A cura di Valter Di Dio

16 bit in un Apple II?

Tutti i possessori di un Apple II sanno che il loro computer contiene un microprocessore 6502 che è poi il cervello della macchina.

Ma esiste all'interno degli Apple II un altro microprocessore, poco conosciuto agli utenti: lo SWEET 16.

Se lo si cerca aprendo la macchina è molto difficile da trovare perché al contrario degli altri microprocessori non è il solito grosso integratone nero con tanti piedini; per vederlo bisognerebbe essere in grado di leggere dentro la RAM del computer; infatti lo SWEET 16 è un metaprocessore a sedici bit interamente software.

Lo SWEET 16

Lo SWEET 16 nasce dalla geniale mente di Stephen Wozniak (e i veri Applisti a questo nome fanno una pausa d'effetto) verso la fine del 1976, quando Stephen era alle prese col problema di infilare in quattro kappa di ROM i dodici e passa del suo Basic. Comunque rigirasse la faccenda la mancanza che più si faceva sentire era quella di un microprocessore con una serie di registri e puntatori a sedici bit. Tutti i grossi programmi (compilatori, editor o assemblatori) fanno largo uso di routine a sedici bit e, come dimostrato anche dalle leggi di Murphy, una routine a sedici bit occupa molto più del doppio dello spazio necessario per una ad otto bit. L'unica soluzione (per quei tempi in cui i computer nascevano con un kappa di ROM e otto di RAM) era la creazione di un metaprocessore software a sedici bit. Un metaprocessore è un programma in linguaggio macchina che, simulando un microprocessore, è in grado di eseguire pezzi di codice come se si trattasse del programma per un vero e proprio. Anzi di solito tutti i microprocessori nascono dopo uno studio su di un metaprocessore, indispensabile alla ricerca dei difetti senza dover ricorrere allo scalpello.

Quello che serviva a Wozniak era un processore con almeno una decina di registri a sedici bit e che fosse in grado di eseguire semplici operazioni come la somma, la sottrazione, il confronto e il movimento dei dati in memoria. Perché fosse poi un vero microprocessore occorrevano le istruzioni per il controllo del flusso e quindi i salti condizionati ed incondizionati.

Con l'uso dello SWEET 16 l'Apple II potè finalmente disporre del suo Basic, e, nonostante pochi programmi ne facciano tuttora uso, è ancora possibile trovare lo SWEET 16 dentro all'Integer Basic.

Per utilizzare lo SWEET 16 si deve prima di tutto caricare, dal disco Master, l'Integer Basic nella Language Card (se non avete l'espansione a 64K andate a vedere sul numero 18 di MCmicrocomputer come si può caricare l'Integer Basic in RAM). A questo punto potete sfilare lo SWEET 16 dal Basic seguendo le istruzioni riportate nel riquadro di pagina 139.

Utilizziamo lo SWEET 16

Per scrivere un programma in SWEET 16 occorre, oltre allo SWEET 16, anche l'elenco delle istruzioni e il loro significato. L'elenco lo trovate in figura 2, per le descrizioni dobbiamo prima vedere come lo SWEET 16 lavora.

Lo SWEET 16 è basato su sedici registri chiamati da R0 ad R15, e formati da due byte ciascuno (16 bit). Nella versione base, i sedici registri occupano le prime trentadue locazioni della pagina zero (inutilizzate nel primo Apple II, quello senza le ROM autostart) che possono però essere sostituite a piacere con altre, purché consecutive, semplicemente modificando il codice oggetto dello SWEET 16. Alcuni di questi sedici registri hanno significati particolari e, a meno di saper bene quello che si sta facendo, è meglio lasciarli in pace. Il primo di questi registri tabù è R15 che è il Program Counter, poi vengono R14 che contiene lo Status, R13 in cui viene messo il risultato dell'ultima operazione di confronto ed R12 che è il puntatore allo Stack delle subroutine. Salvo R0 che corrisponde all'Accumulatore principale dello SWEET 16, tutti gli altri registri, da R1 ad R11, sono a completa disposizione dell'utente.

Le istruzioni dello SWEET 16 che utilizzano i registri sono di due tipi: il primo tipo vede il registro come una locazione in cui mettere o da cui prendere un dato a sedici bit; mentre le istruzioni del secondo tipo utilizzano il contenuto dei registri come un puntatore alla coppia di locazioni di memoria che contiene il dato effettivo.

Ad esempio: INR R5 incrementa il contenuto di R5; mentre STD @R5 scrive il dato contenuto nell'accumulatore (R0) nella locazione di RAM puntata da R5 e nella successiva (come consuetudine prima la parte bassa e poi la parte alta). Come avrete capito la differenza tra l'uso di un registro come dato o come pointer si distingue (oltre ovviamente che dal codice operativo) dal simbolo @ messo prima del nome del registro.

Ci sono poi una serie di istruzioni che non utilizzano i registri (almeno non quelli di utente), le principali sono i BRANCH che secondo la prassi del 6502 permettono spostamenti relativi (+/−127 byte) a seconda del risultato di certi test effettuati sul registro di STATUS (Carry set, positivo, negativo, zero ecc.). Da rilevare anche la presenza di un BRA (Branch Always) che effettuando il salto incondizionato equivale al non implementato JMP. Anche il JSR (Jump to Subroutine) è implementato con indirizzamento relativo e si chiama quindi BRS (Branch to Subroutine).

