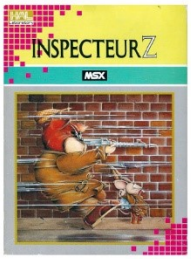
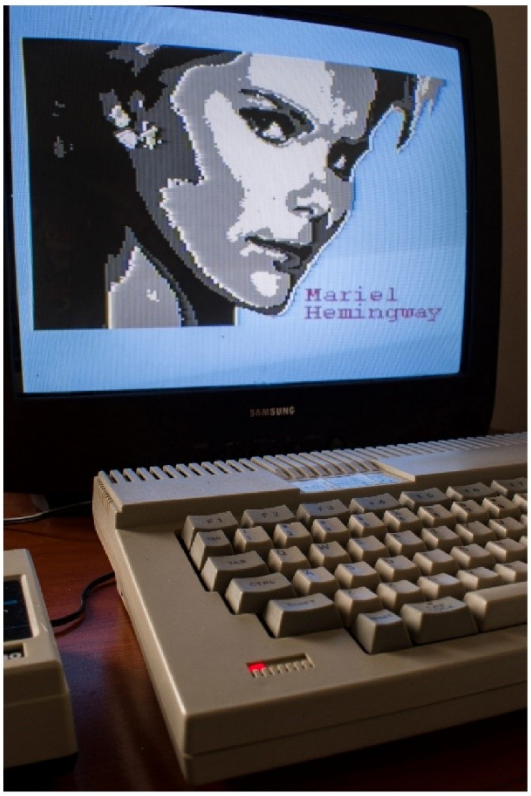
RetroMagazine
semplicemente retro
Numero 12 - Anno 3 - Gennaio 2019 - WWW.RETROMAGAZINE.NET - Pubblicazione gratuita

Home Computer Myarc Geneve 9640

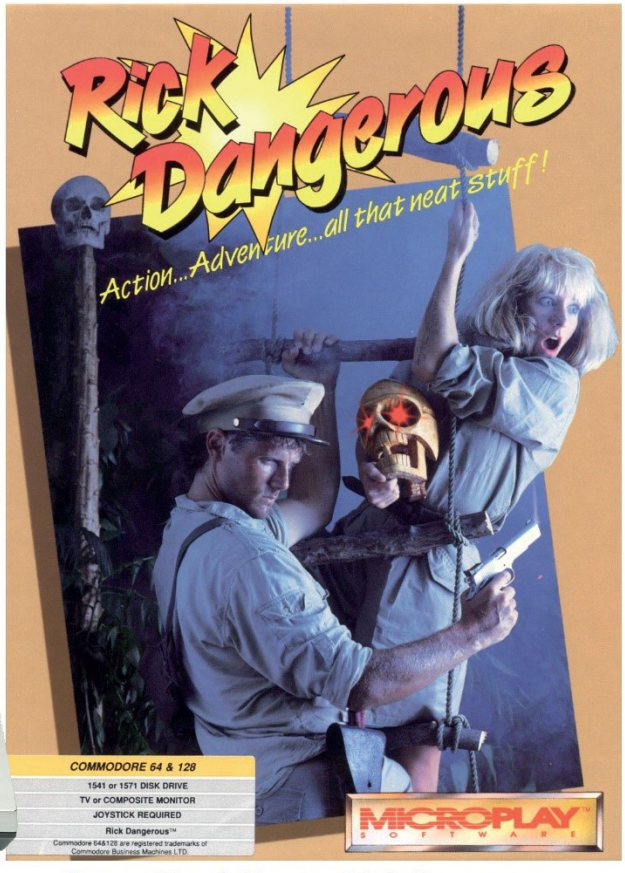


Inspecteur Z (Buru to Marty Kikiippatsu)

MSX - The Dark Side of 8bit (parte 2)



Mariel Hemingway sul LASER 500



RetroGiochiAmo: Rick Dangerous

RetroMagazine
Anno 3 - Numero 12

Hanno collaborato a questo numero:
- Antonino Porcino
- Giuseppe Frisicaro
- Ermanno Betori
- Starfox Mulder
- Fabrizio Caruso
- Daniele Brahimi
- Francesco Gekido Ken Ugga
- Emanuele Bonin
- Marco Pistorio
- Giuseppe Fedele
- Alessandro Paloni|
- David La Monaca/Cercamon
- Federico Gori
- Leonardo Vettori
- Giorgio Balestrieri
- Francesco Fiorentini

Immagine di copertina:
Flavio Soldani

GENNAIO 2019 - WWW.RETROMAGAZINE.NET

IN EVIDENZA IN QUESTO NUMERO

Il paradosso del RetroComputing

di Francesco Fiorentini

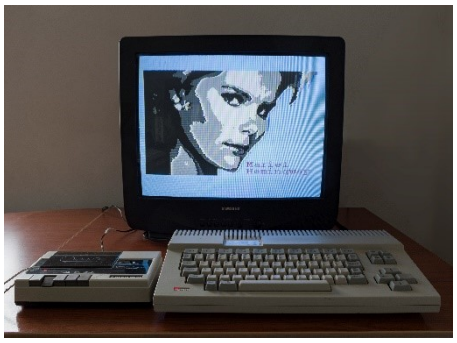
Come molti di voi, probabilmente, sono iscritto a diversi gruppi Facebook che trattano di Retrocomputing e RetroGaming. Alcuni di questi gruppi annoverano tra i loro membri anche migliaia di iscritti, ma fino a qualche tempo fa avevo notato che molti utenti erano silenti o quantomeno 'distratti', cioe' utenti che visionavano qualche post per poi passare immediatemente ad altro. Ma da qualche mese a questa parte mi sembra che il vento sia cambiato. Il numero degli interventi, i commenti, le richieste di spiegazione degli argomenti piu' ostici, sono decisamente aumentati. Segno evidente di un interesse ai massimi livelli. Di contro invece le riviste di settore, dedicate soprattutto al retrogaming, sono scomparse dalle edicole. Ovviamente non ne conosco la ragione, ma e' facile immaginare che la causa sia da ricercare nello scarso numero di riviste vendute e quindi nella difficolta' di rientrare dalle spese sostenute.

La domanda a questo punto pero' sorge spontanea; perche' se l'interesse registrato attorno a questa passione su Facebook e' in costante crescita non c'e' anche un effetto benefico sulle riviste professionali?

Qualcuno potrebbe rispondere che il successo su Facebook e' derivato dal fatto di non dover pagare per accedere alle informazioni, mentre sborsare 8 o 10 euro per acquistare una rivista e' un altro discorso. OK, sono d'accordo, ma soltanto in parte. Credo che questa sia solo una parte del problema e forse nemmeno la piu' preponderante. In molti ci hanno scritto dicendoci che sarebbero anche disposti a pagare per avere un prodotto come RM in formato cartaceo.

Ritengo che le pubblicazioni cartacee abbiano fallito perche' puntavano molto sull'aspetto ludico e poco sull'aspetto serio del retrocomputing. Infatti, a parte le nuove leve che si stanno affacciando a scoprire la magia degli 8 e dei 16 bit, lo zoccolo duro degli utenti e' formato da persone nella fascia di eta' compresa tra i 35/50 anni. Il richiamo colorato dei giochi dell'infanzia funziona un po' come lo specchietto per le allodole, ma dopo un po', se non condito con altro, rimane fine a se stesso e velocemente dimenticato. Sempre piu' sono le persone che oltre a giocare vogliono imparare a programmare o anche soltanto leggere e capire le tecniche che venivano e vengono tuttora utilizzate per creare software per queste macchine meravigliose. Il target delle riviste era probabilmente corretto, ma il linguaggio utilizzato era sbagliato. Forse anche cosi' si spiega il successo della nostra rivista che e' riuscita a trovare il giusto bilanciamento tra ludico e professionale. Che ne dite? Siete d'accordo con la mia analisi?

Prima di lasciarvi, vorrei ringraziare a nome di tutta la redazione di RetroMagazine **Marco Vallarino**, che nel suo blog https://www.marcovallarino.it/retrogaming-cinque-siti/ ci ha menzionato tra i suoi 5 siti italiani preferiti tra quelli che si occupano di avventure testuali. ☺



Mariel Hemingway e...

Quando ci sono la passione, l'abilita' e la volonta' di fare, niente e' impossibile. **Antonino Porcino** ci racconta una storia affascinante carica di pathos.

Articolo a pagina 15

Myarc Geneve 9640

Scopriamo insieme ad **Ermanno Betori** un semi-sconosciuto Family Computer dal nome "amichevole", nato da una costola del TI99/4A.

Articolo a pagina 5

Un Commodore 64C al Cadmio!

di Giuseppe Frisicaro

Premessa

Avendo un parco macchine Commodore popolato da un certo numero di C64 di svariati modelli, tra cui anche alcuni C64C, nasce ad un certo punto l'esigenza di dovere sacrificarne uno per elaborarlo a proprio piacimento. Questa condizione si realizza nel momento in cui si hanno davanti modelli irrecuperabili perché esteticamente malandati (oltre ad essere guasti) per cui lasciandoli in quello stato risulterebbero inguardabili. Accumulare un tale esubero di materiale o, nei peggiori dei casi, buttarlo via perché non ce ne facciamo nulla, sarebbe assurdo e poco produttivo per la nostra collezione.

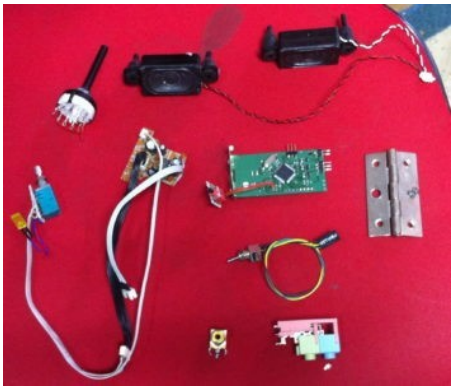
Quindi cosa si fa per recuperare il nostro C64 malconcio? Si passa al "modding " completo della macchina! Questa pratica è già ampiamente diffusa nel mondo dei PC da un po di anni e comprende nella fattispecie il restyling dei case con qualsivoglia artefatto luminoso, widget da aggiungere o l'installazione di sistemi di dissipazione del calore interno futuristici. Ma rimaniamo in ambito Retro con il mio esemplare.

Il C64C Yellow Edition

L'idea era quella di usare un case C64C malconcio, verniciarlo ed elaborare la scheda, aggiungendo vari accorgimenti tecnologici di nuova concezione. Avevo il dubbio iniziale sull' impiego del colore per il case, tra cui il rosso acceso e un giallo lucido, ma dopo svariate prove sulla plastica, la decisione cade sul giallo in quanto l'ho ritenuto piu adatto allo stile del C64c.

Presa questa decisione, ho iniziato a riassumere le modifiche hardware da apportare sull'intero computer, in quanto queste potevano influire sulla verniciatura, che in ogni caso sarebbe stata la fase che avrebbe preceduto l'assemblaggio finale. Il Modding hardware doveva consistere in:

- Altoparlanti stereo integrati ed amplificati (collegati in MONO ma predisposte per un eventuale DualSid)
- 4 Kernal-Rom su unica Eprom e selezionabili mediante deviatore rotativo con indicatori luminosi per ciascuna Rom selezionata
- Controllo del volume, tasto reset e Led amplificatore
- SD2iec incorporata con Led di attività e selezione device 8/9 nonché pulsante reset del dispositivo
- Modulo Radio FM incorporato e controllato via software dedicato
- Display LCD 7" esterno fissato sul case e ribaltabile
- Uscita Audio su Jack 3,5mm
- Uscita Video-C su connettore RCA
- Ingresso audio (che va al Sid)



Avendo chiare tutte le soluzioni tecniche per rendere possibile quest'impresa, non rimaneva che iniziare predisponendo il case per ospitare le varie periferiche e controlli esterni.

Il case utilizzato presentava dei graffi di una certa entità, veri e propri solchi, che andavano eliminati alla meno peggio con carta a smeriglio. La fase successiva è la predisposizione del case per gli altoparlanti. Questi vengono fissati sull' estremità della tastiera, ma era necessario praticare dei fori per sentire l'audio per cui... ecco l'idea: perché non farli con le scritte "64" ?

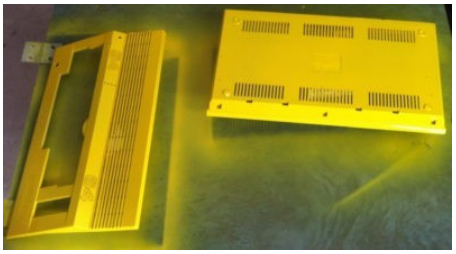
Fatto ciò (con il trapanino e tanta pazienza) si passa ai fori per i Led e la fessura della SD2iec, il controllo volume, il selettore a 4 posizioni per le Rom e i relativi Led, i jack audio IN/OUT, il connettore RCA, nonché il sistema di fissaggio del monitor mediante cerniera.

A quel punto inizia l'installazione della parte Hardware, che consiste nel collegamento interno della SD2iec, i Led, la Eprom con il "Quad Kernal" con il relativo selettore e i Led, l'amplificatore audio con controllo volume e ON/OFF, il modulo Radio FM e le prese Audio e Video-C con i loro cablaggi sulla scheda.

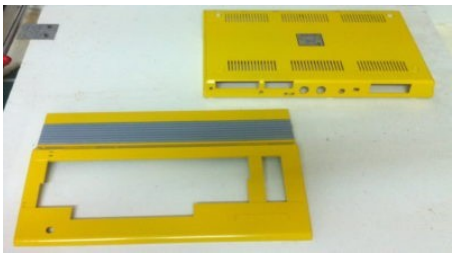
Una volta testate tutte le funzionalità, si ri-smonta il tutto per passare alla verniciatura.

Questa avviene esclusivamente con l'utilizzo di vernice spray a base acrilica, nella fattispecie ho usato il **giallo Cadmio RAL 2942.**

Non è stato necessario alcun pre-trattamento della plastica (come fondo aggrappante o simili) in quanto la vernice utilizzata aderisce alla plastica senza problemi, garantendo un'ottima presa e allo stesso tempo si evita il formarsi di crepe che spesso sono causati proprio dagli aggrappanti. Ovviamente va verniciato anche il monitorino LCD !



Ho voluto che una parte fosse verniciata di grigio opaco, giusto per spezzare con il giallo. Questo ha reso la fase di verniciatura un po più laboriosa, ma l'effetto ottenuto mi è piaciuto moltissimo.



La tastiera, come si può vedere dalle foto iniziali, è stata "mixata" con alcuni tasti derivanti da un C16, proprio per seguire quell' abbinamento col grigio. Questa decisione però è stata scartata successivamente perché risultata un tantino antiestetica, limitando a tale combinazione il solo tasto Spazio, verniciato separatamente. Le ultime foto mostrano lo stato attuale.

L'assemblaggio finale è stata la parte più delicata poiché bisognava rendere l'installazione della componentistica in maniera definitiva, con un occhio di riguardo al cablaggio che doveva venire pulito e privo di confusione fatta da fili "selvaggi" volanti al suo interno, come mi è capitato di vedere in molti altri lavori del genere. Tutto questo con particolare attenzione a non maltrattare il case appena verniciato.

Ultimato l'assemblaggio, ho applicato manualmente le scritte sui vari controlli e aggiunto un pò di decorazioni adesive, giusto per rendere questo modello un esemplare unico.

La foto in intestazione (Che riporto anche qui per completezza - NdR) è quella allo stato attuale, con tanto di autografo di **Petro Tyschtschenko**, general manager logistica & merchandising Commodore Germania.

Gruppi Facebook correlati

Retrocomputer World:
https://www.facebook.com/groups/2398221767222864/

Pianeta Commodore:
https://www.facebook.com/groups/1548452001848020/

Blog personale:
https://www.facebook.com/GiuMacGyver/

Canale Youtube:
https://www.youtube.com/channel/UCnplSbUN18mTJKH4qUPkM4A

Esonero di responsabilita'

L'autore e la redazione di RetroMagazine declinano ogni responsabilita' per eventuali conseguenze dannose che potrebbero derivare agli utenti che decidessero di effettuare una modifica simile a quella illustrata nell'articolo.

In pratica, chi decidesse di eseguire la modifica, lo fara' a proprio rischio: sia per la propria incolumita' (utilizzo di utensili e di vernici), che per il possibile mancato funzionamento dell'hardware coinvolto.

Home Computer Myarc Geneve 9640

di Ermanno Betori

Questo è il racconto di un computer voluto dagli utenti del TI99/4A Home Computer creato dal colosso Texas Instruments, rimasti orfani della casa madre e nato da una micro azienda americana (MYARC) che ambiva ad essere il suo successore e che parzialmente si realizzò.

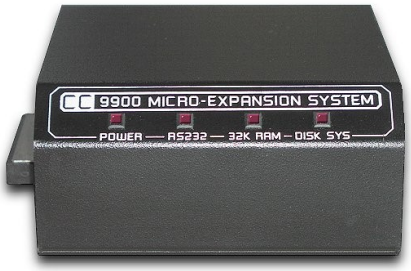
La Storia

La storia nasce nel primo numero della rivista "Home Computer Compendium" (subito dopo ribattezzata "MicroPendium", in quanto la Texas Instruments, contestò subito l'uso del proprio marchio "Home computer"), infatti nel febbraio del 1984 (pochi mesi dopo il ritiro della TI dal mercato) fu annunciato il computer "99/64 (aka Phoenix)" che doveva essere prodotto dalla ditta CorComp. La CorComp era una ditta già affermata come creatrice di espansioni di memoria, di disk controller e di schede RS232 che lavoravano con il Peripheral BOX del TI99/4A.

Le caratteristiche della macchina annunciata erano, 64 KB di RAM espandibili ad 1MB, Interfaccia RS232 e disk controller di serie, una versione migliorata del BASIC esteso e fino a 132 colonne di visualizzazione. Al confronto delle attuali espansione massima di memoria da 32K RAM, periferiche costose come interfaccia RS232, unità dischi, un potente ma talvolta macchinoso Extended BASIC e un display a 32/40-colonne, questa macchina fu chiamata nell'articolo "Dream Machine".

Quanto vi era di vero?

Non molto... Infatti successivamente si scoprì che i dettagli tecnici annunciati si sarebbero solo parzialmente realizzati. CorComp era propensa a rilasciare un modulo di espansione che comprendeva queste caratteristiche, ma niente di più. Nessun nuovo computer. No built-in "super BASIC". Nessun display migliorato. Questo dispositivo che successivamente vedrà la luce, sarà il **Corcomp 9900 Micro-Expansion System** qui sotto mostrato.



In pratica questa compatta espansione che conteneva espansione di memoria da 32k, disk controller e interfaccia seriale/parallela, fu progettata per tutti gli utenti (99ers) che non avevano ancora acquistato l'ingombrante Expansion System della TI. Tuttavia, era evidente che gli utenti erano affamati e desiderosi di un aggiornamento se non addirittura di un nuovo computer più potente! E questo dopo nemmeno un anno che la TI aveva abbandonato il mercato degli Home Computer nell'autunno del 1983. Questo avvenne sia per la brama di prestazioni, sia per non perdere il loro precedente investimento in hardware e software. Infatti dopo che la TI aveva lasciato il mercato, nacquero molti gruppi di utenti in diverse città. I più famosi User Groups avevano la loro rivista con software annesso, cosa che accadde anche in Italia ma con almeno sei anni di ritardo. Ma soprattutto in Boston e Chicago, questi User Groups divennero il motore principale dato che si tennero famose fiere mercato denominate "TI Faires" che furono le vetrine dei nuovi prodotti per il 99/4A che venivano annunciati dalle varie ditte.

Entra in scena la ditta MYARC

La Myarc (Microcomputer Architects), come la CorComp, ha origini nella produzione di espansione di memoria, RS232 e schede controller di disco. In più avevano ampliato la loro gamma di prodotti con un migliorato extended BASIC ed una migliore RAMdisk.

Nel marzo 1984 durante il TICOFF fiera espositiva nel New Jersey, Lou Phillips (nella foto), proprietario della Myarc, annunciò un nuovo computer compatibile 4A. Questa macchina era stata annunciata in due versioni: un modello che ricordava la consolle del TI994A con tastiera ma simile nel design alle macchine Amiga, Atari, ed al 99/8 o altre del periodo e un computer in una scheda che si doveva adattare al sistema di espansione periferica TI. Il PES o "Peripheral-Box", come è stato poi conosciuto, era collegato alla consolle 4A tramite una "scheda di interfaccia Flex cable" ed un enorme cavo che assomigliava ad una manichetta antincendio. Questa seconda versione di computer doveva servire proprio ad eliminare questa macchinosa connessione.

Ulteriori dettagli tecnici di questa fantomatica macchina furono esposti, alla fiera New England 99 Faire in Boston il 5 aprile 1984. Il computer avrebbe avuto 256 KB o 512 KB di RAM (espandibile fino a quasi 2 mb), un chip video più potente il TMS 9938 (sarà successivamente prodotto dalla Yamaha e usato in tutti i computer MSX2 ed altri..), che poteva gestire 80 colonne di testo e grafica a 512 x 424 x 256 colori, RGB e uscita video composito, una porta per mouse, interfaccia di tastiera del PC, il SN76496 come nuovo chip sonoro (compatibile con il chip audio tms9919 presente nel TI99/4A) e una CPU TMS 9995 della TI che è un successore a 12mhz della CPU a 3mhz (TMS9900) usato nella consolle 4A.

Dove era il Computer?

Nel 1985, il Chicago TI Faire era diventato l'evento più importante al mondo riguardante i prodotti per il computer TI99. Il 2 di novembre vi furono la bellezza di circa

1700/1800 visitatori in attesa di assistere al debutto del nuovo computer Myarc.

Che cosa videro invece fu un computer con uno chassis stile computer Amiga avente tastiera integrata ed uno slot cartuccia. Lou Phillips in quella occasione annunciò che la macchina era in produzione e le prime spedizioni sarebbero iniziate a partire dal primo trimestre del 1986 per un prezzo di circa 450 dollari.

I mesi di quel primo trimestre del 1986 andavano e venivano, ma nulla si vedeva... La situazione di vaporware stava diventando così frustrante per i vari 99ers che agli inizi del 1986, la rivista MicroPendium cambiò la sua dicitura nella testata da "rivista specializzata per il Home Computer TI99/4A e compatibili" a "rivista specializzata esclusivamente per il TI99/4A".

La Cometa GENEVE

Verso la metà del 1986 Myarc infine annunciò che il computer sarebbe nato solamente in forma di scheda per il PEBOX e sarebbe stato denominato Myarc Geneve 9640 Family Computer (più comunemente venne detto dagli utenti "il 9640" o "il Geneve"). Si vociferò che volevano usare "99" nel nome, ma la TI non l'aveva permesso. Il "9640" nacque come sigla prendendo il "9" come riferimento al "99/4A" e "640" dalla RAM standard della macchina. "Family Computer" era una bella voce che richiamava alla mente il marchio della TI di "Home Computer". Sentendo che avevano bisogno di un nome "amichevole" per la macchina, Lou Phillips e Jack Riley boss della Myarc (presenti nella foto) videro la città di Ginevra (Geneve in inglese) su un dipinto e la scelsero come nome per il computer.

Volantini, lucidi, biglietti colorati di presentazione di questo nuovo computer furono spediti a migliaia ai proprietari del TI99/4A tramite la mailinglist ufficiale della TI che la Myarc apparentemente gli comprò (la mainlinglist era composta dai nomi di quei proprietari di 4A che spedirono le schede di registrazione del TI dopo l'acquisto).

Nel febbraio 1987, la rivista MicroPendium cambiò ancora la sua intestazione in "rivista specializzata per il TI99/4A, il Myarc 9640 e compatibili". Il 9640 fu il solo computer "compatibile" mai prodotto.

La prima revisione formale della macchina è presente nella edizione di Aprile 1987 di microPENDIUM. Questo articolo era una review di una macchina ancora in fase di beta test. L'articolo iniziava con... "Il Geneve 9640 è qui! Finalmente. E funziona.". Ricordiamo che nello stesso anno come computer antagonisti erano presenti l'Amiga 1000 e l'Atari ST...

Un modello di produzione fu ricevuto il mese dopo Marzo 1987, ma nel numero di maggio 1987 i redattori affermarono che erano ancora in attesa del rilascio del MDOS, che sarebbe diventato il sistema operativo del Geneve.

Il Software

Alcuni pezzi di software furono annunciati con il Geneve, tra cui:

1. **La cartridge Saver** – Questa cartuccia (da usare sulla consolle del TI99/4A) era indispensabile in quanto la maggior parte del software prodotto dalla TI durante il ciclo di produzione del TI99/4A era in formato cartuccia, ma il Geneve non aveva uno slot per le cartucce! La cartuccia Saver aveva un programma che funzionando sulla consolle 4A avrebbe consentito ad un utente del computer classico di creare i dump su disco delle cartucce in un formato che avrebbe funzionato con il Geneve. Questo formato risultava essere lo stesso di quello utilizzato con la cartuccia speciale GRAMkracker (già descritta precedentemente). Ma questo era parzialmente vero.

2. **Advanced BASIC** – questa era una riscrittura del Myarc Extended BASIC II (che si usa con il TI99/4A più una scheda di espansione ram dedicata da inserire nel PEBOX) ed è stato per un tempo noto come Extended BASIC III. È ampiamente compatibile con 4A BASIC e BASIC esteso, anche se consente l'accesso a funzionalità del Geneve quali modalità video avanzate.

3. **Versione 4.21 del linguaggio Pascal runtime** – IL TI99/4A aveva una scheda Pascal USCD (creata dalla TI) da inserire nel PEBOX che gli permise di eseguire programmi USCD Pascal. In sostanza, questa scheda aveva dato al 4A la possibilità di usare un ulteriore linguaggio di programmazione molto potente e il Geneve ne riprendeva l'uso.

4. **TI Writer aggiornato a 80 colonne** – lo standard de facto nell'elaborazione di testi del 4A era il TI Writer questo perché fu rilasciato dalla TI come programma di pubblico dominio quando la TI lasciò il mercato degli Home Computer. IL TI Writer si basava in gran parte sul programma di editing di testo utilizzato sui minicomputer 990 della TI. Invece di visualizzazione WYSIWYG, usava comandi "dot" che erano stati inseriti all'interno del testo per produrre effetti come testo in grassetto o centrato, ecc... simili a quelli utilizzati dal programma di videoscrittura WordStar che si trovava su altri microcomputer vedi IBM e compatibili o Apple. La limitazione principale del TI Writer era la visualizzazione a 40 colonne dello schermo del 4A. Per anni la visualizzazione a 80 colonne era il Santo Graal dei 99ers e questa modifica del TI Writer permetteva finalmente di lavorare in modalità testo 80 colonne. Inoltre, considerando che il computer TI99/4A aveva una tastiera buona ma a 48 tasti che era scomoda da usare per la battitura dei testi; per esempio i tasti FCTN & P sono usati per digitare le virgolette doppie e semplici! La tastiera IBM-stile di Geneve era molto più facile da utilizzare per la videoscrittura e tasti funzione dedicati a questa particolare attività avrebbero reso ancora più semplice l'uso del programma di videoscrittura.

5. **Microsoft Multiplan Upgrade** – poiché Multiplan era ancora un programma protetto da copyright di Microsoft, non poteva essere incluso per intero. Tuttavia, i proprietari della versione di 4A di Multiplan (che comprendeva una cartuccia ed un disco) avrebbero potuto utilizzare Cartridge Saver sulla cartuccia e questa patch sul disco per ottenere una versione di Geneve-compatibile che era più veloce come esecuzione e in modalità a 80 colonne. Un foglio 236 celle richiede 2 minuti e 18 secondi per essere calcolato su un 4A, ma solo 23,73 secondi su un Geneve.

6. **Myarc Disk Operating System** – la consolle 4A non aveva nessun DOS "reale". Sebbene i dati e programmi di base possono essere caricati e salvati su disco e programmi appositi potrebbero essere scritti in XB per ottenere un catalogo del contenuto del disco, tutte le altre funzioni di un sistema operativo DOS (formattazione dei dischi, copia di file, ecc.) dovevano essere raggiunte tramite un programma di gestione dischi, vedi la cartuccia Disk Manager. MDOS cambiava ciò, con una riga di comando e sintassi molto simile a quello di Microsoft MS-DOS si avrebbe avuto tutto ciò.

E qui sorse IL PROBLEMA

Myarc era sotto pressione su due fronti riguardo il Geneve. In primo luogo, aveva annunciato la macchina con 2 anni di anticipo e aveva annunciato nel dicembre del 1986 che la produzione era partita. La Myarc voleva veramente far entrare la macchina in produzione ed essere il rivenditore principale, ma pretendeva il pagamento anticipato. Denaro che Myarc avrebbe usato come salvadanaio per costruire le nuove schede, ma era una pratica di vendita che fu costretta ad abbandonare quando avvennero grandi ritardi nella produzione della macchina con la conseguenza di aver minor denaro da gestire.

In secondo luogo, aveva speso un sacco di soldi nello sviluppo di questo computer. La Myarc aveva pochissime risorse (forse 5 dipendenti e un paio di appaltatori) per la progettazione del Geneve e l'azienda aveva bisogno di recuperare i soldi investiti per non fallire!

Quando viene rilasciato, era dolorosamente ovvio che il Geneve veniva venduto ancora incompleto, oggi lo chiameremmo versione beta avanzata o release candidate. A riprova di ciò nel manuale MDOS era indicato agli utenti che dovevano vedere a schermo un prompt "A:", nei primissimi lotti di vendita vi era la prima pre-release del MDOS con in primo piano un prompt "DSK1.". "DSK1" era come sul 4A l'indicazione dell'unità disco 1. Un utente avrebbe lanciato un programma digitando il suo nome da questo prompt DSK1!! Inoltre la maggior parte dei comandi MDOS menzionata nel manuale non erano stati ancora implementati.

Infatti, quello che Myarc chiamava MDOS era in realtà il suo interprete GPL. GPL – o linguaggio di programmazione grafica – era un linguaggio di programmazione molto vicino all'assembly creato dalla TI che potremmo definire l'antesignano del Java. La maggior parte delle cartucce per il TI99 furono programmate in GPL ed inoltre questo interprete permetteva ad un acquirente del Geneve di eseguire le cartucce che aveva salvato con Cartridge Saver. Così il Geneve poteva eseguire la maggior parte del parco software del 4A altrimenti, essendo una macchina realmente diversa, avrebbe potuto eseguire solo programmi in basic e poco altro.

Il primo articolo ufficiale sul Geneve scritto sulla rivista MicroPENDIUM fu un articolo del settembre 1987 presentato da Mike Dodd che in gran parte parlò negli anni dei vari pezzi di software creati per questa macchina e lo stato di avanzamento di ciascuno.

Infine, MDOS v.97 fu rilasciato nell'ottobre del 1987 e finalmente inseriti la maggior parte dei comandi presenti nel manuale fornito con il computer. La versione V. 99 aggiungeva la elaborazione di file batch. L'aggiornamento del software era distribuito su numerosi servizi di informazione (come CompuServe) e dai dischi che di volta in volta sarebbero stati inviati da Myarc ai proprietari registrati del Geneve.

Ci si aspettava che sarebbe uscito un ROM di avvio contenente la versione "finale" di MDOS in modo che gli utenti non avrebbero avuto bisogno di fare il boot dal disco. Tuttavia, il fatto che il MDOS era incompleto fece accantonare questa idea.

In definitiva, la versione 1.0 di MDOS fu spedita intorno al dicembre del 1987. L'Interprete del GPL era alla versione .98 e funzionava all'interno del MDOS (prima era un programma separato). Con l'eccezione di 6 differenze minori finalmente il MDOS Myarc era ora lo stesso programma descritto nei suoi manuali.

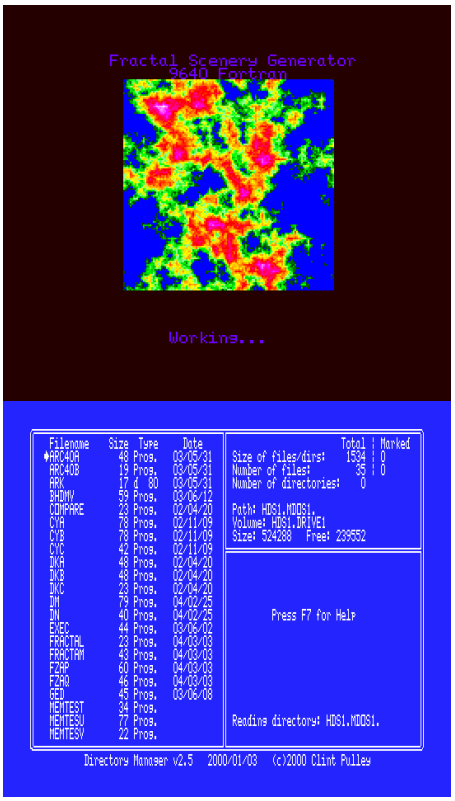
Applicazioni Software e "MY-ART"

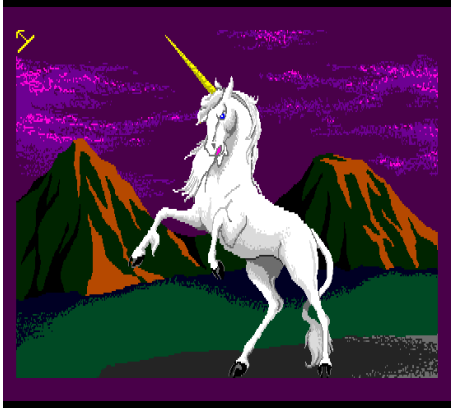
Poco dopo il rilascio del Geneve, la Myarc fa circolare una serie di annunci sul software nei quali diede la colpa dei ritardi di produzione principalmente alla complessità dell'hardware Geneve e la necessità di scrivere MDOS da zero.

Affermo' inoltre di avere un sistema per cui alcuni pezzi di software per PC potevano essere facilmente portati sul Geneve. I primi annunci includevano My Number (un clone di Lotus 1-2-3), My-BASIC (un compilatore BASIC) e My-Data (un dBase III clone), nessuno dei quali fu mai rilasciato.

Altro vaporware includeva un compilatore C (considerato critico, poiché tanto software in quegli anni era stato scritto in C, e in questo modo una grande quantità di software poteva essere portato al Geneve) e My Word Pro, una versione grafica avanzata della MY-Word che implementava l'uso del Myarc Mouse.

Tuttavia la Myarc tra tanti proclami annunciò ed effettivamente rilasciò il suo primo e solo pacchetto di applicazione standalone. Questo era My Art, un programma di disegno venduto all'epoca per circa $149 con incluso il Myarc Mouse.





Sulla scia del Geneve fu annunciato dalla Myarc un HD&FD-Controller (HFDC). Questa scheda permetteva di usare floppy da 3,5 pollici con capacità di 720 KB o 1.44 MB contro i 90 KB/180 KB del controller FDC della TI, fino a tre dischi rigidi in MFM 134mb ed essere utilizzato con un TI99/4A o con il Geneve. Una porta di backup tape streamer era prevista, ma non ha mai veramente funzionato.

Era diventato abbastanza comune per gli utenti che potevano permettersi un Geneve di avere anche una scheda HDFC e per questi utenti godere di una maggior velocità di BOOT del sistema operativo MDOS all'avvio e di avere un caricamento software dal disco rigido anziché dal disco floppy.

La velocità era simile a quella offerta dalla scheda RAMdisk Horizon venduta da Bud Mills. Queste schede permettevano all'utente di simulare un disco con salvataggi e caricamento dati alle velocità delle RAM, il tutto era solitamente dotato di batteria di backup, ma le batterie alla fine sarebbero morte con perdita dei dati e naturalmente le capacità dei floppy simulati erano limitate dal costo elevato dei chip di memoria rispetto allo spazio dato dall'Hard disk.

Il HFDC fu rilasciato intorno al periodo di tempo dove negli U.S.A. la maggior parte dei computer IBM-compatibili utilizzavano questi tipi di HD in quanto più economici e di facile reperibilità contro gli HD a tecnologia SCSI molto più affidabili e performanti ma molto più costosi e contro gli HD IDE che all'epoca soffrivano di una scarsa affidabilità dovuta alla nuova tecnologia.

Anche se altre schede controller dei dischi costruite dalle ditte TI, CorComp e Myarc avrebbero potuto fare uso di floppy disk da 720 KB 3,5" a singola o doppia densità e da 5,25", il HDFC ha consentito l'uso di 1,4 MB (ad alta densità) su dischetti 3.5" con la versione corretta di MDOS.

