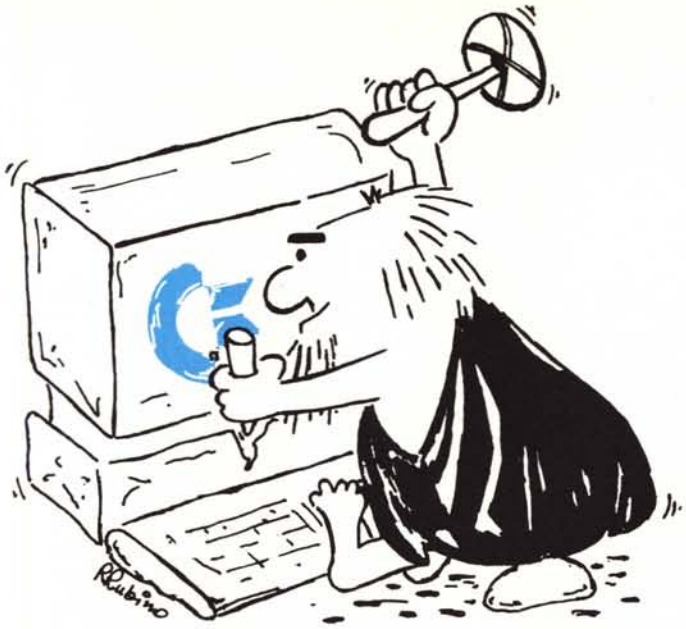

128 da zero

di Andrea de Prisco

Memory management

A partire da questo numero, MCmicrocomputer ospiterà tra le sue pagine una nuova rubrica interamente dedicata al Commodore 128. Come già successo per il VIC-20 e il 64, gli argomenti qui trattati saranno ben lontani dai contenuti del manuale di sistema fornito con la macchina: nel nostro viaggio dentro il 128 cercheremo di scoprire quanto questo computer nasconde sotto le plastiche vesti.
Il 128, infatti, non è un 64 riveduto e corretto ma un calcolatore del tutto nuovo, dalla struttura assai diversa da quella che siamo abituati a vedere in un personal computer tradizionale. Quanto appena detto (per chiarire subito il mio punto di vista) non vuol dire che il 128 è un computer fantastico. Vuol dire solo che (ahimè) è impossibile smanettarvi dentro senza una precisa documentazione della sua architettura. Considerato poi che di Programmer's Reference Guide, almeno fino a questo momento, in Italia manco a parlarne, l'unica cosa è farsela spedire dall'America o continuare a leggere...

Qui comincia l'avventura

Questo mese, del nostro amato 128, discuteremo circa la gestione della memoria. E già ne vedremo delle belle.
Si sa infatti che tutti i calcolatori dispongono di una quantità più o meno grande di memoria centrale per mantenere dati e programmi. Sapere come questa è strutturata può far comodo a chi si occupa di 128 dal lato del Basic, mentre è di primaria importanza per chi, armato del potente cacciavite del linguaggio macchina, vuol programmare qualsiasi applicazione veloce. Considerato poi che la memoria del 128 è veramente quanto di più complesso e contorto una (normale) mente umana riesca ad immaginare, mettere un po' d'ordine a tutta la faccenda non guasta di sicuro. Specialmente quando anche su una nota guida americana di riferimento per il 128, arrivata in Italia chissaccome, si leggono cose non vere su questa benedetta memoria.
Come se la confusione che c'è in giro non fosse abbastanza.

Riassunto dei computer precedenti

Prima di entrare nel merito, vorremmo ricordare brevissimamente come era (ed è) gestita la memoria dei due nobili predecessori del 128: l'amato VIC-20 e l'onnipresente 64.
Punto comune dei tre computer è di usare tre processori della stessa famiglia (nonché compatibili) con indirizzamento della memoria a 16 bit. Ciò significa che in tutti e tre i casi, solo 64K celle di memoria sono direttamente indirizzabili.

