assembler vero e proprio.

Sappiamo infatti che in Assembler tutto è concesso o almeno ciò che non è concesso, beh..., c'è sempre il modo più o meno lecito per ottenerlo: come dire che in Assembler si possono scrivere programmi in cui l'esecuzione passa ad esempio da un segmento all'altro senza alcuna limitazione di sorta.

Tornando al Turbo Pascal, invece, il prodotto finale è un innocuo «.COM», che a meno di necessità impellenti non può essere immediatamente convertito in un file di tipo «.EXE»: basterebbe a tale scopo scrivere il programma «bin2exe», ammesso che ciò sia di pratica utilità.

Dopo aver dunque visto la «necessità» dell'esistenza dei file di tipo «.EXE», continuiamo dunque ad analizzarli in dettaglio dal punto di vista «fisico».

Innanzitutto abbiamo parlato della «carta di identità» del file: in termine tecnico viene detto «header» (testata) e consiste in una tabella la cui parte iniziale è strutturata secondo una serie di word che analizzeremo subito e la cui parte finale è di lunghezza variabile in funzione delle esigenze del programma.

Subito dopo l'header (che per regola ha un'ampiezza che varia secondo multipli di 512 byte) si trova il programma vero e proprio e cioè l'immagine della zona di memoria in cui il programma deve essere caricato e poi eseguito: come dire che un file di tipo «.EXE» è una specie di «.COM» preceduto da un header che serve per la sua caratterizzazione.

Analizziamo dunque la struttura dell'header, la cui parte «fissa» è formata da 14 word, seguita da una parte a lunghezza variabile: sintetizziamo il tutto nella tabella A.

Analizziamo ora, anche se in ordine sparso, il significato di ognuna delle voci della tabella, ognuna delle quali ha una sua «unità di misura», soffermandoci su di ognuna per le considerazioni del caso:

Word n. 1- non è altro che un identificatore di file «.HEX» valido: se il sistema operativo trova che i primi due byte di un file sono proprio 4DH e 5AH (le lettere «MZ», se pensate come codifica ASCII, il che non ci aiuta minimamente) allora, a prescindere dalla sua estensione, lo gestirà sia per il caricamento che poi per l'esecuzione secondo le modalità che vedremo più avanti.

Tanto è vero che se rinominiamo un

Tabella A

HEADER DI UN FILE DI TIPO ".HEX"		
word	significato	unita'
1	valore 5A4DH (esadecimale)	-
2	lunghezza file modulo 512	byte
3	(lunghezza file + header) / 512	page
4	numero "relocation table items"	-
5	ampiezza dell'header	paragr.
6	minima area alla fine	paragr.
7	massima area alla fine	paragr.
8	variazione per SS (Stack Segment)	paragr.
9	offset di SP (Stack Pointer)	byte
10	checksum del file	-
11	offset di IP (Instruction Pointer)	byte
12	variazione per CS (Code Segment)	paragr.
13	puntatore primo "item"	byte
14	numero di overlay	-
...		
i	offset dell'item	byte
i + 1	segment dell'item	paragr.
...		

qualsiasi file di tipo «.EXE» con l'estensione «.COM» il programma gira lo stesso, a riprova che è proprio la prima word che identifica un «.EXE»; tra l'altro è vero anche il viceversa: un «.COM» rinominato come «.EXE» funziona ancora come «.COM».

Sulla scelta dei due valori citati aggiungiamo alcune considerazioni: i byte 4DH 5AH potrebbero essere interpretati come codici operativi di un programma scritto in Assembler e cioè di un programma che inizia con

```
DEC  BP
POP  BX
```

sul cui significato rimandiamo alla rubrica sull'Assembler.

Tutto questo per dire che un programma in Assembler, che vogliamo sia un «.COM», ma che inizi diabolicamente con le due istruzioni citate verrebbe comunque interpretato come un file di tipo «.EXE» ed a seconda di quanto presente successivamente si avrà una delle tante possibili segnalazioni di errore proprie del sistema operativo.