Una versione speciale di MDOS nasce presto per lavorare con il HDFC. Per diversi anni, fu necessario utilizzare una versione "H" di MDOS se si voleva utilizzare un disco rigido o una versione "F" se si voleva utilizzare l'unità disco floppy. Nessuna versione poteva utilizzare entrambi i tipi di unità e vari bugs rimasero nel MDOS.

SPEECH Synthetizer si poteva usare?

La consolle TI99/4A era supportata dal suo sintetizzatore vocale tramite la porta di espansione posta sul lato destro. Questa è la stessa porta utilizzata per collegare il sistema di espansione periferica PE-Box (e relativo cavo flex board interface). Il sintetizzatore aveva un connettore di tipo pass-through, quindi la maggior parte delle console avevano lo speech collegato alla console e il cavo a manichetta antincendio collegato alla porta pass-through del sintetizzatore.

Naturalmente il Geneve non aveva nessuna tale porta di espansione. Una società denominata Rave 99, più nota per un'interfaccia di tastiera che aveva permesso di collegare tastiere di tipo PC alla consolle 4A, aveva sviluppato una "speech adapter card".

Questa scheda permetteva all'utente di rimuovere il sintetizzatore vocale dal suo alloggiamento ed inserire la scheda, quindi

inserire la scheda nel sistema di espansione di TI. Questo rimane l'unico modo possibile per usare lo speech con il Geneve.

Applicazioni aggiuntive

Vario software fu rilasciato che funzionava in modo nativo da MDOS. The Printer's Apprentice by McGann Software era un pacchetto avanzato di desktop publishing che funzionava sul 4A ma era così complesso che pochi 99ers sapevano usarlo. Fu rilasciata una versione MDOS che utilizzava la grafica e la memoria del Geneve che lo rese molto più facile da utilizzare. McGann Software rilasciò anche The Geometer's Apprentice con un hyper-card like system. Tuttavia la pirateria era dilagante nei circoli di 4A e di conseguenza la McGann alla fine si piegò uscendo fuori dal mercato.

TRIAD era un pacchetto bundle che conteneva un editor di testo, un emulatore di terminale ed un disk manager.

Una collezione di giochi che erano state originariamente pubblicate sul computer TOMY TUTOR inoltre furono rilasciati. Queste porting furono fatti abbastanza facilmente perché il TUTOR era stato costruito con gran parte dello stesso hardware usato nel TI99/4A che nel Geneve, compresa la CPU 9995 del Geneve.

Gran parte del migliore software TI è stato scritto in linguaggio Assembly. Anche se alcuni di questi software avevano inclusi dei loader in Extended BASIC, mentre altri necessitavano della cartuccia Editor/Assembler della TI. Mentre il BASIC esteso usava come autoload un programma posto nella prima unità disco denominato "LOAD", il E/A non aveva nessun tipo di sottigliezza, e i programmi spesso erano notoriamente difficile da lanciare. Un program file loader chiamato EXEC fu rilasciato per consentire all'utente di avviare questi programmi dalla riga di comando di MDOS. L'unico altro modo era quello di avviare MDOS, caricare l'interprete GPL, caricare la cartuccia Editor/Assembler e quindi caricare il programma desiderato – aggiungendo difficoltà a un processo già di se ingombrante.

Entra BEERY MILLER ed il BUYOUT MDOS

Beery Miller lanciò **9640 News**, un magazine su disco, debuttando con il numero di agosto 1988. Nel 1990, aveva già pubblicato tre pezzi significativi di software per il Geneve:

-Baricade (un gioco in Advanced BASIC)

-Tetris (che girava direttamente da MDOS ed è uno dei pochi giochi a farlo)

- Windows 9640.

In quel momento storico il MDOS era stato abbasta affinato da essere realmente utilizzabile, il mercato di computer che usavano GUI si stava surriscaldando nel resto del mondo dell'informatica. Amiga, Atari ST, OS/2, Mac e programmi DOS come GEOS e, naturalmente, Windows, stavano dimostrando al mondo che grafica e un mouse rendevano il computer più semplice per gli utenti.

Beery rilasciò Windows 9640 come risposta. Anche se privo di dall'abilità grafica dei sistemi summenzionati (sviluppati da team di programmatori), questo programma permetteva la attività di switching fino a 8 programmi.

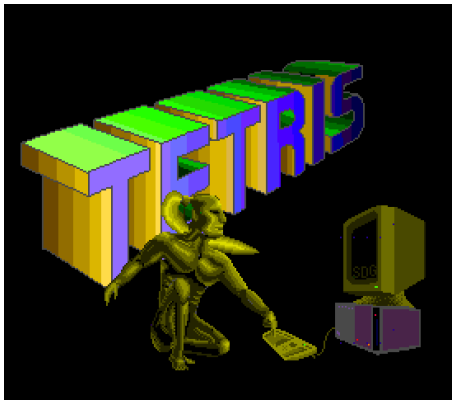
Nello stesso periodo un programma simile era in fase di sviluppo da Myarc denominato GEME, che fu completato e pubblicato da Beery con il permesso di Myarc nel novembre del 1991.

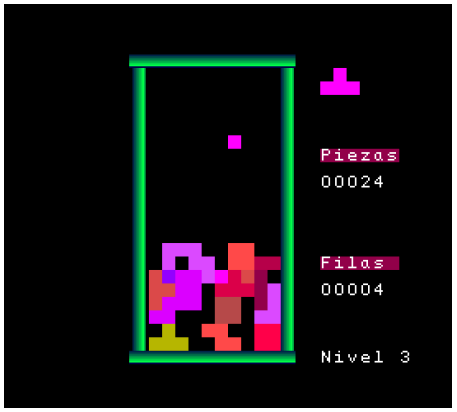
9640 News chiude nel 1999

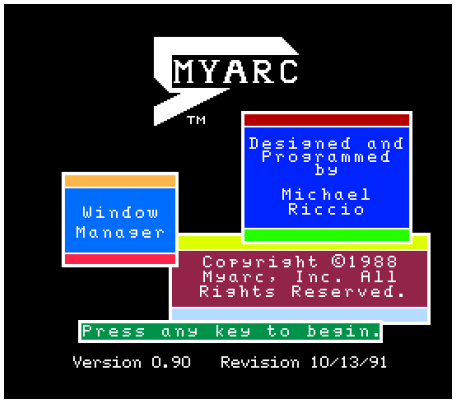
Entro la metà del 1992, lo sviluppo di MDOS era bloccato. Paul Charlton era stato lo sviluppatore di MDOS per Myarc e dopo una disputa con la società, aveva rifiutato di fare ulteriori aggiornamenti per MDOS o di rilasciare il codice sorgente in modo che la Myarc potesse essa stessa creare gli aggiornarsi (questo è un buon motivo in un contratto di chiarire perfettamente di chi è la proprietà intelletuale e di rilasciare adeguata documentazione nella stesura di software).

Beery si adoperò per trovare un modo di acquistare il codice sorgente del MDOS. L'accordo che raggiunse era che Beery avrebbe dovuto raccogliere i fondi ($XXX) per Paul e che Paul avrebbe a sua volta rilasciato il codice sorgente a Beery. Quando abbastanza sostenitori finanziarono il progetto, Beery volò a New York, incontrò Paul, gli diede i soldi, e personalmente gli fu consegnato il codice sorgente del MDOS. Finalmente, il MDOS era nelle mani della Comunità con codice sorgente liberamente disponibile. Le versioni 5 e 6 del MDOS sono i risultati diretti degli sforzi di Beery.

Questa è una parte della storia del Geneve nella descrizione ho volutamente saltato molte parti, basti pensare che questo computer ha avuto lo sviluppo del suo sistema operativo dal 1986 fino al 2000! Per poi a passi alterni avere ancora oggi dei miglioramenti sia sull' Hardware che sul Software. Se siete interessati su questo successore del TI99 scriveteci e vi aggiorneremo sulla sua storia e sul suo Sw/Hw.





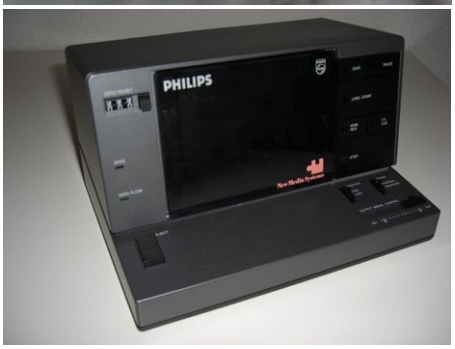


MSX – The Dark Side of 8bit - parte 2

di Ermanno Betori

Riprendiamo la trattazione delle periferiche presenti nei computer MSX; nel numero scorso abbiamo presentato ben sei tipi di espansioni sonore, adesso mostreremo le memorie di massa e la loro evoluzione.

I computer MSX1 adottarono come primario metodo di memorizzazione l'uso dei nastri magnetici che era il più economico e di fatto era quasi la norma nei computer 8 bit dell'epoca.



Vi erano ovviamente vari tipi di registratori a cassette, sia commerciali che altri creati specificamente per il computer ad esempio abbiamo il modello commerciale Philips D6260 che quello dedicato al modello MSX1 VG8020, oppure il Sony che manteneva una cavetteria standard usando le uscite Ear(cuffie), Mic(microfono), ed Rem(controllo remoto). Altresì avevamo invece soluzioni custom dove il registratore a cassette aveva una sua interfaccia vedi lo Spectravideo 728 e il suo predecessore quasi Msx 328.

L'evoluzione delle memorie di massa si ebbe ovviamente con l'uso dei floppy drive, pochi modelli furono da 5.25 con capacità che partivano da 180Kbyte (singola faccia doppia densità prodotti specialmente in Giappone, Brasile, Corea) per finire alla capacità massima di 360K (Doppia faccia doppia densità), per passare quasi subito hai modelli da 3.5 con capacità da 360K o 720Kbyte. Oggi può sembrare scontato che era d'obbligo l'uso dei floppy da 3.5, ma all'epoca pochissimi sistemi avevano tale standard in quanto costoso, infatti ad esempio computer famosi dell'epoca nati dal 1975 fino al 1983-84 usavano quasi tutti i floppy da 5.25 basti ricordare come esempio gli Apple, i Commodore, il Ti99/4A, Osborne, Bondwell, Kaypro, PC IBM ecc.. Per collegare tali periferiche ovviamente era necessaria una circuiteria aggiuntiva che gestisse il floppy drive; questa era integrata nei computer MSX2, mentre nei computer MSX1 venne inserita in una cartridge dalla quale usciva il cavo flat che collegava il floppy drive vero e proprio, come si vede nelle figure seguenti.

Una menzione speciale la facciamo per il floppy drive da 5.25 SVI707 della ditta americana Spectravideo

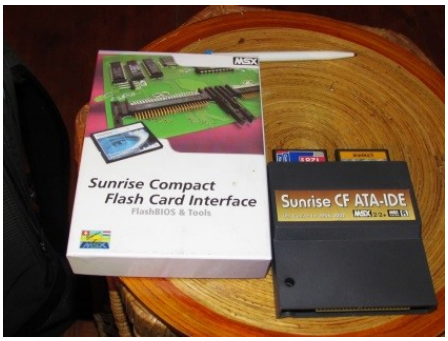
Tale floppy drive è stato fra i primi modelli ad uscire ma aveva una particolarità non rispettava esattamente le specifiche standard sopra enunciate, in quanto aveva il controller del drive integrato nello chassis (tipo il drive commodore 1541), la capacità massima di formattazione che gestiva era di 320Kbyte contro i 360k standard, ma leggeva ed usava correttamente i floppy formattati a 360K da altri computer msx. Inoltre era multi sistema infatti poteva essere usato in ambienti diversi dallo standard MSX in quanto leggeva, scriveva e formattava usando le capacità dei seguenti computer: OSBORNE 1 - 185K, KAYPRO II - 195K, Bondwell 12 - 170K, Bondwell 14 e 16 - 342K ed infine nei sistemi SVI318/328 che usano i drive SVI-605A /SVI-902 - 157K e SVI-605B - 326K permettendo l'interscambio dei programmi specialmente sotto sistema operativo CP/M.

Sempre prendendo come riferimento la ditta Spectravideo come espansioni "storiche" abbiamo una scheda video 80 colonne usabile

principalmente con il CP/M, una scheda seriale RS232 (che successivamente sarà integrata o meno in altri computer MSX vedi ad esempio i computer MSX1 Toshiba), una scheda di espansione Ram da 64K, ed una scheda modem.

Questo era più o meno lo scenario europeo delle periferiche standard MSX fino a quasi la fine degli anni 80, negli anni 90 invece abbiamo una serie di periferiche create dalla ditta svizzera Sunrise che fu l'ultima in Europa a rilasciare prodotti licenziati MSX.

Questa ditta costruì oltre alla scheda sonora MoonSound già vista precedentemente anche una serie di periferiche all'epoca innovative tra le quali ricordiamo l'interfaccia IDE per collegare HardDisk e CD-rom (ne furono fatte varie versioni alcune con l'interfaccia rs232 incorporata, oppure che usavano una Compact Flash al post degli Hard disk), una nuova scheda video e una scheda di video editing avente un chip grafico il V9990 superiore a tutti i modelli MSX, che doveva diventare lo standard di un ipotetico MSX3 ed uno slot expander per collegare fino a otto cartridge/periferiche.



L'importanza dei prodotti rilasciati dalla Sunrise è stata determinante, in quanto a distanza di anni, altre ditte sparse nel mondo quali ad esempio la Tecnobytes brasiliana oppure la 8bit4ever, hanno riprodotto, con le ovvie migliorie della tecnologia attuale, le periferiche create da questa ditta svizzera. Vedi foto allegate:

Con questo abbiamo terminato il nostro piccolo viaggio nel mondo dei computer MSX.

Ovviamente, data la vastità dell'argomento, ho mostrato soltanto quello che a mio parere è l'indispensabile da conoscere per cominciare a capire queste macchine.

Per qualsiasi richiesta, suggerimento o informazione, la redazione di RetroMagazine sarà sempre a vostra disposizione.

Ghost'n COBOL - parte 3 - I file relative

di Francesco Fiorentini

Con questo articolo continua e finisce la nostra avventura con il COBOL del Commodore 64 utilizzando il compilatore **Abacus Cobol 64**. Dopo aver visto come utilizzare l'editor, il compilatore ed aver creato un semplice programma per lavorare con i file sequenziali e' arrivato il momento di utilizzare un tipo di file leggermente piu' complesso, i file **relative**.

La definizione che avevo dato del file relative nel numero scorso era la seguente: *un file **relative** e' un archivio che puo' essere acceduto sequenzialemente, come nei file sequenziali, oppure dinamicamente, usando una chiave come indice. Nel secondo caso per esempio, se volessimo leggere il record numero 10 potremmo, utilizzando l'opportuna chiave, accedere direttamente al record che ci interessa, saltando a pie' pari i 9 record precedenti*. Probabilmente ero stato profetico perche' nell'esempio che vi ho preparato vedremo come scrivere un programma che vada ad utilizzare i file relative per accedere **dinamicamente** ai vari record tramite una chiave ed allo stesso tempo **simuli una lettura sequenziale**.

Perche' ho detto che simuli una lettura sequenziale? Presto detto, il manuale dell'Abacus Cobol 64 ci informa che possiamo accedere i file relative in due modi, con accesso random o con accesso sequenziale, ma i due modi sono incompatibili. Nel momento della definizione del file dovremo scegliere quale approccio utilizzare ed usarlo per tutto il resto del programma. Stara' al programmatore scegliere il modo di accesso migliore in base a quelle che saranno le esigenze del programma che intende realizzare. Onestamente questa limitazione sembrava abbastanza strana anche negli anni 80, quando il COBOL era il linguaggio per eccellenza per creare gli archivi di banche, piccole, medie e grandi imprese, figuriamoci quanto possa sembrare anacronistica adesso.

Ma noi siamo caparbi, non ci facciamo certo fermare da questi *piccoli* inconvenienti in quanto il nostro intento e' didattico e

Figura 1 – Lettura sequenziale partendo dal primo record

vogliamo imparare ad usare il linguaggio COBOL sul Commodore 64, volenti o nolenti questo passa il convento, bando quindi alle ciance e passiamo a vedere come si definisce un file relative nella **input-output section**. ☺

```
input-output section.
file-control.
   select reldata1 assign to disk-1541 drive-8
   organization is relative
   access mode is random
   relative key is rel-key
   file status is file-st.
```

La definizione dei file relative e' piu' complessa di quella dei file sequenziali in quanto l'organizzazione **relative** deve essere esplicitata. Vi ricordo che non specificando nessuna informazione circa la tipologia di file da usare, il compilatore assumera' per default che si tratti di un file sequenziale.

Allo stesso tempo dobbiamo informare il compilatore sul modo di accesso, in questo caso **random**, e su quale sia la chiave da utilizzare per accedere ai record, nel nostro caso **rel-key**. Le altre opzioni le avevamo gia' viste nell'esempio precedente, quindi non credo ci sia la necessita' di spiegarle nuovamente.

Dopo aver definito la tipologia del file dobbiamo assegnargli un nome. Per fare questo dobbiamo, come per i file sequenziali, usare il costrutto **fd** (file description) nella **file section** in **data division**.

```
data division.
file section.
fd  data1
   label record is standard
   value of file-id is "@0:reldata1".
```

A questo punto, sempre nella **file section**, non ci rimane che definire il tracciato record del nostro file relative:

```
01  data-record.
  02 rel-key pic 9(5).
  02 nominativo.
    04 nome pic x(15).
    04 cognome pic x(15).
  02 residenza.
    04 indirizzo pic x(20).
    04 cap pic 9(5).
    04 citta pic x(25).
```

Il tracciato record non dovrebbe presentare sorprese, ormai dovreste avere chiaro come leggerlo. Interessante notare pero' due cose.

La chiave rel-key definita nel tracciato record deve essere un intero (maggiore di zero) ed ovviamente deve avere lo stesso nome come dichiarato nello statement select.

L'altra cosa da notare e' la possibilita' di 'raggruppare' i campi dei record utilizzando i livelli ed i sottolivelli. Nel nostro caso specifico sara' ininfluente ai fini del programma, ma potrebbe sempre tornare utile saperlo (si vedano i livelli 02 e 04).

Ma vediamo adesso il codice del programma vero e proprio (mi scuso per il carattere minuscolo utilizzato, ma per esigenze 'tipografiche' e' stata una scelta obbligata):

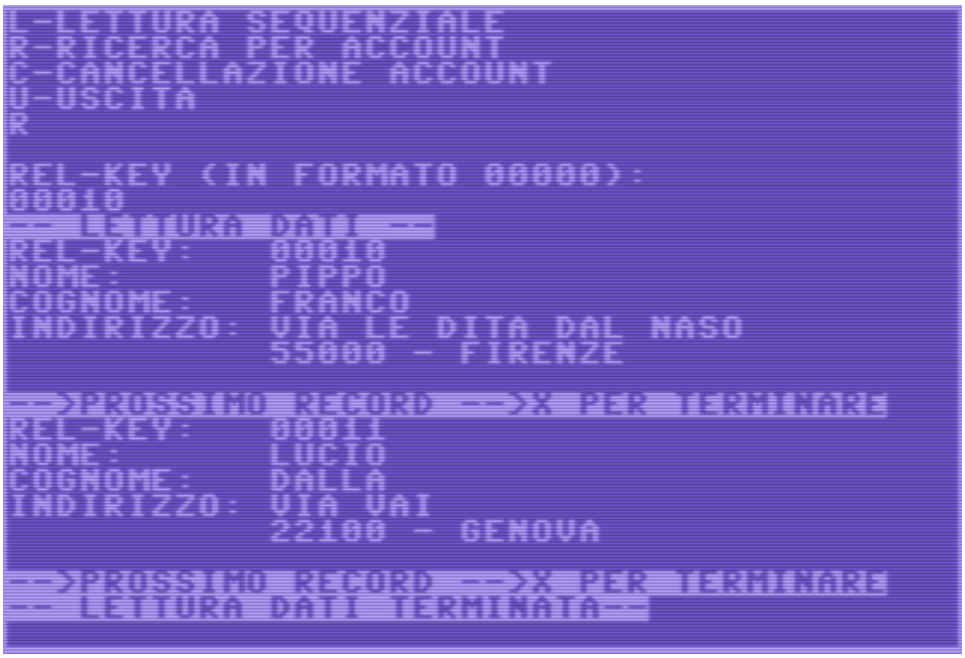


Figura 2 – Lettura sequenziale partendo dal record nr. 10

```
000100 identification division.
000200 program-id. relativefile.
000250 author. retromagazine.
000300 environment division.
000400 configuration section.
000500 source-computer. c64.
000600 object-computer. c64.
000601 input-output section.
000602 file-control.
000603      select reldata1 assign to disk-
1541 drive-8
000604     organization is relative
000605     access mode is random
000606     relative key is rel-key
000620     file status is file-st.
001000 data division.
001001 file section.
001002 fd  reldata1
001003     label record is standard
001004    value of file-id is "@0:reldata1".
001010 01  data-record.
001011     02 rel-key pic 9(5).
001015     02 nominativo.
001016        04 nome pic x(15).
001017        04 cognome pic x(15).
001020     02 residenza.
001026        04 indirizzo pic x(20).
001027        04 cap pic 9(5).
001028        04 citta pic x(25).
001100 working-storage section.
001101 77  scelta pic x.
001102 77  clear-home value chr 147 pic x.
001103 77  return-codice value chr 13 pic x.
001104 77  rvs-on value chr 18 pic x.
001114 77  file-st pic xx.
001115 77  pausa pic x.
001116 77  num-record pic 9(5).
001200 procedure division.
001201 main.
001202     open i-o reldata1.
001203     if file-st not equal to "00"
001204          display "errore in apertura
file"
001205        stop run.
001305 mostra-menu.
001310     display clear-home.
001320          display rvs-on "--> menu
principale - file relative <--".
001330     display " " return-codice.
001335     display " " return-codice.
001340        display "i-inserimento nuovo
record" return-codice.
001350      display "l-lettura sequenziale"
return-codice.
001355      display "r-ricerca per account"
return-codice.
001356     display "c-cancellazione account"
return-codice.
001360     display "u-uscita" return-codice.
001370     accept scelta.
001380     if scelta equal to "u"
001385        close reldata1
001390         display rvs-on "-->programma
terminato. arrivederci..."
001400        stop run.
001410     if scelta equal to "l"
001411        move 00001 to rel-key
001412         perform leggi-file thru end-
leggi-file.
001420      if scelta equal to "i" perform
scrivi-file thru end-scrivi-file.
001430     if scelta equal to "r"
001431        display " " return-codice
001432         display "rel-key (in formato
00000):" return-codice
001433        accept num-record
001434        move num-record to rel-key
001439         perform leggi-file thru end-
leggi-file.
001440      if scelta equal to "c" perform
cancella-record thru end-cancella-record.
001450     go to mostra-menu.
001500**********************
001700 leggi-file.
001701     display rvs-on "-- lettura dati -
-" return-codice.
001707 leggi-loop.
001710      read reldata1 invalid key go to
end-lettura.
001711     if file-st not equal to "00"
001712          display "errore in apertura
file"
001713        stop run.
001715     display "rel-key:    ".
001716     display rel-key return-codice.
001720     display "nome:       ".
001725     display nome return-codice.
001730     display "cognome:    ".
001735     display cognome return-codice.
001740     display "indirizzo: ".
001745     display indirizzo return-codice.
001750      display "            " cap " - "
citta return-codice.
001900    display rvs-on "-->prossimo record
-->x per terminare".
001905     accept pausa.
001910     if pausa not equal to "x"
001911        add 1 to rel-key
001915        go to leggi-loop.
002000 end-lettura.
002005      display rvs-on "-- lettura dati
terminata--" return-codice.
002006     accept pausa.
002030     perform mostra-menu.
002035 end-leggi-file.
002040     exit.
002045**********************
002500 scrivi-file.
002501     display clear-home.
002502      display rvs-on "--> inserimento
dati <--".
002503     display " " return-codice.
002510 accetta-dati.
002511        display rvs-on "rel-key (in
formato 00000):" return-codice.
002512     accept rel-key.
002530    display rvs-on "nome (max 15 chr):"
return-codice.
002540     accept nome.
002550      display rvs-on "cognome (max 15
chr):" return-codice.
002560     accept cognome.
002570     display rvs-on "indirizzo (max 20
chr):" return-codice.
002580     accept indirizzo.
002581        display rvs-on "cap (max 5
numeri):" return-codice.
002582     accept cap.
002584      display rvs-on "citta' (max 25
chr):" return-codice.
```

```
002585     accept citta.
002590        display rvs-on "i dati sono
corretti? s/n" return-codice.
002600     accept scelta.
002605     if scelta equal "n"
002610         go to accetta-dati.
002620     write data-record
002621           invalid key display "chiave
non valida!".
002630     if file-st not equal to "00"
002640         display "errore in scrittura
file"
002650        stop run.
002700     perform mostra-menu.
002710 end-scrivi-file.
002715     exit.
002716*******************************
```

Essendo il COBOL un linguaggio d'alto livello molto discorsivo (verbose) non c'e' necessita' di commenti, in quanto il codice e' facilmente leggibile da chiunque abbia un minimo di dimestichezza con la sua sintassi.

A differenza dei file sequenziali, i file relative permettono l'apertura in **i-o** (input/ouput). Questo ci permette di aprire il file una sola volta all'inizio del programma, compiere tutte le operazioni di lettura e scrittura necessarie e chiudere il file al termine del programma stesso.

Come si evince facilmente dal codice stesso ho implementato una funzione di scrittura ed una funzione di lettura. La funzione di lettura e' interessante in quanto permette, tramite la stessa procedura, di leggere sequenzialmente il file partendo dal primo record (opzione **l-lettura sequenziale**) oppure da un record ben preciso (opzione **r-ricerca per account**). Il *trucco* utilizzato e' quello di ninizializzare la chiave al valore 00001 (primo record) nel primo caso e richiedere invece il valore della chiave all'utente nel secondo caso.

Interessante notare come la lettura sequenziale non sia altro che una lettura random dove di volta in volta viene passato il valore della chiave incrementato di 1. Tanto astuto quanto inutile; per evidenziare infatti un possibile problema ho inserito nel file 8 record. I primi 6 consecutivi e gli ultimi 2 separati da 3 record vuoti. La lettura sequenziale partendo dall'inizio del file si fermera' dopo la lettura del sesto record; per accedere agli ultimi 2 record dovremo usare la funzione **r-ricerca per account** e passare il valore 00010 come chiave. Oppure usare un altro file sequenziale dove tenere traccia dei record memorizzati sul file relative.

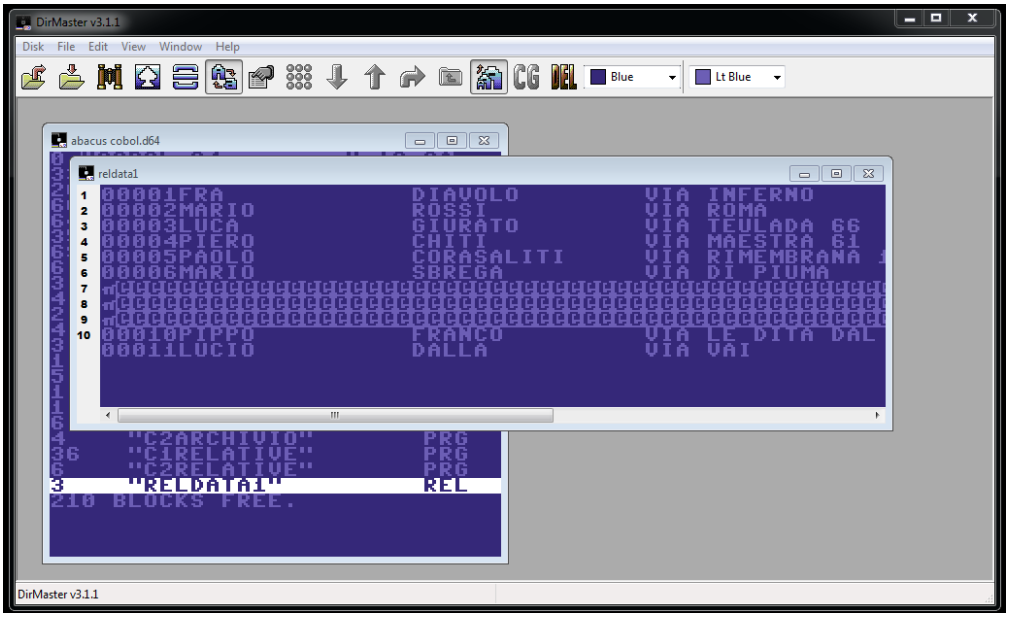


Figura 3 – Il file relative 'reldata1', come si vede dal visualizzatore file di DirMaster ,contiene 8 record, ma solo i primi 6 sono contigui, gli ultimi 2 record sono separati da 3 record vuoti.

Ma le sorprese dell'Abacus Cobol 64 non sono finite qui. Il compilatore manca del tutto della funzione **DELETE** per cancellare un record.

Avevo gia' preparato il menu' principale del programma per supportare la cancellazione di un record, ma quando mi sono reso conto della limitazione ho deciso di desistere dall'intento... Avrei potuto creare una funzione per riempire di spazi un record gia' esistente, ma mi sono chiesto se avrebbe avuto veramente senso. Un record vuoto e' pur sempre un record accessibile, ben diverso quindi dal fatto di essere un record cancellato. Che senso avrebbe avuto? Anche in questo caso sarebbe sato possibile simulare il tutto riscrivendo in un nuovo file tutti i record tranne quello che volevamo cancellare, ma a quale pro? Sarebbe stato soltanto una perdita di tempo, magari un elegante esercizio di stile, ma all'atto pratico poco utile.

Concludendo, posso affermare che l'Abacus Cobol 64 e' un software deficitario, carente di molte funzioni basilari che i programmatori COBOL danno *giustamente* per scontate. Grazie a questa suite possiamo dire con orgoglio che anche il Commodore 64 possiede un compilatore COBOL, ma questo tool e' ben lungi dal risultare efficiente al punto di permettere al COBOL del biscottone di essere utilizzato in ambito professionale. Le limitazioni sono tante e tali da rendere l'Abacus Cobol 64 utile soltanto ai fini della didattica, ma non certo per essere utilizzato anche nelle piu' semplici applicazioni semi-professionali. Quando facciamo una ricerca relativa al C64 ed al suo software siamo abituati a vedere migliaia di risultati, spesso con esempi di codice in BASIC e ASSEMBLY, se il compilatore COBOL fosse stato all'altezza della situazione forse avremmo trovato anche qualche esempio di applicazione pratica sviluppata per il nostro amato 8bit (vista la popolarita' di questo linguaggio negli anni 80) invece non si trova quasi niente. All'inizio mi domandavo come mai; beh, adesso, dopo aver provato a fare qualcosa di concreto credo di aver trovato la risposta. Non a caso il tutorial si chiama **Ghost'n COBOL** ed e' cominciato nel numero di **Halloween**.

Per l'Abacus Cobol 64 e' tutto. Al prossimo compilatore COBOL... ma non subito. ☺

Links utili

Abacus Cobol 64:
https://www.commodoreserver.com/PublicDiskDetails.asp?DID=0FE97DA68C4044D2B30B2106828ACB28

Manuale dell'Abacus Cobol 64:
https://www.lyonlabs.org/commodore/onrequest/COBOL-64_Software_System.pdf

Control character:
https://www.c64-wiki.com/wiki/control_character

NB: le istruzioni per compilare il programma si trovano nella prima parte della guida, nel numero 10 di RetroMagazine.

Mariel Hemingway sul LASER 500

di Antonino Porcino

Era il 1988, e da poco tempo insieme ai miei due amici Agostino e Lorenzo eravamo passati al mondo a 16 bit con il Commodore Amiga. Sembrava incredibile quante cose si potessero fare con quel computer: la grafica, il suono, il multitasking e il mouse. Una vera rivoluzione.

Ma il nostro cuore era ancora nei computer a 8 bit. Ma non per i vari Commodore & c. che si vedevano in giro, no, noi avevamo la nostra piattaforma di nicchia, che per noi era veramente speciale. Si trattava dei computer Laser della Video Technology (VTech), il Laser 310 ma soprattutto il più potente Laser 500.

Laser 310

Laser 500

Erano praticamente sconosciuti in Italia poiché distribuiti solamente con il corso di informatica Scheidegger (che magari qualcuno di voi avrà frequentato). Tutto ruotava intorno al Laser Computer Club, sponsorizzato dalla casa madre, di cui facevamo parte anche noi. C'era pure una rivista omonima bimestrale dove pubblicavamo i listati; online si possono trovare i tre numeri che ho salvato dall'oblio.

La peculiarità del Laser 500 era che, rispetto agli altri computer più diffusi come il C64, non aveva pressoché alcun software. E non essendoci giochi con cui giocare, l'unica cosa che ci si poteva fare era programmare. Da questo punto di vista si rivelò infatti un ottimo strumento didattico. Disponeva di un ricco BASIC Microsoft 3.0 e di una buona combinazione di grafica/colori con cui ci si poteva cimentare (il suono però era penoso – solo un misero buzzer integrato nella tastiera). I programmatori più avanzati potevano avventurarsi a scrivere programmi in assembler Z80 con il semplice ma completo monitor presente nella ROM. Il manuale era estremamente dettagliato, riportando minuziosamente tutte le informazioni utili sulla macchina.

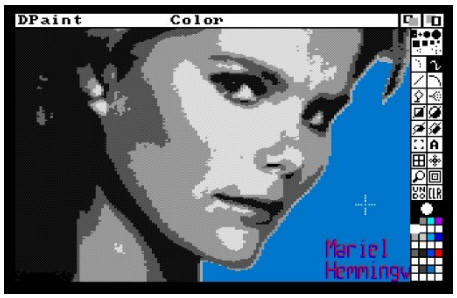
La rivista Laser Computer Club

Ma torniamo al 1988 e all'Amiga. Da qualche giorno stavo sperimentando con l'AmigaBASIC un programma per caricare le immagini in formato IFF (un'altra rivoluzione introdotta con l'Amiga). Tra le immagini con le quali facevo le mie prove vi era un file di nome "Mariel" trovato in uno dei dischi in dotazione con l'Amiga 500. Si trattava di un'immagine digitalizzata a bassa risoluzione di Mariel Hemmingway, l'attrice americana degli anni '70.

La risoluzione ed il numero di colori del file erano molto al di sotto delle potenzialità dell'Amiga, ma questo fece scattare in me l'idea che l'immagine invece potesse essere buona per il Laser 500. Il 500 infatti disponeva di una risoluzione adatta allo scopo di 160x196x16 colori.

La schermata iniziale del Laser 500

All'epoca eravamo fissati con le immagini sul computer, che noi chiamavamo "testate", come quelle popolarizzate dalle cassette del 64 che mostravano un'immagine prima del caricamento dei giochi (un classico erano quelle fatte il con il Koala Paint).



Sul Laser 500 disponevamo già un programma di disegno che avevamo perfezionato nel corso del tempo e con il quale realizzammo una nostra collezione di "testate", ispirandoci a quelle del 64. Ma erano tutte disegnate a mano, non c'era niente di "digitalizzato", così l'idea di

prendere un'immagine come quella di Mariel ci sembrava molto stuzzicante.

Come fare però per trasferire un'immagine dall'Amiga al Laser? Un bel problema! L'Amiga aveva i floppy da 3.5" mentre il Laser 500 quelli da 5.25" che fra l'altro non erano compatibili con nessun altro computer. Di interfacce seriali o parallele neanche a parlarne, il Laser disponeva al massimo della porta registratore a cassette.

Che fare dunque? Ideona. Siccome l'AmigaBASIC faceva sfoggio del nuovo comando "SAY" che pronunciava una stringa di testo, ci venne l'idea di programmare l'Amiga affinché "dettasse" i pixel dell'immagine, i quali potevano poi essere digitati a mano sul Laser. Una cosa lunga e tediosa, ma fattibile! All'epoca avevo 15 anni e mi era sembrata una trovata geniale. Del resto non era poi molto diverso dai listati type-in che fino a qualche anno prima c'erano sulle riviste in edicola.

In breve quindi aggiungiamo una routine al programma BASIC sull'Amiga affinché, dopo aver caricato l'immagine, utilizzi "**SAY A$**" per pronunciare i pixel. Per andar più spediti scegliamo una codifica esadecimale: da `0` a `F` per il colore del pixel, preceduto da un altro numero esadecimale che indicava per quanti punti in orizzontale il pixel doveva essere ripetuto (una sorta di compressione *run-length*). Ad esempio `C8` indicava 12 pixel di colore grigio scuro (8).

Sul Laser 500 modifichiamo il programma di disegno in modo che, digitando sulla tastiera i codici esadecimali dettati dall'Amiga, i pixel vengano caricati nello schermo e poi salvati su disco.