128 da zero

Col VIC-20 ovviamente non esistevano di questi problemi, la memoria si presentava come una cassettiera e solo alcuni cassetti erano forniti compresi nel prezzo: 8K rom di sistema operativo, 8K rom contenenti il basic, 4K rom per il generatore dei caratteri, 5K ram per l'utente e per il sistema, più 1 k-nibble (parole di 4 bit ciascuna) per la mappa dei colori. Il resto dei pericolosi «vuoti di memoria» nella mappa: i cassetti mancanti. Se avevi altri soldi potevi comprare cartucce rom e cartucce ram fino al riempimento completo di tutta la memoria. Anche in configurazione super espansa (a tal proposito nacquero i vari trislot e mother-board) la memoria si presentava come una «vera» memoria, dove se facevi peek di una cella sapevi con certezza che cavolo ti restituiva il sistema. Il primo bagliore di complicazione si ebbe con la nascita del 64, il quale bruciò tutti con la sua memoria composta in partenza da 64K ram e 20K rom: oltre a questa potevi acquistare dell'altra rom contenente ad esempio espansioni basic, programmi o giochi. Tutta la rom di sistema doppiava, per così dire, alcune zone di ram: si sovrapponeva ad essa. In una di queste zone doppie, se effettuavi una peek ottenevi il contenuto della corrispondente cella in rom, se facevi una poke, questa «perforava» la rom e modificava la cella in ram. Infine, per leggere una di queste locazioni ram coperte dalla rom, da linguaggio macchina agendo su di un opportuno registro era possibile far emergere la ram nascosta che così era disponibile sia in lettura che in scrittura (sempre da l.m., s'intende).

Il Commodore 128

Tanto per cominciare, nel Commodore 128, di memoria ce n'è davvero tanta: a giudicare da esperienze tastierecce si contano per lo meno 127K di ram utente (non 128, come vedremo) 32K rom tra basic e monitor di linguaggio macchina, 16K rom per il sistema operativo senza contare gli altri 16K ram del processore video a 80 colonne (argomento che tratteremo in seguito) e gli almeno altri 20K rom del Commodore 64 che il 128 è in grado di simulare (caratteri compresi, che sono diversi).

Quando si programma in basic, sono disponibili all'utente ben 58109 byte di memoria per i programmi più un'area leggermente più grande di questa per le variabili numeriche e di stringa. Già questa è una carta vincente in quanto all'utente basic è assolutamente trasparente il fatto che la memoria è gestita a pagine: per lui ci sono più di 100K ram liberi. Se programma in basic-basic.

Se al contrario smanetta con peek e poke, ancora una volta il basic 7.0 del 128 gli viene incontro offrendo una gustosissima instruzione BANK che permette di selezionare ben 16 configurazioni di memoria diverse prima di una qualsiasi operazione riferita a celle di memoria.

Per mostrare come varia la configurazione switch-ando da un banco all'altro, immaginiamo la memoria suddivisa in 8 zone di memoria, come indicato in figura 1.

«A» si estende dalla locazione $0000 alla locazione $03FF, «B» da $0400 a $3FFF, «C» da $4000 a $7FFF, «D» da $8000 a $AFFF, «E» da $B000 a $BFFF, «F» da $C000 a $CFFF, «G» da $D000 a $DFFF, infine «H» da $E000 a $FFFF.

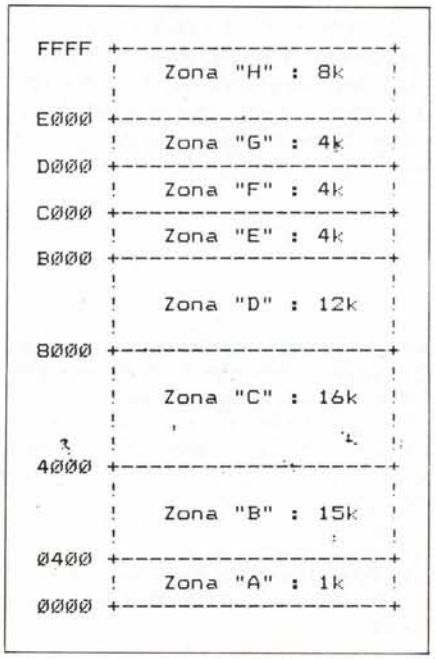

Figura 1 - La memoria del 128 suddivisa in «zone».