Dal momento che lo SWEET 16 è, in fondo, un programma, la chiamata avviene tramite un JSR SWEET effettuato dal nostro programma utente (ovviamente in linguaggio macchina) e il ritorno avverrà tramite un RTN (istruzione di Return dello SWEET

16). Subito dopo il JSR SWEET si scrive la porzione di programma in codice SWEET 16.

Ecco un esempio di programma che effettua una MOVE scritto in SWEET 16.

```
300: 20 00 08          JSR SWEET
303: 41        LOOP    LDR @R1
304: 52                STO @R2
305: F3                DEC  R3
306: 07 FB             BNZ LOOP
308: 00                RTN
309: 60                RTS
```

Come vedete nove byte di programma permettono di scrivere una routine che muove (R3) byte dalla posizione (R1) alla posizione (R2). È chiaro che per una sola routine non conviene portarsi dietro ogni volta 370 byte di SWEET 16; ma pensate al risparmio di memoria che un simile oggetto permette nella stesura di programmi di oltre un kappa.

Per quanto riguarda la velocità di esecuzione lo SWEET 16 è ovviamente molto più lento di un equivalente programma scritto direttamente per il 6502, ma, ricordando che lo SWEET 16 lavora a sedici bit, si ottiene ugualmente una discreta velocità (grazie anche ad una furbizia di Stephen Wozniak che ha predisposto l'incremento o il decremento automatico dei registri quando sono utilizzati come puntatori).