A supporto dell'operazione di trasferimento, facciamo uso di un interfono che ci siamo auto costruiti con dei fili telefonici, collegante casa mia con quella dei miei amici. L'interfono, da noi denominato cavo-tape, sfrutta il circuito di amplificazione audio di due registratori a cassetta del Laser. È utile perché in questo modo possiamo lavorare senza bisogno di spostare fisicamente il computer. L'audio dell'Amiga può essere trasmesso via interfono nell'altra abitazione dove si trova il Laser. Il sistema del cavo-tape poteva anche essere utilizzato per collegare due Laser tramite l'interfaccia registratore, o semplicemente per trasmettere a distanza il segnale video (ma questa è un'altra storia).

Il registratore a cassette DR-30

Parte quindi l'operazione MARIEL HEMMINGWAY: all'inizio tutto fila liscio, io sull'Amiga controllo la velocità di recitazione dei codici esadecimali, mentre i miei amici a casa loro li digitano ascoltando la voce robotica proveniente dall'interfono che l'Amiga pronuncia a ritmo incalzante. I pixel iniziano a comparire sullo schermo e noi ci sentiamo dei piccoli Einstein e dei piccoli Marconi allo stesso tempo.

Ben presto però ci accorgiamo che non è così semplice come pensato inizialmente. I pixel sono decine di migliaia anche se compressi. È un lavoro di parecchie ore, forse giorni. Ci alterniamo, facciamo i turni; ma arrivati circa a metà ci scoraggiamo per l'enorme mole di lavoro rimasta e, non senza esitazione, gettiamo la spugna. L'immagine di Mariel rimarrà così incompleta.

Fast forward di trent'anni... saltiamo dal 1988 al 2018. Adesso gli eventi andranno molto veloci, cercate di starmi dietro.
Mi iscrivo a vari gruppi di retrocomputing su Facebook, e a furia di vedere foto di vecchi 8 bit mi prende la nostalgia di quell'epoca. Mi procuro un Laser 500 su eBay, che figata! Mi rivado a leggere un vecchio thread sul forum *AtariAge* dove ci sono una manciata di utenti che discutono del Laser 500. Le informazioni sono praticamente zero, un tizio però ha il manuale in francese e condivide alcune informazioni di base.

Mi metto quindi a programmare sull'unico emulatore disponibile, il *MAME*, sperando di replicare la piacevole esperienza avuta con il *VICE* in un altro mio passato di retro-programmazione per il VIC-20. Ma ahimè il Laser 500 sul MAME è emulato veramente male: la velocità non è corretta, i file .WAV non caricano, molti tasti non funzionano e si impalla continuamente. È impossibile fare cross-development. Apro alcuni bug report sul sito del MAME ma restano lì in stand-by per un tempo infinito. Intuisco poi che i programmatori che inizialmente hanno implementato il Laser sul MAME sono usciti fuori dal progetto, per cui non c'è speranza che i bug vengano risolti.

Decido allora di scrivere un emulatore tutto mio. Il mio target è il browser perché nel tempo sono diventato un programmatore JavaScript e perché in questo modo l'emulatore può raggiungere un numero maggiore di utenti.

Per documentare i miei progressi con l'emulatore fondo un gruppo su Facebook, e grazie a questo le informazioni sul Laser 500 iniziano ad arrivare a poco a poco da ogni parte. Un utente ritrova la sua collezione di dischetti dalla quale riusciamo a recuperare con molte difficoltà alcuni giochi, che poi sono gli unici esistenti per questo computer. Ma soprattutto riusciamo a recuperare il DOS del computer che si riteneva perso. Con il DOS, un altro utente fa il reverse-engineering dell'interfaccia del floppy disk, che io prontamente implemento nel nuovo emulatore. Adesso si possono caricare i dischetti anche senza avere il disk drive, oggi introvabile. Si scopre fra l'altro che il drive del Laser è praticamente identico a quello dell'Apple II, con solo una minima differenza nella codifica dei settori che però lo rende incompatibile.

Un altro utente ancora ci fornisce il *dump* esatto della ROM caratteri e riusciamo a procuraci anche lo schema elettrico che chiarisce parecchi dubbi sulla macchina. Adesso ho una quasi piena conoscenza dell'hardware e l'emulatore funziona alla grande, mi diverto parecchio ad aggiungergli nuove funzioni. Nel farlo contribuisco a vari progetti open source come *Z80.js* e *z88dk*, il compilatore C per Z80, con il quale inizio a scrivere i primi programmi per il Laser. La mia padronanza dello Z80 cresce esponenzialmente tanto che mi avventuro a sviluppare una routine piuttosto avanzata di *turbo tape*. La sottile soddisfazione di poter

fare quello che all'epoca era impensabile... non ha prezzo!

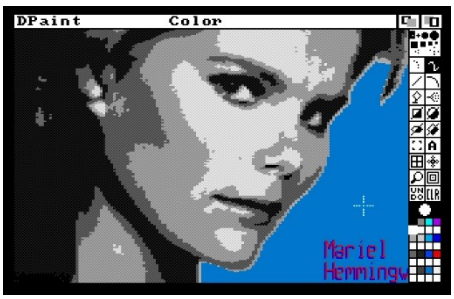
Ma, e Mariel Hemingway?

Ebbene, adesso è arrivato il momento di completare ciò che era rimasto a metà. Sento un impulso che mi spinge a chiudere il cerchio dopo 30 anni: Mariel Hemmingway deve approdare sul Laser 500!

Ok, ma da dove iniziare? La prima cosa è ovviamente ritrovare l'immagine originale di Mariel. Cerco su *Google Immagini* ma non trovo nulla. Mi concentro sui confusi ricordi che ho dell'epoca e mi viene in mente che il disco che conteneva l'immagine era uno di quelli in dotazione all'Amiga. Mi ricordo solamente che era insieme ad un programma spreadsheet chiamato *Logistix*. Faccio una ricerca su internet e dopo un po' scopro che si trattava di una confezione chiamata "lo scrigno del software" della CTO, venduto in offerta insieme ad Amiga e modulatore TV.

Fortunatamente qualcuno anni fa ne ha fatto il dump mettendo le immagini dei dischi sul sito The Zone. Dopo mille link 404 riesco a scaricarli, e mi procuro anche l'emulatore Amiga WinUae. Tiro giù anche il Deluxe Paint, poiché dopo tutti questi anni non mi ricordo più qual era l'utility per aprire i file IFF.

Ci sono, sfoglio tra i dischi ... ed eccola lì. Mariel, ci sei ancora dopo tutto questo tempo!



Non sapendo come estrarla dall'emulatore WinUae, faccio uno screenshot ed importo l'immagine nel Photoshop. Usando il metodo "scala colori" converto nella palette a 16 colori del Laser 500 e riduco la risoluzione da 320x200 a 160x192. Non vi è nessuna perdita di qualità perché come dicevo prima, l'immagine di partenza è già a bassa risoluzione e con pochi colori.

Salvo poi in formato RAW che è quello dove i pixel sono tali e quali, un byte per pixel senza compressione e senza intestazioni varie che ora sarebbero d'impiccio. Adesso mi serve un programma sul Laser 500 per visualizzare le immagini, poiché il programma di disegno che avevamo negli anni '80 è andato perso insieme a tutto il resto del software autoprodotto.

Con il compilatore C abbozzo una routine, ma sorgono i primi problemi. Uno è che nella modalità grafica "GR 3" (160x192) il Laser 500 non ha una memoria lineare: alla riga n non segue la riga n+1, ma la riga n+x, dove x è calcolato secondo uno schema piuttosto convoluto. Probabilmente i progettisti hanno usato questa codifica perché semplificava la costruzione del chip video (un chip custom della VTech), scaricando però sul programmatore l'onere di gestirne la complessità.

Il chip video del Laser 500

Alla fine riesco a ricavare la formula che dalla i-esima riga permette di risalire all'indirizzo in memoria video. È un guazzabuglio di shift e maschere di bit. In linguaggio C diventa:

```
unsigned int indirizzo =
((riga & 0b00000111) << 11)   +
((riga & 0b00111000) <<  5)   +
        ((riga & 0b11000000)) +
        ((riga & 0b11000000) >>  2);
```

Per una maggiore velocità però traduco (a mano) la routine dal C all'assembler dello Z80:

```
; in L=row, out HL=address
get_row_address:
   ld   e, l
   ld   a, e
   and  0b11000000
   ld   l, a
   rra
   rra
   add  a, l
   ld   l, a
   ld   a, e
   and  0b111000
   rra
   rra
   rra
   ld   h, a
   ld   a, e
   and  0b111
   rla
   rla
   rla
   add  h
   ld   h, a
   ret
```

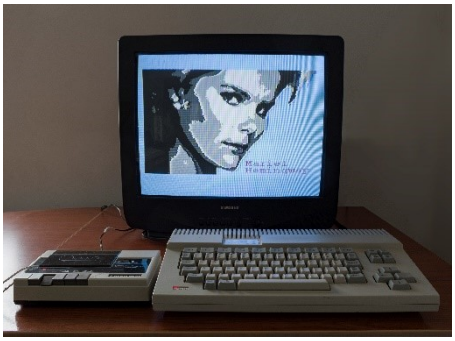
RRA e RLA sono le istruzioni che realizzano i vari bit shifting. Una volta ottenutone l'indirizzo, si può procedere a copiare la riga dal buffer dell'immagine in RAM alla memoria video. Lo si può fare pixel per pixel, oppure vi è un modo molto veloce in assembler Z80 che consente di copiare più byte alla volta con una sola istruzione. Si tratta dell'istruzione *LDIR* (load-increment-repeat). Questa non fa altro che copiare dalla locazione puntata da HL nella locazione puntata da DE, per un numero di byte indicato da BC. Nel codice qui sotto, HL punta al buffer dell'immagine, DE punta a $4000 nella memoria video, BC conta 80 byte (ossia 160 pixel).

```
copyrow:
   push hl
   ld   l, (row)
   call get_row_address
   ld   de, 0x4000
   add  hl, de
```

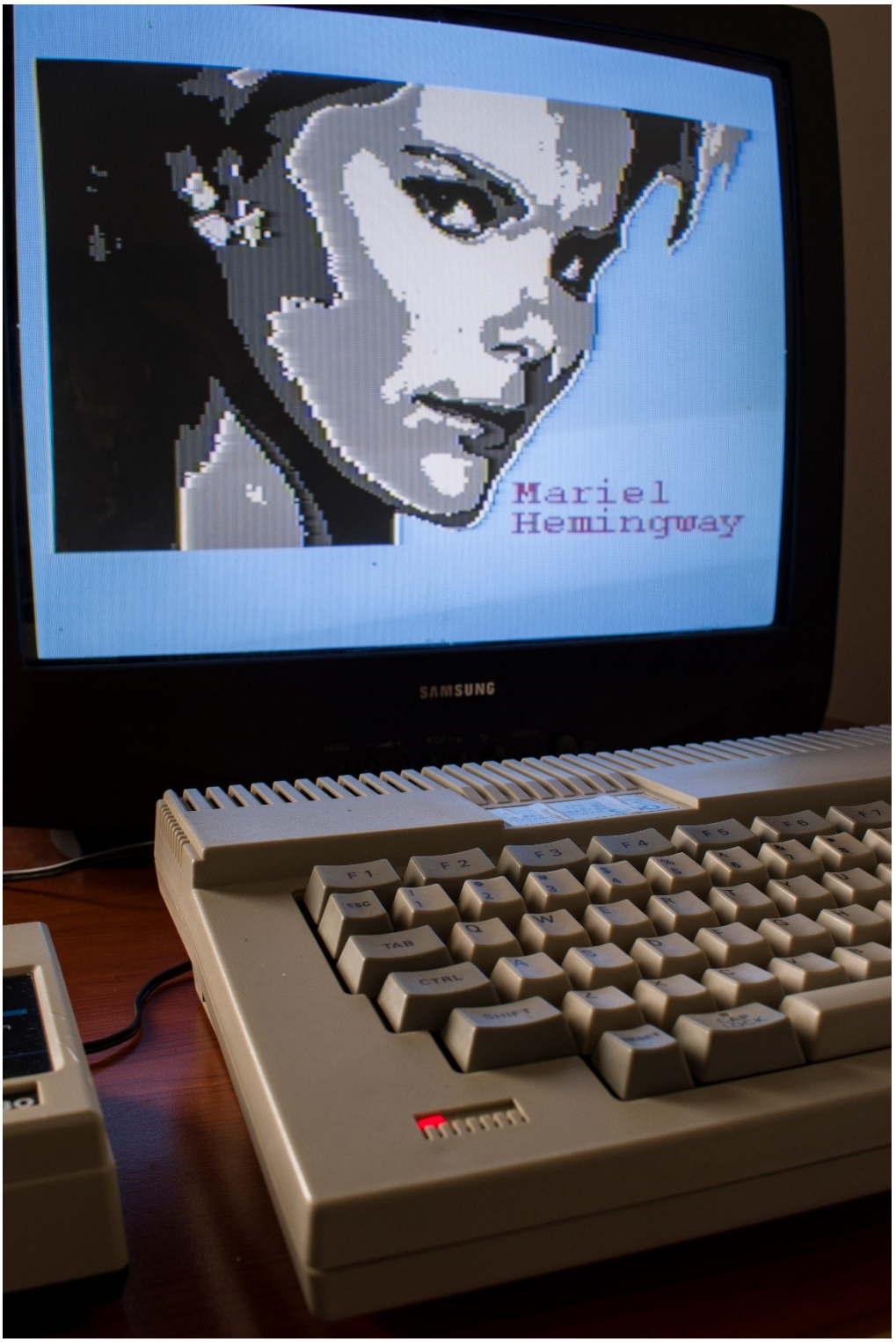
```
ex   de, hl
ld   bc, 80
pop  hl
ldir
```

Impiego un po' prima di arrivare ad una routine funzionante. In assembler quando qualcosa va storto, il computer si pianta o si resetta senza lasciare il minimo indizio di cosa sia successo. Nel Laser c'è un tasto RESET collegato direttamente all'omonimo pin della CPU che è in grado di riportare il prompt *Ready* nella maggior parte dei casi (simile al RUN STOP+RESTORE del C64). Ma è difficile fare il debug con solo il prompt. Usando l'emulatore invece, posso ispezionare lo stato della macchina emulata senza interferire con l'esecuzione del programma e posso collegare routine di *debug* personalizzate al volo, secondo l'errore che di volta in volta si presenta. L'aiuto dell'emulatore è impagabile.

Compilo dunque tutto insieme: bitmap + programma che visualizza l'immagine. Trasformo il risultato in file audio .WAV in modo da poterlo caricare utilizzando il registratore a cassette. Uso l'opzione turbo tape altrimenti ci mette una vita a caricare. Do il comando *CRUN* sul computer e premo PLAY sul registratore. Dopo un po'... voilà!!! MARIEL HEMINGWAY FINALMENTE SUL LASER 500.



Un sussulto di nostalgia mi coglie mentre con soddisfazione osservo l'immagine sflickerante sullo schermo.



Ringrazio personalmente Antonino per questo suo contributo spontaneo che mi ha tenuto letteralmente incollato allo schermo. Io personalmente, gli altri membri della redazione ed i lettori tutti ci auguriamo di poter leggere presto altri tuoi articoli. *Francesco Fiorentini*

Links utili e contatto di Antonino

Antonino Porcino:
nino.porcino@gmail.com

Emulatore online Laser 500:
https://nippur72.github.io/laser500emu

Mariel sull'emulatore:
https://nippur72.github.io/laser500emu?load=mariel

Laser 500 Facebook group:
https://www.facebook.com/groups/263150584310074/

RetroMath: Curve, frattali e tartarughe

Di Giuseppe Fedele e Marco Pistorio

Se qualcuno ci chiedesse una definizione di "curva", intuitivamente diremmo che una curva è un ente geometrico ad una sola dimensione. Più formalmente Descartes definì una curva piana come "l'insieme dei punti del piano che soddisfano un'equazione della forma $F(x,y)=0$". Nella seconda metà dell'800 il matematico francese Camille Jordan formalizzò la nozione intuitiva di traiettoria: "una curva piana è l'insieme dei punti le cui coordinate *x* e *y* sono assegnate da due funzioni continue:

$$x = \phi(t)$$
$$y = \psi(t)$$

con *t* appartenente all'intervallo unitario *[0,1]*.

Con in mente queste definizioni è evidente che una curva non può ricoprire un intero quadrato che invece ha due dimensioni.

Ma attenzione perché l'intuizione comune spesso inganna!!!

Nel 1890 il matematico Giuseppe Peano (1858-1932) pubblicò un articolo dal titolo "*Sur une courbe, qui remplit toute une aire plaine*" (Figura 1) in cui presentava una curva espressa in forma parametrica come la definizione di Jordan ma che fornisce una applicazione suriettiva dell'intervallo *[0,1]* su tutto il quadrato unitario. Riempie tutto il quadrato!!!

Sur une courbe, qui remplit toute une aire plane.

Par

G. Peano à Turin.

Dans cette Note on détermine deux fonctions x et y, uniformes et continues d'une variable (réelle) t, qui, lorsque t varie dans l'intervalle $(0, 1)$, prennent toutes les couples de valeurs telles que $0 \leq x \leq 1$, $0 \leq y \leq 1$. Si l'on appelle, suivant l'usage, *courbe continue* le lieu des points dont les coordonnées sont des fonctions continues d'une variable, on a ainsi un arc de courbe qui passe par tous les points d'un carré. Donc, étant donné un arc de courbe continue, sans faire d'autres hypothèses, il n'est pas toujours possible de le renfermer dans une aire arbitrairement petite.

Figura 1. Articolo di Peano del 1890

Il risultato, fortemente controintuitivo, aprì la strada alla riconsiderazione del concetto di dimensione, concetto chiave in topologia e negli oggetti frattali (che abbiamo visto in un numero precedente di **RetroMagazine**), e nella teoria dei sistemi dinamici e del caos.

Vediamo come si costruisce questa curva (Figura 2):

Prendiamo un segmento *AB* e dividiamolo in tre parti uguali. Costruiamo due quadrati sul lato CD. Si ottiene una figura poligonale formato da nove segmenti che può essere percorsa senza alzare la matita e senza passare due volte sullo stesso segmento. Questa costruzione può essere ripetuta su ciascuno dei nove segmenti, dividendo ognuno in tre parti uguali e costruendo sulla parte centrale un quadrato con lo stesso procedimento.

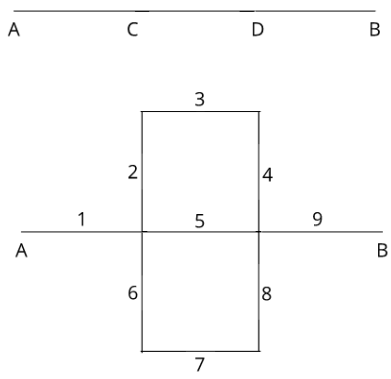


Figura 2. Costruzione della curva di Peano

Per apprezzare meglio il fatto che si tratta di una curva continua (che può essere disegnata senza mai staccare la matita dal foglio) in Fugura 3 sono riportate le prime due iterazioni dl procedimento di Peano troncando gli angoli.

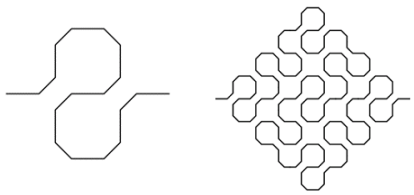


Figura 3. Due iterazioni del procedimento di Peano.

Reiterando il procedimento (un numero teoricamente infinito di volte) la forma della curva di Peano non sarebbe altro che quella di un normale quadrato.

Riprendiamo il concetto di dimensione. Secondo il matematico alessandrino Euclide (come descritto nei suoi *Elementi*) il punto ha dimensione 0, la retta ha dimensione 1, le figure nel piano hanno dimensione 2, i solidi nello spazio hanno dimensione 3. Ma che dimensione ha un albero o una nuvola? Possiamo dire che hanno dimensione pari a 3 visto che occupano un volume? Per gli oggetti irregolari tipo i frattali o le curve che stiamo considerando in questo articolo si introduce il concetto di "dimensione frattale" che misura il grado di irregolarità dell'oggetto.

Facciamo riferimento alla Figura 4. Prendiamo un segmento e dividiamolo in N parti ciascuna delle quali misurerà quindi $1/N$. Abbiamo cioè diviso il segmento in N segmenti simili a quello di partenza.

Consideriamo adesso un rettangolo di lati a e b; dividiamo entrambi i lati in k parti ottenendo così un numero di parti $N = k^2$. Tutte le parti in cui è diviso il rettangolo sono simili al rettangolo di partenza.

L'area del rettangolo di partenza è $A = ab$, mentre l'area di una delle parti in cui è stato diviso è $A_i = \frac{ab}{k^2}$. Il rapporto tra le due aree (rapporto di superficie) è quindi:

$$r^2 = \frac{A}{A_i} = k^2,$$

da cui si ottiene che il rapporto lineare è

$$r = \sqrt{N} = \sqrt{k^2} = k.$$

Facciamo la stessa cosa con il parallelepipedo di lati a, b e c. In questo caso il volume del parallelepipedo iniziale è

$$V = abc$$

mentre il singolo blocchetto avrà un volume

$$V_i = \frac{abc}{k^3}.$$

Il rapporto di volume sarà dunque

$$r^3 = \frac{V}{V_i} = N = k^3$$

da cui si ha

$$r = \sqrt[3]{N} = k.$$

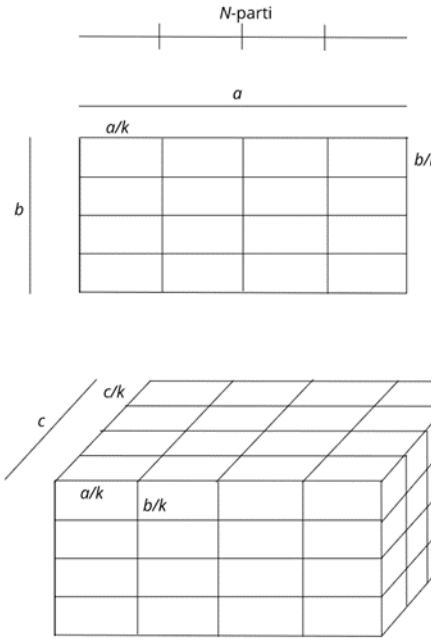


Figura 4. Dimensione frattale

Generalizzando per una qualunque dimensione D, si ha

$$r = N^{\frac{1}{D}} = k.$$

Questa equazione viene detta equazione di Haussdorff, da cui si ottiene che

$$D = \log_k N.$$

Nel caso della curva di Peano, abbiamo che $k = 3$ ed $N = 9$ per cui la dimensione $D = \log_3 9 = 2$.

La curva di Koch (Figura 5) inventata nel 1904 dal matematico svedese Helge von Koch è un'altra curva frattale che si costruisce ricorsivamente come:

1. Dividere il segmento in tre parti uguali;
2. Cancellare il segmento centrale, sostituendolo con due segmenti pari ai lati di un triangolo equilatero;
3. Ripetere la procedura per ogni segmento individuato.

Quanto vale la sua dimensione? In questo caso $k = 3$ e $N = 4$, quindi

$$D = \log_3 4.$$

Attenzione per calcolare $\log_a b$ in base e, basta ricordare che

$$\log_a b = \frac{\log_e b}{\log_e a}$$

per cui

$$D = \log_3 4 = \frac{\log_e 4}{\log_e 3} = 1.26186.$$

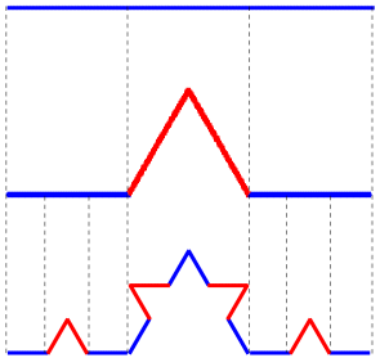


Figura 5. Due iterazioni della curva di Koch

Adesso introduciamo brevemente il linguaggio **LOGO**, che ci permetterà di riprodurre con un approccio semplice le curve illustrate.

- **La "geometria della tartaruga" e sue applicazioni**

La geometria della tartaruga si differenzia dal modo tradizionale di disegnare al computer perché con tale metodologia si parte dal presupposto che il disegno avvenga dall'interno piuttosto che dall'esterno oppure dall'alto.

Ad esempio con il comando "gira a sinistra" non si esprime una direzione assoluta ma una direzione relativa all'orientamento corrente della "tartaruga" (ovvero lo speciale cursore che si adopera con la grafica della tartaruga, che viene raffigurato in genere a forma di triangolo rovesciato). Con il comando "vai avanti di 20 passi" ci si riferisce alla posizione e alla direzione correnti, sempre riferite alla tartaruga.

Quali vantaggi presenta questo approccio? Ad esempio, che disegnare un quadrato con i lati inclinati è facile esattamente come disegnare un quadrato con i lati orizzontali e verticali: la sequenza delle istruzioni sarà la stessa, cambierà solo la posizione iniziale della tartaruga (vedi fig. 1) Un altro vantaggio è che è un modo di procedere analogo al modo con cui ci si muove in uno spazio bidimensionale, quindi risulta abbastanza intuitivo, dal punto di vista pedagogico, apprendere tale metodologia.

Passaggi per ottenere il quadrato	Risultato con tartaruga inizialmente a 0° (immagine sopra) ed a 45° (immagine sotto)
Vai avanti di 10 passi vai a destra di 90 gradi Vai avanti di 10 passi vai a destra di 90 gradi Vai avanti di 10 passi vai a destra di 90 gradi Vai avanti di 10 passi vai a destra di 90 gradi	

Figura A

Negli anni '60 fu sviluppato un apposito linguaggio, denominato LOGO, che permetteva di pilotare un robot che rispondeva ad una serie di comandi come questi elencati di seguito:

- *Avanti e Indietro (forward e back)* seguiti dal numero di "passi" da compiere.
- *Destra e sinistra (right e left)*, seguiti dall'angolo di rotazione espresso in gradi relativi alla rotazione da compiere.
- *Con Pennagiu e Pennasu (pendown, penup)* è possibile ordinare alla tartaruga di tracciare una linea durante il suo cammino oppure no.

Il linguaggio LOGO include anche molti comandi per la gestione di input/output testuale e per l'elaborazione di dati (operatori di confronto, variabili, cicli, selezioni condizionali). E' un linguaggio fortemente procedurale e si presta molto bene per elaborazioni di tipo ricorsivo, ovvero permette di lanciare procedure che richiamano se stesse e ciascuna procedura opera sulle sue variabili all'interno di un proprio "ambito".

Esistono interpreti per diverse piattaforme, a partire dall'Apple II, il Commodore 64, l'MSX, lo ZX Spectrum fino ai PC con Windows, piuttosto che Linux etc. Tali interpreti vengono spesso impiegati a scopo didattico, in particolare per lo studio della geometria.

Vedremo in seguito alcune applicazioni di procedure LOGO, che però non realizzeremo

con un interprete LOGO bensì tramite apposite routines BASIC 2.0 e l'estensione Simon's BASIC. Per poter raggiungere tale obbiettivo però dovremo prima risolvere due ordini di problemi diversi:

A) Riuscire ad implementare almeno un sottoinsieme di comandi del LOGO tramite BASIC.

B) Fare girare le procedure in maniera ricorsiva, dove ogni chiamata ad una procedura permetta di agire SOLO sulle variabili all'interno dello stesso ambito ("scope", per utilizzare un tecnicismo per i più avvezzi...) Ricordiamo che il BASIC del Commodore 64 tratta ciascuna variabile come variabile pubblica (o globale). Ciò significa che il valore di ciascuna variabile è visibile e modificabile da qualsiasi parte del programma BASIC.

Studio dei comandi LOGO per loro implementazione in BASIC

Vediamo come ottenere un sottoinsieme dei comandi LOGO che possono esserci utili.

Il sottoinsieme seguente potrebbe essere già sufficiente per i nostri scopi:

- **FD (forward, Avanti)**
- **BK (Back, Indietro)**
- **RT (Right, Destra)**
- **LT (Left, Sinistra)**
- **PU (Penna Su)**
- **PD (Penna Giù)**

I comandi PU e PD possono essere gestiti facilmente adoperando una variabile di tipo 'flag' che, in funzione del valore ivi contenuto, tracci sullo schermo oppure no.

Ragioniamo intanto sui comandi **LT** ed **RT**, che permettono alla tartaruga di ruotare di un angolo arbitrario a in gradi.

Immaginiamo ci sia quindi una variabile AN che rappresenti l'angolo di rotazione della tartaruga, dato che potrà essere cambiato mediante i comandi LT ed RT.

Potremmo già formalizzare la cosa in questi termini:

LT a → AN=AN + a * PIGRECO / 180

RT a → AN=AN - a * PIGRECO /180

Perchè a deve essere moltiplicato per il fattore PIGRECO/180? Perchè in tal modo convertiamo la misura dell'angolo a espressa inizialmente come gradi in radianti.

Tutte le funzioni trigonometriche presenti nel BASIC 2.0 (ed anche nella maggioranza dei linguaggi ad alto livello) lavorano in radianti e non in gradi, purtroppo.

Per effettuare la conversione, occorre ricordare che 360 gradi equivalgono a 2*PIGRECO radianti.
Da qui, **360 : 2*PIGRECO = a gradi : a radianti**
ovvero, **a radianti = 2*PIGRECO * a gradi / 360**
infine, semplificando nel secondo membro per 2 otteniamo:

a radianti = a gradi * PIGRECO / 180

Trattiamo ora i comandi **FD** e **BK**, anch'essi speculari nel loro comportamento, un po' come i comandi RT e LT già visti.

Teniamo presente il fatto che la "tartaruga" si muove in uno spazio a due dimensioni, caratterizzato da due coordinate, x ed y. Consideriamo inoltre che il suo angolo di rotazione sarà espresso dalla variabile AN.

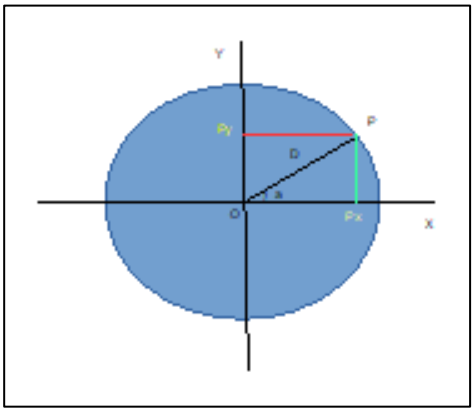
Una prima formalizzazione del funzionamento di tali comandi potrebbe essere il seguente :

FD passi → x_finale = x_iniziale + passi * cos (AN), y_finale=y_iniziale + passi * sin (AN)

BK passi → x_finale = x_iniziale - passi * cos (AN), y_finale=y_iniziale - passi * sin (AN)

Per comprendere meglio il significato di tali relazioni però serve un ripasso veloce di alcuni elementi di trigonometria. Prendete come riferimento la figura riprodotta in alto a destra in questa pagina.

Immaginate che la tartaruga si trovi nel punto O, debba percorrere una distanza D arrivando al punto P. Si dimostra che la coordinata Px del punto P vale **D*cos(a)**, mentre la coordinata Py del punto P vale **D*sin(a)**.



Meccanismo per ottenere delle variabili "locali" con il BASIC 2.0

Questo è il secondo ostacolo da risolvere per gestire in BASIC quanto faremmo sfruttando il linguaggio LOGO, ovvero gestire le chiamate alle procedure assicurandosi che ciascuna procedura possa operare SOLO sulle variabili all'interno del proprio "ambito" o "scope".

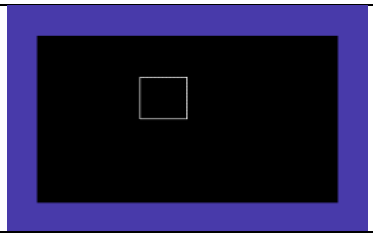
Nativamente, come già detto, il BASIC del C64 gestisce ciascuna variabile come variabile pubblica (o globale). Per far sì che una variabile possa essere gestita come variabile locale in maniera immediata, abbiamo implementato una soluzione molto semplice. Dal momento che le variabili gestite dalle procedure ricorsive si chiamano allo stesso modo, e non potendo "proteggere" in alcun modo il contenuto di una variabile pubblica, abbiamo pensato ad un "trucchetto". Non utilizzeremo variabili ma vettori, ovvero arrays monodimensionali. In tal modo manterremo lo stesso nome della variabile per ciascuna 'istanza'. Invece della variabile v per esempio, avremo il vettore v(n). Inoltre, gestendo all'interno delle chiamate alla procedura una variabile che mi faccia da indice, possiamo far sì che la procedura chiamata la prima volta, legga (ed eventualmente modifichi) v(1), la procedura chiamata la seconda volta legga (ed eventualmente modifichi) v(2) e così via. C'è un prezzo da pagare però. Più memoria. Invece dello spazio necessario per memorizzare il contenuto di una variabile, avremo bisogno dello spazio necessario per contenere un array.

Esempi di procedure LOGO

Iniziamo con il tracciamento di un quadrato, procedura già anticipata all'inizio di questa dissertazione, che possiamo ora vedere più nel concreto.

Esempio 1

```
CLEARSCREEN
FD 10
RT 90
FD 10
RT 90
FD 10
RT 90
FD 10
RT 90
```



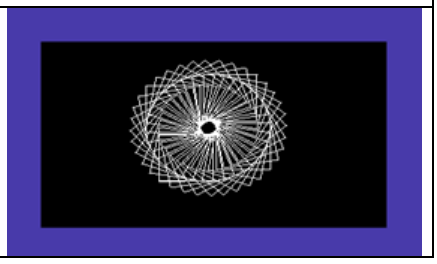
Fin qui probabilmente non vi siete meravigliati molto... :)

Vediamo cosa succede invece se, dopo aver tracciato il quadrato, ruotiamo leggermente la tartaruga e continuiamo. Questo vi piacerà di più!

Esempio 2

```
CLEARSCREEN

REPEAT 360/10 [
FD 10
RT 90
FD 10
RT 90
FD 10
RT 90
FD 10
RT 90
PU
FD 10
RT 10
PD ]
```



Come potete notare, per "spostare" leggermente la tartaruga dal suo punto di arrivo, la spostiamo in avanti di 10 e ruotiamo di 10 gradi verso destra, ma con la penna su e quindi tale movimento non viene tracciato. Avrete anche notato la facilità nel realizzare un disegno che, con una metodologia diversa, richiederebbe un po' di impegno in più. Ma la potenza del LOGO si evince anche dalle sue procedure ricorsive.

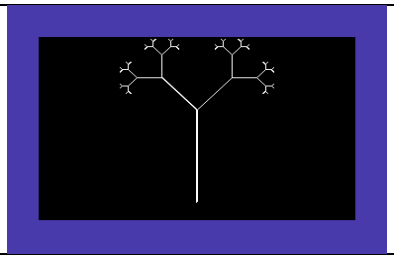
Ricorsione con LOGO

Vediamo come realizzare un albero "binario". Si inizia con un tronco e due diramazioni simmetriche. Le ramificazioni successive sono più corte ma hanno la stessa medesima struttura: uno stelo centrale ed un rametto per parte. Se il procedimento continuasse all'infinito, avremmo l'immagine di un albero con infinite ramificazioni. La procedura RAMIFICAZIONE prevede due parametri, LUNGHEZZA (che sarà la lunghezza del tronco) ed il numero di livello. In ciascun livello la lunghezza delle diramazioni sarà la metà rispetto a quella del tronco centrale. Chiamando la procedura RAMIFICAZIONE con un certo numero di LIVELLO verrà chiamata ricorsivamente la stessa procedura, dimezzando via via LUNGHEZZA e decrementando LIVELLO di 1 unità per volta. Quando LIVELLO arriva al valore o la procedura termina.

