128 da zero

a cura di Andrea de Prisco

Orologi, Pixel, Kernal

128 da zero questo mese tratterà di interrupt, di CIA, di Rom, di grafica ed altro. Presenteremo un comodo orologio in tempo reale (non soggetto cioè a ritardi a causa di operazioni di I/O), da tenere costantemente acceso in un angolo dello schermo, più un pacchetto grafico «640× 200 in 320× 200» che permette a chi non dispone di un monitor di visualizzare una finestra di alta risoluzione sul proprio TV.
Per finire, a chi si occupa di programmazione in linguaggio macchina, dedichiamo un riquadro con alcune routine Kernal del 128 non trattate precedentemente.

Orologio R.T.

di Adriano Asnaghi - Mestre (VE)

L'integrato CIA 6526 contenuto all'interno del 128 (per l'esattezza cercando bene ne troviamo due) dispone di un orologio interno in formato Ore-Minuti-Secondi-DecimiDiSecondo, indipendente dalle altre attività della macchina. Per questo motivo è detto «in tempo reale»: non è il processore a farlo avanzare ma è interno al CIA «a tutti gli effetti». Basta settare l'orario, utilizzando appositi registri, e dare il via al conteggio. La routine in linguaggio macchina inviataci dal lettore di Mestre, incuneandosi nella normale procedura di scansione della tastiera permette la visualizzazione continua nell'angolo in alto a destra dello schermo 40 colonne. Effettuando operazioni I/O che prevedono la disabilitazione degli interrupt, potremo notare qualche ritardo nella visualizzazione, ma non nel conteggio.

Una volta digitato il codice esadecimale mostrato in figura 1, e mandato in esecuzione il programma Basic di listato 1 (di inizializzazione) per ottenere l'orologio sarà sufficiente un:

```
SYS 3072
```

Per ripristinare la normale routine di interrupt (e conseguentemente togliere l'orologio) basta digitare:

```
SYS 3075
```

Il listato 2 mostra il disassemblato commentato della routine orologio. Per finire, come lo stesso autore raccomanda, per non far interferire l'orologio con i comandi da noi impartiti è bene definire una window a partire dalla seconda riga con il comando:

```
WINDOW 1,0,24,39
```

I vettori di INTERRUPT

La routine orologio manipolava il vettore di IRQ locato a $0314. Nelle immediate vicinanze esistono pure due vettori, quello locato a $0316 relativo all'interrupt generato da BRK che attiva il MONITOR e quello locato a $0316 relativo all'interrupt NMI (non mascherabile). I contenuti di questi vettori sono

$0314 → $FA65 (IRQ)
$0316 → $B003 (BRK)
$0318 → $FA40 (NMI)

Esiste un altro vettore locato a $0328 che testa la pressione del tasto STOP e che punta a $F66E. Se si volesse disabilitare sia il tasto STOP che la combinazione STOP-RESTORE (che genera un NMI) basterà cambiare i relativi vettori ed il gioco è fatto. Ad esempio, per il tasto STOP basta porre nel relativo vettore l'indirizzo $F67B che contiene un RTS. In questo modo, tutte le volte che viene premuto il tasto STOP non succede nulla. Lo stesso vale per il NMI ponendo in $0318 il valore $FA62.

```
MONITOR
    PC  SR AC XR YR SP
.
B000 00 00 00 00 F8

>00C00 4C 60 0C 4C 68 0C 48 29:
>00C08 70 4A 4A 4A 4A 09 30 91:
>00C10 FA C8 68 29 0F 09 30 91:
>00C18 FA 60 48 8A 48 98 48 A9:
>00C20 20 85 FA A9 04 85 FB AD:
>00C28 09 DD A0 06 20 06 0C A9:
>00C30 3A A0 05 91 FA AD 0A DD:
>00C38 A0 03 20 06 0C A9 3A A0:
>00C40 02 91 FA A0 00 AD 0B DD:
>00C48 AA 10 07 18 29 7F F8 69:
>00C50 12 D8 20 06 0C AD 08 DD:
>00C58 68 A8 68 AA 68 4C 65 FA:
>00C60 78 A2 1A A0 0C B8 50 05:
>00C68 78 A2 65 A0 FA 8E 14 03:
>00C70 8C 15 03 58 60 FF 00 FF:
```

Figura 1 Assemblato esadecimale del programma Orologio.

