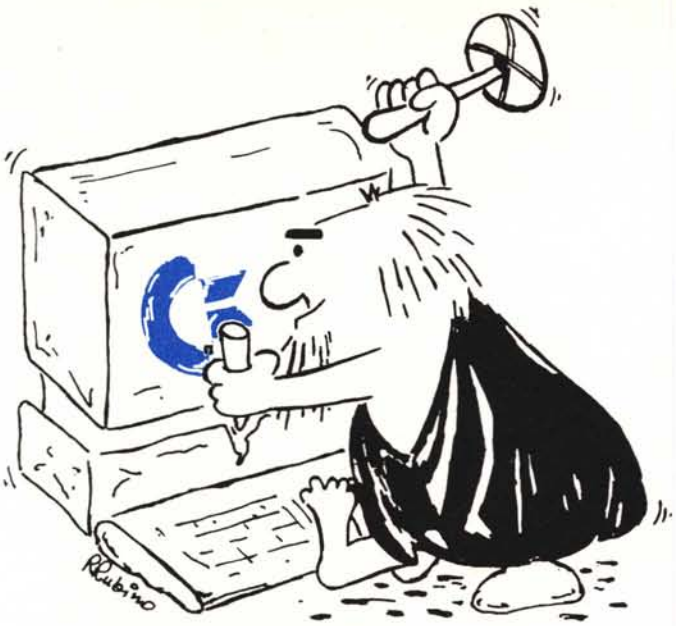

128 da zero

di Andrea de Prisco

Banking da linguaggio macchina

Dopo aver visto il mese scorso come il Commodore 128 mette a disposizione tutta la sua memoria (ram e rom) al livello del linguaggio di programmazione Basic, questo mese ci addentreremo maggiormente nel merito mostrando come siano possibili analoghi «smanettamenti» anche da linguaggio macchina.

Il Monitor

Per programmare in linguaggio macchina occorre un buon monitor: ovviamente i fosfori in questo caso non c'entrano proprio nulla. Ci serve un monitor per il linguaggio macchina, che stavolta mamma Commodore s'è ricordata di infilare dentro alle rom di sistema, come aveva già fatto tanti anni fa col dottor Pet. Un monitor di linguaggio macchina è dunque uno strumento indispensabile per poter manipolare facilmente le celle di memoria o per essere più precisi i contenuti di queste. Il monitor del 128 fa anche qualcosa in più: assembla e disassembla codici mnemonici e porzioni di memoria, converte con estrema facilità numeri da una base all'altra non senza permettere una ragionevole interazione tra computer e la più importante delle periferiche, l'unità a dischi, almeno per quanto riguarda directory e comandi DOS.

Tutto ciò è comunque ben illustrato sul manuale fornito con la macchina al quale naturalmente vi rimandiamo qualora non aveste mai usato questa feature.

Il registro CR della MMU

Ogni manipolazione della configurazione della Memoria coinvolge sempre la MMU del 128 il cui acronimo sta appunto per Memory Management Unit. Per dialogare con questa unità, come avviene anche per gli altri processori interni al 128, si usano dei registri mappati in alcune celle di memoria.

Noi faremo riferimento, per tutto quest'articolo, al registro CR (già nominato il mese scorso) locato all'indirizzo esadecimale $FF00. Sempre come già detto, tale registro è ovviamente accessibile da qualsiasi configurazione di memoria dato che deve essere sempre possibile passare da un banco ad un altro: se in qualche banco questo non fosse accessibile, una volta selezionato quel banco non si potrebbe più «venir via» non potendo dialogare con la MMU.

Di ciò si evince che per passare da un banco ad un altro, da linguaggio macchina basta infilare qualcosa nel registro CR per ottenere il voluto. Quasi.

