Mastercode

Sommario

Nel 1983 viene stampato un testo che sarà una pietra miliare nella storia del C64. Scritto a quattro mani da un programmatore BASIC professionista e un ingegnere elettronico esperto in Assembly 6502, propone non solo un completo assemblatore scritto interamente in BASIC V2, ma anche (come esempio applicativo) un vero e proprio BASIC Extender residente in RAM che implementa oltre una dozzina tra nuovi comandi e funzioni. Col presente articolo si propone un porting per CBM Prg Studio di tutto il codice Assembly relativo a tale Extender, aggiornato ed esteso con ulteriori comandi e comprensivo della correzione di un curioso ma fastidioso bug di progettazione dell'originale.

«Programming is one of the most difficult branches of applied mathematics; the poorer mathematicians had better remain pure mathematicians.»
(Edsger Dijkstra, 1930-2002)

1 Introduzione.

Il testo di cui discutiamo [LE83] è probabilmente noto ai più nell'edizione italiana Jackson Libri del 1985 «Il linguaggio macchina del Commodore 64» [LE85]. La «strana coppia» di autori è costituita da David Lawrence, programmatore BASIC professionista ed autore di numerosi testi di informatica, e Mark England, ingegnere elettronico esperto in Assembly.

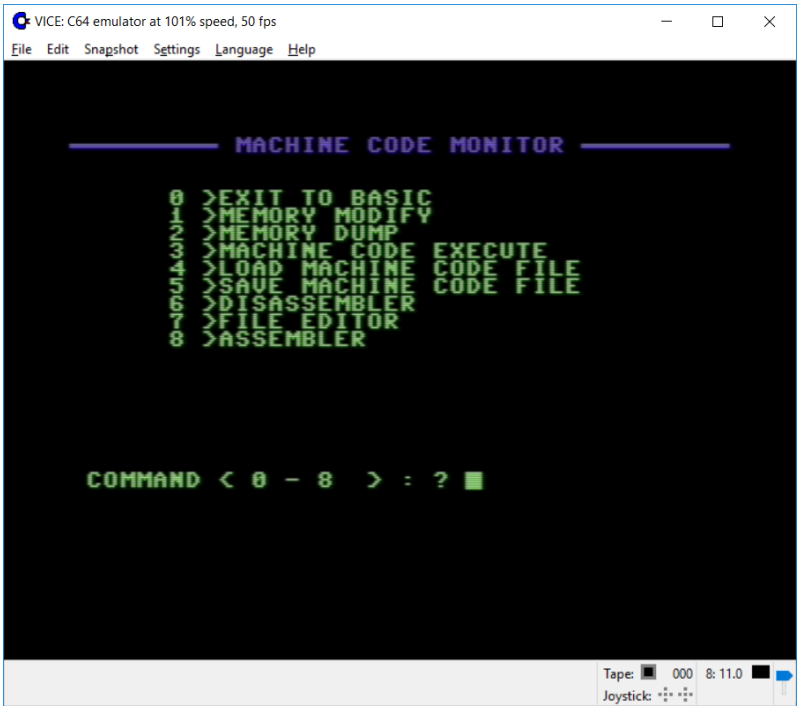
L'idea di un libro dall'approccio così innovativo, che ancora oggi rimane unico nello sterminato panorama della letteratura tecnica applicativa per C64, è venuta a Mark England: evidentemente stufo (come molti di noi) di studiare quei testi della prima ondata sull'Assembly dei PET che si limitavano a duplicare le informazioni su istruzioni, opcode e operandi già presenti nel datasheet della CPU 6510 proponendo poi, nel migliore dei casi, una fantasiosa silloge di brevi routine assembly di scarso o nullo interesse pratico.

Altro punto critico della maggior parte dei testi sull'Assembly della prima metà anni Ottanta: presupponevano che il lettore avesse facile e immediato accesso ad un ambiente assemblatore, cosa in realtà nient'affatto scontata all'epoca, visti anche i costi di licenza dei principali ambienti di sviluppo disponibili in quel periodo.