Orologio R.T.

```
10 SCNCLR:PRINT:PRINT
20 PRINT "INIZIALIZZAZIONE OROLOGIO ":PRINT
30 C=DEC("DD00"): REM CIA2
50 POKE C+14,PEEK(C+14) OR 128: REM 50 HZ CLOCK TRIGGER
60 INPUT "INTRODUCI L'ORA DEL GIORNO (OOMMSS): ";A$
70 O$=LEFT$(A$,2):O=VAL(O$)
80 M$=MID$(A$,3,2):M=VAL(M$)
90 S$=MID$(A$,5,2):S=VAL(M$)
100 IF O>23 THEN 60
110 PM=0:IF O>11 THEN PM=128:O=O-12
120 POKE C+11,6*INT(O/10)+O+PM: REM FORMATO BCD
130 IF M>59 THEN 60
140 POKE C+10,6*INT(M/10)+M: REM FORMATO BCD
150 IF S>59 THEN 60
160 POKE C+9,6*INT(S/10)+S: REM FORMATO BCD
170 POKE C+8,0: REM PARTENZA OROLOGIO
180 PRINT:PRINT "OROLOGIO INIZIALIZZATO "
190 END
```

Listato 1

```
CLOCK    .ORG $0C00
         JMP IR0ENABLE ;abilita TOD routine
         JMP IR0DISABLE ;disabilita TOD routine
;Converte e visualizza
CONVERT  PHA          ;salva valore di A nello stack
         AND #$70     ;decine
         LSR          ;dividi
         LSR          ;per 16
         LSR
         LSR
         ORA #$30     ;trasformalo in ASCII
         STA (*$FA),Y ;e memorizza su video
         INY          ;posizione successiva
         PLA          ;riprendi valore di A dallo stack
         AND #$0F     ;unita'
         ORA #$30     ;
         STA (*$FA),Y
         RTS          ;rientra
IRQ      PHA          ;salva registri
         TXA
         PHA
         TYA
         PHA
         LDA #$20     ;set indirizzo base pagina video
         STA *$FA     ;dove verra' visualizzata
         LDA #$04     ;l'ora del
         STA *$FB     ;giorno
         LDA $DD09    ;preleva secondi
         LDY #$06     ;offset
         JSR CONVERT  ;e converti
         LDA #$3A     ; carattere separatore ':'
         LDY #$05
         STA (*FA),Y
         LDA $DD0A    ;preleva minuti
         LDY #$03     ;offset
         JSR CONVERT  ;e converti
         LDA #$3A     ;separatore
         LDY #$02     ;offset
         STA (*FA),Y
         LDY #$00     ;offset
         LDA $DD0B    ;preleva ore
         TAX          ;test se AM o PM
         BPL BYPASS   ;se AM nessun aggiustamento
         CLC          ;azzera carry
         AND #$7F     ;azzera bit AM/PM
         SED          ;modo decimale
         ADC #$12     ;somma 12
         CLD          ;modo binario
BYPASS   JSR CONVERT  ;e converti
         LDA $DD08    ;libera l'area temporanea
         PLA          ;ripristina registri
         TAY
         PLA
         TAX
         PLA
         JMP $FA65    ;e cedi il controllo all'IRQ di SISTEMA
;
IR0ENABLE SEI         ;disabilita IRQ
         LDX #)IRQ    ;ind. basso routine UTENTE
         LDY #(IRQ    ;ind. alto routine UTENTE
         CLV          ;salto
         BVC STORE    ;incondizionato
IR0DISABLE SEI        ;disabilita IRQ
         LDX #$65     ;ind. basso routine SISTEMA
         LDY #$FA     ;ind. alto routine di SISTEMA
STORE    STX $0314    ;modifica
         STY $0315    ;vettore IRQ
         CLI          ;abilita IRQ
         RTS          ;e rientra al chiamante
```

Listato 2

640× 200 in 320× 200

di Enrico Ceppi - Meda (MI)

Il secondo lettore, ci invia un tool di istruzioni Basic atte a visualizzare una finestra larga 320 pixel della pagina grafica ad alta risoluzione, come noto, larga 640 pixel. Ciò può essere utile quando non si dispone di un monitor ma solo di un televisore. Lasciamo a lui la parola.

Devo premettere che l'apertura di una rubrica che parla di questo computer, e per di più su MC, mi ha subito interessato, dato che conoscevo la qualità della rivista (sono un VICingo...), e quando AdP ha incominciato a trattare il videochip 8563, sono rimasto molto amareggiato. Eggià, perché io NON ho il monitor, e quindi addio 80 colonne. Ma quando ho letto la seconda parte dell'articolo sulla grafica 640 × 200 ho ceduto ed ho digitato tutte le routine proposte. Non potevo però rassegnarmi a non vedere niente, e così, per non arrivare a vertici di masochismo malsani, ho digitato una routine LM che mi trasferisse la pagina grafica 640 × 200 su quella 320 × 200. Dato però che non si può «ficcare» in 320 i 640 punti della pagina, ho pensato di trasportare sulla pagina 320 × 200 solo una finestra di tale dimensione. La routine che fa questo l'ho allocata da $1560 a $15A2, ma inoltre essa usa la subroutine in $15A5-$1645 e quella posta a $1646-$1653. La sintassi per chiamare questa subroutine è:

SYS 5472, ×

dove × è la colonna di caratteri dalla quale si vuole che inizi l'operazione, e quindi è un numero compreso tra 0 e 40.