Esempio 3

```
CLEARSCREEN

TO RAMIFICAZIONE:
LUNGHEZZA :LIVELLO
IF :LIVELLO=0 THEN STOP
FD :LUNGHEZZA
LT 45
RAMIFICAZIONE (:LUNGHEZZA/2)
(:LIVELLO-1)
RT 90
RAMIFICAZIONE (:LUNGHEZZA/2)
(:LIVELLO-1)
LT 45
BK :LUNGHEZZA
END
```

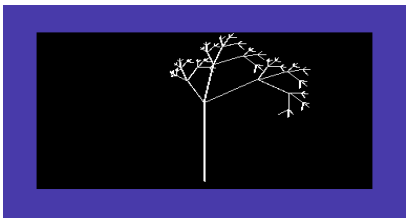


E' possibile rendere l'albero ancora più realistico con qualche accorgimento:

Esempio 4

```
CLEARSCREEN

TO RAMIFICAZIONE:
LUNGHEZZA :LIVELLO
IF :LIVELLO=0 THEN STOP
FD :LUNGHEZZA
LT 30
RAMIFICAZIONE  (:LUNGHEZZA/3)
(:LIVELLO-1)
RT 40
RAMIFICAZIONE  (:LUNGHEZZA/2)
(:LIVELLO-1)
RT 50
RAMIFICAZIONE  (:LUNGHEZZA/1.7)
(:LIVELLO-1)
LT 60
BK :LUNGHEZZA
END
```



I rami adesso sono 3 per ciascun livello, e l'orientamento e la lunghezza di ciascun ramo è leggermente diversa.

A questo punto possiamo realizzare anche alcune curve "frattali". Adesso proviamo ad ottenere la cosiddetta "curva di Koch", conosciuta anche come "curva a fiocco di neve". Il procedimento potrebbe essere il seguente: si disegni un triangolo equilatero, poi si suddivida ciascun lato in tre segmenti uguali e si tracci un nuovo triangolo equilatero sul segmento centrale. Dopo aver cancellato le linee in comune, si esegua la stessa sequenza di operazioni su ogni lato della figura ottenuta. Ripetendo questo procedimento molte volte, otterremo una curva come da illustrazione seguente:

Esempio 5

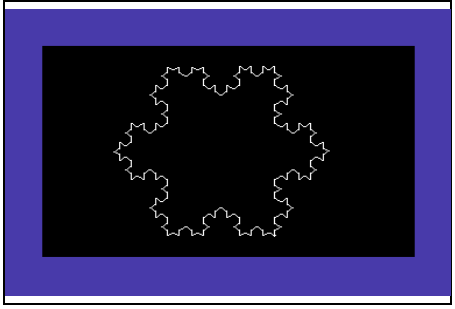
```
CLEARSCREEN

TO FIOCCO: LUNGHEZZA :LIVELLO
REPEAT 3
[LATO :LUNGHEZZA :LIVELLO RT 120]
END

TO LATO :LUNGHEZZA :LIVELLO
IF :LIVELLO=0 THEN FD :LUNGHEZZA
STOP
LATO (:LUNGHEZZA/3) (:LIVELLO-1)
LT 60
```

```
LATO (:LUNGHEZZA/3) (:LIVELLO-1)
RT 120
LATO (:LUNGHEZZA/3) (:LIVELLO-1)
LT 60
LATO (:LUNGHEZZA/3) (:LIVELLO-1)

END
```

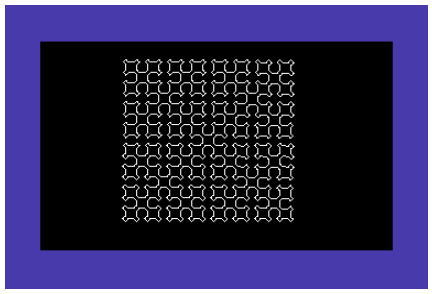


Vediamo un'altra curva classica, la "curva di Sierpinsky". La curva di livello 1 è composta da quattro lati collegati da quattro diagonali. La procedura principale, SIERPSK suddivide il procedimento in quattro parti, che la procedura LATO esegue una alla volta. Ogni lato è composto da tre segmenti, due diagonali ed uno orizzontale o verticale. Quando la procedura passa al livello 2, ciascuna diagonale viene sostituita da una serie di tre segmenti più piccoli ed ogni linea orizzontale o verticale da due serie di tre segmenti simili ai precedenti collegati da una linea. Lo stesso procedimento si ripete per tutti i livelli successivi. Il risultato è quello che vediamo di seguito:

Esempio 6

```
CLEARSCREEN

TO SIERPSK :LATO :LIVELLO
MAKE DIAG :LATO/SQRT(2)
REPEAT 4 [LATO :LIVELLO RT 45
FD :DIAG RT 45]
END
TO LATO :LIVELLO
IF :LIVELLO=0 THEN STOP
LATO (:LIVELLO-1)
RT 45
FD :DIAG
RT 45
LATO (:LIVELLO-1)
LT 90
FD :LATO
LT 90
FD :LATO
LT 90
LATO (:LIVELLO-1)
RT 45
FD :DIAG
RT 45
LATO (:LIVELLO-1)
END
```



Speriamo abbiate compreso, grazie anche a questa veloce "carrellata", le potenzialità del linguaggio LOGO.

Le immagini mostrate tuttavia sono state ottenute da programmi BASIC che girano con l'estensione Simon's Basic su C64 che simulano parzialmente l'ambiente LOGO, secondo quanto illustrato in questo articolo. Sarà possibile scaricare una immagine .D64 contenente tali programmi all'indirizzo: http://www.retromagazine.net/getrm.php?id=logo64

Codice relativo all'esempio 1 "Tracciamento Quadrato"

```
5 PG=4*ATN(1)
100 REM SIMULAZIONE AMBIENTE
LOGO.
110 GOTO 4000
120 :
121 PROC FD
122
XF=XI+PR*COS(AN):YF=YI+PR*SIN(AN)
123 IF FG=1 THEN LINE XI,YI,XF,YF,1
124 XI=XF:YI=YF
125 END PROC
126 :
127 PROC BK
128 XF=XI-PR*COS(AN):YF=YI-
PR*SIN(AN)
129 IF FG=1 THEN LINE XI,YI,XF,YF,1
130 XI=XF:YI=YF
131 END PROC
132 :
133 PROC RT
134 IC=PG/180*PR
135 AN=AN+IC
136 END PROC
137 :
138 PROC LT
139 IC=PG/180*PR
140 AN=AN-IC
141 END PROC
999 :
1000 REM ESEGUI COMANDO
1004 IF CM$="FD" THEN EXEC FD
1005 IF CM$="BK" THEN EXEC BK
1006 IF CM$="RT" THEN EXEC RT
1007 IF CM$="LT" THEN EXEC LT
1008 IF CM$="PD" THEN FG=1
1009 IF CM$="PU" THEN FG=0
1010 RETURN
1999 :
2000 REM SCANSIONE STRINGA
COMANDI SC$
2001 PI=1:PF=LEN(SC$)
2002 P=PI
2003 CM$=MID$(SC$,P,2)
2004 IF CM$="PD" OR CM$="PU" THEN
GOSUB 1000:P=P+3:GOTO2006
2005 GOSUB3000:GOSUB 1000:P=C+1
2006 IF P>PF THEN RETURN
2007 GOTO 2003
2999 :
3000 REM LEGGI PARAMETRO
3001 CI=P+3:CF=PF:C=CI
3002 CR$=MID$(SC$,C,1)
3003 IF CR$<>CHR$(32) THEN 3006
3004 PR=VAL(MID$(SC$,CI,C-CI))
3005 RETURN
3006 C=C+1:IF C-1<CF THEN 3002
3007 GOTO3004
3999 :
4000 REM PROGRAMMA PRINCIPALE
4001 XI=160:YI=50:AN=0:FG=1
4003 HIRES1,0
4004 EXEC QUADRATO
4010 GETA$:IFA$=""THEN4010
4098 END
4999 :
5000 PROC QUADRATO
5001 FOR J=0 TO 3
5002 SC$="FD 50":GOSUB 2000
5003 SC$="RT 90":GOSUB 2000
5004 NEXT
5005 END PROC
```

Bibliografia

Peitgen, H.-O.; Jürgens, H.; and Saupe, D.

Chaos and Fractals: New Frontiers of Science.

New York: Springer-Verlag, 1992.

Flake, G. W.

The Computational Beauty of Nature.

Cambridge, MA: MIT Press, p. 89, 1998.

https://it.wikipedia.org/wiki/Radiante

https://it.wikipedia.org/wiki/Circonferenza_unitaria

C portabile e ottimizzato per gli 8-bit - parte 1

Prima parte: Introduzione e Scrittura di codice C portabile per 8-bit

di Fabrizio Caruso

NdE: per garantire una maggiore leggibilita' del codice, abbiamo preferito formattare l'articolo su 2 colonne.

Questa è la prima parte di una serie di tre articoli che descrivono tecniche per scrivere codice portabile e ottimizzato in ANSI C per **tutti** i sistemi 8-bit *vintage*, cioè computer, console, handheld, calcolatrici scientifiche e microcontrollori dalla fine degli anni 70 fino a metà degli anni 90. L'articolo completo è disponibile on-line su https://github.com/Fabrizio-Caruso/8bitC/blob/master/8bitC.md

Il contenuto dell'articolo originale sarà trattato in tre parti:

1. Introduzione e Scrittura di codice C portabile per 8-bit
2. Tecniche per ottimizzare il codice C per 8-bit
3. Tecniche avanzate per ottimizzare il codice C per 8-bit

Architetture

In particolare ci occuperemo dei sistemi basati sulle seguenti *architetture* (e architetture derivate e retrocompatibili):

- Intel 8080 (*)
- MOS 6502
- Motorola 6809
- Zilog Z80 (*)

(*) Lo Zilog Z80 è una estensione dell'Intel 8080, quindi un binario Intel 8080 sarà utilizzabile anche su un sistema con Z80 ma il viceversa non è vero.

Buona parte di queste tecniche sono valide su altre architetture 8-bit come quella del COSMAC 1802 e quella del microcontrollore Intel 8051.

Obiettivi

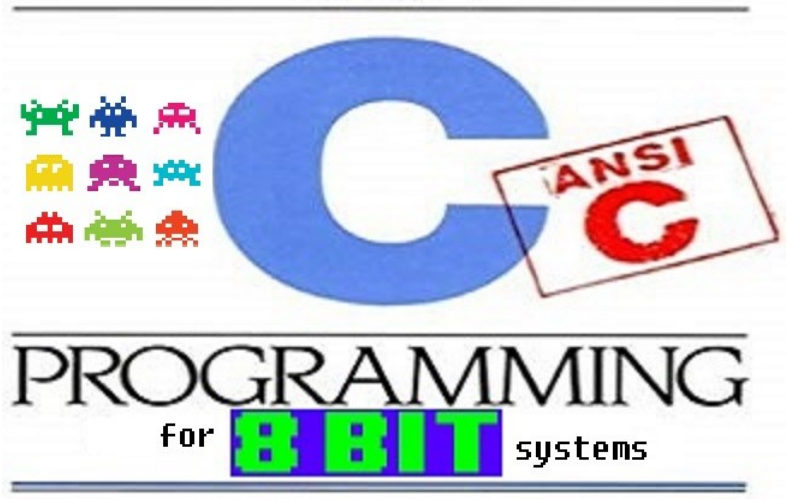
Lo scopo di questa serie di articoli è duplice:

1. descrivere tecniche generiche per scrivere codice **portabile**, cioè valido e compilabile su **tutti** i sistemi 8-bit indipendentemente dal fatto che un sistema sia supportato esplicitamente da un compilatore o meno
2. descrivere tecniche generali per **ottimizzare** il codice C su **tutti** i sistemi 8-bit

Premesse

Questa serie di articoli **non** è un manuale introduttivo al linguaggio *C* e richiede

- conoscenza del linguaggio *C*;
- conoscenza della programmazione strutturata e a oggetti;



- conoscenza dell'uso di compilatori e linker.

Inoltre **non** ci occuperemo in profondità di alcuni argomenti avanzati quali:

- alcuni ambiti specifici della programmazione come grafica, suono, input/output, etc.
- l'interazione tra C e Assembly.

Questi argomenti avanzati sono molto importanti e meriterebbero degli articoli separati a loro dedicati.

Terminologia

Introduciamo alcuni termini che saranno ricorrenti in questa serie di articoli:

- Un *sistema* è un qualunque tipo di macchina dotata di processore come computer, console, handheld, calcolatrici, sistemi embedded, etc.
- Un *target* di un compilatore è un sistema supportato dal compilatore, cioè un sistema per il quale il compilatore mette a disposizione supporto specifico come librerie e la generazione di un binario in formato specifico.
- Un *architettura* è un tipo di processore (Intel 8080, 6502, Zilog Z80, Motorola 6809, etc.) . Un *target* appartiene quindi ad una sola architettura data dal suo processore (con rare eccezioni come il Commodore 128 che ha sia un processore derivato dal 6502 e uno Zilog Z80).

Cross-compilatori multi-target

Per produrre i nostri binari 8-bit consigliamo l'uso di *cross compilatori multi-target* (cioè compilatori eseguiti su PC che producono binari per diversi *target*). Non consigliamo l'uso di compilatori *nativi* perché

sarebbero molto scomodi (anche se usati all'interno di un emulatore accelerato al massimo) e non potrebbero mai produrre codice ottimizzato perché l'ottimizzatore sarebbe limitato dalla risorse della macchina 8-bit.

Faremo particolare riferimento ai seguenti *cross compilatori multi-target*:

Architettura	Compilatore/ Dev-Kit	Pagina
Intel 8080	ACK	https://github.com/davidgiven/ack
MOS 6502	CC65	https://github.com/cc65/cc65
Motorola 6809	CMOC	https://perso.b2b2c.ca/~sarrazip/dev/cmoc.html
Zilog 80	SCCZ80/ZSDCC (Z88DK)	https://github.com/z88dk/z88dk

Inoltre esistono altri *cross compilatori* C *multi-target* che non tratteremo qui ma per i quali buona parte delle stesse tecniche generiche rimangono valide:

- LCC1802 (https://sites.google.com/site/lcc1802/) per il COSMAC 1802;
- SDCC (http://sdcc.sourceforge.net/) per svariate architetture di microprocessori come lo Z80 e di microcontrollori come l'Intel 8051;
- GCC-6809 (https://github.com/bcd/gcc) per 6809 (adattamento di GCC);
- GCC-6502 (https://github.com/itszor/gcc-6502-bits) per 6502 (adattamento di GCC);
- SmallC-85 (https://github.com/ncb85/SmallC-85) per Intel 8080/8085 ;
- devkitSMS (https://github.com/sverx/devkitSMS) per le console Sega basate su Z80 come Sega Master System, Sega Game Gear e Sega SG-1000.

Si noti come il dev-kit Z88DK disponga di due compilatori:

- l'affidabile SCCZ80 che è anche molto veloce nelle compilazione,
- lo sperimentale ZSDCC (versione ottimizzata per Z80 di SDCC sopracitato) che però può produrre codice più efficiente e compatto di SCCZ80 a costo di compilazione più lenta e rischio di introdurre bug.

Quasi tutti i compilatori che stiamo prendendo in considerazione generano codice per una sola architettura (sono *mono-architettura*) pur essendo *multi-target*. ACK è una eccezione essendo anche *multi-architettura* (con supporto per Intel 8080, Intel 8088/8086, I386, 68K, MIPS, PDP11, etc.).

Questo articolo **non** è né una introduzione né un manuale d'uso di questi compilatori e **non** tratterà:

- l'istallazione dei compilatori;
- elementi specifici per l'uso di base di un compilatore

Per i dettagli sull'istallazione e l'uso di base dei compilatori in questione, facciamo riferimento ai manuali e alle pagine web dei relativi compilatori.
Un articolo sul Cross Development (di Giorgio Balestrieri) sul numero 2 anno 1 di RetroMagazine ha presentato CC65 assieme ad altri tools di sviluppo (http://www.retromagazine.net/getrm.php?id=02). Una introduzione all'uso di CC65 (di Fabrizio Lodi) è disponibile su https://www.retroacademy.it/author/fabrizio-lodi/ .

Sottoinsieme di ANSI C
In questa serie di articoli per ANSI C intendiamo sostanzialmente un grosso sotto-insieme dello standard C89 in cui i float e i long long sono opzionali ma i puntatori a funzioni e puntatori a struct sono presenti. Non stiamo considerando versioni precedenti del C come per esempio C in sintassi *K&R*.

Motivazione

Per quale motivo dovremmo usare il C per programmare dei sistemi 8-bit?
Tradizionalmente queste macchine vengono programmate in Assembly o in BASIC interpretato o in un mix dei due.
Data la limitatezza delle risorse è spesso necessario ricorrere all'Assembly. Il BASIC è invece comodo per la sua semplicità e perché spesso un interprete è già presente sulla macchina.

Volendo limitare il confronto a questi soli tre linguaggi il seguente schema ci dà una idea delle ragione per l'uso del C.

	facilità	portabilità	efficienza
BASIC	SI	parziale	poca
Assembly	NO	NO	ottima
C	SI	SI	buona

Portabilità estrema

Quindi la ragione principale per l'uso del C è la sua portabilità assoluta. In particolare se si usa un sottoinsieme dell'ANSI C che è uno standard. In particolare l'ANSI C ci pemette di:

- fare porting semplificato tra una architettura all'altra,
- scrivere codice "universale", cioè valido per diversi target **senza** alcuna modifica.

Buone performance

Qualcuno si spinge a dichiarare che il C sia una sorta di Assembly universale. Questa è una affermazione un po' troppo ottimistica perché del C scritto molto bene non batterà mai dell'Assembly scritto bene. Ciò nonostante il C è probabilmente il linguaggio più vicino all'Assembly tra i linguaggi che permettono anche la programmazione ad alto livello.

Controindicazioni "sentimentali"

Una ragione non-razionale ma "sentimentale" per non usare il C sarebbe data dal fatto che il C è sicuramente meno *vintage* del BASIC e Assembly perché non era un linguaggio comune sugli home computer degli anni 80 (ma lo era sui computer professionali 8-bit come sulle macchine che usavano il sistema operativo CP/M).

Credo che la programmazione in C abbia però il grosso vantaggio di poterci fare programmare l'hardware di quasi tutti i sistemi 8-bit.

Scrivere codice portabile

Scrivere codice facilmente portabile o addirittura direttamente compilabile per diverse piattaforme è possibile in C attraverso varie strategie:

- Scrivere codice *agnostico* dell'hardware e che quindi usi *interfacce astratte* (cioè delle API indipendenti dall'hardware).
- Usare implementazioni diverse per le *interfacce* comuni da selezionare al momento della compilazione (per esempio attraverso *direttive al precompilatore* o fornendo file diversi al momento del linking).

Scrivere codice portabile tra sistemi con la stessa architettura e dev-kit

Questo diventa banale se il nostro dev-kit multi-target mette a disposizione una libreria multi-target o se ci si limita a usare le librerie standard del C (stdio, stdlib, etc.). Se si è in queste condizioni, allora basterà ricompilare il codice per ogni target e la libreria multi-target del del dev-kit farà la "magia" per noi.

Solo CC65 e Z88DK propongono interfacce multi-target per input e output oltre le librerie C standard:

Dev-Kit	Architettura	librerie multi-target
Z88DK	Zilog Z80	standard C lib, CONIO(testuale), vt52 (testuale), vt100 (testuale), sprite software, UDG, bitmap, joystick Z88DK
CC65	MOS 6502	standard C lib, CONIO(testuale), TGI (bitmap), joystick CC65
CMOC	Motorola 6809	standard C lib
ACK	Intel 8080	standard C lib

Librerie specifiche di CC65

CC65 fornisce la libreria *CONIO* per la visualizzazione di testo e la libreria grafica multi-target TGI per grafica bit-map tra alcuni dei suoi target e API proprie per leggere lo stato dei joystick.

Librerie specifiche di Z88DK

Anche Z88DK fornisce la libreria *CONIO* e API proprie per leggere lo stato dei joystick. Inoltre fornisce molti strumenti per la grafica multi-target tra cui API per grafica bitmap, per gli sprite software (https://github.com/z88dk/z88dk/wiki/monographics) e per caratteri ridefiniti per buona parte dei suoi 80 target.

Esempio: Il gioco multi-piattaforma H-Tron (di RobertK) è un esempio (https://sourceforge.net/projects/h-tron/) in cui si usano le API previste dal dev-kit Z88DK per creare un gioco su molti sistemi basati sull'architettura Z80.

Crearsi una libreria multi-target sulla stessa architettura

In tutti i casi in cui le librerie a disposizione non facciano al caso nostro bisognerà crearsi una libreria multi-target sfruttando eventualmente tutto quello che lo specifico dev-kit mette a disposizione.

Scrivere codice portabile tra architetture diverse

Per potere avere codice portabile su target e eventualmente anche su architetture diverse bisogna usare eventuali librerie comuni a dev-kit diversi o scriversi una libreria multi-architettura e quindi anche compilabile da compilatori di dev-kit **diversi**.

Librerie C standard

Se per l'input/output usassimo esclusivamente le librerie C standard (come stdio.h) potremmo avere codice compilabile con ACK, CMOC, CC65 e Z88DK ma saremmo limitati a input e output testaule limitato a comandi come printf e scanf senza controllo preciso della posizione del testo.

Libreria *CONIO* comune tra CC65 e Z88DK

Se per l'input/output, oltre alle librerie standard C, usassimo solo la libreria *CONIO* (che nasce per input/output testuale su *MS-DOS*) avremmo codice compilabile con *CC65* e *Z88DK* ma avremmo input e output testuale limitato a comandi come cprintf, cgetc , gotoxy che consentono il controllo della posizione del testo. Per maggiori dettagli facciamo riferimento a https://www.cc65.org/doc/funcref-14.html .

Crearsi una libreria multi-architettura

In tutti gli altri casi se vogliamo scrivere codice portabile su architetture diverse bisognerà crearsi una libreria multi-target e multi-architettura che quindi non avrà alcuna dipendenza da un dev-kit.

Scrivere una libreria

Scrivere una libreria multi-target o addirittura multi-architettura significa creare un *hardware abstraction layer* il cui scopo e di permettere la **separazione** tra:

- il codice che non dipende dall'hardware (per esempio la logica di un gioco) che usa l'interfaccia della libreria
- dal codice della libreria la cui implementazione dipende dall'hardware (per esempio le funzioni per l'input, output in un gioco) ma la cui interfaccia non dipende dall'hardware.

Questo *pattern* è assai comune nella programmazione moderna. Il C fornisce una serie di strumenti utili per implementare questo *pattern* in maniera che che si possano supportare hardware diversi da selezione al momento della compilazione. In particolare il C prevede un potente precompilatore con comandi come:

- #define per definire una macro,
- #if ... defined(...) ... #elif ... #else...#endif per selezione porzioni di codice che dipendono dal valore o esistenza di una macro.

Inoltre tutti i compilatori prevedono l'opzione -D per passare una variabile al precompilatore con eventuale valore. Alcuni compilatori

come CC65 implicitamente definiscono una variabile col nome del target (per esempio ***VIC20***) per il quale si intende compilare.

Nel codice avremo qualcosa come:

```
...
                        #if defined(__PV1000__)
                                #define XSize 28
                        #elif defined(__OSIC1P__) ||
defined(__G800__) || defined(__RX78__)
                                #define XSize 24
                        #elif defined(__VIC20__)
                                #define XSize 22
...
                        #endif
...
```

e al momento di compilare per il *Vic 20* il precompilatore selezionerà per noi la definizione di XSize specifica del *Vic 20*.

Questo permette al precompilatore non solo di selezionare le parti di codice specifiche per una macchina, ma anche di selezionare opzioni specifiche per configurazione delle macchina (memoria aggiuntiva, modo grafico, debug, etc.).

Cross-Chase* e *CrossLib

Il progetto *Cross-Chase* (https://github.com/Fabrizio-Caruso/CROSS-CHASE) propone il codice di un gioco compilabile su più di 100 sistemi 8-bit diversi con circa 200 configurazioni diverse usando sempre lo **stesso codice** del gioco grazie alla libreria universale *CrossLib*.

Se guardiamo il codice vediamo come sia stato separato in:

- codice del gioco (directory src/chase) totalmente indipendente dall'hardware,
- codice della libreria *CrossLib* (directory src/cross_lib) che implementa i dettagli di ogni hardware possibile.

Scrivere codice portabile su sistemi non supportati dai dev-kit

I nostri dev-kit supportano una lista di target per ogni architettura attraverso la presenza di librerie specifiche per l'hardware. E' comunque possibile sfruttare un dato dev-kit per altri sistemi con la stessa architettura ma saremo dovremo implementare tutta la parte di codice specifica del target:

- codice necessario per gestire l'input/output (grafica, tastiera, joystick, suoni, etc.),
- codice necessario per inizializzare il sistema.

Per fare ciò potremo in molti casi usare le routine già presenti nella ROM (nel terzo articolo di questa serie diamo un semplice esempio che è anche su https://github.com/Fabrizio-Caruso/8bitC/blob/master/8bitC.md).

Inoltre dovremo procurarci o scrivere un convertitore del binario in un formato accettabile per il nuovo sistema.

Per esempio CC65 non supporta né il *BBC Micro* né l'*Atari 7800* e CMOC non supporta l'*Olivetti Prodest PC128* ma è comunque possibile usare (o estendere) questi dev-kit per produrre binari per questi target:

- Cross Chase (https://github.com/Fabrizio-Caruso/CROSS-CHASE) supporta (in principio) qualunque architettura anche non supportata direttamente dai compilatori come per esempio l'Olivetti Prodest PC128.
- Il gioco Robotsfindskitten è stato portato per l'Atari 7800 usando CC65 (https://sourceforge.net/projects/rfk7800/files/rfk7800/).
- BBC è stato aggiunto come target sperimentale su CC65 (https://github.com/dominicbeesley/cc65).

Compilare per sistemi non supportati

Per potere compilare per un sistema non supportato dobbiamo usare alcune opzioni che servono ad eliminare le dipendenze da un target specifico. Qui diamo una lista di alcune opzioni utili a questo scopo. Per maggiori dettagli facciamo riferimento ai rispettivi manuali.

Architettura	Dev-Kit	Opzione
Intel 8080	ACK	(*)
MOS 6502	CC65	+none
Motorola 6809	CMOC	--nodefaultlibs
Zilog 80	SCCZ80/ZSDCC (Z88DK)	+test (generico), +embedded (generico con nuove librerie), +cpm (solo per generico CP/M)

(*) ACK prevede solo il target CP/M-80 per l'architettura Intel 8080 ma è possibile almeno in principio usare ACK per produrre binari generici per Intel 8080 ma non è semplice in quanto ACK usa una sequenze di comandi per produrre il Intel 8080 partendo dal C e passando da vari stai intermedi compreso un byte-code "EM":

1. ccp.ansi: precompilatore
2. em_cemcom.ansi: compila C preprocompilato producendo bytecode
3. em_opt: ottimizza il bytecode
4. cpm/ncg: genera Assembly da bytecode
5. cpm/as: genera codice Intel 80 da Assembly
6. em_led: linker

Un'alternativa a ACK per generare binari Intel 8080 generici è data dal già citato compilatore SmallC-85.

Vorrei ringraziare personalmente ed a nome di tutta la redazione di RetroMagazine, Fabrizio che ha deciso di condividere con noi questo suo lavoro. Sicuramente un documento fondamentale per chiunque voglia provare ad affrontare la programmazione in C con le macchine ad 8bit.

Francesco Fiorentini

Esplorando l'Amiga - parte 4

di Leonardo Giordani

NdE: per garantire una maggiore leggibilita' del codice, abbiamo preferito formattare l'articolo su 2 colonne.

Le funzioni base di Exec

Abbiamo trovato la tabella dei vettori di Kickstart 1.3 (`exec` 34.2) all'indirizzo `0x1a7c`, e le prime 4 voci sono

```
00001a7c: 08a0
00001a7e: 08a8
00001a80: 08ac
00001a82: 08ac
```

Se traduciamo questi valori relativi in indirizzi assoluti, sommando l'indirizzo della tabella stessa, scopriamo l'indirizzo delle 4 funzioni base che ogni libreria dell'Amiga deve fornire, `Open`, `Close`, `Expunge`, e uno spazio riservato che deve contenere una funzione che restituisce 0.

Open

Il primo valore è `0x08a0`, e se sommiamo a questo valore l'indirizzo della tabella stessa otteniamo `0x1a7c + 0x08a0 = 0x231c`. A questo indirizzo troveremo la prima funzione definita nella jump table, ovvero `Open`.

Il codice è il seguente

```
; Open
0000231c: 200e      move.l  a6,d0
0000231e: 526e 0020 addq.w  #0x1,0x20(a6)
00002322: 4e75      rts
```

La routine `Open` assume che l'indirizzo della libreria da aprire sia contenuto nel registro `a6`, e restituisce lo stesso valore in `d0`. Dopodiché aggiunge 1 al numero contenuto 32 byte (`0x20`) dopo l'indirizzo della libreria e termina. Per scoprire cosa rappresenti questo numero possiamo ancora una volta consultare i file include dell'NDK.

Da uno degli articoli precedenti sappiamo che, una volta che la libreria è stata caricata in memoria, ci sono due strutture definite in modo contiguo. La prima è la struttura `LN` che rappresenta un nodo di una linked list, e la seconda è la struttura `LIB` che rappresenta la libreria in questione.

La definizione di `LN` è in `include_i/exec/nodes.i`

```
STRUCTURE    LN,0    ; List Node
    APTR    LN_SUCC ; Pointer to next (successor)
    APTR    LN_PRED ; Pointer to previous (predecessor)
    UBYTE   LN_TYPE
    BYTE    LN_PRI  ; Priority, for sorting
    APTR    LN_NAME ; ID string, null terminated
    LABEL   LN_SIZE ; Note: word aligned
```

mentre la definizione di `LIB` si trova in `include_i/exewc/libraries.i`

```
STRUCTURE LIB,LN_SIZE
    UBYTE   LIB_FLAGS           ; see below
    UBYTE   LIB_pad         ; must be zero
    UWORD   LIB_NEGSIZE     ; number of bytes before LIB
    UWORD   LIB_POSSIZE     ; number of bytes after LIB
    UWORD   LIB_VERSION     ; major
    UWORD   LIB_REVISION        ; minor
    APTR    LIB_IDSTRING        ; ASCII identification
    ULONG   LIB_SUM         ; the system-calculated checksum
    UWORD   LIB_OPENCNT     ; number of current opens
    LABEL   LIB_SIZE    ;Warning: Size is not a longword
multiple!
```

Come si può notare quest'ultima cita la prima riservando dello spazio per essa all'inizio (`LN_SIZE`).

`LABEL` è una macro che crea un alias per la dimensione corrente della struttua, e potete trovare la sua definizione in `include_i/exec/types.i`. Essa funziona in connessione con le macro che definiscono i tipi, le quali incrementano la variabile globale `SOFFSET`. Il codice `LABEL LN_SIZE` nella struttura `LN` produce la definizione `LN_SIZE EQU 14`, che è quindi una semplice etichetta e non contribuisce alla dimensione totale della struttura.

Il commento dopo la macro `LABEL` nella struttura `LN` ci dice che quest'ultima è word aligned (allineata alle word), e infatti la sua dimensione è un multiple di 2 byte (word): 2 `APTR` (8 byte) + 1 `UBYTE` (1 byte) + 1 `BYTE` (1 byte) + 1 `APTR` (4 byte) = 14 byte.

Per trovare il campo aggiornato da `Open` dobbiamo saltare 32 byte. Quindi, dopo aver saltato l'intera struttura `LN`, dobbiamo saltare ancora 18 byte nella struttura `LIB`. A quell'offset troviamo il campo `LIB_OPENCNT` (ricordatevi che `UBYTE` vale 1 byte, `UWORD` vale 2 byte, e sia `APTR` che `ULONG` valgono 4 byte). Questo campo, secondo il commento, è il numero di "aperture" della libreria.

L'ultima istruzione della funzione `Open` è `rts` (return from subroutine, esci dalla procedura) che esce ritornando all'istruzione che segue il comando `jsr` che ha chiamato la funzione.

`Close`, `Expunge`, e lo spazio riservato

Appena dopo la definizione della funzione `Open`, troviamo la definizione di `Close`, elencata nella tabella dei vettori come `0x08a8`, che diventa `0x1a7c + 0x08a8 = 0x2324`. I successivi due valori nella tabella dei vettori contengono lo stesso valore, `0x08ac`, che diventa l'indirizzo assoluto `0x1a7c + 0x08ac = 0x2328`. Questo indirizzo è quindi la posizione sia di `Expunge` che della funzione che deve restituire 0.

Il codice di `Close` è molto semplice, in quanto decrementa solamente il contatore delle aperture (`0x20` nella struttura della libreria). Non c'è un `rts` esplicito in quanto `Close` usa il codice di `Expunge`, immediatamente attiguo, per quello. Essendo impossibile rimuovere la libreria Exec nel sistema Amiga, la funzione `Expunge` di Exec non fa altro che restituire 0, che è esattamente quello che deve fare la funzione nello spazio riservato (da cui il valore duplicato nella tabella dei vettori), e anche quello che deve fare `Close` dopo aver decrementato il contatore delle aperture.

```
; Close
00002324: 536e 0020 subq.w  #0x1,0x20(a6)

; Expunge
00002328: 7000      moveq   #0,d0
0000232a: 4e75      rts
```

`MakeFunctions`

La procedura `MakeFunctions` è usata da Exec per creare la tabella dei vettori all'inizio di una libreria quando questa viene caricata in memoria. Ricorderete che la tabella dei vettori viene creata al contrario partendo dall'inizio della libreria, permettendo quindi di usare uno schema di indirizzamento molto semplice.

Il prototipo della procedura `MakeFunctions` è

```
size = MakeFunctions(addess, vectors, offset)
d0                   a0      a1       a2
```

e il codice che abbiamo trovato in uno degli articoli precedenti è all'indirizzo `0x15b2`

```
000015b2: 2f0b                    move.l  a3,-(sp)
000015b4: 7000                    moveq   #0,d0
000015b6: 220a                    move.l  a2,d1
000015b8: 6716                    beq.b   0x15d0
000015ba: 3219                    move.w  (a1)+,d1
000015bc: 0c41 ffff               cmpi.w  #-0x1,d1
000015c0: 6722                    beq.b   0x15e4
000015c2: 47f2 1000               lea     (0,a2,d1.w),a3
000015c6: 210b                    move.l  a3,-(a0)
000015c8: 313c 4ef9               move.w  #0x4ef9,-(a0)
000015cc: 5c80                    addq.l  #0x6,d0
000015ce: 60ea                    bra.b   0x15ba
000015d0: 2219                    move.l  (a1)+,d1
000015d2: 0c81 ffff ffff          cmpi.l  #-0x1,d1
000015d8: 670a                    beq.b   0x15e4
000015da: 2101                    move.l  d1,-(a0)
000015dc: 313c 4ef9               move.w  #0x4ef9,-(a0)
000015e0: 5c80                    addq.l  #0x6,d0
000015e2: 60ec                    bra.b   0x15d0
000015e4: 265f                    movea.l (sp)+,a3
000015e6: 4e75                    rts
```

Il codice risulta probabilmente più semplice da leggere sostituendo gli indirizzi con delle etichette

```
Setup:
000015b2: 2f0b                    move.l  a3,-(sp)
000015b4: 7000                    moveq   #0,d0
000015b6: 220a                    move.l  a2,d1
000015b8: 6716                    beq.b   Absolute

Relative:
000015ba: 3219                    move.w  (a1)+,d1
000015bc: 0c41 ffff               cmpi.w  #-0x1,d1
000015c0: 6722                    beq.b   Cleanup
000015c2: 47f2 1000               lea     (0,a2,d1.w),a3
000015c6: 210b                    move.l  a3,-(a0)
000015c8: 313c 4ef9               move.w  #0x4ef9,-(a0)
000015cc: 5c80                    addq.l  #0x6,d0
000015ce: 60ea                    bra.b   Relative

Absolute:
000015d0: 2219                    move.l  (a1)+,d1
000015d2: 0c81 ffff ffff          cmpi.l  #-0x1,d1
000015d8: 670a                    beq.b   Cleanup
000015da: 2101                    move.l  d1,-(a0)
000015dc: 313c 4ef9               move.w  #0x4ef9,-(a0)
000015e0: 5c80                    addq.l  #0x6,d0
000015e2: 60ec                    bra.b   Absolute

Cleanup:
000015e4: 265f                    movea.l (sp)+,a3
000015e6: 4e75                    rts
```

Setup

La priam parte del codice è

```
Setup:
000015b2: 2f0b                    move.l  a3,-(sp)
000015b4: 7000                    moveq   #0,d0
000015b6: 220a                    move.l  a2,d1
000015b8: 6716                    beq.b   Absolute
```

La prima cosa che questo codice fa è di salvare il registro `a3` nello stack in quanto esso verrà cambiato durante l'esecuzione della procedura.

```
Setup:
000015b2: 2f0b                    move.l  a3,-(sp)
```

In Assembly, le variabili sono fornite dai registri e quindi non hanno un namespace, essendo i registri gli stessi lungo tutto il programma. Questo è il motivo per cui è necessario salvarli e ripristinarli e lasciare documentazione di quali vengono cambiati, ad esempio per restituire valori.