128 da zero

Il problema infatti è un pò più complesso: infatti anche il flusso di controllo del programma in corso verrebbe catapultato indesideratamente sul banco desiderato. Facciamo un esempio: immaginiamo di aver scritto un programma in linguaggio macchina nel banco 0 della memoria. Sempre per ipotesi poniamo il caso in cui a un certo punto ci serve il contenuto della locazione 4000 del banco 1, interamente riempito di dati (ovvero non-programmi). La situazione è mostrata in figura 1: dopo qualche operazione si pone $7F nel registro CR per cambiare banco prima di prelevare col LDA la locazione 4000. Abbiamo combinato un bel pasticcio: infatti i cambiamenti di banco non riguardano solo gli accessi ai dati da parte delle istruzioni in linguaggio macchina ma quanto effettivamente il processore preleva per eseguire. Nella fattispecie, l'aver eseguito la sequenza:

```
LDX # $7F
STX $FF00
```

corrisponde in pratica (solo quelle due istruzioni) ad aver effettuato un salto, un JMP, all'altro banco di memoria, con le catastrofiche conseguenze che possiamo supporre: sicuramente il blocco del sistema fino a nuovo Reset. Questo perchè siamo saltati in mezzo ai dati (leggi: numeri a casaccio per il processore) e, si sa, non tutti i numeri compresi tra 0 e 255 sono codici operativi di istruzioni di macchina.

Considerato poi che la situazione di cui sopra è tutt'altro che irreale, se non ci fosse una soluzione sarebbe davvero un bel casotto. Fortunatamente di soluzioni ce ne sono due, una hardware e l'altra software.

Spegnete pure il saldatore, non dobbiamo fare nessuna modifica è tutto compreso nel prezzo. Hardware nel senso che ci riferiremo al modo come è stato costruito il 128 e Software rifacendoci ad opportune routine di sistema operativo. In ogni caso farina di mamma Commodore.

Figura 1

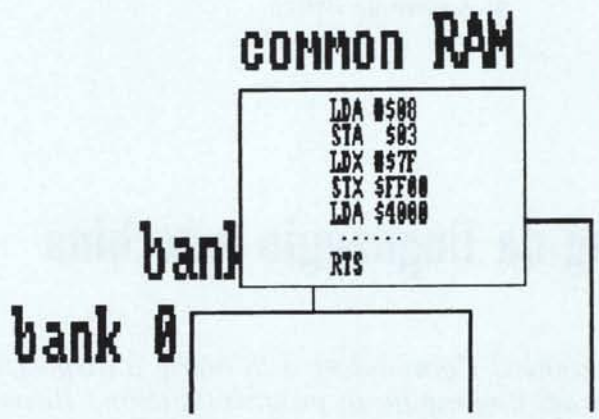

Figura 3

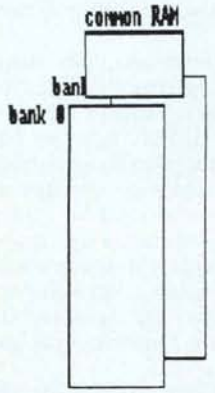

Figura 2

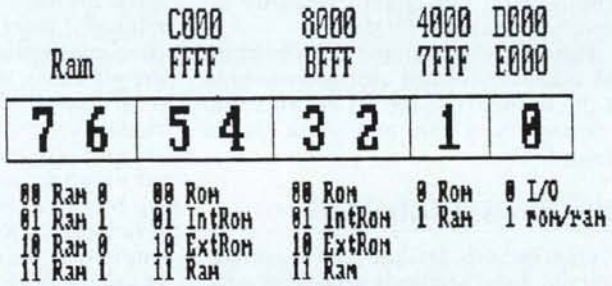

Figura 4: Registro CR della MMU

Prima soluzione

Alla base di tutti i cambiamenti di banco, oltre al registro CR esiste un altro componente altrettanto importante: il primo K di memoria RAM comune sia a RAM 0 che a RAM 1: le due pagine di memoria disponibili su 128. Ciò vuol dire essenzialmente che quanto mostrato in figura 1, se l'avessimo effettuato nelle prime 1024 locazioni di memoria non avrebbe provocato nulla di paranormale.

La figura 2 mostra come stanno i fatti: si nota come la cosiddetta Common RAM sia indipendente dal banco selezionato. Allocando le nostre routine nel primo K di memoria (figura 3), non si hanno tali problemi: di fatto la Commodore stessa usa questa zona per effettuare le sue interazioni tra pagine, a livello di sistema operativo e/o Basic 7.0.

L'unico problema è dato dal fatto che in questo primo K non troviamo molto spazio libero per i nostri programmi essendo quasi interamente occupato da variabili e sottoprogrammi di sistema operativo. In altre parole, se ci serve qualche buco qua e là otteniamo facilmente il nostro scopo, ma se servono grosse zone di memoria non possiamo non ricorrere al metodo software che andiamo subito ad illustrare.