Provate ad esempio questo pro-

```
1E89-  20 4A FF     JSR   $FF4A
1E8C-  68           PLA
1E8D-  85 1E        STA   $1E
1E8F-  68           PLA
1E90-  85 1F        STA   $1F
1E92-  20 98 1E     JSR   $1E98
1E95-  4C 92 1E     JMP   $1E92
1E98-  E6 1E        INC   $1E
1E9A-  D0 02        BNE   $1E9E
1E9C-  E6 1F        INC   $1F
1E9E-  A9 1F        LDA   #$1F
1EA0-  48           PHA
1EA1-  A0 00        LDY   #$00
1EA3-  B1 1E        LDA   ($1E),Y
1EA5-  29 0F        AND   #$0F
1EA7-  0A           ASL
1EA8-  AA           TAX
1EA9-  4A           LSR
1EAA-  51 1E        EOR   ($1E),Y
1EAC-  F0 0B        BEQ   $1EB9
1EAE-  86 1D        STX   $1D
1EB0-  4A           LSR
1EB1-  4A           LSR
1EB2-  4A           LSR
1EB3-  A8           TAY
1EB4-  B9 E1 1E     LDA   $1EE1,Y
1EB7-  48           PHA
1EB8-  60           RTS
1EB9-  E6 1E        INC   $1E
1EBB-  D0 02        BNE   $1EBF
1EBD-  E6 1F        INC   $1F
1EBF-  BD E4 1E     LDA   $1EE4,X
1EC2-  48           PHA
1EC3-  A5 1D        LDA   $1D
1EC5-  4A           LSR
1EC6-  60           RTS
1EC7-  68           PLA
1EC8-  68           PLA
1EC9-  20 3F FF     JSR   $FF3F
1ECC-  6C 1E 00     JMP   ($001E)
1ECF-  B1 1E        LDA   ($1E),Y
1ED1-  95 01        STA   $01,X
1ED3-  88           DEY
1ED4-  B1 1E        LDA   ($1E),Y
1ED6-  95 00        STA   $00,X
1ED8-  98           TYA
1ED9-  38           SEC
1EDA-  65 1E        ADC   $1E
1EDC-  85 1E        STA   $1E
1EDE-  90 02        BCC   $1EE2
1EE0-  E6 1F        INC   $1F
1EE2-  60           RTS

1EE3- 02 F9 04 9D 0D
1EE8- 9E 25 AF 16 B2 47 B9 51
1EF0- C0 2F C9 5B D2 85 DD 6E
1EF8- 05 33 E8 70 93 1E E7 65
1F00- E7 E7 E7

1F03-  10 CA        BPL   $1ECF
1F05-  B5 00        LDA   $00,X
1F07-  85 00        STA   $00
1F09-  B5 01        LDA   $01,X
1F0B-  85 01        STA   $01
1F0D-  60           RTS
1F0E-  A5 00        LDA   $00
1F10-  95 00        STA   $00,X
1F12-  A5 01        LDA   $01
1F14-  95 01        STA   $01,X
1F16-  60           RTS
1F17-  A5 00        LDA   $00
1F19-  81 00        STA   ($00,X)
1F1B-  A0 00        LDY   #$00
1F1D-  84 1D        STY   $1D
1F1F-  F6 00        INC   $00,X
1F21-  D0 02        BNE   $1F25
1F23-  F6 01        INC   $01,X
1F25-  60           RTS
1F26-  A1 00        LDA   ($00,X)
1F28-  85 00        STA   $00
1F2A-  A0 00        LDY   #$00
1F2C-  84 01        STY   $01
1F2E-  F0 ED        BEQ   $1F1D
1F30-  A0 00        LDY   #$00
1F32-  F0 06        BEQ   $1F3A
1F34-  20 66 1F     JSR   $1F66
1F37-  A1 00        LDA   ($00,X)
1F39-  A8           TAY
1F3A-  20 66 1F     JSR   $1F66
1F3D-  A1 00        LDA   ($00,X)
1F3F-  85 00        STA   $00
1F41-  84 01        STY   $01
1F43-  A0 00        LDY   #$00
1F45-  84 1D        STY   $1D
1F47-  60           RTS
1F48-  20 26 1F     JSR   $1F26
1F4B-  A1 00        LDA   ($00,X)
1F4D-  85 01        STA   $01
1F4F-  4C 1F 1F     JMP   $1F1F
1F52-  20 17 1F     JSR   $1F17
1F55-  A5 01        LDA   $01
1F57-  81 00        STA   ($00,X)
1F59-  4C 1F 1F     JMP   $1F1F
1F5C-  20 66 1F     JSR   $1F66
1F5F-  A5 00        LDA   $00
1F61-  81 00        STA   ($00,X)
1F63-  4C 43 1F     JMP   $1F43
1F66-  B5 00        LDA   $00,X
1F68-  D0 02        BNE   $1F6C
1F6A-  D6 01        DEC   $01,X
1F6C-  D6 00        DEC   $00,X
1F6E-  60           RTS
1F6F-  A0 00        LDY   #$00
1F71-  38           SEC
1F72-  A5 00        LDA   $00
1F74-  F5 00        SBC   $00,X
1F76-  99 00 00     STA   $0000,Y
1F79-  A5 01        LDA   $01
1F7B-  F5 01        SBC   $01,X
1F7D-  99 01 00     STA   $0001,Y
1F80-  98           TYA
1F81-  69 00        ADC   #$00
1F83-  85 1D        STA   $1D
1F85-  60           RTS
1F86-  A5 00        LDA   $00
1F88-  75 00        ADC   $00,X
1F8A-  85 00        STA   $00
1F8C-  A5 01        LDA   $01
1F8E-  75 01        ADC   $01,X
1F90-  A0 00        LDY   #$00
1F92-  F0 E9        BEQ   $1F7D
1F94-  A5 1E        LDA   $1E
1F96-  20 19 1F     JSR   $1F19
1F99-  A5 1F        LDA   $1F
1F9B-  20 19 1F     JSR   $1F19
1F9E-  18           CLC
1F9F-  B0 0E        BCS   $1FAF
1FA1-  B1 1E        LDA   ($1E),Y
1FA3-  10 01        BPL   $1FA6
1FA5-  88           DEY
1FA6-  65 1E        ADC   $1E
1FA8-  85 1E        STA   $1E
1FAA-  98           TYA
1FAB-  65 1F        ADC   $1F
1FAD-  85 1F        STA   $1F
1FAF-  60           RTS
1FB0-  B0 EC        BCS   $1F9E
1FB2-  60           RTS
1FB3-  0A           ASL
1FB4-  AA           TAX
1FB5-  B5 01        LDA   $01,X
1FB7-  10 E8        BPL   $1FA1
1FB9-  60           RTS
1FBA-  0A           ASL
1FBB-  AA           TAX
1FBC-  B5 01        LDA   $01,X
1FBE-  30 E1        BMI   $1FA1
1FC0-  60           RTS
1FC1-  0A           ASL
1FC2-  AA           TAX
1FC3-  B5 00        LDA   $00,X
1FC5-  15 01        ORA   $01,X
1FC7-  F0 D8        BEQ   $1FA1
1FC9-  60           RTS
1FCA-  0A           ASL
1FCB-  AA           TAX
1FCC-  B5 00        LDA   $00,X
1FCE-  15 01        ORA   $01,X
1FD0-  D0 CF        BNE   $1FA1
1FD2-  60           RTS
1FD3-  0A           ASL
1FD4-  AA           TAX
1FD5-  B5 00        LDA   $00,X
1FD7-  35 01        AND   $01,X
1FD9-  49 FF        EOR   #$FF
1FDB-  F0 C4        BEQ   $1FA1
1FDD-  60           RTS
1FDE-  0A           ASL
1FDF-  AA           TAX
1FE0-  B5 00        LDA   $00,X
1FE2-  35 01        AND   $01,X
1FE4-  49 FF        EOR   #$FF
1FE6-  D0 B9        BNE   $1FA1
1FE8-  60           RTS
1FE9-  A2 18        LDX   #$18
1FEB-  20 66 1F     JSR   $1F66
1FEE-  A1 00        LDA   ($00,X)
1FF0-  85 1F        STA   $1F
1FF2-  20 66 1F     JSR   $1F66
1FF5-  A1 00        LDA   ($00,X)
1FF7-  85 1E        STA   $1E
1FF9-  60           RTS
1FFA-  4C C7 1E     JMP   $1EC7
```