In secondo luogo il codice imposta il registro `do` a o.

```
000015b4: 7000                    moveq   #0,d0
```

Questo registro, secondo il prototipo della funzione, conterrà la dimensione finale della jump table, e o è un valore iniziale ragionevole.

Infine, il codice sposta il registro `a2` a `d1` per poterlo modificare.

```
000015b6: 220a                    move.l  a2,d1
000015b8: 6716                    beq.b   Absolute
```

Questo però imposta allo stesso tempo le flag del processore al valore di `a2` stesso, e tali flag vengono usate nella successiva istruzione `beq.b`. Se `a2` contiene il valore o la tabella è assoluta (codice a `0x15do`, etichettato `Absolute` in questa sede), altrimenti è relativa (codice a `0x15ba`, etichettato `Relative`).

Relative

La seconda sezione gestisce i vettori relativi

```
Relative:
000015ba: 3219                    move.w  (a1)+,d1
000015bc: 0c41 ffff               cmpi.w  #-0x1,d1
000015c0: 6722                    beq.b   Cleanup
000015c2: 47f2 1000               lea     (0,a2,d1.w),a3
000015c6: 210b                    move.l  a3,-(a0)
000015c8: 313c 4ef9               move.w  #0x4ef9,-(a0)
000015cc: 5c80                    addq.l  #0x6,d0
000015ce: 60ea                    bra.b   Relative
```

Questo codice preleva uno dei vettori dall'indirizzo salvato in `a1` (l'indirizzo della tabella dei vettori), incrementando immediatamente il valore del registro in modo da farlo puntare al vettore successivo.

```
Relative:
000015ba: 3219                  move.w  (a1)+,d1
```

poi lo confronta con `oxffff` (o `#-ox1`) per vedere se è stata raggiunta la fine della tabella. In quel caso il codice salta alla quarta sezione (`ox15e4`, etichettata `Cleanup`), altrimenti l'esecuzione continua con l'istruzione successiva

```
000015bc: 0c41 ffff             cmpi.w  #-0x1,d1
000015c0: 6722                  beq.b   Cleanup
```

La procedura poi carica l'indirizzo effettivo del vettore relativo usando `a2` come indirizzo di base, e salva il risultato in `a3`. Questo registro contiene ora il valore finale che sarà parte della jump table.

```
000015c2: 47f2 1000             lea     (0,a2,d1.w),a3
```

Dato che l'indirizzo della jump table è contenuto in `ao`, il vettore assoluto risultante viene salvato lì, poi il codice salva il valore `ox4ef9` che è il codice dell'istruzione `jmp` (maggiori informazioni su questo più avanti)

```
000015c6: 210b                  move.l  a3,-(a0)
000015c8: 313c 4ef9             move.w  #0x4ef9,-(a0)
```

Si noti che la tabella viene salvata ad indirizzi negativi partendo dall'indirizzo di partenza della libreria, pertanto è necessario prima copiare l'argomento (l'indirizzo) e poi la funzione (il codice di `jmp`). L'ultima cosa che questa parte di codice fa è aggiungere 6 alla dimensione della jump table (1 word per l'istruzione, 2 word per l'indirizzo), poi salta indietro all'inizio del ciclo.

```
000015cc: 5c80                  addq.l  #0x6,d0
000015ce: 60ea                  bra.b   Relative
```

Absolute

La terza sezione è molto simile alla seconda, in quanto effettua le stesse azioni, solamente manipolando indirizzi assoluti invece che relativi.

```
000015d0: 2219                  move.l  (a1)+,d1
000015d2: 0c81 ffff ffff        cmpi.l  #-0x1,d1
000015d8: 670a                  beq.b   Cleanup
000015da: 2101                  move.l  d1,-(a0)
000015dc: 313c 4ef9             move.w  #0x4ef9,-(a0)
000015e0: 5c80                  addq.l  #0x6,d0
000015e2: 60ec                  bra.b   Absolute
```

L'unica differenza con la sezione precedente è che non c'è bisogno di caricare l'indirizzo effettivo, in quanto il valore contenuto in `d1` è già assoluto. Pertanto quest'ultimo può essere salvato direttamente.

Cleanup

L'ultima sezione ripristina il puntatore allo stack, estraendo il valore del registro `a3` dove il puntatore era stato salvato, ed esce dalla procedura.

```
000015e4: 265f                  movea.l (sp)+,a3
000015e6: 4e75                  rts
```

Codice automodificante

La creazione della jump table effettuata dalla procedura `MakeFunctions` sfrutta una caratteristica molto potente di ogni linguaggio Assembly, ovvero la capacità di produrre codice che si automodifica. Questa caratteristica viene da una proprietà del linguaggio macchina chiamata [omoiconicità] https://en.wikipedia.org/wiki/Homoiconicity.

L'idea base dell'omoiconicità è che il linguaggio non considera dati e codice come due cose differenti, e pertanto permette di usare del codice come input delle procedure e di cambiarlo attraverso altro codice. Si faccia attenzione che il codice automodificante è solo una dell implicazioni dell'omoiconicità, non la sua definizione. L'omoiconicità è una caratteristica molto rara dei linguaggi. È molto potente, ma sostanzialmente forza a mantenere il linguaggio al livello del suo AST (Abstract Syntax Tree), che a sua volta significa che non è possibile creare una vera e propria astrazione o, se preferite, un vero e proprio linguaggio ad alto livello, se identifichiamo con questo termine linguaggi di programmazione vicini al linguaggio umano.

Esempi molto famosi di linguaggi omoiconici sono Lisp e Prolog. Lisp è un linguaggio che gestisce liste e la sua sintassi è basata su... liste. Questo significa che possiamo dare in pasto del codice Lisp ad una funzione Lisp e trasformarlo come faremmo con dei semplici dati.

Ritornando all'Assembly Motorola, la riga a cui siamo interessati è

```
000015c8: 313c 4ef9                  move.w  #0x4ef9,-(a0)
```

Questo codice decrementa l'indirizzo contenuto in `ao` di 2 byte, poi salva all'indirizzo risultante il valore esadecimale `ox4ef9`.

La parte interessante è che il numero `ox4ef9` ha un significato specifico per il processore Motorola 68000, ovvero quello dell'istruzione `jmp`. Questo si trova facilmente consultando le tabelle nel Programmer's Reference Manual (Section 8, Instruction Format Summary, 8-15).

Prima di tutto dobbiamo convertire il numero in formato binario, ottenendo

```
0x4ef9 -> 0100 1110 1111 1001
```

I primi 10 bit (`0100 1110 11`) identificano già un'istruzione `jmp`. I restanti 6 bit (`111 001`) identificano il metodo di indirizzamento, che in questo caso è `Absolute Long` o `(xxx).L` (Programmer's Reference Manual, 4-108). Questa istruzione deve quindi essere seguita da un indirizzo assoluto a 4 byte. Con l'istruzione `move.w`, pertanto, il codice scrive in memoria del codice Assembly. Tecniche come questa sono state e sono usate da giochi e virus per oscurare il codice, in quanto un'analisi statica del file binario non rivelerebbe quello che ci sarà solamente a runtime!

Resources

Motorola M68000 Family Programmer's Reference Manual
https://www.nxp.com/docs/en/reference-manual/M68000PRM.pdf

AmigaOS Developer Docs http://amigadev.elowar.com

If you are interested in the English version of this article, it can be found on Leonardo's blog at the url:
http://blog.thedigitalcatonline.com/blog/2018/06/14/exploring-the-amiga-4/

LudoProgrammazione su 6502/6510

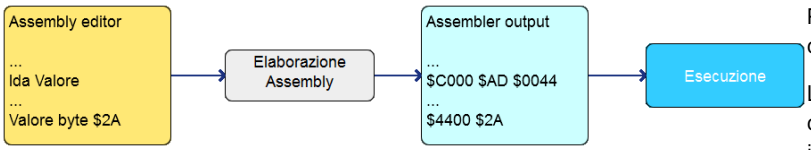
di Emanuele Bonin – Redattore di 8bit RetroProgramming Italia (gruppo Facebook)

Grazie al nostro sodalizio con il gruppo **8bit RetroProgramming Italia**, possiamo offrirvi questo interessante articolo sulla programmazione Assembly a firma di Emanuele Bonin.

NdE: per garantire una maggiore leggibilita' del codice, abbiamo preferito formattare l'articolo su 2 colonne.

Linguaggio Macchina, Assembler ed Assembly

L'assembler (o inguaggio macchina) è l'unico linguaggio che una cpu possa eseguire, qualsiasi altro linguaggio più ad alto livello deve essere per forza "tradotto" in assembler. A seguito delle istruzioni dell'assembler vengono compiute operazioni semplicissime, del tipo leggi/scrivi una cella di memoria, incrementa, somma, confronta, salta e così via. Nonostante la semplicità delle operazioni possibili, programmare in assembler non è assolutamente difficile e può dare grosse soddisfazioni. Se guardassimo un listato scritto in assembler vedremmo una lista di numeri e nulla più, fortunatamente, in fase di scrittura dei programmi, ci viene in aiuto l'assembly. L'assembly ci evita di scrivere direttamente i numeri interpretati dalla CPU come comandi, infatti al loro posto scriveremo delle sigle (codici mnemonici) più adatte ad essere memorizzate dal programmatore, che verranno tradotte opportunamente dal compilatore. L'assembly ci agevolerà anche in altri modi durante la stesutra del listato, ma li vedremo al momento opportuno.



Ambiente di sviluppo

L'assembler a cui farò riferimento è quello del 6502/6510 (i due processori a livello di programmazione base sono praticamente identici) ma molti concetti sono condivisi anche su altri processori a 8 bit. Per programamre in assembly ad 8 bit ci sono vari tool di sviluppo per varie piattaforme, personalmente uso il "CBM prg Studio" (http://www.ajordison.co.uk/download.html) un IDE gratuito per windows che trovo abbastanza completo e semplice da utilizzare, ma questa è solo una scelta personale.

Si comincia

Il programma da realizzare è una versione modificata della classica pallina che rimbalza sullo schermo in modalità testo. La novità sarà che i rimbalzi non avverranno solamente quando la pallina (che è semplicemente un carattere petscii) toccherà i bordi dello schermo, ma anche quando incontrerà un carattere diverso dallo spazio. In seguito faremo in modo che il moto della pallina non impedisca all'utente di continuare il proprio lavoro.

La posizione sul video

Nel C64, la memoria video, in poche parole, non è altro che una porzione della memoria RAM che gode della proprietà di essere continuamente visualizzata sul monitor. Tant'è che tale porzione di memoria può essere modificata in base alle esigenze del programma che si vuole realizzare, nel nostro caso l'impostazione all'accensione andrà benissimo. Il primo carattere in alto a sinistra corrisponde, di default, ala locazione $0400 in esadecimale (1024 in decimale), la $0401 corriponde alla cella accanto a destra, l'ultimo carattere della prima riga corrisponderà quindi alla cella $0427 e il primo carattere a sinistra della seconda riga alla cella $0428... e così via fino all'ultimo carattere in basso a destra che corrispondera $07E7 (2023 in decimale).

Per comodità dovremmo identificare la posizione che di volta in volta avrà il nostro pallino come se fosse su un piano cartesiano in cui il punto (0,0) è il carattere in alto a sinistra ($0400), all'aumentare delle X ci si sposterà sulla destra di un carattere, mentre all'aumentare delle Y ci si sposterà verso il basso sempre di un carattere. Quindi il primo problema che possiamo affrontare è costruire una routine che ci permetta di identificare la cella video avendo a disposizione due coordinate X e Y che andremo di volta in volta a modificare per identificare la nuova posizione.

La formula da utilizzare per calcolare la cella avendo le due coordinate X e Y è: *IndirizzoCellaVideo = $0400 + Y * 40 + X*

Ricordo che il C64 ha 40 colonne per 25 righe con la X che può variare da 0 a 39 ($27) e la Y da 0 a 24 ($18).

Le coordinate X e Y nell'assembler, in buona sostanza, saranno celle di memoria identificate da un numero che non è altro che il loro indirizzo. In assembly ci possiamo facilitare la vita semplicemente dando loro un nome o meglio una label (cioè le etichettiamo). Ogni cella di memoria (nel 6510) è un byte quindi conterrà un valore intero tra 0 e 255 ($00 - $ff), quindi per ognuna delle coordinate sarà sufficiente una byte, mentre per contenere l'indirizzo di memoria di una cella video ci serviranno 2 byte appaiati (o word).

Le label necessarie per il nostro calcolo sono:

```
        Factor1 word $0000      ; Primo fattore per
un'operazione
        Factor2 word $0000      ; Secondo fattore per
un'operazione
        Result  word $0000      ; Risultato delle
operazioni
        PosXY   byte $00,$00    ; x, y Posizione attuale
```

Per ogni riga sopra scritta abbiamo:

```
<label> <tipo> <valore iniziale>{,<valore iniziale>}
```

dove:
<label> è il nome che vogliamo dare al primo dei byte che anremo ad usare come memoria di lavoro
<tipo> indica come devono essere cosiderate le celle di memoria che andremo a inizializzare

<valore iniziale>{,<valore iniziale>} indica che andremo a specificare i valori iniziali delle celle di memoria da riservare, possiamo specificare più di un valore diviso da virgola.

Per esempio Factor1 è l'indirizzo di memoria che punta ad una cella che considereremo parte di una word (2 byte consecutivi), mentre PosXY identificherà la posizione del primo di due byte distinti. Da notare che in entrambi i casi verranno riservati 2 byte in totale, ciò che cambia è il modo in cui l'assembly sndrà a considerarle.

Tutto ciò che è scritto dopo il ; sono dei semplici commenti e in quanto tali verranno ignorati in fase di bulding del programma. In ultima analisi volevo far notare che avremmo potuto identificare due label distinte per la coppia X e Y, scrivendo per esempio:

```
        PosX byte $00 ; Posizione X
        PosY byte $00 ; Posizione Y
```

ma ho optato, per utilizzare una sola label per gestirle entrambe.

Routine di calcolo della posizione a video

Purtroppo sono costretto a spiegare un po' di teoria ancora prima di passare a mostrare un po' di codice. Normalmente se volessimo indicare un certo numero a sedici bit (2 byte) utilizzando la notazione esadecimale, magari per indicare l'indirizzo di memoria del secondo carattere in alto a destra del video, scriveremo $0401, se tale valore lo considerassimo come due byte appaiati potremmo (ma in realtà dobbiamo) dire che $04 è il byte più significativo (Most Significant Byte o MSB) mentre $01 è il byte meno significativo (Least Significant Byte o LSB). Nell'Assembler del 6510 molto spesso troverete invertiti questi due byte quindi in memoria troverete prima l'LSB e successivamente l'MSB, soprattutto quando si tratterà di indicare indirizzi di memoria, quindi per tornare all'esempio precedente il valore $0401 lo troveremmo scritto come $0104.

Ecco a voi il primo assaggio di routine in assembly:

```
        XytoSCPos               ; Calcola la posizione
dello schermo = $0400 + xpos + ypos * 40
                lda  PosXY+1    ; copio il contenuto di
YPos su Factor1
                sta  factor1    ; Byte meno significativo
(LSB)
                lda  #$28       ; Scrivo 40 ($28) in
Factor2 (LSB)
                sta  factor2
                lda  #$0        ; Azzero i byte più
ignificativi (MSB)
                sta  factor1+1  ; su Factor1
                sta  factor2+1  ; su Factor2
                jsr  Moltiplica ; Chiamata alla Routine
Moltiplica
                        ; eseguirà Result = Factor1 *
Factor2
                        ; cioè Result = YPos * 40
                lda  Result     ; Copio LSB di Result su
Factor1
                sta  Factor1    ; LSB
                lda  Result+1   ; Copio MSB di Result su
Factor1
                sta  Factor1+1  ; MSB
                lda  PosXY      ; Copio LSB di XPos in
                sta  Factor2    ; LSB di Factor2 (MSB è
$00)
                jsr  Sum16      ; Chiamata alla routine di
somma a 16 bit
                        ; Factor1 + Result
                        ; cioè XPos + YPos * 40
                        ; metterà il risultato della somma
in Result
                lda  Result     ; Copio Result LSB/MSB in
Factor1
                sta  Factor1    ; copia prima LSB
                lda  Result+1   ; ... poi
                sta  Factor1+1  ; ... MSB
                lda  #$00       ; Copio la LSB di $0400
(inizio memoria video)
                sta  Factor2    ; in Factor2 (LSB)
                lda  #$04       ; Copio la MSB di $0400
(inizio memoria video)
                sta  Factor2+1          ; ... MSB
                jsr  Sum16      ; Chiamata alla routine di
somma a 16 bit
                        ; Result = Factor1 + Factor2 =
(XPos + YPos * 40) + $0400
                lda  Result     ; Finalmente copio Result a
partire da $FB
                sta  $FB; ... manco a dirlo prima LSB su
una Locazione libera della Pagina 0
                lda  Result+1   ; ... poi
                sta  $FC; ... MSB
                ldy  #$00       ; Azzero il registro indice
y
                lda  ($FB),y    ; carico nell'accumulatore
il valore che si trova
                        ; all' indirizzo che trovo a a
partire dell'indirizzo
                        ; $FB (LSB/MSB) in pagina
0, cioè il valore calcolato
                rts     ; ritorno a chiamante
```

La routine proposta prende i valori delle coordinate memorizzate a partire da PosXY e calcola il valore dell'indirizzo di memoria video corrispondente utilizzando la formula sopra descritta.

I registri A,X,Y

Prima di procedere oltre, ancora un po' di teoria. In assembler 6510 non è possibile con un solo comando copiare il contenuto d una cella di memoria in un altro, per fare questa operazione bisogna utilizzare quelli che vengono chiamati registri. I registri possono essere considerati alla stregua di celle di memoria speciali, infatti per accedervi la cpu ci mette meno tempo rispetto all'accesso a qualsiasi altra cella di memoria RAM. Inoltre non hanno un indirizzo fisico, risiedono all'interno del microprocessore e ci si riferisce a loro con un nome. I tre registri del 6502/6510 che ora a noi interessano sono A, X, Y e hanno ampiezza di un byte, ve n'è qualcun altro ma per ora non voglio mettere troppa carne al fuoco. Il registro A, detto anche accumulatore, è il più utilizzato (anche se non ho le statistiche alla mano penso di non poter essere smentito) i registri X e Y possono anch'essi essere utilizzati, per molti aspetti come il registro A, ma hanno delle funzionalità di indicizzazione molto potenti quando vengono utilizzati in coppia con il registro A.

Per caricare con un valore il registro A si utilizza il comando LDA (LoaD Accumualtor) seguito da un argomento, quest'ultimo può essere un valore a 16 o 8 bit, a seconda di come intendiamo reperire il valore da assegnare poi all'accumulatore. Ogni tipo di caricamento si distingue per quella che si chiama indicizzazione, che può essere di tipo:

- Immediato, viene caricato un valore esplicito, l'argomento inizierà con un # e seguirà un valore o una label: lda #$2A ; Carica l'accumulatore con il valore 42 decimale
- Pagina Zero, se l'argomento è un valore di un byte (valore o label), viene eseguita la lettura da quella che viene definita pagina zero di memoria (da $0000 a $00FF), sono stati riservati codici assembler apposta per l'indirizzamento a questa pagina

in quanto il tempo di accesso ad essa è inferiore rispetto ad altre pagine, essendo sempre l'MSB a zero, la CPU risparmierà cicli di lettura, purtroppo, a meno di trucchetti, questa pagina è quasi tutta usata per il BASIC e la gestione del sistema, ci rimangono solo pochi byte a nostra disposizione, ma questo non ha mai fermato nessuno :

```
lda $2A   ; Carica L'accumulatore  col il valore della cella che si trova all'indirizzo $002A
```

- Pagina Zero X-indicizzata, in questo caso entra in gioco anche il registro X, l'accumulatore verrà caricato con la cella di memoria identificata dal calcolo Argomento+X:

```
ldx #$01   ; caricamento immediato del registro X con il valore 1
lda $29,X  ; Carica l'Accumuatore  col il valore della cella che si trova all'indirizzo $002A = $0029 + $01
```

- Assoluto/Assoluto-X/Assoluto-Y, con questa modalità, l'argomento son sarà più un byte, bensì un valore di 2 byte che indicheranno il valore di una cella di memoria di riferimento dalla quale prelevare il valore se in modalità Assoluta o utilizzando X o Y come indice per spostarsi rispetto alla cella indicata:

```
ldx #$01   ; caricamento immediato del registro X con il valore 1
ldy #$02   ; caricamento immediato del registro Y con il valore 2
lda $29,X  ; Carica l'Accumulatore  col il valore della cella che si trova all'indirizzo $002A = $0029 + $01
lda $29,Y  ; Carica l'Accumulatore  col il valore della cella che si trova all'indirizzo $002B = $0029 + $02
```

- Indiretto Indicizzato o Indiretto-Y, questo metodo di caricamento dell'Accumulatore è molto potente e spesso utilizzato. L'argomento, lungo un byte, punterà ad un indirizzo di memoria della pagina zero che assieme al byte successivo a quell'indirizzo formeranno un nuovo indirizzo di memoria al quale verrà sommato il registro Y, il risultato sarà l'indirizzo di memoria dal quale prelevare il valore e portarlo nell'accumulatore, come precedentemente indicato i due byte che verranno utilizzati per comporre l'indirizzo di memoria dovranno essere nela forma LSB-MSB, cioè prima il meno significativo e poi il più significativo:
  supponendo di avere i seguenti valori nella memoria (il numero a sinistra è l'indirizzo fisco quello a destra il valore al suo interno);

```
$00FB $01
$00FC $C0
....
....
$C001 $20
$C002 $2A
```

possiamo utilizzare i seguenti codici mnemonci per caricare l'accumulatore:

```
ldy #$01   ; carico nel registro Y il valore 1
lda ($FB),Y;  nell'accumulatore verrà inserito il valore $2A che è contenuto nella cella $C001 + $01 = $C002
```

- Indicizzato Indiretto o Indiretto-X, anche in questo caso viene utilizzata la pagina zero, viene utilizzato il registro X come indice di offset, ma in questa modalità viene prima sommato l'argomento alla X e poi viene composto l'indirizzo da cui prendere il valore da caricare nell'accumulatore: considerando la stessa configurazione di memoria dell'esempio precedente

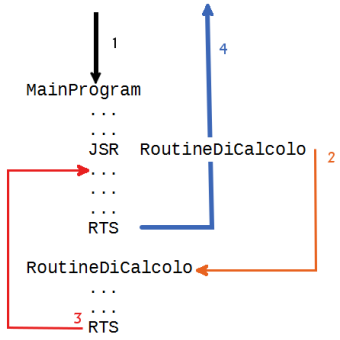
```
ldx #$02   ; X=$02
lda ($F0, X) ; carico l'accumulatore con il valore presente nella cella di memoria $C001 cioè $20
; infatti $F0 + $02 = $FB indirizzo contente $01 (LSB) che assieme al valore $C0 contenuto in $FC (MSB)
; formato l'indirizzo $C001
```

Come avrete capito, leggendo alcuni esempi, anche i registri X e Y possono essere caricati tramite i comandi LDX e LDY. Questi due registri hanno comunque un numero minore di metodi di indirizzamento.

Tornando all'analisi della routine, nella prima riga abbiamo la label XYtoSCPos che "marca" l'inizio della routine stessa, d'ora in poi se volessimo "chiamare" la nostra routine useremo il comando oppiortuno seguito da questa label.

Nelle 7 righe successive si predispongono i valori delle label affinché venga eseguito il calcolo coordinata y * 40, quindi viene reperito il valore della coordinata y e copiata in Factor1 mentre in Factor2 dovrà essere memorizzato il valore 40 ($28) . Abbiamo visto come reperire un valore tramite il comando LDA, per fare l'opposto, cioè scrivere in memoria il valore di un registro, si utilizzano i comandi STA, STX e STY (Store A,X,Y), i quali hanno anche loro vari metodi di indirizzamento (tranne ovviamente l'immediato). Nella nostra routine carichiamo innanzitutto il valore della nostra y del piano cartesiano (che non ha nulla a che fare con il registro Y) la quale si trova alla posizione PosXY+1. Fortunatamente l'assembly ci permette di eseguire alcune operazioni sulle label (ATTENZIONE non è un nuovo modo di indirizzamento, è semplicemente l'assembly che calcolerà il valore opportuno prima del building del programma) quindi con il comando LDA PosXY+1 non facciamo altro che dire all'assembly che la locazione di memoria da indirizzare sarà il byte succesivo a quello puntato dalla label PosXY, eseguendo un caricamento con indicizzazione assoluta.

Piccola annotazione i fattori Factor1 e Factor2 sono delle word per cui i valori cartesiani della x e y, costituiti ognuno da un byte, dovranno essere memorizzati nell'LSB dei rispettivi fattori mentre gli MSB dei fattori dovranno essere azzerati.



Il prossimo comando è "JSR Moltiplica", il comando JSR (Jump to SubRoutine) esegue il codice all'indirizzo indicato nell'argomento (la JSR permette il solo indirizzamento assoluto, quindi va specificato sempre un valore di 2 byte), una volta terminato il codice chiamato dalla JSR il nostro programma ripartirà dall'istruzione successiva alla JSR.

Nel nostro caso, senza entrare nei dettagli del codice chiamato, basti sapere che la chiamata alla SubRoutine "Moltiplica" non farà altro che moltiplicare i due fattori che abbiamo caricato e il risultato (una word) ce lo ritroveremo dall'indirizzo "Result" sempre nella forma LSB/MSB.

Eh già!, forse non ve lo aspettavate, ma il 6510 non ha un comando mnemonico per seguire le moltiplicazioni, tocca rimboccarsi le maniche, e costruirsi la propria routine.

Se avete letto fin qui, ammesso e non concesso di essere stato chiaro, avrete le conoscenze sufficienti per interpretare il codice fin'ora scritto,

per ora basti sapere che la label "Sum16" è una SubRoutime che mette in Result il valore di Factor1+Factor2.

Alla fine della Routine di calcolo, il comando RTS (ReTurn from Subroutine) farà in modo che il programma prosegua a partire dall'istruzione succesiva alla'ultima chiamata fatta alla nostra routine tramite JSR, infatti anche la nostra routine (come per "Moltiplica" e "Sum16") verrà chiamata tramite il comando JSR.

Nella penultima riga c'è un comando di caricamento dell'accumulatore indiretto indicizzato, riuscite a capire il significato del valore caricato nell'accumulatore ?
Si, proprio così, prima di ritornare al chiamante viene fatto in modo che nel registro A venga caricato il valore contenuto all'indirizzo video puntato dalle nostre coordinate PosX e PosY, questo ci servirà per capire (una volta usciti dalla routine) se in quella posizione c'è un carattere diverso dallo spazio, controllo essenziale al fine del del nostro programma.

Dapprima le somme e le sottrazioni ...

Come abbiamo visto ogni cella di memoria può contenere valori interi da 0 a 255, per sommare tra loro due valori (che siano essi presi dalla memoria o valori immediati) bisogna passare per l'accumulatore (il suo nome è proprio dovuto a questo), vediamo subito un esempio:

```
                lda #$2A; carico in A il valore $2A
                clc             ; azzero il riporto (carry)
                adc #$02; A = A + $02 => A = $2C
```

La prima riga ci dovrebbe essere già familiare, tralasciamo per un attimo la seconda riga e passiamo alla terza, dove incontriamo l'istruzione ADC (ADd with Carry – somma con il riporto) che è la responsabile della somma tra l'accumulatore, il valore determinato dall'argomento ed il riporto (carry) rimettendo il risultato nell'accumulatore. Non esistendo un comando che esegua la somma senza sommare anche il carry siamo costretti, in questo caso, a "pulire" il carry con il comando CLC (CLear Carry), altrimenti avremmo rischiato di trovarci nell'ammontare della somma anche un riporto dovuto ad operazioni precedenti. In effetti il carry non è altro che un bit-flag che quindi può assumere il valore 0 o 1 e il cui significato può variare a seconda delle operazioni che stiamo eseguendo. Il carry in realtà ha molteplici utilizzi, dipende dal contesto in cui stiamo operando, proprio per questo può venire modificato anche da operazioni che con la somma esplicita non hanno nulla a che fare.

Vediamo ora come sommare due valori a 16 bit analizzando la routine Sum16:

```
        Sum16
                clc     ; Mi assicuro che non vi sia un
riporto (Carry)
                lda  Factor1    ; A = LSB(Factor1)
                adc  Factor2    ; A = A + LSB(Factor2) +
Carry
                sta  Result     ; LSB(Result) = A  e viene
impostato il carry
                lda  Factor1+1  ; A = MSB(Factor1)
                adc  Factor2+1  ; A = A + MSB(Factor2) +
Carry
                sta  Result+1   ; MSB(Result) = A  e viene
impostato il carry
                rts     ; Ritorno al chiamante
```

Innanzitutto, nelle prime tre righe di codice, sommiamo i due byte meno significativi dei due addendi (Factor1 e Factor2) con la tecnica spiegata sopra, tale somma potrebbe avere un riporto, tale riporto quindi andrà sommato nel byte + significativo, ma di questo non dobbiamo preoccuparci più di tanto in quanto la nostra ADC lo prende già in considerazione. Infatti nelle righe successive, dopo esserci salvati il registro sull'LSB di Result, non facciamo altro che ripetere le operazioni precedenti (escluso il clc, ora il carry è importante che venga effettivamente sommato) sugli MSB dei Fattori e del Risultato, questo è tutto. Alla fine, al ritorno dalla nostra subroutine (RTS alla fine) nei due byte a partire dalla label/indirizzo "Risultato" avremo la somma di factor1 + Factor2. Se avessimo avuto la necessita di eseguire una sottrazione avremmo dovuto utilizzare il codice mnemonico SBC (SuBtract with Carry) , con la differenza che il carry in questo caso funziona come un prestito invertito, quindi viene sottratta la sua negazione, il che rende necessario settare il bit di carry con SEC (Set Carry) prima di eseguire la prima (se la differenza va fatta su più byte di seguito) delle differenze.

```
Per capire meglio come funziona il riporto con ADC
prendiamo ad esempio la somma di due "grandi"
numeri (non enormi, ma in un byte ci stanno
giusti), 254 + 254. Il "giochetto" del riporto si
vede bene se il 254 lo scriviamo in binario e
facciamo le somme dei singoli bit in colonna:

Ripassiamo giusto come si esegue la somma di bit:

0 + 0 = 0 (nessun riporto)
0 + 1 = 1 (nessun riporto)
1 + 1 = 0 (con riporto di 1)
(1+1) + 1 = 1 (con riporto di 1)

     Somma di 254 + 254:
     1 1 1 1 1 1 1 0
     1 1 1 1 1 1 1 0 +
     1 1 1 1 1 1 1 0 =
     -----------------
     1 1 1 1 1 1 1 0 0
In rosso sono evidenziati i riporti, della somma
della colonna alla loro immediata destra. Il
riporto più a sisnistra è il famoso carry-bit.
```

Carry e i suoi fratelli

Fin'ora abbiamo visto un utilizzo del carry, è venuto il momento di approfondire da dove viene ma soprattutto andiamo a fare la conoscenza degli altri flag che assieme al carry formano un altro importante registro del processore 6502: Il registro di stato del processore (per gli amici P). Questi non è altro che un byte in cui vengono immagazzinate le informazioni sullo stato del processore. A parte uno, gli altri bit di P indicano la condizione in cui si trova una ben determinata proprietà. I valori questi bit (0 o 1) vengono modificati implicitamente da alcune operazioni, ma possono essere modificate esplicitamente tramite opportuni istruzioni, vedi CLC e SEC ad esempio. I significati di ogni flag di P sono nell'ordine:

- 7: N Negative, questo flag è a 1 se l'ultima istruzione ha prodotto un risultato negaitvo
- 6: O OverFlow, nel caso di operazioni con segno viene settato a 1 se il valore è al di fuori del range (-128 – 127)
- 5: - Non utilizzato
- 4: B Break Command, 1 indica che a richiesta di interrupt è stata fatta tramite l'istruzione BRK

- 3: D Decimal mode, 1 Indica che il processore è nella modalità decimale
- 2: I IRQ disabled, 1 indica che le richieste di interrupt sono disabilitate
- 1: Z Zero, 1 indica che l'ultima operazione ha prodotto 0 come risultato
- 0: C Carry, 1 indica che l'ultima operazione ha prodotto un riporto

La spiegazione dettagiata del significato di ogni flag esula dall'intento che ha questo articolo, quindi per ora prenderemo in considerazione il carry (di cui ne abbiamo già visto un utilizzo) e il flag Z, che come descritto sopra indica quando l'ultima istruzione che ne modifica il valore ha avuto come risultato esattamente zero.

… quindi le moltiplicazioni

Come dei bravi scolaretti, a questo punto, è arrivato il momento di eseguire le moltiplicazioni, con i nostri byte. Genericamente, la moltiplicazione di due numeri N * M, avendo a disposizione la sola somma come operazione, si ottiene sommando M volte N (o viceversa). Sin d'ora vorrei chiarire che c'è almeno un altro algoritmo da poter utilizzare per fare le moltiplicazioni con l' assembler 6502, che risulta sicuramente più performante, ma per i nostri fini va benissimo il metodo "classico".
Scriviamo la nostra routine di moltiplicazione:

```
Moltiplica              ; Moltiplica LSB(Factor1) *
LSB(Factor2)
        lda  #$00       ;
        sta  Result     ;
        sta  Result+1   ; Risultato = 0
        ldx  Factor2    ; x = Factor2
        cpx  #$00       ; x == 0 ?
        beq  FineMoltiplica     ; Se x==0 fine
moltiplicazione (Result = 0)
        lda  Factor1    ; a = Factor1
        cmp  #$00       ; a == 0 ?
        beq  FineMoltiplica     ; se a == 0 vai a fine
moltiplicazione (Result = 0)
        sta  Result     ; altrimenti Result = Factor1
LoopMoltiplica
        dex             ; x=x-1 (x inizialmente conteneva
LSB(Factor2))
        beq  FineMoltiplica     ; Se x==0 vai a fine
Moltiplicazione
        clc             ; Pulisci il Carry
        lda  Result     ; ----------------
        adc  Factor1    ; Somma Result con Factor1
        sta  Result     ; metti il risultato su Result
        lda  #$00       ; Caricamento immediato (più
veloce)
        adc  Result+1   ; al posto di Factor+1 che so
essere a 0
        sta  Result+1   ; ----------------
        jmp  LoopMoltiplica     ; Vai a LoopMoltiplica
FineMoltiplica
        rts             ; Torna al chiamante
```

La routine di moltiplicazione eseguirà la moltiplicazione di Factor1 e Factor2 intesi come singoli byte, infatti prima della chiamata a questa subroutine, il nostro programma (vedi il listato più sopra) inserirà in queste word le coordinate dello schermo che ci stanno comodamente su di un byte, quindi per ottimizzare le cose (e renderle più semplici) viene preso in considerazione solo l'LSB dei due fattori.
Ciò che volevo mettere in evidenza con questa routine erano i comandi di confronto e di salto condizionato.
Nelle prime tre istruzioni non si fa altro che azzerare Result (il nostro contenitore del risultato finale) successivamente incontriamo l'istruzione CPX (ComPare X, per inciso esiste anche CPY per il registro Y), la quale esegue una comparazione tra il registro X (che è stato caricato col valore di Factor2) e l'argomento (che può essere di tipo immediato, pagina zero o assoluto). Ci riferiamo a CMP come una comparazione, Ma possiamo pensarla come una semplice operazione di differenza tra X è l'argomento, la quae va a modificati alcuni dei flag di P e per la precisione verranno modificati i flag di Segno, Zero e Carry.
L'istruzione successiva BEQ (Branch on Equal tradotta malamente sarebbe "Salta se uguale") esegue il controllo del Flag Z, e se Z=1 (nel nostro caso se la differenza Registrox-$00 = 0 il che implica che Registro-x sia $00) allora la CPU andrà ad eseguire le istruzioni che si trovano a partire dall'indirizzo indicato con l'argomento.
Oltre a BEQ esistono anche altre istruzioni di salto condizionato che eseguono controlli sugli altri flag del registro di stato P, per ora li tralascio, sebbene siano importantissimi, altrimenti questo articolo diverrebbe un libro.
Riprendendo il filo del nostro discorso, con l'istruzione "BEQ FineMoltiplica" quindi si indica al programma di andare ad eseguire l'istruzione RTS e quindi tornare al chiamante con Result = 0 (qualsiasi numero moltiplicato per 0 dà come risultato 0). Nelle istruzioni successive eseguiamo lo stesso controllo su Factor1, solo che stavolta la comparazione la facciamo utilizzando l'accumulatore e quindi l'istruzione CMP (CoMPare accumulator, la quale ha più tipi di indirizzamento rispetto a CPY e CPX).
Se entrambi i fattori sono diversi da 0 (in questo caso li consideriamo sempre come valori positivi) allora procediamo l'algoritmo andando a sommare Factor1 per Factor2 volte. Innanzitutto Inizializziamo Result con il valore di Factor1, poi per tenere il conteggio del numero di volte in cui dovremmo la somma viene tenuto dal registro X (inizialmente eguagliato a Factor2) decrementandolo di una unità tramite l'istruzione DEX (Decrement X), quando X raggiungerà il valore 0, la successiva istruzione BEQ prenderà la decisione di terminare il Loop, se ciò non avvenisse (X > 0) allora il numero di sommatorie da effettuare non è ancora sufficiente a raggiungere lo scopo, quindi eseguiremo la sommatoria di Factor1 con Result e riporteremo il risultato su Result (qui si mi preoccupo di usare LSB eMSB di Result in quanto la moltiplicazione potrebbe raggiungere valori non contenibili in un singolo byte) e poi, per chiudere il ciclo, verrà eseguita l'istruzione JMP (JuMP) a LoopMoltiplica, da dove si si eseguirà nuovamente il decremento di X e il controllo, tutto questo finché X non raggiungerà il valore di 0.
Questa tecnica di eseguire i loop "al contrario", cioè decrementare un contatore piuttosto che incrementarlo, è molto usata in assembler, oltre a DEX, esistono le istruzioni DEY e DEC che vanno ad agire rispettivamente sul registro Y e sull'accumulatore.