Nasce così l'idea di sviluppare un completo assemblatore scritto interamente in BASIC, che il lettore potrà digitare gradualmente come salutare esercizio, aiutato anche da un verificatore di checksum: e questa è solo la prima parte del divertimento, perché gli «esempi applicativi», invece di essere il solito inconcludente florilegio di scarne routine aritmetiche e di I/O, sono in realtà una completa applicazione che guida il lettore nei meandri dell'interprete BASIC e dell'interazione al più basso livello col Kernal, aggiungendo ben 15 nuovi comandi e funzioni al CBM BASIC V2.

2 Mastercode.

La prima parte del testo è interamente occupata dai listati dell'Assembler Mastercode, il quale oggigiorno riveste unicamente interesse storiografico e didattico: digitarne il codice per intero, possibilmente su una macchina reale, costituirebbe un ottimo esercizio per molti neofiti. Si tratta di un ambiente modulare, basato su menu, fondamentalmente completo nell'impostazione ma molto limitato sia nella sintassi Assembly accettata, sia nelle prestazioni. Il suo pregio fondamentale è stato quello di conferire una totale autonomia e indipendenza al testo rispetto ai costosi ambienti commerciali all'epoca esistenti. Non occorre però neppure rimarcare che l'editor di memoria, il disassembler e il monitor risultano realmente utilizzabili solo in un ristretto novero di casi, dal momento che risiedono nella memoria dedicata al codice BASIC, e risentono di limitazioni drastiche sia rispetto ai monitor software dedicati, sia a maggior ragione rispetto a quelli su cartuccia, spesso perfino rilocabili e con caratteristiche decisamente avanzate. L'immagine seguente mostra l'aspetto originale del menu principale di Mastercode.



3 BASIC Extender.

La programmazione di un BASIC Extender differisce profondamente dalla normale programmazione applicativa e richiede una solida consapevolezza di numerosi dettagli inerenti il codice in ROM (Kernal e BASIC). Una piena comprensione del codice richiederà, oltre allo studio del testo in questione, anche la consultazione di mappe di memoria e disassembly commentati come [Hee84], [Hee85], [Lee87] e una conoscenza dei meccanismi fondamentali dell'interprete residente.

I comandi e le funzioni del CBM BASIC V2 constano di parole chiave o *keyword*, quelle che normalmente digitiamo nei programmi e al prompt interattivo, come **LIST** o **PRINT**. Codeste stringhe però non vengono memorizzate letteralmente nel programma, per ovvi motivi di ottimizzazione in spazio: ciascuna di esse è invece sostituita da un codice univoco, rappresentato da un piccolo intero (un singolo byte) detto *token*. Tale valore numerico è caratterizzato dall'avere il bit più significativo posto a 1, per evitare ambiguità con altri elementi sintattici previsti dal linguaggio, come nomi di variabile o operatori. In fase di esecuzione, il *token* viene utilizzato per indicizzare correttamente la *jump table* o tabella dei vettori che contiene gli *entry point* delle routine assembly che implementano i singoli comandi BASIC; durante il listing il valore del *token* viene invece usato per un lookup inverso sulla tabella delle keyword, in modo che l'utente veda esattamente il comando che ha digitato in fase di immissione, rendendo del tutto trasparente il meccanismo di tokenizzazione o *crunching*.

Per gli usuali motivi di ottimizzazione in spazio (8kib di ROM erano pochi anche all'epoca), la tabella delle keyword è memorizzata usando uno dei tanti accorgimenti intelligenti dell'era Commodore: invece di sprecare un byte ponendolo a zero per ogni entry, rendendola così null-terminated o ASCII-zero che dir si voglia, il bit più significativo MSB dell'ultimo carattere è semplicemente posto a 1. Ciò, ovviamente, limita drasticamente ad ASCII-7 (in realtà PETSCII-7) il range dei caratteri effettivamente rappresentabili, il che tuttavia non costituisce affatto un problema per tale tipo di applicazione. Si tratta di una prassi talmente radicata che molti assemblatori offrono una apposita direttiva a tale scopo, come la *.shift* del Turbo Assembler.

Tenendo in mente quanto appena succintamente ricordato, i principali problemi che ci si trova ad affrontare nel progettare una espansione BASIC che rispetti il più possibile l'esistente sono i seguenti:

1) Il CBM BASIC risiede su una memoria di sola lettura (ROM o EPROM), tuttavia per rendere possibile l'interoperabilità con l'Extender occorre modificarne il codice in vari punti.