Figura 1 - Listato del metaprocessore a sedici bit chiamato da Stephen Wozniak «SWEET 16». Tramite lo SWEET 16 è possibile scrivere dei programmi che fanno largo uso di routine a sedici bit risparmiando una quantità enorme di memoria e molto tempo di sviluppo.

grammino che riempie lo schermo con una specie di mosaico:

```
300: 20 89 F6          JSR SWEET
303: 10 BC BE          SET  R0 = "><"
306: 11 00 04          SET  R1 = $400
309: 12 FF 1           SET  R2 = $1FF
30C: 71          loop  STD @R1
30D: F2                DCR  R2
30E: 07 FC             BNZ loop
310: 00                RTN
311: 60                RTS
```

Come vedete si sfrutta il fatto che l'accumulatore può contenere due byte per dimezzare il numero di cicli necessari a riempire tutta la pagina grafica (R2 = $1FF invece che $3FF).

Le istruzioni dello SWEET 16

Cominciamo per ordine (numerico) la descrizione delle istruzioni dello SWEET 16.

Codice 00 **RTN** (1 byte)
Ritorna al programma 6502 nel punto immediatamente successivo al RTN; i registri e lo Status sono quelli precedenti alla chiamata dello SWEET 16.

Codice 01 **BRA** (2 byte)
Branch Always: effettua la diramazione alla locazione indicata (relativamente) senza alcun controllo (sempre). Equivale ad un JMP con indirizzamento relativo.

Codice 02 **BNC** (2 byte)
Branch No Carry: effettua la diramazione se il Carry è zero.

Codice 03 **BIC** (2 byte)
Branch If Carry: effettua la diramazione se il Carry è settato.

Codice 04 **BIP** (2 byte)
Branch If Plus: effettua la diramazione se l'ultimo risultato (o l'ultimo dato trasferito) era positivo (bit 15 = zero).

Codice 05 **BIM** (2 byte)
Branch If Minus: effettua la diramazione se l'ultimo risultato era negativo (bit 15 = 1).

Codice 06 **BIZ** (2 byte)
Branch If Zero: effettua la diramazione se l'ultimo risultato era uguale a zero.

Codice 07 **BNZ** (2 byte)
Branch If No Zero: effettua la diramazione in caso di risultato diverso da zero.

Codice 08 **BM1** (2 byte)
Branch if Minus 1: il salto viene effettuato se il risultato è −1 ($FFFF).

Codice 09 **BNM** (2 byte)
Branch if Not Minus 1: esegue la diramazione se il risultato è diverso da $FFFF.

Codice 0A **BKS** (1 byte)
Break: esegue un Break del 6502 (corrisponde al codice 0 del 6502).

Tabella dei codici operativi dello SWEET 16

Operazioni sui registri			Operazioni senza registri		
			00	RTN	
1n	SET	Rn	01	BRA	d
2n	LDR	Rn	02	BNC	d
3n	STO	Rn	03	BIC	d
4n	LDR	@Rn	04	BIP	d
5n	STO	@Rn	05	BIM	d
6n	LDD	@Rn	06	BIZ	d
7n	STD	@Rn	07	BNZ	d
8n	POP	@Rn	08	BM1	d
9n	STP	@Rn	09	BNM	d
An	ADD	Rn	0A	BKS	d
Bn	SUB	Rn	0B	RSB	d
Cn	PPD	@Rn	0C	BSB	d
Dn	CPR	Rn	0D	NUL	
En	INR	Rn	0E	NUL	
Fn	DCR	Rn	0F	NUL	

Figura 2 - Lista dei comandi dello SWEET 16 divisi in codici che utilizzano i registri e codici che non utilizzano i registri.

Codice 0B **RSB** (1 byte)
Return from SuBroutine: il controllo passa al programma principale nel punto successivo al GOSUB.

Codice 0C **BSB** (2 byte)
Branch to SuBroutine: esegue un JSR con indirizzamento relativo (l'indirizzo di ritorno viene immagazzinato nello Stack e il controllo passa alla subroutine).

Codici 0D 0E 0F **NOP** (2 byte)
No Operation: questi codici non sono implementati; possono essere utilizzati come NOP oppure implementati dall'utente.

Codice 1N **SET** RN = costante (3 byte)
Mette nel registro N il valore contenuto nei due byte che seguono l'istruzione (es. SET R5, $FDED –> 15 ED FD). Questa è l'unica istruzione lunga tre byte.

Codice 2N **LDR** RN (1 byte)
LoaD Register: carica nell'accumulatore (Registro R0) il contenuto (a 16 bit) del registro N.

Codice 3N **STO** RN (1 byte)
STOrage: scarica nel registro N il contenuto dell'Accumulatore (R0).

Codice 4N **LDR** @RN (1 byte)
LoaD Register indirect: carica in accumulatore il contenuto della locazione di memoria (solo 8 bit) puntata dal registro N, la parte alta dell'Accumulatore viene azzerata. Attenzione! Dopo il trasferimetno RN è incrementato di uno automaticamente.

Codice 5N **STO** @RN (1 byte)
STOrage indirect: scarica la parte bassa dell'Accumulatore (8 bit) nella locazione di memoria puntata da RN; RN è incrementato di uno.