La routine posizionata a $1664 - $16A8 fa l'operazione inversa, e viene chiamata con:

SYS 5732, ×

dove per la × valgono le stesse considerazioni di prima.

Un inconveniente che ho subito riscontrato nella grafica 640 × 200 era quello della perdita dello schermo grafico (magari ottenuto con svariati quarti d'ora di calcolo) al ritorno in modo testo, ed inoltre l'impossibilità di salvarlo su memoria di massa. Per risolvere questo inconveniente ho pensato di sacrificare la memoria da 1C000 a 1FEFF e di salvare lì la pagina grafica, che ora è anche raggiungibile da un

BSAVE "nomechevuoi", B1, P49152 TO P65152

Per fare questo

SYS 5812

e per riottenere la nostra bella videata

SYS 5859

Le routine che si incaricano di ciò sono quelle poste a $16B4-$16E2 per il salvataggio, e quella posta a $16E3-$170E per il ri-posizionamento.

Tutto questo era bello, ma il dover lavorare con le SYS è certamente molto scomodo, anche se, grazie alla possibilità di assegnare dei valori ai registri A, X ed Y si lavora infinitamente più comodi che sul C64. Le ultime parole di AdP, quelle relative a un ipotetico Basic, mi hanno sollecitato, e quindi mi sono messo all'opera.

Conoscevo l'indirizzo della routine CHRGET, di quella routine cioè, che si incarica di prelevare il Byte di testo Basic quando il C128 deve eseguire un comando. Il primo tentativo è stato quello di preporre ad ogni nuovo comando il carattere &, e di farlo intercettare da questa routine, tentativo però miseramente fallito: il computer eseguiva sempre i comandi impartiti, anche durante la tokenizzazione di una linea, se presenti dopo una REM, e, addirittura inaccettabile, anche se posti dopo un THEN non verificato. La strada da percorrere non era evidentemente quella, ed allora ho scelto di intercettare la routine che il Basic usa per eseguire i comandi. Questa routine viene chiamata tramite un salto indiretto in RAM con JMP ($0308), quindi per intercettarla basta cambiare quelle locazioni (non da Basic ma, da LM) per farle puntare alla nostra routine personale di decodifica comandi che poi vedrà il da farsi. Un tentativo in questo senso ha portato ad un insuccesso per un semplice motivo: il Basic non usa il banco 15 per operare, ma il 14, che è identico al 15 tranne per il fatto che ha la ROM carattere al posto dell'I/O che tanto ci serve per comunicare con il 8563. Per verificare