2) La tabella delle keyword standard ha dimensione limitata a 256 bytes, in quanto viene scandita entro l'interprete originale in modalità di indirizzamento indicizzato tramite il registro Y, ovviamente a 8 bit. Sfortunatamente, tale tabella risulta inutilizzabile ai fini di una espansione, in quanto già quasi totalmente occupata.

3) Occorre scrivere il codice Assembly per ciascuno dei nuovi comandi aggiunti all'Extender, in modo ovviamente compatibile con il firmware residente sulla macchina.

Il punto 1) (che su quasi ogni altro PET sarebbe l'epitaffio di ogni tentato progetto del genere) è pressoché insignificante nel nostro caso, dal momento che il C64 possiede infatti 64kib effettivi di memoria RAM contigua ed indiscriminatamente accessibile, alcuni segmenti della quale sono rimappati su ROM o port di I/O. Ma è possibile (e molto facile) copiare integralmente il contenuto della ROM nella RAM "sottostante", abilitando poi definitivamente quest'ultima e rendendo modificabile ogni byte copiato, ovviamente solo entro la sessione corrente.

La sola copia in RAM, tuttavia, non risolve il problema della tabella delle keyword e quello parallelo della corrispondente tabella dei vettori (come già ricordato, i puntatori alle singole routine corrispondenti a ciascuna keyword), la quale è parimenti quasi piena. Vi è poi un ulteriore problema: i *token* già occupati dalle keyword standard vanno dal 128 al 202, più 255 che rappresenta il pigreco. In considerazione di tutti questi vincoli, è strettamente necessario predisporre altre indipendenti tabelle, ed alterare parzialmente (seppure in modo minimo) il codice dell'interprete esistente, come già anticipato. Fortunatamente, l'intelligenza e la lungimiranza dei progettisti Commodore hanno fatto sì che buona parte del codice coinvolto sia revettorizzabile con estrema semplicità, in quanto puntato da variabili collocate entro i primi 2 kb di RAM, nell'area di sistema. In particolare, il vettore a 16 bit alla locazione 306h (ossia agli indirizzi 306 e 307 esadecimali) indirizza la routine PRTTOK, e risulta facilmente reindirizzabile per i nostri scopi. I *token* per le nuove keyword assumeranno valori da 203 in su, il che li rende immediatamente distinguibili da quelli standard.

Per le tabelle aggiuntive, la soluzione adottata dalla maggior parte dei BASIC Extender (incluso quello di cui discutiamo) consiste nell'uso di tabelle alternative indirizzate dinamicamente. Si deve partire dal presupposto che la routine di tokenizzazione («crunching» nella letteratura Commodore) del BASIC V2 non è particolarmente sofisticata, né tantomeno ottimizzata per le prestazioni: invece di ricorrere a strutture dati dedicate (come un albero binario) e algoritmi efficienti, si limita ad effettuare una banale ricerca lineare sulla tabella delle keyword usando come chiave la potenziale keyword attualmente in elaborazione. Un semplice contatore, inizializzato a monte del loop di scansione, viene incrementato ad ogni mancata corrispondenza, ed è proprio tale contatore che al termine della routine (salvo errori di sintassi) conterrà il valore del *token* corrispondente alla keyword esaminata nella linea BASIC corrente.

Oltre a questa struttura di base, decisamente elementare, vi sono poi nel codice dell'interprete residente varie euristiche per gestire le eccezioni e le singolarità del linguaggio: tra questi, il trattamento del separatore ':' che non è un *token* vero e proprio, ma viene gestito separatamente nel codice del cruncher/tokenizer.

La tabella aggiuntiva delle keyword viene indirizzata dinamicamente da una apposita routine CRUNCH nel codice di Lawrence & England, quando la keyword correntemente analizzata non ha trovato riscontro nella tabella originale. Di fatto, la scansione di ogni singola keyword e l'associazione con il relativo codice numerico (tokenizzazione) in una linea di codice seguono uno schema logico estremamente semplice e lineare:

1. Il tokenizer confronta la stringa della potenziale keyword presente nella linea di codice correntemente analizzata con le keyword della tabella originale del BASIC V2 (con inizio alla locazione $A09E): END, FOR, NEXT, DATA, ...