Routine principale

La routine principale del nostro programma si occuperà di tre cose essenzialmente, porre dei valori iniziali alle coordinate x e y ed eseguire un loop (per ora infinito) che cancelli il pallino alla posizione precedente e lo stampi in quella attuale. A questo punto dovrà essere dichiarata un'altra label che punterà a due byte su cui memorizzare le coordinate "future" (NewXY) del pallino, il loro utilizzo sarà chiaro più avanti, e una Label di un byte (Delay) il cui scopo è semplicemente quello di farci perdere tempo … nel senso che ferma per un po' il ciclo del programma, altrimenti il pallino sarebbe talmente veloce da non poter essere visto.
Oltre alle nuove label, ora sfruttiamo un altro vantaggio offerto dalla programmazione in assembly, cioè la definizione di alcuna costanti da poter utilizzare al posto di un valore esplicito all'interno del listato. In particolare ci servirà definire una variabile SCPos da utilizzare al posto del valore $FB (che, se avete fatto attenzione, abbiamo incontrato nel primo listato dell'articolo assieme a $FC). La dichiarazione è semplice:

SCPos = $FB ; Locazione libera alla pagina zero assieme a $FC conterrà il valore della cella video
; calcolata dalla subroutine XYToSCPos

Nel Commodore 64, come accennato in precedenza ci sono poche locazioni libere da sfruttare per caricare e salvare tramite l'indirizzamento Indiretto indicizzato, $FB e $FC sono moto utilizzate essendo 2 tra le 5 libere che io conosco (naturalmente questo è vero se non si libera la pagina 0 dal BASIC).

Questo il listato della routine principale nella versione con Loop Infinito:

```
Start
        lda  #$13               ; mi posiziono al centro
dello schermo
        sta  PosXY              ;
        sta  NewXY
        lda  #$09
        sta  PosXY+1
        sta  NewXY+1
        jsr  XYtoSCPos  ; calcolo la cella dello schermo
relativa a x,y
                                ; quindi ne metto
l'indirizzo in SCPos ($00FB/$00FC)
LoopSenzaFine
        dec  Delay              ; Contatore di ritardo (già
inizializzato nella sua dichiarazione)
        bne  SkipNewPos ; Se diverso da 0 Non fare nulla
        ldy  #$05               ; Reinizializza il numero
di cicli di ritardo
        sty  Delay              ; in Delay
        ldy  #$00
        lda  #$20               ; Metto in A il valore
screen code di spazio
        sta  (SCPos),Y  ; Indiretto Indicizzato Pagina zero
visualizzo lo spazio
                                ; alla posizione appena
calcolata e posta in SCPOS
        jsr  IncDec             ; Calcolo la prossima
coppia coord. x,y
        jsr  XYtoSCPos  ; Calcolo Cella in funzione di x,y
        lda  #$51               ; A = Pallino
        sta  (SCPos),Y  ; Stampa il pallino nella cella
puntata da SCPos
SkipNewPos
        jsr  Wait               ; Attendi
        jmp  LoopSenzaFine      ; torna all'inizio del
ciclo
```

Possiamo tranquillamente saltare allariga di LoopSenzaFine dove iniziamo subito a decrementare il contatore Delay per poi andarne a testare il valore tramite l'istruzione BNE (Brach on Not Equal), la quale esegue il controllo esattamente opposto a BEQ, cioè continuerà l'esecuzione partendo dalla linea SkipNewPos nel caso in cui il flag Z (Zero) sia uguale a 0, cioè quando Delay è diversa da 0. Se fosse uguale a zero procederebbe reinizializzando Delay a 5 (per ricominciare a saltare il calcolo della nuova posizione al prossimo ciclo).

Arrivati a Questo punto non ci resta che analizzare la subroutine IncDec e Wait.

Subroutine decisionale IncDec

La subroutine IncDec, pur essendo corposa no ha quasi nulla che non sia stato affrontato nell'articolo tranne unica istruzione non ancora incontrata, l'istruzione EOR (Exclusive OR) la quale esegue un'operazione logica di OR esclusivo bit a bit tra l'argomento indicato e l'accumulatore mettendo il risultato nell'accumulatore. Nel caso qualcuno di voi fosse a digiuno di algebra binaria, basti sapere che l'or esclusivo tra due bit, restituisce 1 se i due bit confrontati sono diversi altrimenti restituisce 0. Nello specifico, il punto in cui viene usato (che è in realtà è una subroutine di IncDec) viene sfruttato per eseguire l'operazione di negazione bit a bit di una particolare locazione di memoria (infatti non esiste un'istruzione che lo faccia di suo). Infatti eseguendo EOR tra un Byte (es: 00110010) con l'accumulatore = $FF (11111111) ci ritroveremmo sempre nell'accumulatore un valore che è la negazione bit a bit del valore iniziale (nell' es:11001101).

Detto questo mi limiterò a descrivere a grandi linee il semplice algoritmo implementato nella subroutine IncDec.
Il pallino oltre ad essere ad avere una posizione identificata tramite due coordinate x y (poi tradotte in una cella video) ha un'altra proprietà essenziale per capire la direzione dello stesso, tramite la label DirXY identifichiamo due byte che indicano se stiamo "avanzando" o "indietreggiando" sull'asse delle x e delle y (si lo ammetto sono uno sprecone, sarebbero bastati due bit), anche in questo caso, come noterete, la scelta è quella di usare un'unica label per identificare le due proprietà. Questa scelta di usare un'unica label (come per PosXY, NewXY e altre ancora che noterete nel listato completo) è dovuta al fatto che in questa maniera potrò gestire le due coordinate separatamente pur senza scrivere lo stesso codice due volte. Infatti, la gestione di una coordinata piuttosto che l'altra è demandata al valore del Registro X, il quale farà da indice nelle varie istruzioni all'interno della subroutine.

Inizialmente il pallino avanzerà in entrambe le direzioni x e y (quindi verrà incrementata di uno la posizione ad ogni passaggio) percorrendo la diagonale verso il basso a destra. IncDec, ad ogni ciclo, testerà (per ognuna delle coordinate) se il pallino "sforerà" il bordo sia superiore o inferiore mantenendo quella direzione, in tal caso viene cambiata la direzione, label DirXY. Ed è proprio qui che interverrà l'operazione EOR sopra descritta, infatti ho conveuto che per l'avanzamento, il byte direzione dovesse essere $00, mentre per l'arretramento sarà $FF, due valori facilmente convertibili l'uno nell'altro. Da notare che se lo sforamento potenziale avvenisse nel bordo inferiore, dovremmo testare che la coordinata vada a -1, ma non è necessario impelagarsi troppo con le varie operazioni di complemento (in questo caso), in quanto possiamo semplicemente testare che x o y non diventi $FF. Questo perché in un byte con valore $00, se andassimo a sottrarre il valore $01, ci ritroveremmo con il valore $FF (e i vari bit di stato modificati di conseguenza). Altre verifiche che verranno fatte riguardano il contenuto delle celle video dove il pallino si ritroverebbe, anche in questo caso Cambierà direzione qualora incontrasse un carattere diverso da spazio, con in aggiunta il controllo del tocco su uno degli spigoli oppure nel caso si ritrovasse il un "angolo" formato dalle lettere, in entrambi i casi dovrà invertire la direzione sia su x che y.

Subroutine Wait

La subroutine Wait, il cui scopo è semplicemente tenere in sospeso il programma per rendere più fluida la visualizzazione del pallino, sfrutta il tempo che il Commodore 64 ci mette a "ridisegnare" una riga di pixel dello schermo, detta raster line. Senza addentrarci troppo nello specifico, il Commodore 64 ha un paio di celle di memoria ($D011/$D012) in cui viene scritta su quale raster line sta eseguendo in refresh. Per precisione il numero di righe varia da 0 a 319, l'LSB di tale valore è D012, mentre nel $D011 si deve tenere conto solo del bit meno significativo come MSB.

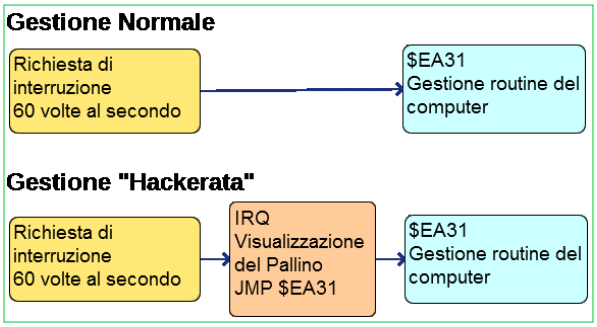
Il corpo di Wait è semplicissimo.

```
Wait
        lda  $d012      ; Carico in A l'attuale linea di
raster LSB
WaitLoop
        clc             ; pulizia carry
        adc  #$FE       ; A = A + $FE
```

```
        cmp  $D012      ; l'attuale raster line = A
        bne  WaitLoop   ; No, attendi ancora
        rts             ; torna al chiamante
```

La routine non fa altro che leggere il valore dell'attuale raster line quindi ci somma $FE, lo mette nell'accumulatore e continua a leggere il valore dell'attuale raster line finché non è uguale a quella dell'accumulatore (CMP $D012/BNE WaitLoop). L'intento era quello di aspettare poco meno di 1/50 di secondo, il tempo che il Commodore 64 ci mette a refreshare tutto il video, la routine comunque non è stata fatta come se stessimo progettando lo Space Shuttle, ma per quello che serve a noi svolge il suo compito più che bene.

Interagiamo con il pallino



Come da progetto iniziale ora ci occuperemo di come fare in modo che il nostro pallino non ci impedisca di svolgere i nostri lavori. Questo è possibile grazie ad una particolare coppia di bytes, l'interrupt vector ($0314/$0315), i quali, in condizione di normalità, contengono l'indirizzo di memoria (nella forma LSB/MSB) $31 e $EA, richiamato 60 volte al secondo. Queste chiamate sono essenziali al sistema affinchè possa per esempio ricevere input da tastiera o semplicemente far lampeggiare il cursore. Il trucchetto è quello di andare a modificare questi 2 bytes inserendo la locazione del corpo principale della nostra routine, assicurandosi che alla fine di essa si vada a fare un JMP alla locazione originaria $EA31. In questa maniera la nostra routine verrà eseguita 60 volte al secondo.

Per fare tutto ciò dobbiamo introdurre una routine di inizializzazione che vada a modificare l'indirizzo dell'interrupt vector, con l'accortezza di disabilitare momentaneamente proprio le richieste di interruzione e riabilitarle quando abbiamo finito.

```
Start
        sei             ; disabilitazione dell'interrupt
request
        lda  #$00       ; inizializzo alla posizione 0,0
schermo
        sta  PosXY      ; le varie coordinate
        sta  NewXY
        sta  PosXY+1
        sta  NewXY+1
        jsr  XytoSCPos  ; salvo su SCPos il valore della
corrispondente cella video
                        ; Mofico l'intrerrupt vector in
modo
                        ; che punti alla mia routine di
visualizzazione del
                        ; pallino (che ho chiamato IRQ)
        lda  #<IRQ      ; indirizzamento immediato LSB
        ldx  #>IRQ      ; indirizzamento immediato MSB
        sta  $314       ; Indirizzi interrupt vector
        stx  $315       ;
        cli             ; ripristino l'interrupt
        rts             ; ritorno al chiamante (BASIC)
```

La disabilitazione dell'interrupt avviene con l'istruzione SEI (Set Interrupt) che va a modificare il relativo bit nel registro di stato del processore, la riabilitazione degli interrupt avviene con l'istruzione contraira CLI (Clear Interrupt). IRQ è la label che ho dato alla routine principale da eseguire 60 volte al secondo. La nomenclatura "<IRQ" o ">IRQ" è una istruzione assembly che indica semplicemente di sostituire quei caratteri con LSB (<) e MSB (>) della label che segue.

Siccome la nostra routine principale ora verrà eseguita all'interno di un processo più ampio di cui non conosciamo (o comunque possiamo non conoscere) i dettagli e la nostra routine principale va ad utilizzare i registri A X Y senza remora alcuna, potremmo andare in conflitto con quelle routine che seguiranno la nostra (a partire da $EA31). Per non saper ne leggere ne scrivere bisogna quindi salvare tali valori per poi ripristinarli prima del salto finale.
Per fare ciò ci viene in aiuto lo Stack. Con questo termine si identifica un'area di memoria che viene gestita tramite alcune apposite istruzioni mnemoniche implicitamente o esplicitamente e sostanzialmente permette di salvare e riprendere dei valori in un ben determinato ordine e cioè facendo in modo che l'ultimo valore che è stato chiesto di salvare (operazione di Push) sia anche il primo ritornato quanto viene richiesto di tornare un valore (operazione di POP). Le istruzioni implicite che vanno ad utilizzare lo stack sono JSR e RTS, infatti quando viene eseguita l'istruzione JSR, prima di saltare alla subroutine viene salvato l'indirizzo successivo a quello di JSR (quindi il push di 2 bytes), nel momento in cui la subroutine chiamata esegue un RTS, il sistema prende dallo stack l'indirizzo salvato e riparte ad eseguire il codice da quell'indirizzo. Se vi fossero subroutine annidate in questo modo il gioco JSR/RTS potrà sempre funzionare. Nel C64 allo stack è riservata l'area di memoria che va da $0100 a $01FF, quindi risulta abbastanza intoccabile dai programmi, altrimenti vi è il rischio di bloccare tutto. Per tenere traccia di dove si trovi l'ultimo dato inserito, il sistema utilizza un ulteriore registro detto Stack Pointer, il quale si incrementa o descrementa in base alle operazione di Push o di Pop effettuate. Le istruzioni esplicite che vanno a lavorare sullo stack sono PHA (PusH Accumulator), PLP (PuLl Accumulator), che permettono il Push e il Pop dell'accumulatore, PHP (PusH Processor status) e PLP (PuLl Processor status). Come possiamo vedere non ci sono operazioni dirette sui registri X e Y, quindi dovremmo passare sempre per l'accumulatore nel momento in cui vogliamo farne il Pesu/Pop. Per traferire il registro X in A e viceversa esiste la coppia di istruzioni TXA e TAX (Tranfer X to A e Transfer A to X) in questo modo riusciamo a salvare tutti i registri senza problemi. Dobbiamo comunque avere l'accortezza di eseguire le istruzioni di Pop in ordine inverso alle istruzioni di Push.

La nostra routine di IRQ è quindi:

```
IRQ
                        ; Salvataggio sullo stack dei
registri
        php             ; Processor status
        pha             ; Salvo l''Accumulatore
        txa             ; A=X
        pha             ; salvo X (tramite il salvataggio
dell'accumulatore)
        tya             ; A=Y
        pha             ; Salvo Y
        dec  Delay      ; Ritardo per rendere un pò
persistente
        bne  SkipIrq    ; la visualizzazione del pallino
        ldy  #$03
        sty  Delay
        lda  #$20       ; A=Spazio Code screen
        ldy  #$00       ; pongo a 0 l'indice per
        sta  (SCPos),y  ; pongo A in SCPos con
l'indirizzamento Indiretto indicizzato
        jsr  IncDec     ; calcolo prossima possizione PosXY
        jsr  XYtoSCPos  ; pongo in SCPos l'indirizzo della
cella video
```

```
        lda  #$51       ; A = Pallino
        sta  (SCPos),y  ; pongo A in SCPos con
l'indirizzamento Indiretto indicizzato
SkipIrq
        pla             ; prendo il quarto valore salvato
nello stack e lo metto in A
        tay             ; Y=A
        pla             ; prendo il terzo valore salvato
nello stack e lo metto in A
        tax             ; X=A
        pla             ; prendo il secondo valore salvato
nello stack e lo metto in A
        plp             ; prendo il primo valore salvato e
lo rimetto nel registro di stato
        jmp  $ea31      ; Salto alla normale gestione
dell'interrupt
```

Arrivati a questo punto basta mettere il tutto all'interno di un progetto assembly, non prima di aver deciso da quale indirizzo far partire il nostro programma in assembler. Nel C64 ci sono varie opzioni su dove far iniziare un programma in assembler senza far danni, ogni uno ha pregi e difetti, ma in definitiva la scelta dell'entry point dipende da fattori tipo l'ampiezza del programma stesso ma soprattutto dipende da quello per cui è stato progettato. Per i nostri scopi possiamo sfruttare lo spazio libero definito come santo graal per il C64, tale spazio inizia dall'indirizzo $C000 (49152 in decimale).

L'indicazione dell'indirizzo (nel CBM studio ma anche in altri programmi di assembly) si effettua ponendo la direttiva:

```
*=$C000
```

Una volta assemblato il tutto e caricato in memoria, da BASIC possiamo scrivere SYS 49152 (ch altri non è che l'istruzione JSR $C000) per far eseguire il nostro programma.

Il programma presentato ha margini di ottimizzazione e soprattutto ha dei problemini per quanto riguarda per esempio l'incontro del pallino con il cursore oppure il problema di proliferazione dei pallini (statici stavolta) nel caso in cui avvenisse uno scroll del video.

Listato del programma

```
Screen  = $0400     ; Indirizzo prima cella video
SCPos   = $FB       ; indirizzo della pagina 0 libero per i
nostri scopi $fb e $fe
Sid     = $0D400    ; registro SID     (Bonus)


               *=$C000
;------------- versione con utilizzo dell'IRQ
                    ; Routine Iniziale per incastonare
                    ; la visualizzazione del pallino
                    ; tramite interrupt
Start   sei         ; disabilitazione dell'interrupt
request

        lda  #$00       ; inizializzo alla posizione 0,0
schermo
        sta  PosXY      ; le varie coordinate
        sta  NewXY
        sta  PosXY+1
        sta  NewXY+1
        jsr  XYtoSCPos  ; salvo su SCPos il valore della
corrispondente cella video
                    ; Mofico l'intrerrupt vector in modo
                    ; che punti alla mia routine di
visualizzazione del
                    ; pallino (che ho chiamato IRQ)
        lda  #<IRQ      ; indirizzamento immediato LSB
        ldx  #>IRQ      ; indirizzamento immediato MSB
        sta  $314       ; Indirizzi interrupt vector
        stx  $315       ;
        cli             ; ripristino l'interrupt
        rts             ; ritorno al chiamante

IRQ
                    ; Salvataggio sullo stack dei
registri
        php             ; Processor status
        pha             ; Salvo l''Accumulatore
        txa             ; A=X
        pha             ; salvo X (tramite il salvataggio
dell'accumulatore)
        tya             ; A=Y
        pha             ; Salvo Y
        dec  Delay      ; Ritardo per rendere un pò
persistente
        bne  SkipIrq    ; la visualizzazione del pallino
        ldy  #$03
        sty  Delay
        lda  #$20       ; A=Spazio Code screen
        ldy  #$00       ; pongo a 0 l'indice per
        sta  (SCPos),y  ; pongo A in SCPos con
l'indirizzamento Indiretto indicizzato
        jsr  IncDec     ; calcolo prossima possizione PosXY
        jsr  XYtoSCPos  ; pongo in SCPos l'indirizzo della
cella video
        lda  #$51       ; A = Pallino
        sta  (SCPos),y  ; pongo A in SCPos con
l'indirizzamento Indiretto indicizzato
SkipIrq
        pla             ; prendo il quarto valore salvato
nello stack e lo metto in A
        tay             ; Y=A
        pla             ; prendo il terzo valore salvato
nello stack e lo metto in A
        tax             ; X=A
        pla             ; prendo il secondo valore salvato
nello stack e lo metto in A
        plp             ; prendo il primo valore salvato e
lo rimetto nel registro di stato
        jmp  $ea31      ; Salto alla normale gestione
dell'interrupt
;------------Fine versione con IRQ

; ------ Versione senza interazione
;Start
;        lda  #$13       ; mi posiziono al centro dello
schermo
;        sta  PosXY      ;
;        sta  NewXY
;        lda  #$09
;        sta  PosXY+1
;        sta  NewXY+1
;        jsr  XYtoSCPos  ; calcolo la cella dello schermo
relativa a x,y
;                        ; quindi ne metto l'indirizzo in
SCPos ($00FB/$00FC)
;LoopSenzaFine
;        dec  Delay      ; Contatore di ritardo
;        bne  SkipNewPos ; Se diverso da 0 Non fare nulla
;        ldy  #$05       ; Reinizializza il numero di cicli
di ritardo
;        sty  Delay      ; in Delay
;        ldy  #$00
;        lda  #$20       ; Metto in A il valore screen code
di spazio
;        sta  (SCPos),Y  ; Indiretto Indicizzato Pagina zero
visualizzo lo spazio
;                        ; alla posizione appena calcolata e
posta in SCPOS
;        jsr  IncDec     ; Calcolo la prossima  coppia
coord. x,y
;        jsr  XYtoSCPos  ; Calcolo Cella in funzione di x,y
;        lda  #$51       ; A = Pallino
```

```
;        sta  (SCPos),Y  ; Stampa il pallino nella cella
puntata da SCPos
;SkipNewPos
;        jsr  Wait       ; Attendi
;        jmp  LoopSenzaFine ; torna all'inizio del ciclo
; ------ Fine versione senza irq



XYtoSCPos               ; Calcola la posizione dello schermo
= $0400 + xpos + ypos * 40
        lda  PosXY+1    ; copio il contenuto di YPos su
Factor1
        sta  factor1    ; Byte meno significativo (LSB)
        lda  #$28       ; Scrivo 40 ($28) in Factor2 (LSB)
        sta  factor2
        lda  #$00       ; Azzero i byte più significativi
(MSB)
        sta  factor1+1  ; su Factor1
        sta  factor2+1  ; su Factor2
        jsr  Moltiplica ; Chiamata alla Routine Moltiplica
                        ; eseguirà Result = Factor1 *
Factor2
                        ; cioè Result = YPos * 40
        lda  Result     ; Copio LSB di Result su Factor1
        sta  Factor1    ; LSB
        lda  Result+1   ; Copio MSB di Result su Factor1
        sta  Factor1+1  ; MSB
        lda  PosXY      ; Copio LSB di XPos in
        sta  Factor2    ; LSB di Factor2 (MSB è $00)
        jsr  Sum16      ; Chiamata alla routine di somma a
16 bit
                        ; Factor1 + Result
                        ; cioè XPos + YPos * 40
                        ; metterà il risultato della somma
in Result
        lda  Result     ; Copio Result LSB/MSB in Factor1
        sta  Factor1    ; copia prima LSB
        lda  Result+1   ; ... poi
        sta  Factor1+1  ; ... MSB
        lda  #<Screen   ; Copio la costante screen Screen
(LSB/MSB)
        sta  Factor2    ; in Factor2 (LSB)
        lda  #>Screen   ; ... poi
        sta  Factor2+1  ; ... MSB
        jsr  Sum16      ; Chiamata alla routine di somma a
16 bit
                        ; Result = Result + Screen
        lda  Result     ; Finalmente copio Result in SCPos
        sta  SCPos      ; ... manco a dirlo prima LSB
        lda  Result+1   ; ... poi
        sta  SCPos+1    ; ... MSB
        ldy  #$00       ; Azzero il registro indice y
        lda  (SCPos),y  ; carico nell'accumulatore il valore
che si trova
                        ; allindirizzo che trovo a a partire
dall'indirizzo
                        ; SCPos, cioè il valore calcolato
        rts             ; ritorno a chiamante
Sum16
        clc             ; Mi assicuro che non vi sia un
riporto (Carry)
        lda  Factor1    ; A = LSB(Factor1)
        adc  Factor2    ; A = A + LSB(Factor2) + Carry
        sta  Result     ; LSB(Result) = A  e viene impostato
il carry
        lda  Factor1+1  ; A = MSB(Factor1)
        adc  Factor2+1  ; A = A + MSB(Factor2) + Carry
        sta  Result+1   ; MSB(Result) = A  e viene impostato
il carry
        rts             ; Ritorno al chiamante
```

```
Moltiplica              ; Moltiplica LSB(Factor1) *
LSB(Factor2)
        lda  #$00       ;
        sta  Result     ;
        sta  Result+1   ; Risultato = 0
        ldx  Factor2    ; x = Factor2
        cpx  #$00       ; x == 0 ?
        beq  FineMoltiplica ; Se x==0 fine moltiplicazione
(Result = 0)
        lda  Factor1    ; a = LSB(Factor1)
        cmp  #$00       ; a == 0 ?
        beq  FineMoltiplica ; se a == 0 vai a fine
moltiplicazione (Result = 0)
        sta  Result     ; altrimenti Result = Factor1
LoopMoltiplica
        dex             ; x=x-1 (x inizialmente conteneva
LSB(Factor2))
        beq  FineMoltiplica ; Se x==0 vai a fine
Moltiplicazione
        clc             ; Pulisci il Carry
        lda  Result     ; -----------------
        adc  Factor1    ; Somma Result con Factor1
        sta  Result     ; metti il risultato su Result
        lda  #$00       ; Caricamento immediato 00 (più
veloce)
        adc  Result+1   ; Sommo il solo carry
        sta  Result+1   ; ----------------
        jmp  LoopMoltiplica     ; Vai a LoopMoltiplica
FineMoltiplica
        rts             ; Torna al chiamante



IncDec                  ; incr./decr. (in base al verso)
coordinate x o y
        lda  PosXY      ; Salvo la coordinata x
        sta  BakXY      ; su posizione di Backup
                        ; giro di test su x per vedere se
andanado
                        ; avanti per la direzione x attuale
incontrerò
                        ; un carattere diverso da spazio
                        ; la routine IncDecXY tiene conto di
eventuali bordi
                        ; dello schermo raggiunti
        ldx  #$00       ; registro X indica a IndDecXY se
lavorare con x o y
        jsr  IncDecXY   ; registro X=0 => lavora su
coordinata x
        lda  NewXY      ; assegna la nuova coordinata x
(calcolata da IncDecXY)
        sta  PosXY      ; alla coord. x attuale
        jsr  XYtoSCPos  ; calc.cella schermo e mette in A il
carattere
        ldy  BakXY      ; ripristino la coordinata x
        sty  PosXY      ; al valore prima della routine per
test
        cmp  #$20       ; in A ho il carattere delle
coordinate calcolate
        beq  TestYDir   ; se spazio allora proseguo col
vedere la stessa cosa su Y
        jsr  RestoreXY  ; prima di proseguire ritorno nei
miei passi in x e cambio direzione x
TestYDir
        lda  PosXY+1    ; Salvo y attuale
        sta  BakXY+1    ;
        ldx  #$01       ; indico di lavorare su y
        jsr  IncDecXY   ; Decrementa o incrementa y in base
alla direzione y attuale
        lda  NewXY+1
        sta  PosXY+1
        jsr  XYtoSCPos  ; A = Carattere in prossimità
seguendo nuova y
        ldy  BakXY+1    ; rispristino la y prima del test
        sty  PosXY+1
```

```
        cmp  #$20       ; A contiene uo spazio ?
        beq  TestDiagonale ; se contiene uno spazio test
carattere diagonale in direzione
        jsr  RestoreXY  ; ritorno sui miei passi per y
TestDiagonale           ; controllo del carattere che si
trova nella diagonale
                        ; calcolata rispetto alla direzione
attuale
        lda  NewXY      ; Copio x e y possibili calcolate
rispetto al test sui caratteri
        sta  PosXY      ; prossimi a x e y attuali ...
        lda  NewXY+1
        sta  PosXY+1
        jsr  XYtoSCPos  ; ... calcolo nuova cella schermo
A=car.sulla cella
        cmp  #$20       ; A contiene uno spazio ?
        beq  IncDecExit ; Si, tutto a posto le coordinate
nuove vanno bene
        ldx  #$00       ; No
        jsr  ChgDirXY   ; Cambio Direzione delle x
        ldx  #$01       ;
        jsr  ChgDirXY   ; e delle y (rimbalzo)
        lda  BakXY      ; e ripristino le "vecchie"
coordinate x y
        sta  PosXY
        lda  BakXY+1
        sta  PosXY+1
IncDecExit
        rts             ; ritorno al chiamante

IncDecXY
        lda  DirXY,X    ; carico in A x o y in base
all'indice Registro X
        cmp  #$00       ; coordinata == 0 ?
        beq  IncTestCoord ; Si, Incremento la coordinata di
1 e check sforamento bordi
        jmp  DecTestCoord ; No, Decremento la coordinata di
1 e check sforamento bordi

IncTestCoord
        inc  NewXY,X    ; Incremento la coordinata puntata
dal registro X
        lda  NewXY,X    ; la carico in A
        cmp  MaxXY,X    ; controllo col numero di colonne
                        ; coordinata parte da 0 quindi se =
numero colonna ho sforato
        beq  ChgDirDec  ; se ho "sforato" il bordo massimo
inverto direzione
                        ; su quella specifica coordinata e
la decremento
        rts             ; ritorno

DecTestCoord
        dec  NewXY,X    ; Decremento la coordinata puntata
dal registro X
        lda  NewXY,X    ; Carico in A
        cmp  #$FF       ; ho sforato il bordo minimo ($00-
$01 = $FF) ?
        beq  ChgDirInc  ; Cambia direzione e incrementa la
coordinata
        rts             ; ritorno


ChgDirInc
        jsr  ChgDirXY   ; Cambia direzione rispetto alla
coordinata
        inc  NewXY,X    ; incrementa la coordinata
        rts             ; ritorno

ChgDirDec
        jsr  ChgDirXY   ; Cambio direzione
        dec  NewXY,X    ; Decremento la coordinata in
accordo con il registro X
        rts             ; ritorno


RestoreXY               ; ritorno sulla coordinata
temporalmente precedente
        lda  PosXY,X    ; copio la "vecchia" coordinata
        sta  NewXY,X    ; su quella nuova
ChgDirXY
        lda  #$FF       ; A = $FF
        eor  DirXY,X    ; inverto i bit della direzione
della coordinata
                        ; se era $00 diverrà $FF e viceversa
        sta  DirXY,X    ; aggiorno la direzione della
coordinata
        jsr  Boing      ; effetto sonoro
        rts             ; ritorno

Boing                   ; Piccolo effetto sonoro
        php             ; Salvataggio dei vari registri
        pha
        txa
        pha
        ldx  #$1C
        lda  #$00
BoingLoop
        sta  SID,x
        dex
        bne  BoingLoop
        sta  SID
        lda  #$0f
        sta  SID + 24
        lda  #$14
        sta  SID + 1
        lda  #$00       ; 0*16+0
        sta  SID + 5
        lda  #$f9       ; 15*16+9 (249)
        sta  SID + 6
        lda  #$11       ; 1+16
        sta  SID + 4
        lda  #$10       ; 16
        sta  SID + 4
        pla              ; Ripristino dei registri
        tax
        pla
        plp
        rts


                        ; Attesa senza far niente
                        ; sfruttando la raster line
Wait
        lda  $d012      ; Carico in A l'attuale linea di
raster LSB
WaitLoop
        clc             ; pulizia carry
        adc  #$FE       ; A = A + $FE
        cmp  $D012      ; l'attuale raster line = A
        bne  WaitLoop   ; No, attendi ancora
        rts             ; torna al chiamante

Delay   byte $01        ; Indicatore del numero di cicli
Wait a vuoto pe rla persistenza a video
Factor1 word $0000      ; Primo fattore per la
moltiplicazione
Factor2 word $0000      ; Secondo fattore per la
moltiplicazione
Result  word $0000      ; Risultato delle operazioni
PosXY   byte $00,$00    ; x, y Posizione attuale
NewXY   byte $00,$00    ; x, y Poszione futura
MaxXY   byte $28,$19    ; col/row number 40, 25 (remember
lsb &msb are inverted)
DirXY   byte $00,$00    ; x, y directions 00 forward ff
backward
BakXY   byte $00,$00    ; Backup dei valori XY
```

RetroTool: DFM Database 464 (Amstrad CPC)

Di Francesco Fiorentini

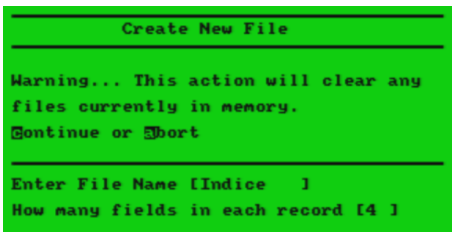
E' esattamente dal numero 7 di Maggio 2018 che non scrivevamo per la rubrica RetroTool. In quell'occasione parlammo di un moderno Spreadsheet per il Commodore 64, questa volta invece vogliamo parlarvi di un Database per l'Amstrad CPC, il **DFM Database 464** creato dalla Dialog Software e pubblicato da Amsoft.

Come dice il manuale stesso, DFM Database e' un potente **Data File Management System residente in memoria**, da usarsi in ufficio o a casa. Devo ammettere che il manuale non esagera assolutamente nella sua definizione perche' il tool, pur limitato dalla quantita' di memoria disponibile, offre una completa gamma di servizi a corredo.

Scordiamoci pero' le interfacce grafiche, il tutto viene gestito da un'essenziale interfaccia a caratteri. Essenziale quanto pulita e perfettamente funzionale allo scopo.

Il menu' iniziale ci accoglie con una serie di opzioni accessibili tramite la pressione del numero corrispondente. Alcuni di questi comandi accedono immediatamente alla relativa funzione, mentre altri ci portano ad un sotto-menu' con ulteriori scelte a disposizione (e.g. Reports ed Utilities).

Per testare il softwarer e le sue funzionalita', direi di provare a creare subito un nuovo archivio: **1- Create new file**

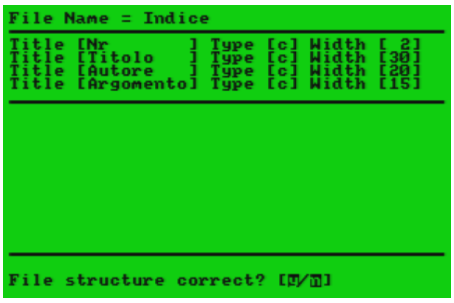


Nel processo di creazione del nuovo archivio, il sistema ci avverte che questa operazione cancellera' tutti i file presenti in memoria. Infatti, per garantire una velocita' di accesso alle informazioni accettabile, tutti gli archivi utilizzati verranno caricati in memoria e gestiti in RAM; potremmo si' memorizzarli e leggerli da cassetta, ma tutte le operazioni verranno comunque eseguite in memoria.

Anche se ultimamente e' passato in secondo piano, grazie alle funzionalita' dei moderni software, in DFM Database dobbiamo avere ben chiara in mente la struttura della nostra base dati prima di cominciare a lavorare. Questo perche' le modifiche successive alla struttura dati si limitano alla possibilita' di rinominare il campo ed I file. Scordatevi modifiche alla natura dei campi la rimozione o l'aggiunta degli stessi. L'archivio, una volta creato, rimane invariato.

Ecco quindi spiegato il motivo del successivo messaggio: il programma, per calcolare il numero massimo di record a disposizione, deve sapere a priori il numero dei campi e la loro natura (dimensione).