Codice 6N **LDD** @RN (1 byte)
LoaD Double: carica in accumulatore il contenuto della locazione puntata da RN e della successiva (16 bit); RN è automaticamente incrementato di due.

Codice 7N **STD** @RN (1 byte)
Storage Double: scarica il contenuto dell'Accumulatore (16 bit) nella locazione puntata da RN e nella successiva; RN è poi incrementato di due.

Codice 8N **POP** @RN (1 byte)
POP indirect: RN viene decrementato di uno, dopodiché la parte bassa dell'accumulatore viene caricata con il contenuto della locazione puntata da RN e la parte alta dell'Accumulatore azzerata.

Codice 9N **STP** ®RN (1 byte)
STore and Pop indirect: RN viene decrementato di uno, il contenuto della locazione puntata da RN viene caricato nella parte bassa dell'Accumulatore; poi RN viene ancora decrementato e il contenuto della locazione indicata da RN, caricato nella parte alta dell'accumulatore.

Codice AN **ADD** RN (1 byte)
ADD: il contenuto di RN viene sommato al contenuto dell'Accumulatore (R0) il diciassettesimo bit diventa il Carry, il risultato torna nell'Accumulatore.

Codice BN **SUB** RN (1 byte)
SUBtract: il contenuto del registro N viene sottratto dal contenuto dell'Accumulatore, se il contenuto di RN è strettamente minore del contenuto di R0 (Acc.) il Carry viene settato.

Codice CN **PPD** ®RN (1 byte)
PoP Double: RN è decrementato di uno, la parte alta dell'Accumulatore viene caricata con il contenuto della locazione di memoria puntata da RN; quindi RN viene nuovamente decrementato e viene caricato il contenuto della nuova locazione nella parte bassa dell'Accumulatore.

Codice DN **CPR** RN (1 byte)
ComPaRe: al contenuto dell'Accumulatore viene sottratto il contenuto di RN, il risultato viene perso, ma se R0 > = RN viene settato il Carry. Nessun registro viene modificato (escluso R14).

Codice EN **INR** RN (1 byte)
INcRement: il contenuto di RN viene incrementato di uno.

Codice FN **DCR** RN (1 byte)
DeCRement: il contenuto di RN viene decrementato di uno.

Nota 1: tutte le istruzioni sui registri (da 1x a Fx) tolte ADD, SUB e CPR puliscono il Carry.

Nota 2: Tutti gli mnemonici delle istruzioni differiscono un poco da quelli originali pubblicati su Byte maggio 1977 perché ho preferito illustrare quelli dell'Assembler LISA che permette di usare lo SWEET 16 nei propri programmi.

Conclusioni

Avere a disposizione un microprocessore come lo SWEET 16 permette di sviluppare più rapidamente le proprie applicazioni in linguaggio macchina, risparmiando oltretutto una notevole quantità di codice oggetto. Infatti la maggior parte delle istruzioni dello SWEET 16 è lunga un solo byte (contro le circa due del codice 6502) pur lavorando a sedici bit. La velocità di esecuzione è a livello di una routine in linguaggio macchina e si nota il rallentamento solo durante grossi trasferimenti di dati. Mancano purtroppo le istruzioni di ingresso e di uscita, ma se proprio servono è facile implementarle utilizzando i codici lasciati liberi.

Come si recupera lo SWEET 16

Per prima cosa bisogna caricare in memoria l'Integer Basic; chi ha la Language CARD o comunque 64 kappa di RAM (Apple IIe e Apple IIc) deve solo effettuare il bootstrap con il disco Master, e si troverà automaticamente con l'Integer Basic caricato nella RAM a partire da $D000 fino a $F7FF. Chi invece non ha l'estensione di memoria deve procurarsi una copia di MC numero 18 dove è spiegato come si può caricare l'Integer Basic sotto al DOS.

Per vedere se l'Integer Basic è stato effettivamente caricato basta battere direttamente da tastiera il comando INT (è un comando del DOS); in risposta il Prompt dell'Apple si trasforma dalla parentesi quadra al simbolo di maggiore (>) che è il prompt dell'Integer.

Eseguite ora, passo passo, i seguenti comandi:

]INT return

l'Apple risponde con:

>

>CALL −151 return

l'Apple risponde con:

*

*1E89<F689.F7FFM return

Chi avesse l'Integer rilocato a $ 6000 deve battere invece:

*1E89<9289.93FFM return

adesso ci sono 18 locazioni da modificare (vedi elenco qui a fianco).