2. Se il confronto va a buon fine, si esce dal loop principale del tokenizer. A quel punto il numero contenuto in $0B (con l'aggiunta di un offset pari a $80, che per convenzione caratterizza i token BASIC) è il valore del *token* corrispondente al comando trovato nella linea di codice BASIC appena analizzata e viene quindi memorizzato assieme al resto della riga di codice.

3. Ad ogni confronto fallito il *token counter* (contenuto nella locazione di pagina zero $0B) viene incrementato e si passa alla keyword successiva in tabella.

4. Se al termine del loop la stringa non ha trovato corrispondenza, la scansione della tabella delle keyword è giunta a fine e il *token counter* ha raggiunto il suo valore massimo per il BASIC standard, pari nel nostro caso a 202 ($CA). Normalmente a questo punto si avrebbe un errore sintattico, ma in questo caso potrebbe ancora trattarsi di un nuovo comando. Una piccola modifica al codice ROM originale impone quindi di richiamare la routine CRUNCH sopra menzionata, la quale sostituisce dinamicamente nel tokenizer l'indirizzo di partenza della tabella delle keyword con il valore della label KEYWRD nel nostro sorgente (di norma $C000) e lo costringe ad eseguire un nuovo loop di scansione, ricominciando dal punto 1 ma con una differente tabella delle keyword e **senza azzerare il token counter**. In questo modo, per costruzione, è garantita l'assegnazione di token nell'intervallo riservato all'Extender, dal 203 in poi.

5. Se uno dei confronti all'interno della tabella delle keyword estese va a buon fine, siamo in presenza di una keyword dell'Extender, che viene regolarmente tokenizzata. In caso contrario: **SYNTAX ERROR**.

In fase di esecuzione viene implementato un meccanismo analogo per la scelta della tabella dei vettori, in questo caso semplificato dalla immediata discriminazione tra gli intervalli di valori dei *token* standard ed estesi.

La soluzione originariamente prevista da Lawrence e England prevedeva un totale di 15 nuove keyword, di cui tre funzioni, oltre a una coppia di istruzioni puramente rappresentative (FAST e SLOW) che fin dall'inizio l'autore del presente articolo non ha mai ritenuto opportuno implementare. La tabella seguente riassume sinteticamente le nuove funzionalità disponibili: poiché alcuni comandi hanno un numero elevato di parametri, per esigenze tipografiche si è scelto di ometterli quando potevano compromettere la leggibilità.

UNDEAD	Ripristina un programma BASIC dopo un (accidentale) comando NEW.
SUBEX	Elimina l'ultimo indirizzo di ritorno dallo stack.
RKILL	Compressione sorgente BASIC, elimina spazi e REM.
DOKE <addr>,<val>	Versione a 16 bit della POKE.
PLOT <row>,<col>	Posiziona il cursore alle coordinate specificate.
DELETE <inizio>,<fine>	Cancella intervalli di linee contigue da un sorgente BASIC.
BSAVE <...>	Salva un blocco di memoria su file.
BLOAD, BVERIFY <...>	Carica (verifica) un blocco di memoria da file.
MOVE <to>,<from>,<len>	Copia un blocco di memoria ad un nuovo indirizzo.
FILL <start>,<len>,<val>	Riempie un blocco di memoria con il valore dato.
RESTORE <line>	Versione estesa della RESTORE originale.
VARPTR(<var>)	Restituisce un puntatore alla variabile referenziata.
YPOS()	Complementare alla POS() del BASIC V2, restituisce la riga del cursore.
DEEK(<addr>)	Complementare a DOKE, è una PEEK() a 16 bit.

Nella implementazione originale era previsto, oltre al file binario prodotto da Mastercode e contenente il codice della vera e propria estensione BASIC, un loader scritto in BASIC V2 che si occupava del caricamento da disco (o nastro), della copia da ROM a RAM del BASIC V2 originale, della rilocazione dell'estensione a partire dall'indirizzo $C000 e della revettorizzazione.