C'è da dire che in realtà è ben raro iniziare un programma in Assembler con le due istruzioni date: non si capisce chi avrebbe interesse a decrementare subito il registro BP e poi fare la POP senza avere prima effettuato una PUSH con il rischio di stravolgere il sistema operativo dal momento che eliminiamo subito un livello di stack.

Comunque nulla ci impedisce di iniziare un programma con

```
DEC   BP
POP   BX
PUSH  BX
INC   BP
```

con il che abbiamo più che diabolicamente ripristinato il tutto...

Word 2,3 - rappresentano la lunghezza del file espressa in una combinazione tra byte e blocchi di 512 byte: in particolare la word numero 3 rappresenta l'ampiezza del file, compreso l'header, espressa in blocchi di 512 byte, mentre la word n. 2 non è altro che il resto della divisione tra la lunghezza totale del file in byte e 512. Così se un file è lungo 10000 byte (header compreso), allora nella word n. 2 troveremo 272 (il resto di 10000/512), mentre nella word n. 3 si avrà 19.

Queste informazioni servono all'MS-DOS per allocare un numero opportuno di segmenti in cui caricare successivamente il programma.

Word n. 6 - serve a sapere quant'è lungo l'header, con un valore espresso in paragrafi (che ricordiamo essere pari a 16 byte).

È un'informazione che serve al sistema operativo per il calcolo dell'effettiva ampiezza del programma da caricare (vedasi il punto precedente) ed inoltre per indicare il punto in cui, all'interno del file, inizia il programma vero e proprio.

Abbiamo già detto che l'header può avere un'ampiezza pari ad un multiplo di 512 byte ed in generale è proprio ampio 512 byte, con il che il valore della word 6 è generalmente 0020H (32 in esadecimale), espresso come detto in paragrafi.

Word n. 10 - è la cosiddetta «checksum» del file: si tratta di un valore che serve a vedere se nel caricamento da disco si sono verificati degli errori che hanno alterato il valore di uno o più byte.

È ottenuta effettuando la somma, word a word, di tutto il contenuto del file, ignorando volta per volta eventuali overflow (trabocchi) ed alla fine complementando a 2 la somma così ottenuta: in tal modo il sistema operativo, mano a mano che legge da disco una word, ne somma il valore alla somma parziale fin lì ottenuta. Alla fine del caricamento sommerà al risultato totale proprio il valore della checksum (che ha letto all'inizio dall'header, appunto) e dovrà ottenere, escludendo l'overflow, un valore nullo: un qualsiasi altro valore indicherà un errore nel caricamento.

Tutto questo in teoria, in quanto anche andando a variare uno o più byte all'interno di un certo file, questo gira come se niente fosse accaduto: quindi possiamo dire che l'uso di questa word è decaduto nel tempo, il che a pensarci bene è anche giusto, visto l'alto grado di affidabilità raggiunto dai supporti magnetici di massa.

Word n. 14 - si tratta del numero di overlay del programma in esame ed è pari a 0 nel caso che il file rappresenti la parte principale (quella residente) del programma eventualmente dotato di overlay. Questa word non ha un significato granché utile per le nostre analisi, perciò passiamo alla successiva.

Word n. 6,7 - sono due word rappresentanti altrettanti valori, espressi in paragrafi, relativi a due quantità non direttamente verificabili dall'utente: in definitiva si tratta rispettivamente del numero minimo e del numero massimo di paragrafi di cui il programma necessita e vengono considerati alla fine fisica del programma stesso. Rappresentano dunque la minima e la massima quantità di memoria di cui il programma ha bisogno a partire dalla sua «fine» fisica, dove poter allocare variabili, vettori, ecc. a seconda delle necessità: grazie a questi valori il sistema operativo allocherà per il file in esame la quantità di memoria richiesta, che in un certo senso rimane di proprietà del programma stesso.