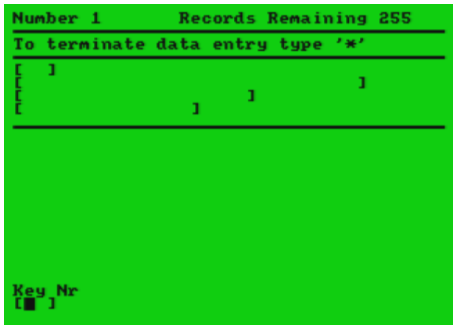
Per cominciare proviamo a creare un archivio contenente l'indice degli articoli pubblicati su RetroMagazine.



Come si vede dalla schermata precedente, per semplicita' ho creato 4 campi alfanumerici corrispondenti alla struttura:

Nome campo	Tipo	Dimensione
Numero	Char	2
Titolo	Char	30
Autore	Char	20
Argomento	Char	15

Una volta terminata la definizione del nostro archivio, passiamo a popolarlo:



Ed ecco invece come appare il nostro archivio una volta inserito il primo record:

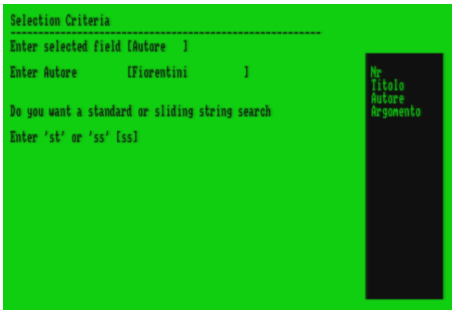
Niente affatto male; il tutto appare leggibile e con diverse opzioni a portata di mano per spostarsi velocemente tra i record ed usufruire dell'archivio nel migliore dei modi. Vediamo insieme quali sono le opzioni offerte dal programma:

- **n**ext – va al record successivo
- **l**ast – ritorna al record precedente
- **f**ind – ricerca un record tramite il primo campo (trattato come campo chiave)
- **g**oto – va ad uno specifico record

- **a**mend – permette di modificare il nome di un campo o I dati contenuti nello stesso
- **m**enu – ritorna al menu principale
- **c**alcs – opzione molto interessante; permette di effettuare un'operazione di somma su un campo individuale o sull'intero archivio
- **i**nput – aggiunge nuovi record al file; premendo '=' in ogni campo permette di ripetere il valore inserito nell'ultimo record
- **d**elete – cancella il record corrente
- **p**rint – stampa il record corrente
- **s**croll – scorre automaticamente l'archivio
- **<>** – nei campi contenenti piu' di 24 caratteri permette di scorrere orizzontalmente per leggere il contenuto completo

E' innegabile che le opzioni a disposizione sono veramente molte e capaci di soddisfare le basilari esigenze di gestione di un semplice archivio. Ma torniamo velocemente al menu principale e proviamo un paio di ulteriori interessanti feature di DFM Database:

L'opzione **4. Select** permette di effettuare una ricerca all'interno dell'archivio. Devo ammettere che ero abbastanza prevenuto riguardo questa feature, ma complice il fatto che tutto l'archivio viene gestito in RAM, il risultato non e' poi cosi' male. Anche l'interfaccia e' all'altezza della situazione:



Dobbiamo innanzitutto scegliere il campo su cui effettuare la ricerca; se non vi ricordate I nomi dei campi del vostro archivio nessun problema, sulla destra una lista vi fornira' visivamente questa informazione.

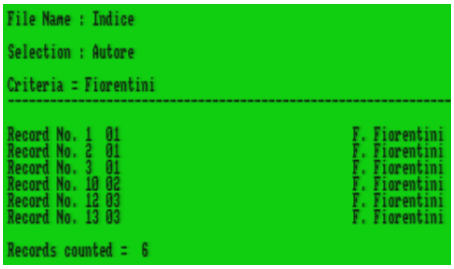
Dopodiche' dovremo immettere la stringa da ricercare e decidere tra una ricerca 'standard string' oppure una ricerca 'sliding string'.

La ricerca standard ricerchera' esattamente la stringa a partire dalla prima posizione nel campo, ma ritornera' esito positivo solo se esattamente corrispondente a quanto trovato nel record (e.g. stringa%).

La ricerca sliding invece ricerchera' la stringa a prescindere da dove e' contenuta all'interno del campo (e.g. %stringa%).

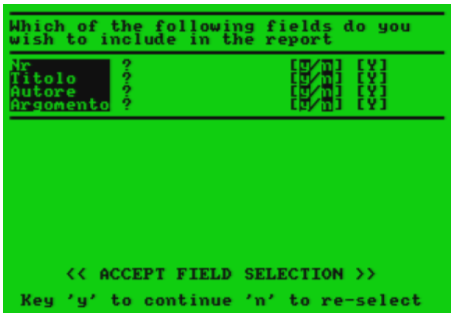
Contenuto	Ricerca	Tipo	Esito
F. Fiorentini	F.Fio	st	OK
F. Fiorentini	F.Fio	ss	OK
F. Fiorentini	Fio	st	KO
F. Fiorentini	Fio	ss	OK

Si veda di seguito un esempio di una ricerca sliding string all'interno del campo autore:



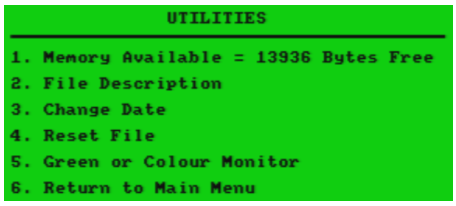
Come si puo' notare la stringa 'Fiorentini', contenuta all'interno del campo Autore come 'F. Fiorentini' e' stato trovata con successo in 6 record all'interno dell'intero archivio. Come dicevo prima, la ricerca non e' il massimo della velocita', soprattutto nel caso di una ricerca di tipo sliding, ma dato che il tutto viene effettuato in RAM, i tempi di attesa sono accettabili.

Un'altra opzione interessante di questo tool e' la possibilita' di stampare, a video o su carta, il contenuto dell'achivio: **6. Reports**.



Come si vede dalla schermata precedente, il tool permette addirittura di selezionare quali campi includere nel report e di generare un report a video in 80 colonne con tanto di intestazione e totale (se presenti campi numerici). Niente male affatto.

Prima di chiudere con il DFM Database pero' diamo un'occhiata al menu' **8. Utilities**.



Da qui e' possibile accedere ad informazioni utili come la data visualizzata nel report di stampa, la descrizione del file attualmente in memoria (giusto il nome...), la quantita' di memoria RAM libera e la possibilita' di resettare l'archivio.

Gli anni passano per tutti e soprattutto per i tool di office automation! Attualmente un prodotto del genere difficilmente potrebbe trovare spazio in un ufficio moderno, ma e' sicuramente utile per rammentare come si lavorava con gli home computer ad 8bit. ☺

Link Utili

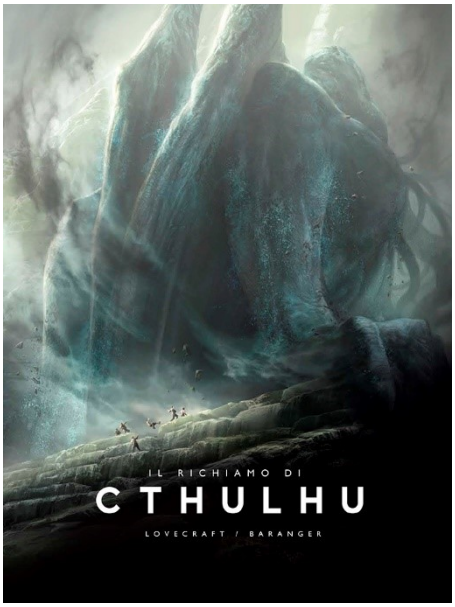
Download DFM Database: http://www.cpc-power.com/index.php?page=detail&num=5038

Manuale: http://www.cpcwiki.eu/imgs/8/8b/DFM_Database_%2B_Labels_%28Amsoft_UK%29_Manual.pdf

IL RICHIAMO DI CTHULHU (Chaosium, 1981, Sandy Petersen)

-Come smisi di dedicarmi ad elfi e draghi abbracciando la follia.-

di Starfox Mulder



La più antica e potente emozione umana è la paura, e la paura più antica e potente è la paura dell'ignoto.

1997, Rimini.
Avevo da pochi mesi conosciuto i giochi di ruolo grazie a Dungeons and Dragons quando decisi di recarmi, solitario, nell'unico negozio, tra quelli raggiungibili facilmente in treno da casa mia, che vendesse manuali per i suddetti. Conobbi Lovecraft ed i suoi racconti solo l'anno prima e da adolescente vorace ne feci incetta molto rapidamente. Trovarmi davanti ad un manuale chiamato come uno dei suoi racconti più significativi fu una sorpresa ma non ci pensai a lungo prima di investire in esso. Il mio primo tomo, la prima volta in cui feci da "Custode" (così si chiamava il Master nel gdr della Chaosium, edito in Italia da Stratelibri).

Passarono gli anni e solo recentemente potei giocare un'avventura ambientata nel tetro mondo del solitario di Providence, poiché nei decenni passati il mio ruolo era sempre e solo stato quello del creatore di storia, mai di colui che le viveva.

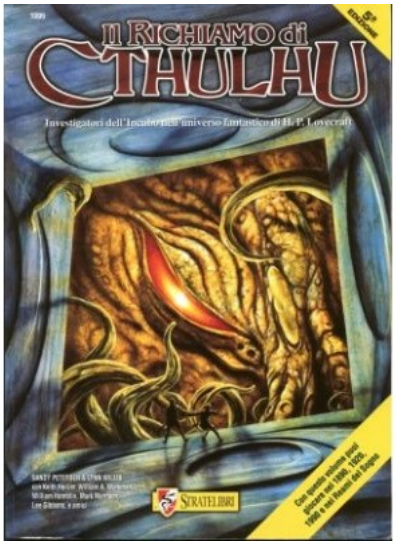
Orrori inenarrabili, antichi tomi, tombe inesplorate, miti ancestrali...sì insomma, una serie di aggettivi e sostantivi che il buon Howard amava decisamente piazzare come il prezzemolo nelle sue opere, capaci oggi di riassumerne velocemente lo stile.
Ora che son tornato "serio", andiamo a parlare del gioco in se ma prima un piccolo capitolo sui suoi autori.

Che imbecille sono stato a intraprendere con tanta incoscienza lo studio di misteri che l'uomo non dovrebbe affatto conoscere!

Perchè dico suoi? Perchè il gioco fu creato da Sandy Petersen, un nome che tutti i videogiocatori dovrebbero conoscere per la sua partecipazione a titoli come Darklands, Doom o Quake, ma senza Howard Philips Lovecraft non sarebbe mai potuto neppure essere pensato.

Petersen ebbe l'incredibile merito di lanciare nell'immaginario nerd i miti di cthulhu in un periodo "non sospetto". Scordatevi l'era moderna in cui siamo stati abituati a vedere la grande seppia anche in compagnia di Hello Kitty (sono serio...purtroppo), nel 1981 erano in pochi a conoscere Lovecraft e l'opera di Petersen contribuì molto alla sua celebrazione nella cultura popolare. Lovecraft d'altra parte dovreste conoscerlo un po' tutti.

Scrittore dall'immaginazione fervida come pochi altri nella storia, morì poco celebrato e solo nei decenni a venire venne riscoperto per il grande narratore che era. Il suo stile aveva mille difetti ma la sua capacità di creazione nessun rivale e questo lo portò ad essere fondamentalmente uno scrittore di nicchia. Penso che pochi oggi non abbiano mai sentito nominare Cthulhu, ma quanti di costoro hanno letto i racconti del suo inventore?



I Miti Lovecraftiani erano perfetti per l'intrattenimento che ne sarebbe seguito ma non per lo scopo originale. Sia chiaro, sono un fan sfegatato di Lovecraft e mi rileggo periodicamente i racconti che preferisco della sua collezione, titoli come "La maschera di Innsmouth" o "L'orrore di Dunwich" sono eterne gemme di letteratura, ma sarei sciocco a definirli di facile fruizione.

Semplice o complessa che fosse la narrazione, il suo lascito è eterno e come tale ispirò uno dei migliori giochi di ruolo di sempre.

Non è morto ciò che può vivere in eterno, e in strani eoni anche la morte può morire.

Il richiamo di Cthulhu di Petersen arrivò alla quinta edizione quando lo comprai (ora è alla settima) e si basò sin dall'inizio sul D100 system, altrimenti detto Basic-Role Playing. Le dinamiche sono semplici e facilmente comprensibili: ogni personaggio ha una serie di caratteristiche e abilità.

Le prime sono da tirarsi con un numero variabile di dadi mentre le seconde sono scelte dal giocatore stesso in base ai punteggi ottenuti in Educazione e Intelligenza (due caratteristiche) ed alla professione scelta. Niente guerrieri, maghi o ladri quindi ma bensì Investigatori, Professori e Parapsicologi. Le professioni sono tante di più ad essere precisi ma il concetto è semplice: il nostro personaggio sa quello che ha senso che abbia imparato. Ogni abilità ha un punteggio base (che può essere alto come il 50% dell'abilità Pugno o nullo come l'abilità Lingue Straniere) a cui aggiungeremo i punti liberi disponibili e otterremo così una serie di punteggi variabili da 0% a 100%.

Ogni qual volta il nostro personaggio cercherà, nel corso delle sue avventure, di compiere un azione difficile che si basi su una delle abilità contemplate, effettuerà un tiro di dado lanciando due dadi da 10 facce, il primo per le decine ed il secondo per le unità. Se il risultato ottenuto sarà uguale o inferiore al suo punteggio di abilità la prova sarà riuscita mentre se sarà superiore avrà fallito.

Semplice, immediato e poco cervellotico una volta fatta la scheda.

Ma tolto il sistema di regole, che cosa farà il nostro personaggio? Indagherà su misteri al di sopra dell'umana comprensione, cercando al contempo di uscirne vivo e soprattutto...sano di mente!

Gli uomini di più ampio intelletto sanno che non c'è netta distinzione tra il reale e l'irreale, che le cose appaiono come sembrano solo in virtù dei delicati strumenti fisici e mentali attraverso cui le percepiamo.

Un gioco di ruolo investigativo a tutti gli effetti, il primo di cui abbia mai sentito parlare e sicuramente il primo di stampo horror.

Scordiamoci i potenti guerrieri in armature magiche pronti ad affrontare orchi e stregoni, il nostro investigatore è mortale ed un colpo di artiglio indirizzato alla sua faccia rischierà di staccargli di netto la testa al primo tentativo. Peggio ancora, l'avversario potrebbe non dover neppure arrivare allo scontro fisico per aver ragione del nostro alter-ego, dal momento che la sola visione di orrori inaspettati rischierà di far vacillare il senno del malcapitato. Ogni personaggio è infatti dotato, tra le altre caratteristiche, di un punteggio nuovo e minaccioso: i punti sanità.

La sanità mentale dello stesso viene infatti calcolata in fase di creazione ma subirà continue sollecitazioni nel corso delle avventure. La visione di un mostro, di un massacro o di qualsiasi scena particolarmente spaventosa lo porteranno ad un "tiro sanità" e fallirlo (ma in certi casi pure riuscirlo, sebbene in maniera minore) comporterà una perdita di punti sanità.

Troppi punti sanità persi ed ecco che affioreranno le follie, vere e proprie patologie mentali come la schizofrenia, capaci di affliggere il personaggio da lì alla fine dei propri giorni. Curarle? Possibile ma con grossi rischi e disponibilità di tempo.

Come se non bastasse il punteggio massimo di sanità andrà a calare in modo permanente all'aumentare della conoscenza in "Miti di Cthulhu". La consapevolezza di ciò che realmente si cela nello spazio profondo o negli abissi oceanici renderanno il nostro

investigatore sempre più instabile e non esisterà cura per la spirale di follia in cui prima o dopo finirà.

Non di rado i giocatori saranno costretti a cambiare personaggio in corso d'opera e nella campagna più lunga da me condotta ricordo bene che nessuno dei miei amici riuscì ad arrivare alla fine della storia con lo stesso personaggio con cui la cominciò.

Morti ammazzati per un tiro andato a male, impazziti per le troppe rivelazioni, i personaggi del Richiamo di Cthulhu fanno tutti una brutta fine prima o dopo e solo i giocatori, spesso, avranno il privilegio di scoprire dove le trame ordite dal Custode siano dirette.

Arrivato ai miei ultimi giorni, e spinto verso la follia dalle atroci banalità dell'esistenza che scavano come gocce d'acqua distillate dai torturatori sul corpo della vittima, cercai la salvezza nel meraviglioso rifugio del sonno.

Trattata la questione regolamento non resta che parlare delle avventure in se e sta qui la vera forza del gioco. I giocatori sono chiamati ad investigare in una delle tre ambientazioni proposte dal manuale (1890 – 1920 – Oggi) e lo faranno inizialmente in punta di piedi.

Inconsapevoli degli orrori che si annidano nelle profondità, il loro coinvolgimento con le trame occulte potrebbe partire da una vecchia magione da perlustrarsi per ragioni assicurative (ogni riferimento ad Alone in the Dark è fortemente voluto) o nel rapimento di un ereditiere per mano dei membri d'una strana setta. Gli spunti sono infiniti ma ovviamente i racconti di Lovecraft sono la base da cui far partire tutto. Città dimenticate, villaggi isolati in cui serpeggia uno strano credo, navi scomparse al largo di Boston o antichi tomi ritrovati in degli scavi archeologici del nord Africa che han fatto impazzire il traduttore... infinite possibilità!

Il Custode deve conoscere i racconti che hanno ispirato il lavoro di Petersen mentre i giocatori meno ne sanno e più potranno venirne rapiti. Perchè se da un lato Lovecraft era contestabile come stile di scrittura, vivere in prima persona i suoi orrori è un esperienza che può davvero coinvolgere e terrorizzare, se il Custode saprà rendersi all'altezza del compito ed i giocatori sapranno calarsi nel giusto ruolo.

In un universo senza scopo, tutto è uguale e nulla vale la pena di un serio pensiero.

Se dico che è, ad oggi, uno dei miei GDR preferiti c'è un motivo. Ogni avventura è una nuova scoperta, ogni giocatore un nuovo modo di interpretare la paura nei panni di un alterego distante al proprio carattere ma vicino alla propria psiche. Il richiamo di Cthulhu fu il gioco con cui passai dall'impeto adolescenziale del "avanzo ed attacco" al più ragionato approccio che una vita fugace e fragile poteva impormi.

Sconsigliato ai powerplayer di ogni sorta, consigliato a chi ama investigare sull'ignoto, qualsiasi cosa questo comporti...

[Tutte le parti in corsivo sono citazioni tratte dai racconti di H.P. Lovecraft]

Magnetic Scrolls e The Pawn

di Giorgio Balestrieri

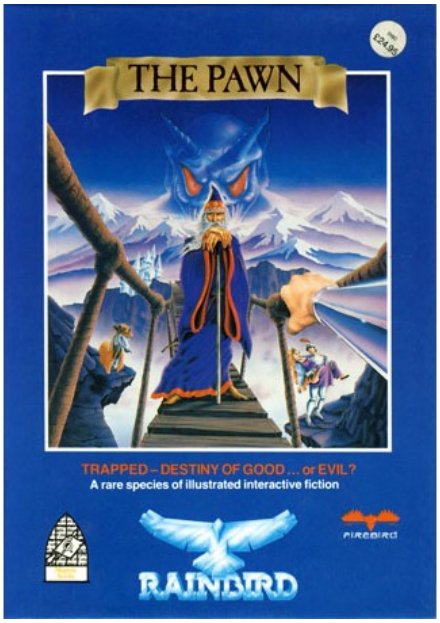
Nel nostro percorso di ri-scoperta delle avventure testuali, le cui tappe sono piazzate assolutamente a caso lungo di esso, è giunto il momento di parlare della Magnetic Scrolls e dei fantastici giochi che grazie ad essa videro la luce. In questo articolo ripercorreremo brevemente la storia di questa eccellente software house e recensiremo *The Pawn*, il gioco che ebbe l'onore di annunciare al mondo l'esistenza di un degno rivale della Infocom.

Magnetic Scrolls

A metà degli anni '80, più precisamente nel 1984, Anita Sinclair, Ken Gordon e Hugh Steers decisero che l'ora di sfidare la Infocom (nata nel 1979) era giunta, dando vita a Londra ad una società orientata alla produzione di avventure testuali che potessero competere con quelle della casa americana sia sul piano narrativo, sviluppando storie con trame articolate e coinvolgenti, sia su quello tecnico, grazie ad un parser estremamente avanzato ed una progettazione che desse al giocatore la maggior libertà possibile, ampliando l'esperienza utente oltre il mero obiettivo di gioco. L'azienda partì con un team di otto dipendenti fissi e un numero variabile di collaboratori free-lance, con al timone la Sinclair nel ruolo di direttore generale, Gordon in quello di direttore tecnico e Steers, che dei tre possedeva le maggiori abilità di programmazione, come capo programmatore. Il piano prevedeva di produrre giochi principalmente per Spectrum QL, microcomputer di cui Anita possedeva un modello, ma l'arrivo sul mercato dell'Atari ST e del Commodore Amiga, con la loro superiorità sia sul fronte hardware che del sistema operativo, convinse i tre soci fondatori ad ampliare i loro obiettivi iniziali ed aumentare gli sforzi di sviluppo per supportare il prima possibile anche queste due piattaforme. Il risultato del loro lavoro fu *The Pawn*, il primo gioco prodotto dall'azienda, sviluppato da un'allora diciottenne Rob Steggles, autore delle quattro più apprezzate avventure della casa. Come Infocom ed in generale la maggior parte delle software house che si occupavano della produzione di giochi d'avventura testuali, anche i ragazzi della Magnetic Scrolls scelsero la strada di creare un'interprete (o macchina virtuale se preferite, visto che di fatto di quello si trattava) che permettesse l'esecuzione dei giochi su più piattaforme hardware senza necessità di una riscrittura totale ossia che evitasse, o perlomeno minimizzasse, gli oneri del porting. L'intero sistema di gioco (oggi diremmo *game engine*) fu creato da Steers e Gordon e disponeva di uno dei migliori parser mai visti, paragonabile a quello della Infocom. Inoltre, era possibile inserire immagini grafiche e suoni nel giochi, anche se queste caratteristiche aggiuntive non erano disponibili per tutte le piattaforme. Nelle prime versioni, i programmatori compilavano i file sorgente dei giochi con l'ausilio di un normale editor di testi, ma in quelle successive fu possibile sfruttare un tool grafico, prodotto grazie al lavoro congiunto del trio Sinclair/Gordon/Steers, per accelerare il processo di produzione e permettere agli sviluppatori di concentrarsi di più sul game design che sulla scrittura del codice. Durante il suo periodo di attività, la Magnetic Scrolls sviluppò 8 giochi:

- **The Pawn**, 1985, pubblicato da Rainbird per i sistemi: Amiga, Amstrad CPC, Amstrad PCW, Apple II, Acorn Archimedes, Atari ST, Atari 8-bit, Commodore 64, MS-DOS, Macintosh, Sinclair QL e ZX Spectrum
- **The Guild of Thieves**, 1987, pubblicato da Rainbird per i sistemi: Amiga, Amstrad CPC, Amstrad PCW, Apple II, Acorn Archimedes, Atari ST, Atari 8-bit, Commodore 64, MS-DOS, Macintosh e ZX Spectrum
- **Jinxter**, 1987, pubblicato da Rainbird per i sistemi: Acorn Archimedes, Amiga, Amstrad CPC, Amstrad PCW, Apple II, Macintosh, Atari 8-bit, Atari ST, Commodore 64, MS-DOS, ZX Spectrum
- **Corruption**, 1988, pubblicato da Rainbird per i sistemi: Amiga, Amstrad CPC, Amstrad PCW, Apple II, Acorn Archimedes, Atari ST, Commodore 64, Apple Macintosh, MS-DOS, ZX Spectrum
- **Fish!**, 1988, pubblicato da Rainbird per i sistemi: Amiga, Amstrad PCW, Apple II, Acorn Archimedes, Atari ST, Commodore 64, Macintosh, MS-DOS, ZX Spectrum
- **Myth**, 1989, pubblicato da Rainbird per i sistemi: Amiga, Amstrad PCW, Atari ST, Commodore 64, MS-DOS, ZX Spectrum
- **Wonderland**, 1990, pubblicato da Virgin Interactive per i sistemi: MS-DOS, Acorn Archimedes, Amiga, Atari ST
- **The Legacy: Realm of Terror**, 1993, pubblicato da MicroProse per il solo sistema MS-DOS. Ne fu prevista anche una versione per Amiga che però non fu mai completata.

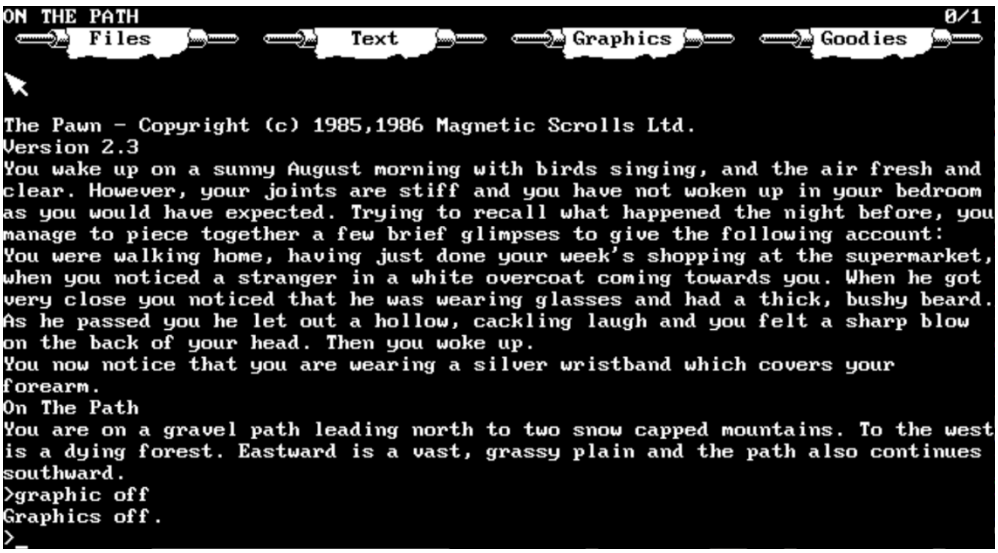
Tutti erano caratterizzati da un livello di difficoltà superiore alla media, da un eccellente parser dotato di un vocabolario davvero ampio e da illustrazioni grafiche di altissima qualità. La brillante carriera dei ragazzi della Magnetic Scrolls terminò quando l'interesse del pubblico si spostò su altri generi di giochi, avventure grafiche in primis, e la produzione di avventure testuali smise di generare profitti sufficienti alla sopravvivenza del settore. I giochi che videro la luce nel suo periodo di attività sono però apprezzati e giocati ancora oggi e considerati tra i migliori di sempre.



The Pawn

"Ti risvegli in una bella e soleggiata mattina di agosto ma, contrariamente a quanto ti saresti aspettato, non ti trovi nella tua camera da letto. I tuoi ultimi ricordi della sera prima sono di uno straniero con una folta barba che ti passa avanti mentre tornavi a casa dopo aver fatto la spesa settimanale al supermercato. Poi un forte colpo alla nuca e ora ti ritrovi qui, all'aperto, in un luogo sconosciuto, con un bracciale d'argento all'avambraccio."

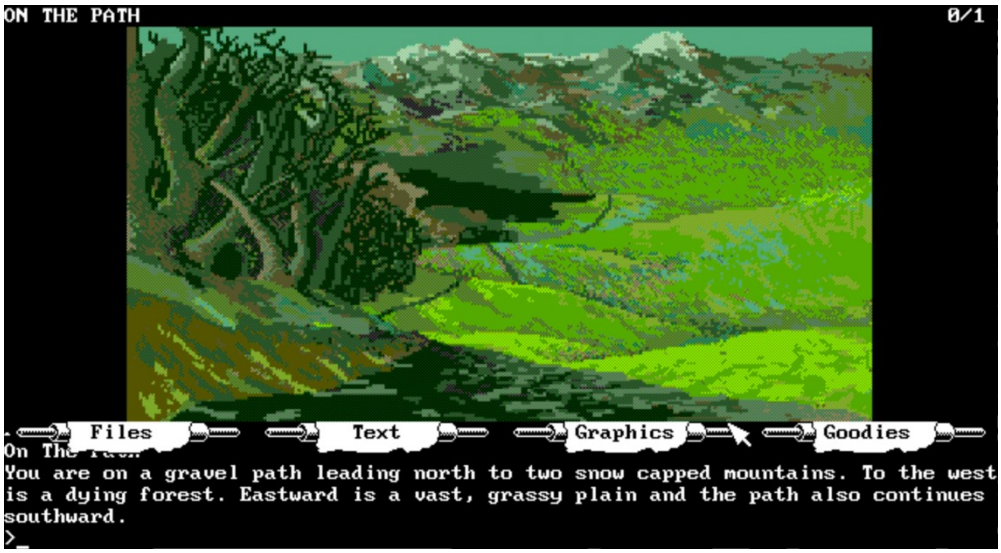
Inizia così *The Pawn*, il primo e più amato gioco della Magnetic Scrolls (il testo è un riassunto del paragrafo di apertura del gioco). Ideato e programmato da Rob Steggles, con il supporto di Steers e Gordon, durante le sue vacanze estive (Steegles aveva solo 18 anni



all'epoca), questa prima opera della Magnetic Scrolls è apprezzata per la sua storia divertente, ben costruita e raccontata, per l'uso di strabilianti (per l'epoca) schermate grafiche nel gioco e per un parser di altissima qualità, tranquillamente in grado di competere con quello della Infocom. L'impiego di immagini nelle avventure testuali fu subdolamente introdotto dalle case di produzione minori, nel tentativo di attrarre un maggior numero di giocatori che si sperava si lasciassero sedurre dalle illustrazioni e magari sorvolassero sulla qualità della trama e dei parser, il cui il livello della maggior parte di essi era sull'orridamente scarso, spesso al limite dell'irritante. Sia i ragazzi della Infocom, il cui motto era "un paragrafo vale mille immagini", che quelli della Magnetic Scrolls erano contrari all'uso della grafica nei loro giochi, convinti del fatto che un testo ben scritto e una robusta trama non avessero bisogno di ricorrere a questi mezzucci di bassa lega per stimolare i giocatori. Per la pubblicazione di *The Pawn* però la casa editrice, la Rainbird, pretese l'adozione di illustrazioni grafiche all'interno del gioco pena il mancato accordo sulla distribuzione. Benché fortemente contrariati da questo out-out, i ragazzi della Magneti Scrolls affidarono l'incarico a Geoff Quilley e quando il nostro dinamico trio ne vide i bozzetti mutò opinione circa l'uso delle immagini. Quilley sottopose alla Sinclair, Steers e Gordon dei disegni di incredibile qualità, magnificamente resi sui potenti Atari ST ed Amiga. La decisione fu presto presa: invece di utilizzarli in maniera funzionale per mostrare fedelmente le scene di gioco, compito lasciato al testo e all'immaginazione dei giocatori, essi vennero adottati per arricchire e valorizzare il prodotto finale alla stessa maniera in cui erano impiegate le illustrazioni nei grandi classici della narrativa, costituendo un gradevole divertissement per il lettore/giocatore, donandogli qualche istante di svago dall'impegno richiesto dall'avventura. La maestria di Quilley permise in seguito di adottare con successo le schermate grafiche anche sui microcomputer ad 8 bit su cui il gioco fu reso disponibile.

Ma non fu questo l'unico tocco di classe che contraddistinse *The Pawn* e in generale, tutti i giochi della casa londinese. Una confezione ben curata, una novella introduttiva scritta dalla Sinclair, un elegante uso di humor (british, of course) ed un originale sistema di aiuto contribuirono al successo del gioco, garantendogli un posto nell'olimpo delle più belle avventure testuali di sempre ed un Golden Joystick Awards assegnatogli nel 1986 per essere stato il miglior gioco d'avventura dell'anno.

Girovagando per le locazioni che costituiscono il discretamente esteso mondo di gioco e leggendo la novella presente nella scatola (cosa che consigliamo caldamente di fare prima di iniziare a giocare), scoprirete di essere finiti a Kerovnia, una terra fantastica appartenente ad una non meglio specificata dimensione e governata da Re Erik, in un momento particolarmente caldo dal punto di vista sociale. Ben presto vi renderete conto che realizzare il vostro desiderio di tornare a casa non sarà un'impresa facile. In *The Pawn* il pericolo è pane quotidiano e, sebbene la morte non vi attenderà dietro ogni angolo, le possibilità di finire all'altro mondo (no, non il vostro natio) sono abbastanza alte da consigliarvi di salvare il gioco con una certa frequenza. Muovere i primi passi non vi sarà difficile, sia perché la dimensione della mappa di gioco vi concederà un certo grado di libertà prima di imbattervi in un enigma da risolvere per accedere ad una nuova zona, sia perché i primi enigmi che incontrerete non saranno così difficili da superare. Nelle prime fasi di gioco, dopo circa mezz'ora, avrete già fatto conoscenza del mago mostrato sul frontespizio della scatola e del Re di Kerovnia, ma da qui in poi le cose inizieranno a farsi decisamente più complesse. Benché i puzzle abbiano sempre una soluzione logica, risolverli non sarà affatto banale: vi toccherà spremere per benino le meningi e trarre vantaggio dall'uso creativo del parser di gioco, a volte un tantinello pedante. Ad esempio (ALLERTA SPOILER!) nella ricerca di un modo per sbarazzarvi del bracciale, quando incontrerete il mago intento a svolazzare su una prateria, dopo averlo salutato e accettato l'incarico che vi offre, provate a MOSTRARGLI il bracciale ("show the wristband to kronos"); il mago vi dirà che è carino. Ma se gli CHIEDETE del bracciale ("ask kronos about the wristband"), la risposta sarà completamente diversa e decisamente più utile ai vostri scopi (o almeno così sembrerà). Perché porre un simile tranello al giocatore? E' chiaro che se si mostra il bracciale a qualcuno è per cercare di ottenere informazioni utili a capire perché ve lo ritrovate addosso e come fare per rimuoverlo, che bisogno c'era di complicare le cose ponendo l'attenzione a come se ne parla con gli altri personaggi? Bell'esercizio di stile



certo, utile a mostrare quanto sia intelligente il parser, ma sinceramente un'inutile complicazione a danno del giocatore a nostro avviso. Di situazioni simili ne troverete più d'una in quest'avventura, perciò non arrendetevi se le vostre istruzioni non otterranno il risultato sperato. Se avete un'intuizione che valutate fortemente plausibile, provate a spiegare al parser cosa intendete fare riformulando le frasi, potrebbe funzionare.

Aldilà di questi difetti di gioventù (stiamo sempre parlando di un'opera prima), il gioco è assolutamente godibile e coinvolgente e man mano che avanzerete nella storia, vi apparirà ben chiaro perché sia stato battezzato con un nome simile (the pawn, il pedone). Durante tutta l'avventura avrete la sensazione di essere manipolati per scopi ignoti, che diventerà una certezza con il procedere degli eventi.

La quantità di locazioni, oggetti e persone nonché la varietà delle azioni possibili vi permetteranno un elevato grado di libertà all'interno del mondo di gioco che vi darà modo di scoprire gli aspetti umoristici con cui l'autore ha permeato l'avventura. Per i giocatori esperti, che amano immergersi nell'universo creato dal programmatore piuttosto che limitarsi ad imbroccare le scelte giuste per far avanzare la storia fino al suo naturale epilogo, questo è un grande pregio; d'altro canto gli avventurieri meno smaliziati potrebbero restare disorientati da una tale ventaglio di possibilità ed avere non poche difficoltà ad arrivare alla fine del gioco. In generale, nel gioco c'è una grande attenzione ai dettagli, il che può portare a conseguenze diverse a seconda di cosa avete fatto in precedenza e per come l'avete spiegato al gioco. Il modo in cui parlate del bracciale a Kronos ne è un esempio, ma non è il solo caso. Se per qualche motivo (su cui non desideriamo affatto indagare) vi venisse in mente di girare in tenuta adamitica, potete farlo chiedendo al vostro alter ego di togliersi i vestiti e proseguire l'avventura in mutande (o almeno supponiamo che ne indossi), senza che nessun personaggio nel gioco si meravigli più di tanto per i vostri gusti in fatto di abbigliamento. Se per sventura però dimenticate che state andando in giro vestiti solo della vostra nuda pelle e provate ad attraversare le montagne innevate, ben presto la morte per assideramento vi coglierà, a meno che non vi rivestiate velocemente. Questo tipo di attenzione del gioco per lo stato delle cose e per come vi relazionate con i personaggi riesce ad ampliare magnificamente la gamma delle azioni da tentare ed i modi in cui risolvere gli enigmi, aumentando la profondità e la longevità del gioco, fattore non facile da sviluppare in un'avventura testuale.