A questo punto possiamo salvare lo SWEET 16 su disco, quindi:

BSAVE SWEET 16, A$1E89, L$176

La zona in cui abbiamo rilocato lo SWEET 16 è appena sotto la prima pagina grafica; se qualcuno volesse spostarlo altrove lo può fare a patto di non cambiare la parte bassa dell'indirizzo di partenza (deve sempre essere $89). Chi volesse cambiare anche questa si dovrà rilocare anche la tabella codici-indirizzo che inizia a $1EE3 e termina a $1F02 facendo attenzione al fatto che, per risparmiare spazio, tutte le entrate degli Op-Code dello SWEET 16 si devono trovare nella stessa pagina di memoria, pagina che è indicata nella locazione $1E9F.

```
*1E94:1E   *1F54:1F
*1E97:1E   *1F5B:1F
*1E9F:1F   *1F5E:1F
*1EB6:1E   *1F65:1F
*1EC1:1E   *1F98:1F
*1F36:1F   *1F9D:1F
*1F3C:1F   *1FED:1F
*1F4A:1F   *1FF4:1F
*1F51:1F   *1FFC:1E
```

Apple-posta

Hard copy in alta risoluzione

Molti lettori ci scrivono chiedendo una routine per effettuare l'Hard-copy delle pagine grafiche in alta risoluzione sulla loro stampante. Purtroppo non è possibile rispondere con una unica routine a tutte le richieste, infatti ciascuna stampante richiede un proprio programma di gestione, e, a volte, il programma deve essere diverso a seconda della scheda di interfaccia utilizzata. Chi possiede una stampante abbastanza comune e una interfaccia originale può sicuramente trovare in commercio la routine necessaria (in genere viene fornita con l'acquisto della stampante); inoltre i più importanti programmi di grafica prevedono già l'uso di molte delle stampanti in commercio.

Per quanto riguarda il problema del Sig. Morocuti di Brescia (la sua SPIRIT 80 fa l'hard copy «allungata» in verticale), se la stampante è effettivamente EPSON compatibile (che poi non si sa bene cosa significhi, visto che la mia EPSON MX80 non è compatibile con la nuova EPSON RX80), dovrebbe provare ad accendere la stampante e lanciare un programmino in Basic che sistemi l'interlinea, prima di utilizzare i programmi grafici (sperando che questi non resettino la stampante).

CP/M o non CP/M

Tralasciando i complimenti di rito avrei alcune domande da porvi:

1) Ho letto alcuni annunci pubblicitari su

una scheda CP/M da applicare all'Apple IIc, è veramente utile, e a cosa?

2) Come si usano i caratteri speciali denominati MOUSE SET?

3) E come si usa il Mouse?

Silvano Dotti, Faenza (RA)

La scheda CP/M per l'Apple IIc (che va installata internamente al posto del 65C02) permette di utilizzare tutti i programmi scritti in CP/M o comunque utilizzanti, come microprocessore, lo Z80. È quest'ultima possibilità forse la più interessante, infatti i programmi in CP/M devono essere stati scritti appositamente per l'Apple (la formattazione dei dischetti è diversa) e, a parte il WordStar e un compilatore FORTRAN, non ci sono poi molti programmi che non siano stati riscritti in versione 6502. Chi invece voglia interessarsi alla programmazione dello Z80 trova nella scheda la possibilità di sviluppare proprie routine in linguaggio macchina. Ma a questo punto forse costa meno comprare uno ZX80!

Per quanto riguarda i caratteri speciali denominati «mouse set» questi sostituiscono il set di caratteri maiuscoli in INVERSE.

Quindi, una volta attivati tramite il seguente programmino Basic, basta dare il comando INVERSE quando serve un carattere speciale.

```
10  PRINT  CHR$ (4)"PR#3"
20  PRINT  CHR$ (17): PRINT  CHR$
    (27)
100 HOME
110 PRINT "Questi sono i caratte
    ri del MOUSE:": PRINT
120 FLASH : PRINT "ABCDEFGHIJKLM
    NOPQRSTUVWXYZ\^[@]_"
130 NORMAL
```

La gestione del mouse da Basic si ottiene aprendo un canale di input dallo slot del mouse (IN#4 nel IIc) e utilizzando il semplice comando INPUT"";X,Y,T.

Il software della scheda del mouse metterà automaticamente in X e Y le coordinate del mouse (0...1024) e in T un valore che indica l'attuale posizione del tasto e quella precedente secondo la seguente tabella:

Rilasciato Rilasciato	4
Rilasciato Premuto	3
Premuto Rilasciato	2
Premuto Premuto	1

Inoltre se il valore diventa negativo vuol dire che è stato premuto un tasto sulla tastiera (usare PEEK(−16384) per leggere il tasto e POKE −16368, 0 per togliere il segno meno).

Quanto al difetto incontrato dal lettore (la cancellazione della riga in cui si trova il simbolo mosso col mouse) questo dipende da un Carriage Return che la scheda invia al termine della risposta alla INPUT"";X,Y,T. Per eliminare il difetto occorre effettuare un VTAB 1:HTAB 0 prima di ogni chiamata alla lettura del mouse; naturalmente la prima riga di schermo verrà cancellata e non può quindi essere utilizzata come area lavoro dal programma.

Quali dischetti?

Perché si vendono dischetti a singola e a doppia faccia (con un costo molto maggiore) se poi è possibile trasformare un disco a singola faccia in uno a doppia faccia tramite l'apposito attrezzino che costa poche migliaia di lire? C'è veramente differenza tra i due tipi o è solo una scusa per «soffiare» un altro po' di soldi all'acquirente?