Soluzione software

Lasciamo dunque il nostro primo K di memoria e scriviamo comodamente il nostro programma in linguaggio

Tabella 2
Corrispondenza tra numero banco e valore di CR

n.banco	valore cr
0	$3F
1	$7F
2	$BF
3	$FF
4	$16
5	$56
6	$96
7	$D6
8	$2A
9	$6A
10	$6A
11	$EA
12	$06
13	$0A
14	$01
15	$00

Tabella 1

Tabella riassuntiva delle routine del kernal per la manipolazione dei banchi di memoria

Routine	Descrizione	Indirizzo	Parametri IN	Parametri OUT
GETCFG	Trasforma il n.ro del banco nel corrispondente valore di CR	$F7EC 63438	X = # banco	A = Valore CR
JSRFAR	Chiama dal banco 15 una subroutine posta in qualsiasi banco	$02CD 717	$02 = # banco $03 = Hi addr. $04 = Lo addr. $05 = status $06 = registro A $07 = registro X $08 = registro Y	$05 = status $06 = registro A $07 = registro X $08 = registro Y $09 = stack pointer
JMPFAR	Salta dal banco 15 ad una locazione posta in qualsiasi banco	$02E3 739	$02 = # banco $03 = Hi addr. $04 = Lo addr. $05 = status $06 = registro A $07 = registro X $08 = registro Y	

INDFET	Legge il contenuto di una cella posta in qualsiasi banco	$F7D0 63440	A = Puntatore al puntatore al byte
			X = # banco
			Y = offset
				A = Byte cercato

INDSTA	Scrive un byte in una cella posta in qualsiasi banco	$F7DA 63450	A = Byte da scrivere
			$2B9 = Puntatore al puntatore alla cella
			X = # banco
			Y = offset	

INDCMP	Confronta l'accumulatore con una cella posta in qualsiasi banco	$F7E3 63459	$2CB = Puntatore al puntatore alla cella A = carattere da comparare X = # banco	Status register

Listato 1

```
. FF7EC  BD F0 F7 LDA $F7F0,X
. FF7EF  60       RTS
```

Listato 2

```
. 002CD  20 E3 02 JSR $02E3
. 002D0  85 06    STA $06
. 002D2  86 07    STX $07
. 002D4  84 08    STY $08
. 002D6  08       PHP
. 002D7  68       PLA
. 002D8  85 05    STA $05
. 002DA  BA       TSX
. 002DB  86 09    STX $09
. 002DD  A9 00    LDA #$00
. 002DF  8D 00 FF STA $FF00
. 002E2  60       RTS
. 002E3  A2 00    LDX #$00
. 002E5  B5 03    LDA $03,X
. 002E7  48       PHA
. 002E8  E8       INX
. 002E9  E0 03    CPX #$03
. 002EB  90 F8    BCC $02E5
. 002ED  A6 02    LDX $02
. 002EF  20 6B FF JSR $FF6B
. 002F2  8D 00 FF STA $FF00
. 002F5  A5 06    LDA $06
. 002F7  A6 07    LDX $07
. 002F9  A4 08    LDY $08
. 002FB  40       RTI
```

Listato 3

```
. FF7D0  8D AA 02 STA $02AA
. FF7D3  BD F0 F7 LDA $F7F0,X
. FF7D6  AA       TAX
. FF7D7  4C A2 02 JMP $02A2


. 002A2  AD 00 FF LDA $FF00
. 002A5  8E 00 FF STX $FF00
. 002A8  AA       TAX
. 002A9  B1 66    LDA ($66),Y
. 002AB  8E 00 FF STX $FF00
. 002AE  60       RTS
```

macchina dove ci pare (o quasi). Per interagire con gli altri banchi possiamo usare alcune routine di sistema operativo atte allo scopo.

La tabella 1 riassume tali routine che, come vedremo, sono tutte piuttosto semplici sia da capire che da usare. A tale scopo ci riferiremo anche ai listati in linguaggio macchina che le descrivono, presenti in queste pagine.

La prima routine serve per trasformare il numero di un banco (0-15) nel corrispondente valore da «pok-are» nel registro CR della MMU. Per usarla è sufficiente mettere in X il banco desiderato ed effettuare un JSR all'indirizzo $F7EC del banco 15. Al ritorno da questa, troveremo nel registro A il valore corrispondente. Dando uno sguardo al listato 1 possiamo notare quanto sia banale tale trasformazione che non è altro che la lettura di un valore in una tabella posta a partire dall'indirizzo $F7F0.