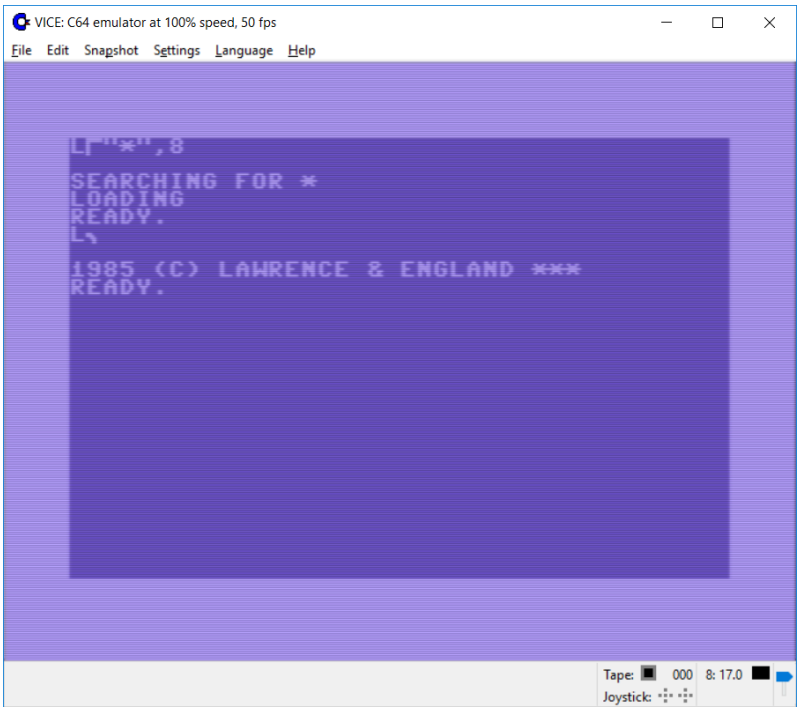
4 Porting ed estensione del progetto.

Nel 1985, dopo avere pedissequamente digitato tutto il codice originale proposto dal testo, si è provveduto al porting dei sorgenti dell'Extender su Turbo Assembler, in un unico file. Tale lavoro mirava a superare le inerenti limitazioni sintattiche di Mastercode, rendendo più facile la manutenzione e l'espansione del progetto. Tutte le operazioni di caricamento e predisposizione dell'ambiente possono infatti essere centralizzate ed eseguite direttamente in Assembly senza il minimo problema, con un singolo eseguibile «mascherato» da programma BASIC ovvero caricato interamente da disco a partire dalla locazione di default $0801 e successivamente eseguito con un semplice comando RUN. Il codice di startup elaborato all'epoca è qui riportato in una moderna versione per CBM Prg Studio.

```
;************************************
; Codice "universale" per startup BASIC
;************************************
*=$0801
;------------------------------------
        WORD BASEND                     ; Indirizzo della prossima linea
        WORD 1985                       ; Numero di linea -> anno di stesura...
        BYTE $9E                        ; Token per "SYS"
        TEXT "2102:"                    ; Indirizzo di avvio: $0836
        BYTE $8F                        ; Token per "REM"
        BYTE $22                        ; Token per '"'
; Rendo illeggibile il listing con una serie di DEL ($14)
        BYTE 20,20,20,20
        BYTE 20,20,20,20
        BYTE 20,20,20,20
; Solo la stringa seguente apparira' nel listato:
        TEXT "(c) lawrence & england ***"
```

```
        BYTE 0                          ; Terminatore di linea
BASEND  WORD 0                          ; Terminatore di programma
;-------------------------------------
;*************************************
;** Inizio codice applicazione      **
;*************************************
```

Esaminando il codice il lettore più smaliziato noterà come, già in quel lontano 1985, l'allora giovane programmatore univa costantemente il divertimento all'apprendimento. Andando oltre la banale funzionalità di avvio da BASIC del codice Assembly, il listato con pochi bytes aggiuntivi risulta anche «protetto» da sguardi indiscreti (un programmatore minimamente esperto userà comunque un monitor con disassembler per esaminare a piacimento il codice dopo il caricamento) con una tecnica all'epoca universalmente diffusa per i software da edicola e commerciali. Se si esegue un LIST subito dopo il caricamento del codice, si ottiene unicamente quanto illustrato nello screenshot seguente:



La sezione di codice successiva si occupa delle operazioni preliminari necessarie alla modifica del BASIC residente su ROM: copia il BASIC nella RAM sottostante e in seguito modifica alcuni vettori, reindirizzando alle nuove routine dell'espansione che ne permettono la coesistenza col BASIC originale.

```
;*************************************
; Espansione BASIC / RELEASE 2.0
;*************************************

;-------------------------------------
; Rilocazione codice in $C000
;-------------------------------------
START   LDA #<LDR_END+1
        STA MVSTART
        LDA #>LDR_END+1
        STA MVSTART+1
```

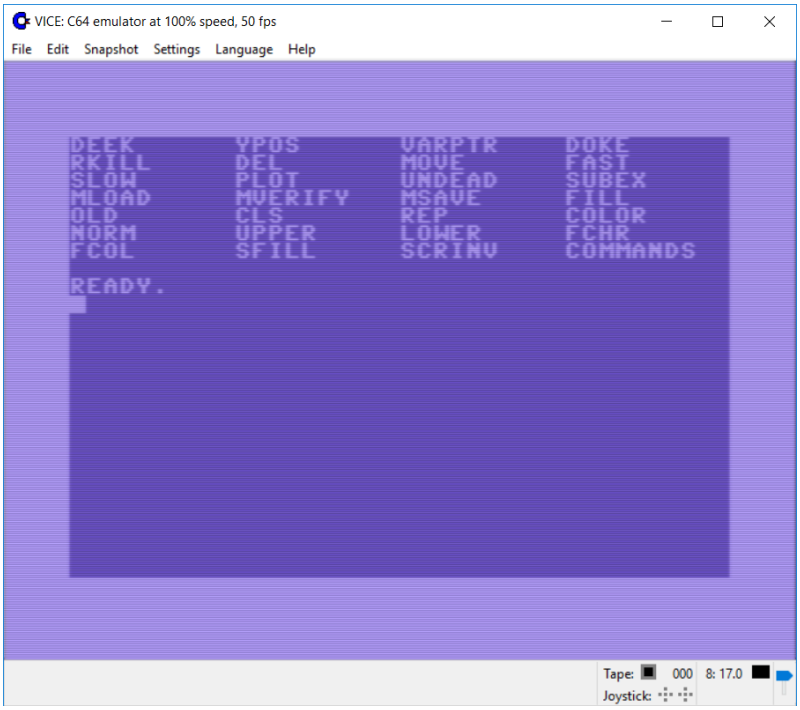
```
            LDY #$00
            STY MVDEST
            LDA #$C0
            STA MVDEST+1
            LDX #LEN

MOVE1       LDA (MVSTART),Y
            STA (MVDEST),Y
            INY
            BNE MOVE1
            INC MVSTART+1
            INC MVDEST+1
            DEX
            BNE MOVE1
;-----------------------------------------
; Modifiche all'interprete BASIC C64:
; copia BASIC ROM in RAM e revectoring
;-----------------------------------------
            LDA $01         ; Seleziona la ROM BASIC
            ORA #$01
            STA $01
            LDY #$00

; Non occorre modificare il registro $01 ad ogni lettura:
; l'hardware del C64 e' progettato per scrivere
; comunque su RAM, anche se viene selezionata la ROM.
BSTART      LDA BASIC_START,Y
BDEST       STA BASIC_START,Y
            INY
            BNE BSTART
            INC BSTART+2
            INC BDEST+2
            LDX BDEST+2
            CPX #$C0
            BNE BSTART

MODIFY      LDA $01         ; Seleziona la RAM in $A000-$BFFF
            AND #$FE
            STA $01
            LDX #_JMP
            STX $A7E1
            LDA #<EXECUT
            STA $A7E2
            LDA #>EXECUT
            STA $A7E3
            STX $AFAD
            LDA #<FUNEVL
            STA $AFAE
            LDA #>FUNEVL
            STA $AFAF
            STX $A604
            LDA #<CRUNCH
            STA $A605
            LDA #>CRUNCH
            STA $A606
            LDA #<PRTTOK
```

```
            STA  $0306
            LDA  #>PRTTOK
            STA  $0307
            LDA  #<RESTORE-1
            STA  $A024
            LDA  #>RESTORE
            STA  $A025
             ...
LDR_END     BYTE 0
```

L'estensione BASIC, nella nuova versione per Turbo Assembler, è poi stata arricchita con ulteriori nuovi comandi, di ispirazione didattica o mutuati da altre estensioni BASIC. Vale la pena di sottolineare nuovamente che questo progetto ha un valore precipuamente didattico e non presenta la benché minima velleità di porsi al livello di un Simon's BASIC o di qualsiasi altro Extender commerciale.