Innanzitutto, qualsiasi banco selezioniamo, il primo K di memoria, ovvero la zona A, è sempre lo stesso: ecco perché troviamo 127K e non 128K. In figura 2 tale K, per non dimenticarlo, è stato indicato con COMMON (comune).

Dato che i 128K ram (o 127 che siano) in nessun caso sono visti come un unico insieme contiguo di celle (anche da basic: non possiamo scrivere un programma da 90K e usare 10K per i dati) d'ora in poi li indicheremo come due distinte zone: RAM 0 e RAM 1, di 64K l'una (o 63?).

La configurazione bank 0 mette a disposizione i 63K di ram 0 più, come detto, il primo K di memoria comune. Analogamente bank 1 permette di utilizzare ram 1 (si tenga sempre sott'occhio la figura 2). Per quanto riguarda bank 2 e bank 3, questi dovrebbero permettere l'utilizzo di un ipotetico ram 2 e ram 3 non disponibile sulle macchine fino ad ora prodotte: forse nella mente della Commodore c'era anche un «128» con 256 K ram. Nell'attuale configurazione, ram 2 corrisponde a ram 0 e ram 3 a ram 1: in altre parole, bank 3 e bank 1 sono la stessa cosa così come lo sono bank 2 e bank 0. Passiamo avanti.

A partire da bank 4, comincia ad emergere qualcosa: innanzitutto la zona di memoria G diventa zona di I/O, da dove è possibile comandare i chip intelligenti del 128. Sempre in bank 4 incontriamo internal rom che a lume di naso dovrebbe significare rom interna. Peccato che non c'è. Nel senso che andando a curiosare col monitor di linguaggio macchina lì dentro non leggiamo nulla, se non valori fittizi funzione dell'indirizzo di memoria. Succede proprio la stessa cosa che avveniva col VIC-20 quando si eseguiva una peek in una zona di memoria mancante. La differenza con bank 5, 6 e 7 è la stessa che intercorre tra bank 1, 2, 3 e 4: riguarda la ram delle zone C e B e anche qui un riferimento a ram 3 e a ram 2 è riportato alla ram effettivamente esistente: 1 e 0.

Bank 8, 9, 10, e 11 permettono di accedere alla stessa ram di prima e alla rom esterna. Non c'è nemmeno questa: forse il riferimento è ad una eventuale cartuccia inserita nella porta espansione.