640 x 200

```
01554  a9 24    lda #$24  modifica il
01556  8d 08 03 sta $0308 vettore della
01559  a9 17    lda #$17  routine del
0155b  8d 09 03 sta $0309 BASIC che
0155e  60       rts       esegue un
0155f  ea       nop       comando
01560  8d 16 0b sta $0b16 salva A
01563  38       sec       lo confronta
01564  e9 29    sbc #$29  con 40
01566  90 01    bcc $1569 se >40
01568  60       rts       non esegue,
01569  a9 00    lda #$00  altrimenti
0156b  8d 17 0b sta $0b17 pone X=0
0156e  8d 18 0b sta $0b18 pone Y=0
01571  a2 12    ldx #$12  si posiziona
01573  a9 00    lda #$00  sul giusto
01575  20 cc cd jsr $cdcc carattere in
01578  a2 13    ldx #$13  pag.  640x200
0157a  ad 16 0b lda $0b16
0157d  20 cc cd jsr $cdcc
01580  20 a5 15 jsr $15a5 calcola indirizzo byte 320x200
01583  20 d8 cd jsr $cdd8 carica byte 640x200
01586  a0 00    ldy #$00  lo salva in
01588  91 fe    sta ($fe),y pag.320x200
0158a  ee 17 0b inc $0b17 X=X+1
0158d  ad 17 0b lda $0b17
01590  c9 28    cmp #$28  X=40 ?
01592  d0 ec    bne $1580 se no next X
01594  20 46 16 jsr $1646 altr. jsr1646
01597  ee 18 0b inc $0b18 Y=Y+1
0159a  ad 18 0b lda $0b18
0159d  38       sec
0159e  e9 c8    sbc #$c8  Y<200
015a0  90 de    bcc $1580 se si' next Y
015a2  60       rts
015a3  ea       nop
015a4  ea       nop       ----------
015a5  a9 20    lda #$20
015a7  8d 1a 0b sta $0b1a
015aa  a9 00    lda #$00
015ac  8d 19 0b sta $0b19
015af  8d 1c 0b sta $0b1c
015b2  ad 18 0b lda $0b18
015b5  29 f8    and #$f8
015b7  ea       nop
015b8  0a       asl
015b9  8d 1b 0b sta $0b1b
015bc  2e 1c 0b rol $0b1c
015bf  0e 1b 0b asl $0b1b
015c2  2e 1c 0b rol $0b1c
015c5  0e 1b 0b asl $0b1b
015c8  2e 1c 0b rol $0b1c
015cb  ad 1b 0b lda $0b1b
015ce  0a       asl
015cf  8d 1d 0b sta $0b1d
015d2  ad 1c 0b lda $0b1c
015d5  2a       rol
015d6  8d 1e 0b sta $0b1e
015d9  0e 1d 0b asl $0b1d
015dc  2e 1e 0b rol $0b1e
015df  18       clc
015e0  ad 1b 0b lda $0b1b
015e3  6d 1d 0b adc $0b1d
015e6  8d 1b 0b sta $0b1b
015e9  ad 1c 0b lda $0b1c
015ec  6d 1e 0b adc $0b1e calcola il
015ef  8d 1c 0b sta $0b1c
015f2  18       clc       byte di
015f3  ad 19 0b lda $0b19
015f6  6d 1b 0b adc $0b1b coordinate
015f9  8d 19 0b sta $0b19
015fc  ad 1c 0b lda $0b1c X e Y e ne
015ff  6d 1a 0b adc $0b1a
01602  8d 1a 0b sta $0b1a pone
01605  a9 00    lda #$00
01607  8d 1c 0b sta $0b1c l'indirizzo
0160a  ad 17 0b lda $0b17
0160d  0a       asl       in $fe-$ff
0160e  8d 1b 0b sta $0b1b
01611  2e 1c 0b rol $0b1c
01614  0e 1b 0b asl $0b1b
01617  2e 1c 0b rol $0b1c
0161a  0e 1b 0b asl $0b1b
0161d  2e 1c 0b rol $0b1c
01620  18       clc
01621  ad 1b 0b lda $0b1b
01624  6d 19 0b adc $0b19
01627  8d 19 0b sta $0b19
0162a  ad 1c 0b lda $0b1c
0162d  6d 1a 0b adc $0b1a
01630  8d 1a 0b sta $0b1a
01633  ad 18 0b lda $0b18
01636  29 07    and #$07
01638  18       clc
01639  6d 19 0b adc $0b19
0163c  85 fe    sta $fe
0163e  ad 1a 0b lda $0b1a
01641  69 00    adc #$00
01643  85 ff    sta $ff
01645  60       rts       ----------
01646  a9 00    lda #$00  X=0
01648  8d 17 0b sta $0b17 legge 40
0164b  a0 28    ldy #$28  bytes e cosi'
0164d  20 d8 cd jsr $cdd8 va a capo
01650  88       dey       in pagina
01651  d0 fa    bne $164d 640x200
01653  60       rts       ----------
01654  ea       nop