Veronesi Nicola (TN)

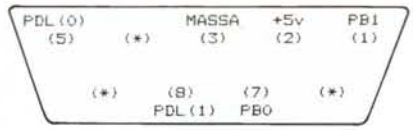
No la differenza esiste, i dischi a doppia faccia sono stati testati, e quindi garantiti, su tutte e due le superfici magnetiche. Il retro dei dischetti a singola faccia, di solito, è utilizzabile (dopo aver creato la tacca di scrittura), ma non è assolutamente certo che vada sempre tutto bene; diciamo che c'è una probabilità su dieci di incappare in una traccia difettosa. Dipende, a questo punto, da quanto valutiamo il contenuto del dischetto, ma penso che sia ben più di 1000 o 2000 lire. Inoltre bisogna ricordare che i driver Apple non sono a doppia faccia per cui il disco scorre tra la testina ed un feltrino anziché tra due testine. In questo modo il lato del disco che non viene letto è a contatto con il feltrino che, trattenendo la polvere che si trova sul supporto, può creare delle scalfiture sulla superficie magnetica. Attenzione quindi: utilizzare un disco da tutte e due le parti su un DISK II può danneggiare anche la faccia «buona».

Presa giochi sul IIc

Volendo utilizzare un Joystick sull'Apple IIc avevo pensato alla vostra interfaccia apparsa sul numero 29. Come devo modificarla per collegarla al mio computer che dispone di un ingresso Cannon a 9 poli.

Giuseppe Vilardi (LT)

Lo schema del connettore del IIc (guardando il computer dal retro) è il seguente:



Quale kit per le minuscole

Alcuni lettori chiedono ulteriori informazioni sul kit per le minuscole. Rispondiamo un po' a tutti qui di seguito.

1) La scelta del tipo di kit (con o senza il circuitino stampato) dipende dal numero di revisione della piastra Apple; il numero si legge, inciso direttamente sullo stampato, tra lo slot 0 e l'alimentatore. Se questo numero termina con 07, o più, ci vuole il kit con la sola Eprom che sostituisce direttamente la ROM originale, se invece il numero termina con 06, o meno (Apple molto vecchi), occorre il secondo tipo di kit che ha anche un piccolo circuito stampato necessario a correggere alcune piste scambiate tra la ROM originale e la EPROM.

2) Per quanto riguarda gli Apple compatibili non è possibile sapere a priori quale kit vada montato, infatti alcuni compatibili montano già una EPROM 2516, ma l'organizzazione interna dei dati è differente da quella Apple. Conviene quindi copiare la EPROM e poi bruciarne una con le minuscole rispettando l'organizzazione di quella originale.

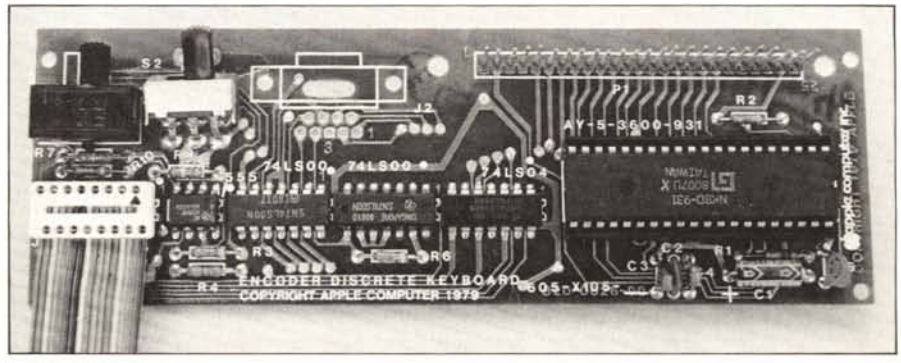
3) Le lettere minuscole occupano i caratteri ASCII da 92 a 127, non interferiscono quindi in alcun modo con i programmi commerciali, anzi molti di questi le utilizzano già regolarmente.

4) Chi volesse utilizzare il set di caratteri minuscoli direttamente da tastiera (con il tasto SHIFT) anche nei programmi protetti (e quindi senza possibilità di utilizzare il MINUS.CODE di BO) possono effettuare una semplice modifica hardware alla scheda che supporta l'encoder di tastiera (è quella fissata proprio sotto la tastiera e da cui parte il filo per l'Apple).

Guardando la scheda con l'integratone a destra e il cavo di uscita a sinistra cercate, in alto a sinistra, il deviatore che seleziona il RESET semplice o il CONTROL-RESET; a destra di questo si vedono sei fori per un secondo deviatore denominato S2. Tagliate con un cacciavite affilato i due Jumper (quelle piccole farfalline stagnate) che si trovano sopra e sotto i fori per il deviatore e montate il deviatore direttamente sulla scheda (coricato come quello del RESET) oppure con uno spezzone di cavo a sei poli lungo 15 o 20 centimetri.

Rimontate il tutto e piazzate il deviatore, in modo che sia facilmente accessibile, vicino alla tastiera (per esempio a sinistra). A seconda della posizione del deviatore la tastiera dell'Apple si trasforma da «Tipo Apple» a «Tipo macchina da scrivere».