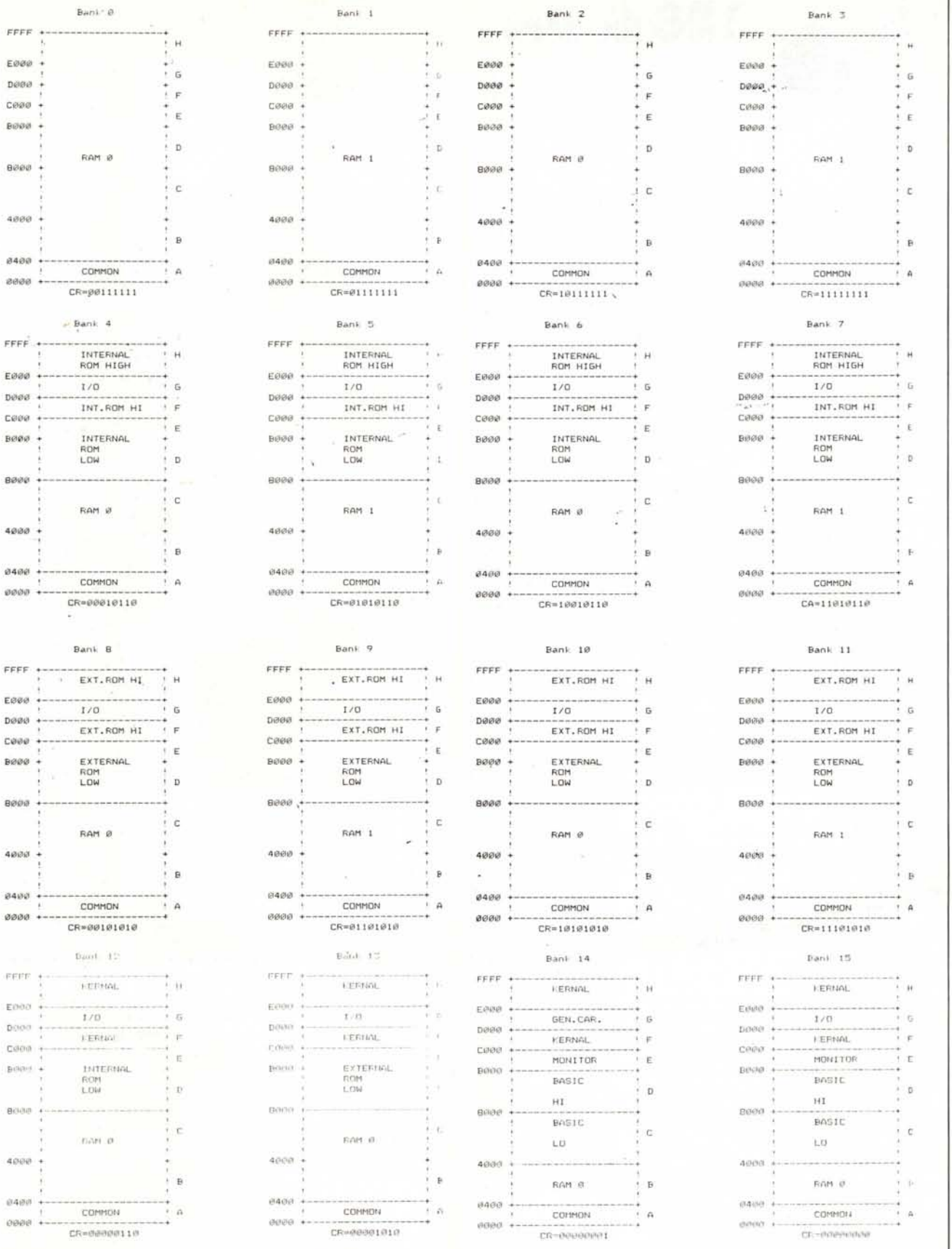

Figura 2 - Le 16 configurazioni di memoria disponibili da basic.

128 da zero

Bit 7 & 6	Bit 5 & 4	Bite 3 & 2	Bit 1	Bit Ø
ØØ = Ram Ø	ØØ = Kernal	ØØ = Bas.hi+mon.	Ø=bas.lo	Ø= I/O
Ø1 = Ram 1	Ø1 = Int.Rom hi	Ø1 = Int.Rom low	1=Ram	1=ram/rom
1Ø = Ram 2 (!)	1Ø = Ext.Rom hi	1Ø = Ext.Rom low		
11 = Ram 3 (!)	11 = Ram			
Ram Controller	$CØØØ - $FFFF	$8ØØØ - $BFFF	$4ØØ-$7FFF	$DØØØ-DFFF

Figura 3 - Registro CR della MMU.

L'affare migliora un poco con bank 12: nelle zone F e H spunta fuori il Kernal del 128, il sistema operativo. D e E continuano a mostrare l'inesistente rom interna che in bank 13 si trasforma in rom esterna ancor più inesistente, come già detto.
Il culmine lo troviamo in bank 14, dove effettivamente, di roba ce n'è un bel po': troviamo oltre al kernal, il generatore dei caratteri, il monitor di linguaggio macchina e i 28K rom del basic 7.0.
Bank 15, che tra l'altro è il banco selezionato al momento dell'accensione, come unica differenza rispetto a bank 14, troviamo I/O al posto del generatore dei caratteri.