01655  ea       nop
01656  ea       nop
01657  ea       nop
01658  ea       nop
01659  ea       nop
0165a  ea       nop spazio per
0165b  ea       nop
0165c  ea       nop eventuali modifiche
0165d  ea       nop
0165e  ea       nop
0165f  ea       nop
01660  ea       nop
01661  ea       nop
01662  ea       nop
01663  ea       nop
01664  8d 16 0b sta $0b16 salva A
01667  38       sec
01668  e9 29    sbc #$29  se >40
0166a  90 01    bcc $166d
0166c  60       rts       non esegue
0166d  a9 00    lda #$00
0166f  8d 17 0b sta $0b17 X=0
01672  8d 18 0b sta $0b18 Y=0
01675  a2 12    ldx #$12  si posiziona
01677  a9 00    lda #$00  nella pagina
01679  20 cc cd jsr $cdcc 640x200
0167c  a2 13    ldx #$13
0167e  ad 16 0b lda $0b16
01681  20 cc cd jsr $cdcc
01684  20 a5 15 jsr $15a5 calcola byte
01687  a0 00    ldy #$00  preleva byte
01689  b1 fe    lda ($fe),y da 320x200
0168b  a2 1f    ldx #$1f  e lo mette
0168d  20 cc cd jsr $cdcc in 640x200
01690  ee 17 0b inc $0b17 X=X+1
01693  ad 17 0b lda $0b17
01696  c9 28    cmp #$28  X=40 ?
01698  d0 ea    bne $1684 se no next X
0169a  20 46 16 jsr $1646 se si jmp1646
0169d  ee 18 0b inc $0b18 Y=Y+1
016a0  ad 18 0b lda $0b18
016a3  38       sec
016a4  e9 c8    sbc #$c8  Y>199 ?
016a6  90 dc    bcc $1684 se no next Y
016a8  60       rts se si' -> fine
016a9  ea       nop
016aa  ea       nop
016ab  ea       nop
016ac  ea       nop
016ad  ea       nop
016ae  ea       nop
016af  ea       nop
016b0  ea       nop
016b1  ea       nop
016b2  ea       nop
016b3  ea       nop
016b4  a9 c0    lda #$c0 byte iniziale
016b6  85 ff    sta $ff  =c000
016b8  a9 00    lda #$00
016ba  85 fe    sta $fe
016bc  a2 12    ldx #$12 posiziona
016be  a9 00    lda #$00 cursore in
016c0  20 cc cd jsr $cdcc 640x200
016c3  a2 13    ldx #$13
016c5  20 cc cd jsr $cdcc preleva byte
016c8  a9 fe    lda #$fe  prepara per
016ca  8d b9 02 sta $02b9 indsta
016cd  a0 00    ldy #$00
016cf  20 d8 cd jsr $cdd8
016d2  a2 01    ldx #$01
016d4  20 da f7 jsr $f7da esegue indsta
016d7  c8       iny incrementa byte in.
016d8  d0 f5    bne $16cf se<>0 next
016da  e6 ff    inc $ff inc.hi byte in.
016dc  a6 ff    ldx $ff   arrivato a
016de  e0 ff    cpx #$ff  feff ?
016e0  d0 ed    bne $16cf se no next
016e2  60       rts       se si' ->fine
016e3  a9 c0    lda #$c0  byte iniziale
016e5  85 ff    sta $ff   =c000
016e7  a9 00    lda #$00
016e9  85 fe    sta $fe
016eb  a2 12    ldx #$12  posiziona
016ed  a9 00    lda #$00  cursore in
016ef  20 cc cd jsr $cdcc 640x200
016f2  a2 13    ldx #$13
016f4  20 cc cd jsr $cdcc
016f7  a0 00    ldy #$00  prepara per
016f9  a2 01    ldx #$01
016fb  a9 fe    lda #$fe  indfet
016fd  20 d0 f7 jsr $f7d0 esegue indfet
01700  20 ca cd jsr $cdca scrive byte
01703  c8       iny inc.low indir.
01704  d0 f3    bne $16f9 next
01706  e6 ff    inc $ff inc.high indir.
01708  a6 ff    ldx $ff
0170a  e0 ff    cpx #$ff >feff
0170c  d0 eb    bne $16f9 se no next
0170e  60       rts       se si' ->fine
0170f  ea       nop
01710  a9 00    lda #$00  seleziona
01712  8d 00 ff sta $ff00 banco 15
01715  60       rts
01716  20 80 03 jsr $0380 chargot
01719  20 96 af jsr $af96 valuta espres
0171c  20 0c af jsr $af0c tras. in int.
0171f  a5 16    lda $16   A=low
01721  a6 17    ldx $17   X=high
```