All'inizio abbiamo detto che i valori non sono direttamente verificabili: infatti mentre per la word n. 6, nei vari esempi verificati, ci possono essere valori più disparati (dipendenti da moltissimi fattori e calcolati dai programmi che a loro volta hanno generato il file in esame), per la word n. 7 abbiamo sempre trovato il valore FFFFH, come dire che al massimo un file richiede 65535 paragrafi pari ad 1 Mbyte di ram.

Word n. 8,9 - si tratta di due valori molto importanti e servono a dare le necessarie informazioni riguardanti lo stack al sistema operativo, subito prima dell'esecuzione: in particolare la

word n. 8 rappresenta di quanto deve essere alterato il valore proposto dal sistema operativo per lo «Stack Segment» (SS), per ottenere quello invece richiesto dal programma per poter funzionare correttamente.

Analogamente la word n. 9 rappresenta l'offset che deve essere assegnato allo «Stack Pointer» (SP) sempre prima di dare il via all'esecuzione.

Per capire meglio il significato di questi concetti, analizziamo subito le word nn. 10 e 11, allorché vedremo un esempio di applicazione.

Word n. 11, 12 - sono tutto sommato i valori più importanti in quanto definiscono qual è l'indirizzo iniziale da cui il programma deve essere eseguito.

In particolare la word n. 11 rappresenta l'offset che deve essere assegnato all'«Instruction Pointer» (IP) subito prima dell'esecuzione del programma, mentre la word n. 12 rappresenta di quanto deve essere alterato il valore proposto dal sistema operativo per il «Code Segment» (CS), ancora una volta per ottenere il funzionamento del programma.

Questo perché il nostro file potrà avere all'inizio una parte del «Code Segment» in cui sono contenute le costanti di programma, e solo dopo potrà esserci il vero e proprio «Code Segment».

In generale infatti, scrivendo un programma in Assembler, dapprima creeremo il «Data Segment» contenente le variabili e poi creeremo il «Code Segment» con all'interno le costanti e le istruzioni del nostro programma: tradotto il tutto dall'assemblatore, il linker a questo punto genererà un file contenente appunto nella prima parte la zona relativa al «Data Segment» e poi quella relativa al «Code Segment».

Il fatto che la word n. 8 e la word n. 12 rappresentino un valore con cui si devono alterare rispettivamente i registri di stack e di codice, deriva dal fatto che il sistema operativo, all'atto di assegnare un certo segmento al nostro file, a quel tale valore pone tutti e quattro i registri di segmento ed in particolare il CS e l'SS (del DS e dell'ES non ci si deve preoccupare): questo si può verificare analizzando con il «debug» in file di tipo «.COM», nel qual caso i valori per i quattro registri segmento coincidono.

Nel caso invece di un «.EXE», tanto il CS che l'SS devono essere alterati sommandoci il valore posto all'interno delle word relative: anche questo si può verificare analizzando, sempre con il «debug», un file di tipo «.EXE». In questo caso i valori contenuti in CS e SS sono proprio i valori contenuti in DS o ES (che sono come visto lasciati inalterati), aumentati dei rispettivi valori forniti appunto nell'header del file.

Per quanto riguarda i valori da assegnare ad SP e ad IP, questi devono essere forniti, in quanto altrimenti per default lo Stack Pointer viene posto ad FFFFH (e cioè si considera lo stack posto a partire dal «fondo» del segmento assegnato al file: ricordiamoci che lo stack «cresce», «evolve» verso indirizzi inferiori e cioè «verso» la parte del segmento occupata dal nostro prezioso programma), mentre, come già sappiamo, l'Instruction Pointer verrebbe posto a 0100H.

Sono rimaste a questo punto da analizzare due word relative ai cosiddetti «relocation table items».

Per i soliti problemi di spazio ed anche per concedere un certo periodo di riflessione su quanto detto finora, interrompiamo l'analisi della tabella, per riprenderla nella prossima puntata, quando parleremo di altre caratteristiche dei file sin qui analizzati.

MC