La seconda routine, mappata nel primo K (guardacaso) a partire dall'indirizzo esadecimale $02CD permette di chiamare una subroutine posta in qualsiasi banco a condizione però che la chiamata avvenga dal banco 15 (a tal proposito fra un po' dirò qualcosa). Prima di utilizzarla è necessario caricare un pò di celle di memoria con il desiderato, nella fattispecie metteremo nella cella $02 il numero del banco desiderato (0-15), nelle due successive l'indirizzo dove è posta la nostra subroutine, nelle celle $05 e seguenti rispettivamente il registro di stato e i registri A-X-Y coi quali desideriamo che venga eseguita la subroutine. Settati tali parametri, possiamo eseguire dal nostro banco (qualsiasi) JSR $02CD per ottenere quanto voluto. I risultati dell'esecuzione della subroutine li ritroveremo nelle celle $05-$09 come mostrato in tabella 1.

Proviamo ora a commentare quanto succede al momento della chiamata a $02CD, tenendo sott'occhio il listato 2. La prima operazione è una chiamata di subroutine all'indirizzo $02E3 (per la cronaca questo è l'indirizzo della prossima routine che mostreremo). Essenzialmente lì succede che le celle contenenti indirizzo e status vengono infilate nello stack (vedremo tra poco perché); di seguito a questo è effettuata la trasformazione da numero di banco desiderato a corrispondente valore di CR (JSR $FF6B corrisponde a saltare al listato 1, anche se con un passaggio in più) per poi aggiornarlo ovvero switch-are banco; infine sono caricati i registri A-X-Y coi corrispondenti valori immessi precedentemente

128 da zero

alla chiamata nelle celle $06-$08.

L'RTI di fondo quasi per magia effettua il salto alla subroutine desiderata, nonostante non vi sia stato alcun interrupt: è solo che alla Commodore sono molto furbi. Infatti l'RTI non fa altro che scaricare dallo stack un primo valore per porlo come status, di seguito a questo preleva i due successivi valori e, interpretati come un indirizzo di 16 bit, salta alla locazione così ottenuta.

Badaben-badaben-badaben che la subroutine a cui siamo diretti prima o poi terminerà con un RTS (se no, che subroutine è?) e la domanda da porci è naturalmente: dopo tutti questi spippolamenti dove mai torneremo?

Semplice, a $02D0 ovvero all'istruzione successiva al JSR $02E3 che abbiamo effettuato qualche centinaio di parole fa. In effetti ancora non è finita: occorre mettere in $06-$08 il contenuto dei registri A-X-Y, in $05 lo status e in $FF00 il valore 0 per ripristinare il banco chiamante, 15.

Apriamo, prima di continuare, una piccola parentesi: per la scrittura di articoli specifici come questi, è d'obbligo una massiccia documentazione in tema prima di cimentarsi in tali «prodezze». Nel caso di 128 da zero, dato che di programmer's references guide della Commodore, fino a questo momento manco a parlarne, il «riferimento» è un libro edito dalla Abacus Software denominato Tricks and Tips. Peccato che se non si sta veramente attenti a quello che c'è scritto si finisce per confondersi le idee più che senza libri affatto. Tanto per citarne una, la routine appena commentata è spacciata per «chiamata di subroutine da qualsiasi banco in qualsiasi banco» con tanto di cella $09 atta a contenere il banco chiamante... Oppure i 128 americani sono diversi... oppure è diverso il 128 del sottoscritto...

Bene, tornando alla tabella 1, la successiva routine permette di saltare a una qualsiasi locazione posta in qualsiasi banco. Inutile dirvi che non commenteremo il listato dato che esso corrisponde alla porzione $02E3-$02FB che abbiamo già mostrato. Passiamo oltre: tocca a INFET, mappata a partire dall'indirizzo $F7D0 (vedi listato 3) che permette di leggere il contenuto di una cella posta in qualsiasi banco. Come al solito tutta la complessità sta nel settare i parametri, tra cui il puntatore al puntatore (!) al byte cercato da porre nel registro A. Ovvero ci scegliamo due celle contigue in pagina zero e posto in esse l'indirizzo voluto, scriviamo in A l'indirizzo della prima di queste due celle. Come è consuetudine del linguaggio macchina, indirizzi a 16 bit vengono spezzati mettendo la parte bassa nella prima delle due celle e la parte alta nella seconda. Caricheremo poi in X il banco desiderato e in Y un eventuale offset per accessi indiciati. Al ritorno da questa routine troveremo in A il byte cercato.