(continua a pagina 198)

(segue da pagina 197)

```
01723  60       rts
01724  ad 00 ff lda $ff00 salva banco
01727  85 fd    sta $fd
01729  a5 3d    lda $3d   salva punt.
0172b  48       pha       al testo
0172c  a5 3e    lda $3e   BASIC
0172e  48       pha
0172f  20 80 03 jsr $0380 chargot
01732  c9 26    cmp #$26  = & ?
01734  f0 0e    beq $1744 se si' >start
01736  68       pla recupera puntatore
01737  85 3e    sta $3e
01739  68       pla
0173a  85 3d    sta $3d
0173c  a5 fd    lda $fd   recupera
0173e  8d 00 ff sta $ff00 banco
01741  4c a2 4a jmp $4aa2 vai basic 7.0
01744  68       pla scarica stack
01745  68       pla
01746  20 80 03 jsr $0380 chargot
01749  c9 de    cmp #$de  =graphic ?
0174b  d0 3b    bne $1788 se no salta
0174d  20 80 03 jsr $0380 chargot
01750  c9 30    cmp #$30  = 0 ?
01752  d0 14    bne $1768 se no salta
01754  20 10 17 jsr $1710 banco 15
01757  a5 3a    lda $3a   controlla
01759  ea       nop
0175a  38       sec spazio per save
0175b  e9 c0    sbc #$c0  area 640x200
0175d  b0 03    bcs $1762 se no salta
0175f  20 b4 16 jsr $16b4 se si' salva
01762  20 07 13 jsr $1307 setta modo tx
01765  4c 3c 17 jmp $173c jmp basic 7.0
01768  c9 31    cmp #$31  = 1 ?
0176a  d0 0c    bne $1778 se no salta
0176c  20 10 17 jsr $1710 banco 15
0176f  20 00 13 jsr $1300 setta 640x200
01772  20 e3 16 jsr $16e3 load 640x200
01775  4c 3c 17 jmp $173c jmp basic 7.0
01778  c9 32    cmp #$32  = 2 ?
0177a  d0 e9    bne $1765 se no basic 7.0
0177c  20 10 17 jsr $1710 banco 15
0177f  20 00 13 jsr $1300 setta 640x200
01782  20 16 13 jsr $1316 pulisce 640x200
01785  4c 3c 17 jmp $173c jmp basic 7.0
01788  c9 e5    cmp #$e5  = draw ?
0178a  d0 53    bne $17df se no salta
0178c  20 80 03 jsr $0380 chargot
0178f  85 fc    sta $fc   salva comando
01791  20 16 17 jsr $1716 valuta X
01794  48       pha       salva X
01795  8a       txa       nello stack
01796  48       pha
01797  20 16 17 jsr $1716 valuta Y
0179a  48       pha       Y nello stack
0179b  20 10 17 jsr $1710 banco 15
0179e  a5 fc    lda $fc   recupera com.
017a0  c9 a4    cmp #$a4  = to ?
017a2  d0 0b    bne $17af se no salta
017a4  68       pla       recupera Y
017a5  a8       tay
017a6  68       pla       recupera X
017a7  aa       tax
017a8  68       pla
017a9  20 50 14 jsr $1450 draw
017ac  4c 3c 17 jmp $173c jmp basic 7.0
017af  c9 50    cmp #$50  = p ?
017b1  d0 1a    bne $17cd se no salta
017b3  a9 38    lda #$38  sec
017b5  8d f8 13 sta $13f8
017b8  a9 00    lda #$00  #$00
017ba  8d fa 13 sta $13fa
017bd  a9 0d    lda #$0d  ora
017bf  8d 15 14 sta $1415
017c2  68       pla recupera Y
017c3  a8       tay
017c4  68       pla recupera X
017c5  aa       tax
017c6  68       pla
017c7  20 40 13 jsr $1340  p/u draw
017ca  4c 3c 17 jmp $173c jmp basic 7.0
017cd  a9 18    lda #$18  clc
017cf  8d f8 13 sta $13f8
017d2  a9 ff    lda #$ff  #$ff
017d4  8d fa 13 sta $13fa
017d7  a9 2d    lda #$2d  and
017d9  8d 15 14 sta $1415
017dc  4c c2 17 jmp $17c2 vai p/u draw
017df  c9 93    cmp #$93  = load ?
017e1  d0 0e    bne $17f1 se no salta
017e3  20 16 17 jsr $1716 valuta colon.
017e6  48       pha       salva colonna
017e7  20 10 17 jsr $1710 banco 15
017ea  68       pla       load colonna
017eb  20 64 16 jsr $1664 esegue load
017ee  4c 3c 17 jmp $173c jmp basic 7.0
017f1  c9 94    cmp #$94  = save ?
017f3  d0 f9    bne $17ee se no jmp basic 7.0
017f5  38       sec       c'e' l'area
017f6  a5 2e    lda $2e   grafica ?
017f8  e9 3f    sbc #$3f
017fa  90 f2    bcc $17ee se no jmp basic 7.0 -> sintax error
017fc  20 16 17 jsr $1716 valuta colon.
017ff  48       pha       salva colonna
01800  20 10 17 jsr $1710 banco 15
01803  68       pla       load colonna
01804  20 60 15 jsr $1560 esegue save
01807  4c 3c 17 jmp $173c jmp basic 7.0
```

che il banco usato è il 14 (io credevo fosse il 15 e che fatica per accorgersene!), basta disassemblare il pezzo iniziale della routine di CHRGET, già menzionata, routine che inizia a $0380.

```
00380 E6 3D     INC $3D
00382 D0 02     BNE $0386
00384 E6 3E     INC $3E
00386 8D 01 FF  STA $FF01
00389 A0 00     LDY #$00
0038B B1 3D     LDA ($3D),Y
0038D 8D 03 FF  STA $FF03
```

Capire questo pezzo di routine è facile, ma non capivo il perché degli STA $FF0X in $00386 ed in $0038D.