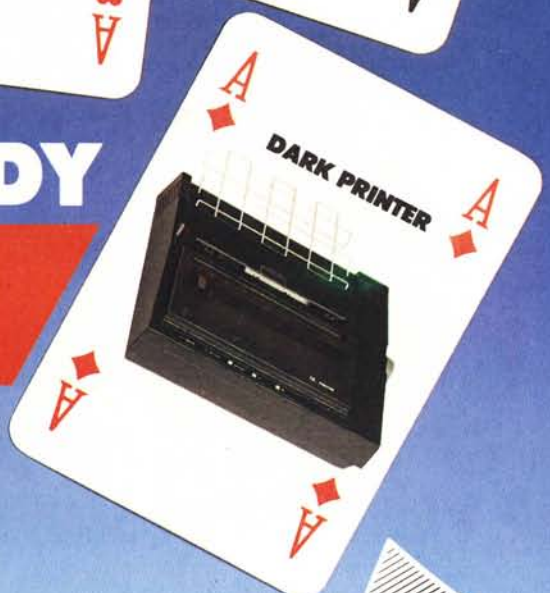
Il deviatore deve essere facile da raggiungere perché con la tastiera e con le minuscole non sono più accessibili i caratteri] ^ e @ che si trovano sopra la M, la N e la P (tasti che adesso funzionano regolarmente) e, se la @ e la] non servono quasi a niente, il simbolo ^ serve nei programmi in Basic come elevamento a potenza e non se ne può fare a meno. MC



LOOK AT THESE TRUMP CARDS FOR YOUR QL ...

... FROM SANDY

QUALITÀ AL MIGLIOR PREZZO!
DIMOSTRATECI IL CONTRARIO
E SAREMO LIETI DI OFFRIRVI
CONDIZIONI ECCEZIONALI.
Offerte speciali per acquisti multipli
telefonare per quotazioni
garanzia completa per un anno



QL FLOPPY DISK SYSTEM

- COMPLETA EMULAZIONE DEI MICRODRIVE
- COMPATIBILITÀ ASSOLUTA CON TUTTO L'HARDWARE ED IL SOFTWARE SINCLAIR
- CAPACITÀ 720 K FORMATTATI
- DIMENSIONI ECCEZIONALMENTE RIDOTTE

TWIN EXPANSION UNIT

- BASSO PROFILO NON INTERFERISCE CON LA TASTIERA
- COLLEGABILE ALLA THRU - CON RAM CARD
- ACCETTA NEL SUO INTERNO FINO A DUE SCHEDE DI ESPANSIONE

THRU - CON RAM CARD

- DUPLICAZIONE DEL CONNETTORE INTERNO
- 256 E 512K DI MEMORIA
- IDEALE PER L'USO CON QUALSIASI FLOPPY DISK CONTROLLER

DARK PRINTER

- 120 CPS BIDIREZIONALE
- EPSON COMPATIBILE
- TRASCINAMENTO A FRIZIONE E TRATTORE
- COMPLETA DI CONVERTITORE SERIALE PARALLELO

☐ QL FLOPPY DISK SYSTEM L. 756.000
☐ SECONDO DRIVE L. 390.000
☐ THRU - CON RAM CARD 256K L. 335.000
☐ THRU - CON RAM CARD 512K L. 490.000
☐ 256K UPGRADE A 512K L. 220.000
☐ TWIN EXPANSION UNIT L. 105.000
☐ DARK PRINTER COMPLETA DI INTERFACCIA L. 621.000

Compilare e inviare questo Coupon (si accettano anche fotocopie)

a: SANDY VIA ERBA 21 - 20037 PADERNO DUGNANO (MI) - TEL. 9105617

NOME _____ COGNOME _____

INDIRIZZO _____

TEL. _____

PREZZI COMPRENSIVI DI IVA E SPESE DI SPEDIZIONE
PAGAMENTO IN CONTRASSEGNO

NUOVISSIME DALLA CITIZEN
ECCO LE VOSTRE PROSSIME STAMPANTI A MATRICE

Progettate con la stessa cura e precisione con cui per oltre 50 anni sono stati costruiti milioni di orologi CITIZEN.
Sia la MSP-20 (80 colonne) che la MSP-25 (136 colonne) stampano alla incredibile velocità di 200 cps in DRAFT MODE e 50 cps in NLQ MODE mentre la MSP-10 (80 colonne) e la MSP-15 (136 colonne) stampano normalmente a 160 cps (40 cps in NLQ).
Inoltre tutte le stampanti CITIZEN sono IBM e EPSON compatibili, usano tutte il sistema di avanzamento della carta a spinta, stampa proporzionale per produrre documenti così nitidi da farvi dubitare che siano stati eseguiti con una stampante a matrice.
L'estetica molto curata conferisce alle stampanti CITIZEN una linea moderna ed essenziale che si adatta perfettamente in ogni ambiente.
...e tutte sono coperte da 2 ANNI DI GARANZIA!!!

CITIZEN
COMPUTER PRINTERS

TELAV
INTERNATIONAL S.r.l.

COMPUTER GRAPHICS DIVISION
MILANO: Via L. da Vinci, 43 - 20090 Trezzano S/N
Tel. 02/4455741/2/3/4/5 - Tlx: TELINT I 312827
ROMA: Via Salaria, 1319 - 00138 Roma
Tel. 06/6917058-6919312 - Tlx: TINTRO I 614381

ADV Studio