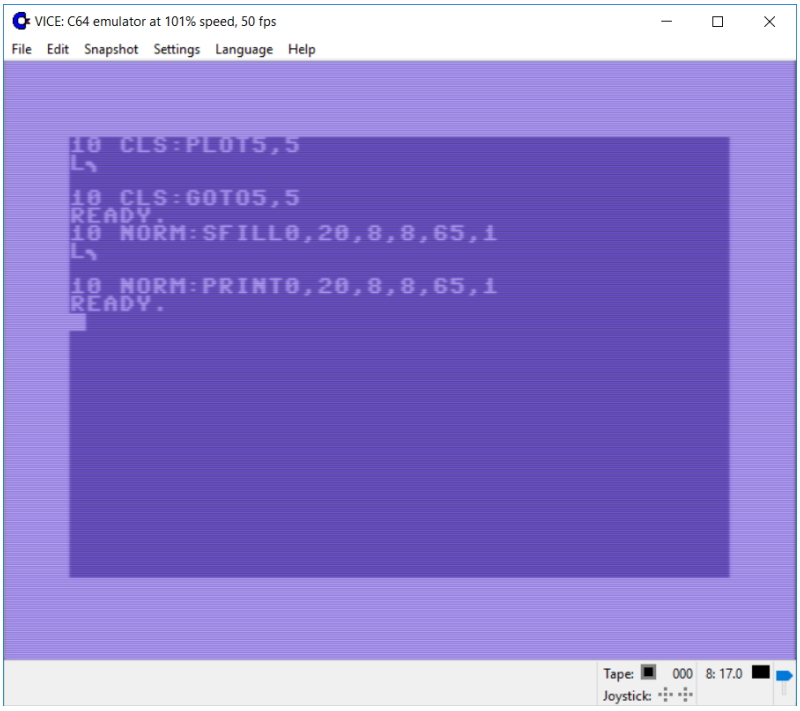
CLS	Cancella lo schermo.
COLOR <sfondo>,<bordo>,<testo>	Imposta i colori usando i codici CBM.
COMMANDS	Elenca a video tutti i nuovi comandi e funzioni dell'estensione.
FCHR <...>	Riempie con un carattere un'area rettangolare a video.
FCOL <...>	Riempie con il colore dato un'area rettangolare a video.
LOWER	Imposta il set di caratteri minuscolo.
NORM	Comoda scorciatoia per impostare colori e case di default.
REP	Imposta ciclicamente l'autorepeat della tastiera.
SCRINV <...>	Inverte sfondo e testo in un'area rettangolare a video.
SFILL <...>	Versione specializzata di FILL per la memoria video.
UPPER	Complementare a LOWER, imposta il set maiuscolo.

Si è inoltre introdotto un comando OLD, semplice e più sobrio sinonimo per UNDEAD. L'immagine seguente mostra il risultato dell'esecuzione di COMMANDS dopo il caricamento dell'estensione.



5 Un curioso bug.

All'epoca l'aggiunta di nuovi comandi all'Extender rivelò presto un problema non previsto dagli autori, in quanto il bug apparentemente non era riproducibile nel contesto delle 14+3 keyword originali. L'immagine seguente mostra in modo immediato il problema:



Come si vede, esisteva un problema di errata tokenizzazione: il comando PLOT viene tokenizzato e quindi listato come GOTO, mentre la PRINT prende il posto di SFILL, e così via. Questo nel caso in cui sulla riga di programma siano presenti due o più nuove keyword.

In base ai meccanismi accennati al paragrafo §3, si possono evidentemente creare situazioni di concatenazione di nuovi comandi su una singola linea a causa di cui l'associazione tra *token counter* e indirizzo della tabella delle keyword non è più sincronizzata: ciò avviene perché esistono (rari) percorsi di esecuzione nel codice del tokenizer ROM che non transitano dal punto di iniezione in cui viene richiamato il codice aggiuntivo CRUNCH previsto dagli autori. Ci si trova quindi in una situazione incongruente in cui il *token counter* viene inizializzato a zero, ma nel codice del tokenizer viene ancora puntata la tabella delle keyword estese $C000 e non quella originale: ciò genera esattamente il tipo di errore evidenziato.