Infatti il registro FF00 NON È IL REGISTRO DELL'MMU: esso è solo una copia del vero registro dell'MMU posizionato a D500 nei banchi che contengono l'I/O. O meglio, dei registri. Eggià, perché la MMU è composta da vari registri a partire da D500, ma a me interessano solo quelli che vanno da D500 a D504 e che sono «copiati» in FF00-FF04 in ogni banco. A dire il vero il registro FF00 è identico a D500, ma i registri FF01-FF04 non sono identici agli originali D501-D504. Se infatti noi li leggiamo, i registri FFII contengono lo stesso valore del corrispondente D5II, ma se facciamo una «POKE» in uno di essi, otteniamo il risultato di sbattere in D500, e quindi in FF00, ciò che contiene il corrispondente registro D5II, ed il contenuto del registro FFII rimane invariato (questo non avviene se operiamo sui registri D5II). Perciò una STA $FF01, manda in FF00 il contenuto di D501, e cioè seleziona il banco 0, mentre una STA $FF03, manda in D501 il contenuto di D503, e cioè seleziona il banco 14.

Questo inconveniente ha richiesto di salvare il banco nel quale ci si trova (istruzioni $1724-$1727) per poi ri-settarlo prima di uscire (istruzioni $173C-$173E). In compenso l'idea di usare il carattere & per identificare i nuovi comandi è risultata particolarmente facile ed anche abbastanza breve. Purtroppo non è tutto oro quello che luccica e anche così nascono problemi. Il primo problema è che prima di ogni nuova istruzione ci vuole ob-

Pubblichiamo il listato di due programmi che utilizzano queste routine: uno è il vostro linee 1, il secondo mostra un possibile utilizzo dei comandi & SAVE e & LOAD per disegnare cerchi in 640 × 200. Bisogna notare che essi vanno fatti funzionare sulle 40 colonne se non li si vuole modificare, e che occorre premere i tasti cursore <— /—> per visualizzare e spostare il video ed il tasto E per uscire.

Demo Linee

```
100 :& GRAPHIC 2:GRAPHIC 1,1
110 X=100:Y=100
120 :& DRAW P X,Y:
130 Y=199*RND(1):& DRAW TO X,Y:
140 X=639*RND(1):& DRAW TO X,Y:
150 GET A$:IF A$="" THEN 130
160 IF A$="(LEFT)" AND X0>9 THEN X0=X0-10
170 IF A$="(RGHT)" AND X0<631THEN X0=X0+10
200 IF A$="E" THEN GRAPHIC 0:END:ELSE:& SAVE X0:GOTO 130
```

Demo Cerchi

```
100 :& GRAPHIC 2::GRAPHIC 1,1
110 FOR I=20 TO 300 STEP 10
120 CIRCLE 1,320,100,I,I/3
130 NEXT:& LOAD 00::SCNCLR 1
140 FOR I=20 TO 300 STEP 10
150 CIRCLE 1,000,100,I,I/3
160 NEXT:& LOAD 40:
170 DO:GET KEY A$
180 IF A$="(RGHT)" AND X<31 THEN X=X+10
190 IF A$="(LEFT)" AND X>09 THEN X=X-10
200 :& SAVE X::LOOP UNTIL A$="E"
210 GRAPHIC 0
```

bligatoriamente il : altrimenti il computer non riconosce il nuovo comando, il secondo, generato dal modo che uso per ritornare al Basic 7.0 è che anche dopo ogni comando è necessario il :, e quindi tra due nuovi comandi continui ci vorranno i doppi duepunti [::]. Il terzo problema è dato dalla memoria disponibile: mettere da $1300 a $18FF qualcosa di decente non è facile, soprattutto se si pensa di espandere ulteriormente la routine (come «suggeriva» AdP). Il voler ficcare tutto nell'esiguo spazio da $1710 a $1809 ha portato alla quasi totale mancanza di gestione degli errori: quando il nuovo interprete non capisce o non può eseguire un comando, o lo esegue in modo errato (routine DRAW), o emette un SINTAX ERROR (routine SAVE, quando non è stata riservata la memoria per il video 320 × 200). Per il resto la routine funziona egregiamente, non è sensibile agli spazi (grazie CHRGET) e, permettendo di dare la coordinata X senza spezzarla in due, si rivela abbastanza comoda ed utile da usare.

Ora un commento sulle routine Commodore impiegate:

$0380: Routine che preleva un carattere dal testo Basic; usa come puntatore $3D-$3E.

$AF96: Routine che valuta un'espressione aritmetica e ne pone il valore nell'accumulatore I floating point ($63-$69).

$AF0C: Routine che converte il valore presente nell'accumulatore I in intero compreso tra 0 e 65535 e lo pone in $16-$17.