Detto questo

Qualcuno si sarà sicuramente chiesto cos'è quel numero binario sotto ad ogni mappa di memoria in figura 1 e più in generale cos'è CR.
Sarà bene svelarvi un piccolo segreto: vi abbiamo detto qualche bugia. Non per cattiveria, s'intende, era solo per complicare subito le cose. In tutte le mappe di memoria, descritte prima e mostrate in figura 2, c'è un'altra piccola zona comune con tutte le configurazioni: 5 byte a partire dall'indirizzo $FF00. Sono i registri che controllano la MMU, memory management unit, un chip che permette di manipolare la configurazione della memoria. È ovvio che tali registri devono essere visibili in ogni mappa, altrimenti, una volta selezionato un banco non si potrebbe tornare indietro. A tal proposito, vorremmo anticiparvi un'altra delle stranezze del 128 (questa però era altamente prevedibile): immaginiamo di scrivere una routine in linguaggio macchina e mettiamola in ram 0, ad una qualsiasi locazione. Diamo run al nostro programmino (nella fattispecie sarebbe meglio dire diamo SYS) e, sempre per ipotesi, ammettiamo che il programmino a un certo punto, manipolando i registri della MMU, selezioni un altro banco. Non l'avesse mai fatto: di colpo il flusso del programma verrebbe catapultato sul nuovo banco con le disastrose conseguenze che possiamo immaginare. A meno che, non provvediamo a scrivere la continuazione del programma nel nuovo banco selezionato o meglio tutto il programma sia in ram 0 che in ram 1 in modo da non avere problemi nel passare dall'uno all'altro. Solo così possiamo manipolare «facilmente» tutta la ram disponibile.
Stiamo scherzando, ovviamente. Non s'è mai visto che per programmare un computer bisogna scrivere o caricare due volte il medesimo programma. Fortunatamente ci viene incontro mamma Commodore col suo bravo K di memoria comune a tutti i banchi. Solo da lì potremo swhitc-are ciò che ci piace senza dare conto a nessuno.
Comunque, anche perché lo spazio disponibile nel primo K è davvero poco, opportune routine di sistema operativo (anche di questo ne riparleremo più dettagliatamente) permettono di accedere a celle di banchi diversi, senza stare a impazzire con la mmu e i suoi modi di fare. A me serve la locazione $A876 del banco 14? Basta rivolgersi al sistema operativo.

Il registro CR (configuration register)

Locato alla posizione $FF00, in tutte le configurazioni di memoria disponibili, il registro CR permette di manipolare le zone di memoria B...H di cui sopra ed è mostrato in figura 3. Settando opportunamente i bit di tale registro è così possibile da linguaggio macchina configurarsi la memoria a piacimento.
Il bit 0 di tale registro, controlla la zona di memoria G ovvero quella compresa tra $D000 e $DFFF. Se tale bit è a 0, indipendentemente dagli altri bit, selezioneremo l'I/O.
Se è a 1, a secondo dei bit 4 & 5 avremo dell'altro.
Il bit 1, che controlla la zona C della memoria, più semplicemente, se è a 0 seleziona la rom low del Basic, se è ad 1 la ram.
Quale ram (0 o 1) dipende, come vedremo, dai bit 6 & 7.
I bit 2 & 3, controllano simultaneamente la zona D e E. Posti a 00, mettono in luce la rom hi del basic e il monitor di linguaggio macchina; a 01 la non meglio specificata rom interna, a 10 quella esterna, a 11 la ram specificata coi bit 6 & 7.
I bit 4 & 5 si occupano delle zone F, G e H: per la zona G ricordiamo che se il bit 0 e a 0 è selezionato l'I/O. Posti a 00 selezioniamo il Kernal (e il generatore dei caratteri bit 0 permettendo); posti a 01 la rom interna, a 10 la rom esterna, a 11 la ram, anche in questo caso quella specificata nei bit 6 & 7. Per quanto riguarda quest'ultimi, se sono posti a 00, per tutta la ram che c'è da mostrare, selezionano ram 0; se posti a 10, ram 1; per 10 e 11, non essendo disponibili ram 2 e ram 3 otteniamo, come sempre, ram 0 e 1.
Più difficile di così... (Arrivederci!)

MC