Effettuando tramite monitor un accurato reverse engineering del codice ROM interessato all'analisi di una linea di codice BASIC e alla tokenizzazione, si è quindi determinata la necessità di un ulteriore punto di iniezione nel codice del tokenizer stesso, in modo da garantire in qualsiasi caso la sincronizzazione tra l'inizializzazione del *token counter* (contenuto come abbiamo visto nella locazione di pagina zero $0B) e l'indirizzo della tabella delle keyword originali alla locazione $A09E. Il codice seguente, aggiunto al sorgente dell'estensione BASIC riveduta e corretta, risolve efficacemente anche tale bug.

```
;*************************************************
;** Il codice seguente non fa parte del testo ed
;** e' stato aggiunto per risolvere un curioso bug
;** del BASIC Extender. L'associazione originale tra
;** tabella delle nuove istruzioni e contatore usato
;** dal tokenizer BASIC non e' sufficientemente
;** robusta, perché esiste almeno un caso in cui
;** si crea una desincronizzazione, ad esempio
```

```
;** digitando una linea di programma come segue:
;** 10 CLS:SFILL 0,20,8,8,65,1
;** il secondo comando viene tokenizzato in modo
;** errato. Questo avviene, in breve, perche' gli
;** autori non hanno considerato tutti i possibili
;** percorsi di esecuzione nel tokenizer ROM,
;** cio' che di norma si attua effettuando una
;** analisi full code coverage con un tracer.
;** Per ovviare basta comunque un ulteriore punto
;** di iniezione, in corrispondenza della
;** inizializzazione del token counter. Si aggiunge
;** una brevissima routine che garantisce che
;** all'azzeramento del counter la tabella corrente
;** delle keyword sia quella originale del BASIC.
;*************************************************
         STX $A5AE
         LDA #<TOKSTR2
         STA $A5AF
         LDA #>TOKSTR2
         STA $A5B0
...

;*************************************************
;** Routine aggiuntiva per eliminare il bug
;** nel tokenizer.
;*************************************************

TOKSTR2  JSR PUTREG
         LDA #$A0
         LDX #$9E
         JSR TOKSTR
         JSR GETREG
         LDY #$00
         STY $0B
         JMP $A5B2
;*************************************************
```

6 Conclusioni.

Si è brevemente presentata la storia del BASIC Extender didattico di Lawrence & England, accompagnata da una succinta descrizione di una possibile espansione di notevole interesse per l'apprendimento e lo studio dell'Assembly. Si è anche illustrata l'insorgenza di un bug progettuale del tutto imprevisto e la sua risoluzione tramite reverse engineering, sempre lecitamente consentito quando si tratti di garantire l'interoperabilità del codice.

Il codice Assembly allegato, predisposto per il moderno ambiente di cross-development CBM Prg Studio, viene messo a disposizione per lo studio e la sperimentazione, con l'augurio che molti lettori studiandolo si sentano invogliati a progettare ed implementare nuovi comandi per migliorare i propri skill in Assembly e comprendere meglio l'interazione col codice BASIC e Kernal presenti in ROM. Il codice è brevemente commentato in alcuni punti salienti e quasi tutti gli indirizzi di memoria significativi (originariamente tutti hardcoded, senza distinzioni) sono referenziati tramite label descrittive, ma ci si attende comunque che il lettore studi il sorgente avendo a disposizione almeno i testi elencati in bibliografia per una comprensione puntuale.

Riferimenti bibliografici

[Hee84] D. Heeb, *Vic and commodore 64 tool kit: Basic*, Compute!, 1984.

[Hee85] Dan Heeb, *Compute!'s vic-20 and commodore 64 tool kit: Kernal*, Compute, 1985.

[LE83] David Lawrence and Mark England, *Commodore 64 machine code master: A library of machine code routines*, Reston Pub Co, 1983.

[LE85] ———, *Il linguaggio macchina del commodore 64. con floppy disk*, Jackson Libri, 1985.

[Lee87] Sheldon Leemon, *Mapping the commodore 64 & 64c*, Compute, 1987.