$4AA2: È la normale routine che esegue un comando, la chiamo per uscire (ecco perché ci vuole un: dopo il comando, per farlo eseguire a questa routine!) in modo da rientrare in modo abbastanza «pulito» nel Basic 7.0.

Diamo ora qualche accenno sui nuovi comandi e la loro sintassi:

:& GRAPHIC 0: = Salva la pagina grafica (se è stata riservata memoria sufficiente), indi setta il modo testo.
:& GRAPHIC 1: = Setta la pagina 640 × 200 e recupera l'ultima utilizzata da $1C000-$1FEFF.
:& GRAPHIC 2: = Setta la pagina grafica e la pulisce.
:& DRAW P X,Y: = Plotta il punto X,Y e definisce le coordinate di partenza di DRAWTO.
:& DRAW U X,Y: = Cancella il punto X,Y e definisce le coordinate di partenza di DRAWTO.
:& DRAW TO X,Y: = Se il comando precedente aveva plottato, allora plotta la retta che congiunge il punto precedentemente plottato per ultimo con quello di coordinate X,Y; se invece il comando precedentemente aveva cancellato, allora anziché plottare cancella la retta tra i due punti visti prima.
:& SAVE C: = Trasporta lo schermo 640 × 200 a partire dalla colonna C (0 < = C < = 40) e per 320 punti, sullo schermo 320 × 200.
:& LOAD C: = Esegue l'operazione inversa della precedente.

Questi nuovi «comandi» si abilitano con

```
BANK 15 : SYS 5460
```

Se inoltre si vuole usare la routine & SAVE, bisogna dare un GRAPHIC1,1 del normale Basic 7.0 (se non capite il perché del secondo 1, provate ad ometterlo!). Se inoltre si vuole salvare la pagina 640 × 200 nella zona $1C000-$1FEFF, bisogna dare

```
POKE 58,191 : CLR
```

Bisogna stare attenti, perché il comando & GRAPHIC 0: non esegue il salvataggio se non si è dato quel comando prima (controlla il puntatore 57-58), mentre la routine richiamabile con SYS5812 non controlla niente e pasticcia le variabili del Basic.

MC

Routine Kernal

Commenteremo brevemente alcune routine Kernal non trattate precedentemente. Come noto, chi programma in linguaggio macchina, ha quasi sempre la necessità di interagire col sistema operativo della macchina per compiere operazioni implementate da questo. Un esempio classico sono le funzioni di I/O, cosi come l'editing, la gestione della memoria, del video o della tastiera.

Nome: **C64 MODE** - Indirizzo: **$FF4D**
Descrizione: Permette di passare al modo 64 da linguaggio macchina. Nessuna conferma è richiesta da parte del sistema.

Nome: **PHOENIX** - Indirizzo: **$FF56**
Descrizione: Provoca la partenza a freddo della macchina: Se il drive è collegato al 128 è tentato il boot del dischetto in esso contenuto.

Nome: **SWAPPER** - Indirizzo: **$FF5F**
Descrizione: Equivale alla sequenza ESC + X: permette di dirigere l'output sullo schermo 40 o 80 colonne.

Nome: **DLCHR** - Indirizzo: **$FF62**
Descrizione: Copia il generatore dei caratteri nella video ram del processore video a 80 colonne 8563. Si usa per ripristinare i caratteri dopo aver usato la grafica 640 × 200.

Nome: **CINIT** - Indirizzo: **$FF81**
Descrizione: Inizializza i due processori video (40 e 80 colonne), i tasti funzione, pulisce i due schermi video e mostra il cursore nello schermo selezionato dal tasto 40/80 DISPLAY.

Nome: **RESTOR** - Indirizzo: **$FF8A**
Descrizione: Inizializza i vettori di sistema locati a partire dall'indirizzo esadecimale $0314.

Nome: **READST** - Indirizzo: **$FFB7**
Descrizione: Immette nell'accumulatore il valore della variabile di stato ST.

Nome: **SETTIM** - Indirizzo: **$FFDB**
Descrizione: Scrive nelle locazioni A0..A2 (orologio di sistema in 60-esimi di secondo) il contenuto dei registri A, X, Y.

Nome: **RDTIM** - Indirizzo: **$FFDE**
Descrizione: Immette nei registri A, X, Y il valore dell'orologio interno.

Nome: **STOP** - Indirizzo: **$FFE1**
Descrizione: Una chiamata a questa routine permette di stabilire (testando di seguito il bit di ZERO) se è stato premuto il tasto di STOP. Il bit è settato in caso affermativo, resettato in caso negativo.

Nome: **CLALL** - Indirizzo: **$FFE7**
Descrizione: Chiude tutti i file precedentemente aperti.