Di fattura assai simile, la successiva routine, mappata all'indirizzo $F7DA permette di scrivere un byte in una cella posta in qualsiasi banco. Metteremo come prima in X il banco desiderato, in Y un eventuale offset mentre in A il byte da scrivere e nella cella $02B9 (come prima in A) il puntatore al puntatore alla cella. Semplice, no?

Stiamo finendo: l'ultima routine permette di confrontare l'accumulatore col contenuto di una cella posta in qualsiasi banco. Il risultato lo otterremo naturalmente nello status register come per qualsiasi altra comparazione semplice. Come prima in $02C8 va messo il puntatore al puntatore alla cella e in X il banco desiderato.

MC

Listato 4

```
. FF7DA  48          PHA
. FF7DB  BD F0 F7    LDA $F7F0,X
. FF7DE  AA          TAX
. FF7DF  68          PLA
. FF7E0  4C AF 02    JMP $02AF


. 002AF  48          PHA
. 002B0  AD 00 FF    LDA $FF00
. 002B3  8E 00 FF    STX $FF00
. 002B6  AA          TAX
. 002B7  68          PLA
. 002B8  91 FF       STA ($FF),Y
. 002BA  8E 00 FF    STX $FF00
. 002BD  60          RTS
```

Listato 5

```
. FF7E3  48          PHA
. FF7E4  BD F0 F7    LDA $F7F0,X
. FF7E7  AA          TAX
. FF7E8  68          PLA
. FF7E9  4C BE 02    JMP $02BE


. 002BE  48          PHA
. 002BF  AD 00 FF    LDA $FF00
. 002C2  8E 00 FF    STX $FF00
. 002C5  AA          TAX
. 002C6  68          PLA
. 002C7  D1 FF       CMP ($FF),Y
. 002C9  8E 00 FF    STX $FF00
. 002CC  60          RTS
```

Per chi non sa

Brevemente dedicheremo questo riquadro a coloro i quali non conoscono (ancora) il linguaggio macchina del 6502 e gentile famiglia, tra cui il processore del 128 l'8502. Ovviamente non faremo un corso d'Assembler neppure alla lontana, ci limiteremo semplicemente a spiegare le istruzioni che il processore è in grado di eseguire.

LDA, LDX, LDY, STA, STX, STY, servono rispettivamente per caricare un byte in uno dei tre registri dell'8502 (LD sta per load) o per scaricare il contenuto di uno di questi tre registri in una cella di memoria (ST sta per store).

PHP, PHA, PLP, PLA servono per immettere nello o togliere dallo stack il puntatore al top dello stack stesso o il contenuto del registro A. Esistono poi istruzioni per eseguire trasferimenti tra registri come TSX, TXS, TAX, TXA, TAY, TYA che spostano i contenuti dei registri A, X, Y e S detto anche status register.

A questi si aggiungono operazioni per incrementare un registro (INX, INY) o per il decremento (DEX, DEY) o riferite a una qualsiasi cella di memoria (INC, DEC). È possibile confrontare un registro con un dato (CMP, CPX, CPY, BIT) così come effettuare un salto a seconda di una condizione verificatasi precedentemente la quale ha settato o resettato uno dei bit (C, Z, N, V) dello status register. Tali operazioni di salto condizionato sono: BCC, BCS, BVC, BVS, BNE, BEQ, BMI, BPL.

A queste aggiungiamo un paio di operazioni per eseguire somma e sottrazione di byte (ADC, SBC), operazioni logiche come l'and, l'or inclusivo, l'or esclusivo (AND, ORA, EOR), shift a destra e a sinistra di una posizione (ASL, LSR), rotazione di byte (ROR, ROL), manipolazione del registro di stato (SEC, CLC, SED, CLD, SEI, CLI, CLV), salto incondizionato (JMP), interrupt da programma (BRK), salto a sottoprogramma (JSR) e relativo ritorno (RTS), ritorno da routine di manipolazione delle interruzioni (RTI) e per ultima operazione NOP che non fa un bel nulla.