Conclusioni

The Pawn è un classico della narrativa interattiva ed uno dei migliori esponenti del genere. Nonostante non sia un titolo adatto ai principianti o a chi si avvicina alle avventure testuali per la prima volta, merita di essere giocato da tutti, anche superando la difficoltà linguistica (è disponibile solo in inglese), aspetto che tutto sommato potrebbe non essere considerato un difetto. Certo, se non avete una buona dimestichezza con la lingua d'Albione, la difficoltà complessiva del gioco aumenterà, ma d'altro canto questo può rappresentare un ottimo stimolo per incrementare il vostro livello di inglese, sia nel vocabolario che nella grammatica. Tanti giocatori hanno imparato o migliorato il proprio inglese grazie alle IF (chi vi scrive in primis), trainati dalla volontà di giocare e divertirsi con questo tipo di opera ludica. Ma skill linguistici a parte, molti sono gli aspetti che un neofita, soprattutto se ha meno di 30 anni, può scoprire ed apprezzare. In primis la difficoltà del gioco, legata all'esercizio intellettivo che l'avventura richiede, non utilizzata in maniera artificiosa, tanto per allungarne la longevità, ma ben strutturata e parte integrante dell'esperienza e dello sviluppo della trama. L'ampio respiro delle possibilità offerte dal software e la capacità di profondo coinvolgimento che un titolo praticamente privo di grafica, audio, video ed azione riesce a dare sono altri fattori che possono sorprendere chi è abituato a vedere animazioni spettacolari ed affrontare un gioco grazie più alla propria abilità manuale che all'uso del ragionamento e della pensiero laterale. La capacità di *The Pawn* di intrattenere e coinvolgere giocatori ancora oggi, dopo più di 30 anni, è un ottimo esempio dell'universalità del videogioco e già solo per questo andrebbe provato.

Bene, se tutto questo pistolotto avesse convinto qualcuno dei lettori a tentare di cimentarsi con questo titolo, siamo lieti di informarvi che il gioco è ancora in vendita, insieme agli altri titoli della Magnetic Scrolls e disponibile anche per piattaforme mobili come iOS ed Android. E' persino possibile giocarlo (legalmente) on-line grazie ad un interprete scritto in javascript, capace di girare in un moderno browser web, reperibile all'indirizzo: https://msmemorial.if-legends.org/msa2/msa2.html

Buona avventura a tutti!

Giocabilità 95%

Una tastiera e la vostra immaginazione è tutto quello che vi occorre per giocare (e la capacità di comprensione della lingua inglese, ovviamente). *The Pawn* è allo stesso tempo divertente, impegnativo, stimolante e coinvolgente. Una volta iniziato a giocare, sarà difficile smettere, trascinati dalla voglia di vedere come va a finire.

Longevità ??

E' un'avventura testuale e sapete come la pensiamo sulla longevità di un'opera di questo tipo. *The Pawn* però possiede in qualche modo la capacità di non essere sempre esattamente uguale a se stesso e qualcosa di nuovo viene fuori ad ogni partita. Scoprire tutti i dettagli del gioco vi richiederà molto più tempo di quello necessario a finirlo (che già non è poco).

JACK THE NIPPER II – COCONUTS CAPERS

La schermata iniziale

Ecco la schermata introduttiva del gioco, dove campeggia, tra gli altri, il nome di **Ben Daglish**, grande compositore di musiche per i nostri amati 8bit, che ci ha lasciato purtroppo di recente

"Pierino" all'opera...

Qui possiamo apprezzare la qualità grafica del gioco, i suoi colori, i dettagli assai curati, nonché il famigerato "Cattivometro"!

GIUDIZIO SUL GIOCO

GIOCABILITA'

90%

Il gioco veloce, il personaggio reattivo e l'ambiente vasto rendono l'esperienza di gioco estremamente stimolante e divertente, provare per credere.

LONGEVITA'

90%

La mappa veramente immensa, la libertà di azione e la vasta gamma di attività durante il gioco vi faranno stare incollati a lungo al joystick, senza alcun dubbio.

Jack the nipper II – Coconuts capers

Gremlin Graphics sofware Ltd. - Anno 1987 - Piattaforma C64

Il venerdì era il giorno in cui, tornando a casa da scuola, facevo un giro un pò più largo per passare all'edicola del mio quartiere e porre al giornalaio la fatidica domanda: "Buongiorno, è uscita la cassetta per il Commodore 64 ? " Il giornalaio si voltava, frugava per qualche istante dentro uno scaffale e finalmente mi consegnava la tanto desiderata rivista!

Beh, da quel preciso momento iniziava per me un nuovo sogno, a cadenza settimanale !! Eh si perché già alla prima lettura dei titoli e successivamente dalla presentazione e dalle didascalie all'interno del fascicolo, la fantasia prendeva il sopravvento ed iniziavo a fantasticare su quanto quei giochi sarebbero stati divertenti e a quanto svago mi avrebbero assicurato. Alle volte il divertimento aveva un lasso di tempo talmente breve da durare il solo tempo del caricamento del gioco altre, invece, mi divertono ancora oggi!

Tornai quindi frettolosamente a casa per verificare il contenuto del fascicolo. Si trattava del numero 2 di Game 2000, (che uscì esattamente 31 anni fa, ovvero nel Gennaio 1988, e conteneva oltre a "Pierino" altri ottimi titoli, tutti rigiocabili con piacere anche oggi, come : Freddy Hardest, Histerya e Star Paws). Erano contenuti 8 giochi e ce n'era per tutti i gusti, tra questi ne compariva uno dal titolo che mi lasciò un po' incuriosito: "Pierino la peste".

Quando venivo in possesso di una nuova cassetta sceglievo i giochi che dal titolo e dalle note a corredo mi stuzzicavano di più e iniziavo a giocare proprio da quelli. I giochi che venivano prima li caricavo, mi appuntavo i numeri del registratore e riavviavo il Commodore fino ad arrivare al gioco prescelto, solo più tardi, una volta giocati quelli che immaginavo essere i TOP game, riavvolgevo il nastro per provare gli altri e devo dire ogni tanto ci azzeccavo nella selezione dei migliori. "Pierino la peste" richiedeva ben 77 interminabili giri di registratore, sufficienti a fare una veloce merenda e sosta in bagno, il fatto di impiegare tanto tempo faceva ben sperare!

Una volta terminato il caricamento il gioco si presentava con una schermata statica che riportava i nomi dei realizzatori e la Software House (la "Gremlin Graphics") con una musica di fondo di Ben Daglish azzeccata al nome del gioco !! Nella parte in basso compariva invece in una cornice di pietra la scritta "Cattivometro".

Joystick in porta 2 e finalmente sono pronto a giocare! Il primo impatto è spettacolare, la grafica originale, ricercata e con elementi e dettagli che cambiano frequentemente. Pierino è un bambinone che indossa ancora il pannolino (all'occorrenza anche paracadute) che si trova nel bel mezzo della foresta Australiana, dove dovrà girovagare tra templi, ponti tibetani e sotterranei, circondato da aborigeni, alligatori, vespe, leoni, elefanti, gufi, esploratori e perfino Tarzan! Iniziando il gioco la musica cambia, alla sbarazzina sigla iniziale prende posto un tappeto di percussioni tribali che fa calare perfettamente il ragazzo tredicenne di ieri (e l'adulto quarantaquattrenne di oggi) nell'ambientazione del gioco. Inizio così a vagare senza metà nella foresta e ben presto mi rendo conto che la mappa è veramente immensa!

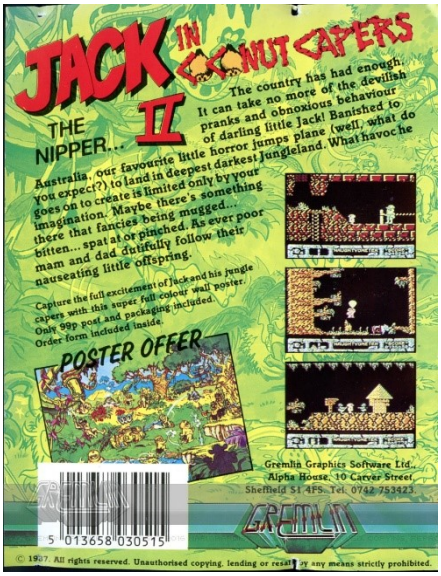
Il gioco è un platform avventuroso a scorrimento multidirezionale, si può praticamente fare di tutto, salire e scendere da liane e scalini, saltare su sabbie mobili, percorrere con un vagone i binari di una vecchia miniera e molto altro.

Credo all'epoca di aver passato oltre un mese a giocare esclusivamente a questo gioco senza riuscirne a capire il vero scopo ma tanto era bella e curata la grafica e tante le cose che si potevano scoprire girovagando che mi divertivo un mondo anche così.

Persino rimanere fermi aveva una particolarità, Pierino infatti dopo qualche secondo si guardava intorno e ingannava il tempo con gesti poco eleganti tipo dita nel naso e gesto delle corna ... non avevo mai visto cose del genere in un videogame.

Nel gioco è possibile recuperare diversi tipi di armi come sassi e frecce e di tanto in tanto si trovano altri oggetti da prendere, tra questi ci sono particolari barattoli di grease (si di grease! ovvero grasso che una volta recuperato fa partire un jingle tratto dal musical Grease) e poi caramelle, dinamite, topolini ecc che dovevano avere a che fare con il gioco ma in che modo? Nella presentazione del gioco veniva suggerito di compiere dei dispetti ma come e a chi ? Con le armi che si raccoglievano ci si poteva fare strada fra gli aborigeni aumentando il punteggio ma era ovvio che lo scopo del gioco doveva essere altro.

E poi quella scritta Cattivometro... non riuscivo proprio a dargli un significato, fino a quando, un bel giorno, ho scoperto su ZZap il vero nome di "Pierino la peste" ovvero "Jack The Nipper II : Coconut capers", con tanto di mappa del gioco, significato ed uso degli oggetti... e in quel momento, come Belushi, ho esclamato "Ho visto la luce !!!!"



Arrivato a casa, questa volta dal lavoro, apro il Vice e trascino il file .Tap (posseggo ancora il Commodore con relativa cassetta, è il tempo che è risicato) inizio la partita, vado a prendere il famoso barattolo di grease e finalmente ... Tarzan scivola giù dalla liana che è una bellezza !! Mi fermo e un sorriso si stampa sul mio viso, ripensando al ragazzo che si divertiva a prendere quel barattolo per ascoltare il jingle Grease ☺.

Per completare il gioco bisogna raccogliere alcuni oggetti e compiere azioni da birbantelli (lanciare appunto il grasso a Tarzan che, arrampicato su una liana, scivolerà giù, portare un topolino vicino ad un elefante per terrorizzarlo al punto di farlo finire sopra un albero, far cadere in un fossato un aborigeno, ecc,). Tutto ciò fa incrementare il famoso "Cattivometro" che una volta arrivato a fondo scala, ci consente di recarci nel tempio finale e finire il gioco.

Una menzione speciale va a Ben Daglish, che è stato uno dei più importanti compositori del C64 e non solo. Sue sono le colonne sonore di giochi come: The Last Ninja, Cobra, Return of the Mutant Camels e molti altri. Nei videogames, come nei film, se un'opera riesce bene è anche grazie alla componente musicale, anzi a volte è la musica stessa trainante per il gioco.

A presto!

di Alessandro Paloni

Collegamenti

Pagina su zzap
https://www.zzap.it/mappe/jackthenipper2

Pagina su c64 wiki
https://www.c64-wiki.com/wiki/Jack_the_Nipper_II_%E2%80%93_Coconut_Capers#Trick_1_.E2.80.93_the_grease

Pagina su Specialprogramsipe (Sovox)
http://specialprogramsipe.altervista.org/gioco.php?collana=game_2000_nuova_serie&console=c64&codice=pierino_la_peste&id_rivista=2

INSPECTEUR Z

Ecco il fiero e infallibile Buru, ovvero l'Ispettore "Z"!

Buru e Marty che si accingono a entrare nel covo dei mafiosi.

Ecco Marty che trae in salvo il suo superiore, ancora una vittoria per l'Ispettore "Z".

GIUDIZIO SUL GIOCO

GIOCABILITA'

95%

Tra i titoli arcade MSX più immediati e divertenti, dove grafica e gameplay si combinano in un prodotto di alto livello nonostante l'appartenenza alla cosiddetta categoria "budget".

LONGEVITA'

80%

La difficoltà iniziale e qualche incertezza nelle collisioni potrebbero in principio scoraggiare, ma una volta entrati nella meccanica di gioco si sarà portati a completare tutti e cinque i livelli.

Inspecteur Z (Buru to Marty Kikiippatsu)

HAL - Anno 1986 - Piattaforma MSX

Avete mai immaginato che per sgominare un'intera banda di malfattori o associazione a delinquere basterebbe far saltare in aria l'intera loro sede? E' l'idea su cui si basa il gioco di cui vi parlerò.

Si chiama Inspecteur Z (Buru to Marty Kikiippatsu), prodotto nel 1986 per lo standard MSX di prima generazione dalla leggendaria giapponese HAL Lab., la rinomata software house che orientava le proprie risorse su molti sistemi senza mai legarsi a qualcuno e di cui anche il Commodore 64 e il Vic20 hanno goduto delle produzioni (Le Mans, Star Battle, Radar Rat Race), dissolta poi dall'avvento dei Pokemon sul settore ludico ai quali ha rivolto il suo futuro.

Il gioco pone ai controlli dell'Ispettore Buru denominato "Z" il quale dovrà infiltrarsi nel palazzo dove risiede l'organizzazione mafiosa a cui da la caccia per sgominarli definitivamente, ma per stanarli dal loro nascondiglio ha escogitato un sistema tanto efficace quanto azzardato, piazzare esplosivi per tutto il palazzo e attivarne l'innesco, cercando ovviamente di fuggire in tempo prima di venire coinvolto nell'esplosione.

Per fare questo il dobbiamo guidare il nostro "Z" all'interno del palazzo e raccogliere tutte le bombe disseminate un po' a caso, per poi piazzarle nei punti chiave indicati sulla parete con dei riquadri tratteggiati rappresentanti le bombe. Non è necessario trovare prima tutte le bombe, man mano che si raccolgono possono poi essere piazzate a seconda delle nostre esigenze del momento, l'importante è comunque sistemare tutte quelle esistenti nel palazzo in ogni punto indicato. A complicare la vita vi saranno gli scagnozzi del boss che si accorgeranno della presenza di "Z" e usciranno dalle loro stanze per farcirlo di piombo, ma non hanno fatto i conti con l'infallibile mira di Buru che con grande destrezza può difendersi con la sua fedele pistola e colpire tutti i nemici che lo attaccano. Oltre ai vari guerci altri ostacoli possono compromettere la missione, come pavimenti cedevoli che potranno far precipitare "Z" al piano inferiore o peggio crollargli addosso, o come le enormi pale dei ventilatori accesi sul soffitto e sparsi lungo i corridoi della struttura, il cui contatto costerà la vita.

Durante la ricerca e il piazzamento delle bombe è possibile trovare delle monete che potranno essere utili con la slot machine presente in uno dei corridoi e che se la fortuna vuole verranno concessi punti e invulnerabilità temporanea.

Una volta completata la missione e posizionato tutti gli ordigni, si dovrà cercare in fretta l'uscita che condurrà l'ispettore sul tetto dove lo attende Marty, il suo fedele assistente, che lo trarrà in salvo a bordo di un elicottero.

La struttura del gioco è in realtà semplice, ma non è facile eliminare tutti i nemici, evitare le trappole e piazzare le bombe senza rischiare ogni momento di perdere una vita. Il gameplay è come sempre equilibrato e i controlli sono perfetti, come tutti i giochi di produzione Hal, qualche incertezza sulle collisioni nei punti di connessione tra le scale e le piattaforme, ma nulla a cui non ci si abitua e adattarsi.

La grafica è molto simpatica, colorata e definita, in stile cartonesco con cani umanizzati come personaggi, infatti Buru dalla pronuncia giapponese di "bul" definisce l'ispettore come un buldog, un ottimo uso degli accostamenti cromatici, manca uno scroll video progressivo, presente solo nel passaggio da una stanza all'altra, gli sprite sono davvero ben fatti, con un design tipico anime giapponese, leggero flikering quando troppi personaggi si trovano sulla stella linea per il consueto limite dei quattro sprite in linea di cui la VDP dell'MSX1 soffre, tutto sommato abbastanza sopportabile.

Le colonne sonore sono perfettamente in tema poliziesco investigativo, con un sound incalzante ed effetti sonori nella media.

Non esistono multi caricamenti come la buona tradizione ludica MSX vuole, con ben cinque livelli da esplorare uno più difficile dell'altro.

Un prodotto semplice ed immediato che rientra nel tipico palinsesto ludico arcade giapponese e quindi rendendo bene l'idea su quale contesto e supporto poteva contare lo standard MSX..

di Francesco Gekido Ken Ugga
della rivista Re.BIT
www.rebitmagazine.it

EUROSTRIKER

GIUDIZIO SUL GIOCO

GIOCABILITA'

75%

L'immersione nell'atmosfera del campionato a squadre è pressoché immediata. E si resta attaccati allo schermo almeno per tutta la prima stagione, cercando di raggiungere la prima promozione. La fase clou del gioco in cui bisogna sfruttare al meglio le occasioni sotto porta è ben animata graficamente e la meccanica di tiro, ben realizzata, consente di sperimentare e migliorare col susseguirsi delle partite.

LONGEVITA'

70%

Eurostriker richiama alla memoria molti titoli che, dopo gli albori di Football Manager, hanno cercato negli anni di conquistare i cuori degli appassionati di calcio digitale. Il fatto che non vi siano molte opzioni per dirigere ed allenare tutta la squadra ma soltanto l'attaccante in grado di portare il club al successo, toglie un po' di longevità rispetto ai titoli manageriali puri.

Eurostriker

Homebrew - Anno 2018 - Piattaforma ZX Spectrum

Pur essendo sempre stato un felice possessore di home computer fin dai tempi del primo (un C64), non sono mai stato un videogiocatore accanito prediligendo spesso la parte didattica e di programmazione a quella puramente ludica. I miei generi preferiti sono sempre stati quelli con una maggiore attinenza alla realtà (simulatori e puzzle games), ma soprattutto ho sempre avuto un debole per i titoli di simulazione sportiva, sia manageriali che d'azione. E da sempre, tempo permettendo, dò uno sguardo interessato ai titoli (anche quelli nuovi e non legati alle piattaforme retro) che trattano degli sport più diffusi e conosciuti, basket e calcio fra tutti. Da quando coltivo l'hobby del retrocomputing ho potuto sperimentare di persona i vecchi titoli sportivi usciti per le varie piattaforme a 8 e 16 bit e ho notato come anche i giochi homebrew di recente produzione, di tanto in tanto propongono titoli che hanno come sfondo campi e stadi sportivi.

È questo il caso di Eurostriker, un gioco uscito dalle mani di un programmatore portoghese, Valdir, una vecchia conoscenza del mondo dei titoli homebrew per ZX Spectrum, già autore di due titoli recenti: ZX Nights e ZX Striker. Quest'ultimo, un mix di azione e strategia calcistica, è il gioco da cui trae spunto EuroStriker, ma a differenza di ZX Striker, con il quale si può giocare fino a 16 in contemporanea, solo un giocatore è consentito durante la sessione di gioco. Ma non si può avere tutto e le buone caratteristiche di Eurostriker compensano ampiamente questo divario.

Chi conosce ZX Striker non sarà sorpreso dalla meccanica di gioco, almeno per quanto riguarda della fase di gioco in cui bisogna cercare di mettere a segno il maggior numero di reti (fig. 4). In Eurostriker assumiamo il ruolo di capocannoniere per una squadra che milita in un immaginario campionato europeo per club. Si parte dalla quarta divisione e lo scopo è ovviamente quello di scalare le posizioni in classifica, match dopo match, per raggiungere la promozione e passare alla serie maggiore. Le somiglianze con il più famoso Footballer Of The Year, uscito nel 1986, sono molte, perché segnare tanti gol e sprecare il meno possibile aiuta la nostra squadra a vincere le partite e ad aumentare il morale del team. Obbiettivo ultimo: scalare la classifiche minori, raggiungere la prima divisione e vincere il campionato. Ma non sarà facile, perché dovremo scontrarci via via con tutte le squadre più forti e blasonate d'Europa, fino a sfidare le formazioni più forti della Champions League, come Juventus, Bayern Monaco, Barcellona, Real Madrid, Paris Saint-Germain, Manchester United, Liverpool e Chelsea. All'inizio del gioco abbiamo la possibilità di scegliere il nome del top scorer che andremo ad impersonare. In seguito viene mostrato il menu principale delle opzioni di gioco: si può accedere ai dati e alle statistiche della nostra squadra e del nostro capocannoniere, c'è la possibilità di allenare e migliorare le nostre abilità nel segnare gol, verificare la posizione attuale della squadra nel campionato (e la classifica delle altre divisioni), visualizzare l'elenco delle partite in programma nel campionato e in coppa (sì, abbastanza incredibilmente, c'è anche questa competizione), scorrere la lista dei migliori marcatori e, infine, procedere con il gioco ed affrontare le partite successive. La sensazione è che la struttura dei campionati provenga da un gioco manageriale "classico", con divisioni a 16 squadre, tipiche di una lega europea degli anni '80. Il programmatore ha scelto di mescolare le carte e proporre 4 divisioni piene zeppe di squadre di ogni parte d'Europa, che oggi normalmente accedono alla Champions o Europa League.

Andando avanti con il campionato, sullo schermo appare la presentazione del match che la nostra squadra dovrà affrontare. Il risultato finale di ciascuna partita è legato al livello tecnico raggiunto dal nostro team e allo stato del morale di squadra (che naturalmente aumenta o diminuisce a seconda dei risultati). Durante la partita ogni tanto si presenta un'opportunità in attacco ed è qui che entra in gioco il nostro apporto. Poiché non si tratta di un gioco basato sulla pura strategia sportiva, ma piuttosto di un gioco che richiede abilità nel realizzare le opportunità che si presentano, lo svolgimento di un campionato è relativamente veloce e coinvolgente.

La fase di gioco in cui bisogna cogliere le occasioni in attacco presenta una grafica essenziale ma ben animata. Il campo di gioco si limita ad inquadrare l'area di rigore avversaria, il nostro capocannoniere, il portiere della squadra avversaria ed un cursore in movimento nei pressi della palla, che rappresenta la direzione di tiro. Un contatore decrescente, sempre accanto al giocatore, misura il tempo rimasto per tirare in porta. Basta usare i tasti cursore (o i tasti WASD) per effettuare il tiro, imprimere effetti e più o meno potenza (influenzando anche l'altezza del tiro). In ogni caso, la pratica vi porterà a migliorare la tecnica ed il tempismo e vi avvicinerà alla perfezione. Ed è proprio quello di cui avremo bisogno per iniziare a vincere giochi ed acquistare prestigio per noi stessi e per tutta la squadra.

Eurostriker, nella sua immediatezza e intuitività, coinvolge fin da subito, per la possibilità di vedere i propri progressi aumentare con il nostro impegno, per il fascino di partecipare e di completare (si spera vincenti) una stagione in modo rapido, e, soprattutto, perché è uno di quei giochi nei quali non ci si accorge neppure del tempo che passa. Possiamo garantire che non rimarrai deluso.

Eurostriker è un titolo distribuito gratuitamente dal suo autore e può essere ottenuto all'indirizzo web:
https://www.dropbox.com/s/m7vetx4oolqtoxp/Estriker.tap?dl=o

di David La Monaca/Cercamon

MENACE

Questo è il titolo che appare all'inizio del gioco, già dal solo titolo si evidenziano colori che non appartengono alla palette MSX1.

Anche dalle schermate della storia si evidenziano ben più di 16 colori...Ma non è il caso di indugiare, la nostra colonia su Marte è stata attaccata da forze sconosciute.

GIUDIZIO SUL GIOCO

GIOCABILITA'

100%

Si tratta di un remake perfettamente riuscito di Galaxy Wars della TAITO, nulla più, nulla meno.

LONGEVITA'

80%

Sarete tentati dal finire tutti i dieci livelli ma successivamente vi rivolgerete ad altro.

Menace

Homebrew - Anno 2009 - Piattaforma MSX

Torniamo a parlare ancora di MSX e del suo settore ludico, ma questa volta sotto un profilo tendenzialmente più tecnico, che pone come contesto le capacità grafiche a molti ancora sconosciute, riguardanti la prima generazione di questo meraviglioso standard.

Negli anni '80 si è erroneamente usato il settore ludico come metro di misura delle prestazioni di un sistema informatico e quando si parlava di MSX di prima generazione (MSX 1), saltavano fuori tipici luoghi comuni come la pessima gestione degli sprite o dello scrolling video; tutta colpa del generoso ma datato TMS9918, la VDP che nel 1978 fu una rivoluzione in campo grafico, tant'è che in origine era di uso militare. Presentando 16kbyte di VideoRAM dedicata, sprite, grafica multicolore e persino capacità di superimpose per le sovrapposizioni video (sì, avete capito bene), questa VDP infatti mostrò il meglio di sé su macchine come il TI99/4A della stessa Texas Instruments e sulla leggendaria console Colecovision, di cui tutti ricordano le performance dei suoi videogame molto simili agli arcade da sala giochi che circolavano all'epoca. Col tempo però queste prestazioni iniziarono a soffrire di vecchiaia, ma era ancora considerata una VDP economica e allo stesso tempo all'avanguardia grazie alla gestione VRAM separata, tanto che venne utilizzata per il progetto MSX. Tuttavia nonostante i limiti di cui sopra, programmatori professionisti (per lo più giapponesi) seppero sfruttare al meglio ogni singola caratteristica del sistema, producendo videogiochi che hanno fatto la storia, ma che per i più critici mettevano ancora una volta in evidenza i famosi limiti grafici della macchina.

Oggi lo standard MSX è tutt'altro che morto e tra le varie produzioni recenti, più o meno regolari, esistono contest annuali che danno l'occasione a molti appassionati di affinare le tecniche di programmazione e sviluppare videogiochi dove i limiti tradizionali di sistema vengono abilmente compensati: ed è il caso di MENACE, vincitore del secondo posto all'MSX Dev' 2009.

Il gioco in se è palesemente ispirato allo storico arcade Galaxy Wars del 1979, sviluppato da TAITO, nel quale il giocatore affronta orde di alieni mediante missili teleguidati. Il missile partiva da una postazione mobile posizionata in basso allo schermo che permetteva di allineare il missile nel punto desiderato e poi lanciarlo verso gli alieni che occupano le ultime due file in alto allo schermo. Durante la guida del missile bisogna accuratamente evitare gli asteroidi che vagano al centro dello schermo che al contatto farebbero esplodere il missile. La guida del razzo prevede il lancio dalla rampa mobile, lo spostamento laterale e l'avanzamento potenziato grazie al propulsore che viene azionato in qualsiasi momento dal giocatore, in questo modo è possibile evitare lo sciame di asteroidi vaganti e i proiettili che gli alieni lanciano verso la nostra base. I nemici da eliminare per ogni livello sono otto, suddivise in due file da quattro che si muovono lateralmente in direzioni opposte, la cui velocità di spostamento aumenta nel momento in cui restano in totale solo quattro nemici.

Non è presente un numero definito di livelli, ad ogni completamento aumenta gradualmente la velocità degli alieni e il numero degli asteroidi da cinque iniziali diventano anche dieci, dopodiché anche la velocità di questi ultimi inizierà a progredire fino ad un punto di non gestione del gioco, in quanto troppo veloce da non poter garantire un gameplay equilibrato.

La grafica del primo Galaxy Wars si presenta in bianco e nero, ma a molti cabinati, come in Space Invaders, veniva applicata sullo schermo dei lucidi trasparenti che aderendo alla superficie di vetro suddivideva varie zone in diversi colori, un artefatto che rendeva il gioco più accattivante.

Per quanto riguarda l'audio, erano presenti pochi effetti sonori dedicati soprattutto al rumore del razzo propulsore ed alle esplosioni dei nemici o del razzo stesso.

Tre vite in tutto, tanta abilità e fantasia per un gioco che impegnava e divertiva ogni tipo di videogiocatore.

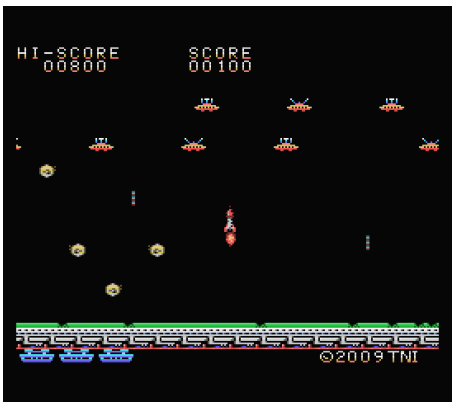
Bene, a questo punto torniamo al nostro Menace, che per quanto riguarda il gameplay non vi è molto da aggiungere alla descrizione di Galaxy Wars, se non per una diversa configurazione degli asteroidi a centro schermo ma tutto sommato rimanendo nei canoni, dedicando però una particolare attenzione alla grafica e contrariamente all'arcade Taito, vengono proposti dieci livelli.

Molti accusano l'MSX e tutte le macchine dotate del TMS9918, di una pessima gestione degli sprite, non tanto per la definizione di 16x16 pixel, ma per l'assegnazione di un singolo colore ciascuno, del limite di quattro sprite visualizzabili sulla stessa linea e dell'unico rilevatore di collisioni hardware relativo a tutti e 32 sprite disponibili su schermo. Dei suddetti limiti il più grande è quello della visualizzazione orizzontale, infatti se sulla stessa linea raster vengono posizionati più di quattro sprite, quello con priorità di visualizzazione più basso viene oscurato.

I sistemi di difesa sono pronti e ubicati in tutto il mondo!

Il governo ci concede l'uso dell'arma più potente! Ancora una volta si notano colori aggiuntivi alla palette base, come le quattro tonalità di grigio quando la palette ne prevede uno solo.

Ecco le orde di alieni in alto.



Missile lanciato!

Esiste comunque un registro nella VDP che genera un multiplex automatico, ovvero dal quinto sprite in poi sulla stessa linea raster, alterna "l'accensione" di quelli precedenti generando però uno sfarfallio chiamato flickering, effetto spesso fastidioso che fortunatamente si interrompe nel momento i cui gli sprite allineati tornano ad essere non più di quattro. È chiaro che detto limite incide anche sulla necessità di avere un oggetto multicolore sovrapponendo più sprite, creando quindi non pochi problemi ad uno sviluppatore grafico.

Tutto quanto descritto però, non riguarda il gioco Menace, nel quale a differenza di Galaxy Wars, non solo esistono sprite multicolore visualizzandone anche più di quattro sulla stessa linea orizzontale, ma addirittura il gioco prevede ben cinquanta colori circa a fronte dei sedici della palette base disponibile.

Come è possibile tutto questo? Poco alla volta ci arriviamo.

Ad una gestione sprite così limitata, si contrappone un sistema grafico molto più efficiente: iniziamo col dire che la matrice grafica dello schermo è di 256x192 pixel suddivisi in 768 celle carattere da 8x8 pixel l'una, ogni carattere è ridefinibile indipendentemente dagli altri direttamente nella VRAM mappata su tre set di caratteri da 256 l'uno, prevedendo un sistema multicolor che permetteva per ogni cella carattere di avere due colori per ogni fila di 8x1 pixel. In questo modo era persino possibile visualizzare tutti e 16 i colori della palette in un unico carattere, un sistema che non rendeva facile il lavoro del grafico ma che permetteva di non avere ulteriori legami con la palette stessa, come invece avveniva in altri sistemi dell'epoca, nei quali spesso alcuni dei colori scelti dovevano essere comuni al resto del set caratteri. Questo sistema però, è stato considerato anche una soluzione ottimale alla ridotta gestione degli sprite, permettendo di poter generare tile come sprite software di qualsiasi grandezza e avvalendosi del multicolor che grazie alla gestione indipendente della VRAM sulla mappa caratteri e del bus dedicato alla VDP, potevano essere mossi con una velocità impressionante senza appesantire particolarmente la CPU. Il movimento di questi tile avveniva tramite le celle carattere, risentendone un po' sulla fluidità, ma se programmati ad una certa velocità l'effetto "scattoso" era quasi impercettibile.

Tuttavia questo sistema ha avuto una sua ulteriore evoluzione, generando sprite software che riescono a muoversi anche di un pixel alla volta e quindi in modo fluido e sempre multicolore. Come avviene? Un semplice trucco, ogni cella occupata dal disegno dell'oggetto o parte di esso, viene fatto "shiftare" ovvero traslare all'interno stesso della sua griglia di definizione un pixel alla volta, riportando i pixel "uscenti" nella cella carattere successiva, che si trovi in alto o di lato.

È proprio quello che avviene in Menace per MSX1, dove tutti gli sprite, ad eccezione dei proiettili alieni, sono software e multicolor. Grazie ancora una volta all'accesso indipendente alla VRAM, questa operazione può avvenire senza che il sistema risenta di rallentamenti, restituendo un framerate più che sufficiente per garantire il gameplay, ma le sorprese non finiscono qui: grazie al clock sincrono dello Z80 e all'aggiornamento raster del video obbligato a 50/60hz, è possibile usare un sistema interlacciato dei colori, la cui sovrapposizione di alcune delle tonalità base, ha consentito di generare all'occhio umano un numero totale di cromie di circa 50 tonalità con immagini abbastanza stabili, che è possibile apprezzare sia nelle schermate di introduzione demo che durante il gioco. Il tutto senza compromettere in alcun modo l'efficienza del gameplay e la velocità di gioco, con un risultato finale di gran lunga migliore del cabinato storico Taito.

Il gioco è freeware, in quanto voleva essere solo un esempio di cosa si puo' fare per ovviare ad alcuni limiti imposti dall'hardware nello sviluppo di videogiochi sullo standard MSX di prima generazione.

È possibile scaricarlo dal sito ufficiale del contest MSX Dev' oppure dal sito ufficiale della TNI ai seguenti link:

https://www.msxdev.org/msxdev-archive/msxdev09/

http://www.tni.nl/products/menace.html

Buon divertimento e ricordate, "MSX, NO LIMITS!"

di Francesco Gekido Ken Ugga
della rivista Re.BIT
www.rebitmagazine.it

Angolo Oscurita' - Burnin' Rubber

Ocean - Anno 1990 – Piattaforma Amstrad CPC Plus

BURNIN' RUBBER

Siamo nella seconda metà degli anni '80.

La Amstrad di Alan Sugar ha riscosso un buon successo con il "CPC", una linea di computer ad 8 bit introdotta nel 1984. Pur affrontando una spietata concorrenza, le macchine vendono bene sia tra gli appassionati di videogiochi, sia nel settore lavorativo, vista la compatibilità con il sistema operativo CP/M.

Nel tentativo di dare ancora un po' d'ossigeno alla serie CPC, Amstrad decide, nel 1990, di rilasciare un trio di nuove macchine con caratteristiche inedite. Nasce così la linea "Plus", che è formata da due computer (464 Plus e 6128 Plus) e da una console (GX4000).

La serie Plus sembra proprio indirizzata verso i videogiocatori, dato che propone caratteristiche molto avanzate per un sistema ad 8 bit. Tra le varie possiamo citare ad esempio il supporto hardware per gli sprites, il soft scrolling, 32 colori contemporaneamente su schermo da una tavolozza di 4096 (col CPC standard erano 16 da un massimo di 27), un chip audio che sfrutta l'accesso diretto alla memoria (DMA) ed una porta cartuccia. Da citare anche la forma molto più accattivante dei computer e la strana ma interessante scelta per il design del GX4000, che appare simile ad un'astronave.

Ma come dimostrare le caratteristiche avanzate di queste macchine? La risposta arriva proprio con "Burnin' Rubber", un gioco di corse della Ocean che viene scelto per essere venduto in bundle con tutta la linea Plus.

La base del titolo nasce dal buon WEC Le Mans, uscito nel 1988, ma le migliorie sono tali che l'originale è difficilmente riconoscibile.

Graficamente il gioco fa davvero una bella impressione, soprattutto se pensiamo che stiamo parlando di un titolo ad 8 bit. Dopo una coreografica sequenza introduttiva, scenderemo in pista e fin da subito potremo notare la qualità estetica della nostra vettura, così come le altre che ci troveremo a superare. Gli effetti che ci accompagnano sono di buona qualità: dal fumo che esce durante le curve più secche alle scintille che schizzano quando abbiamo un contatto con un avversario. Impressionante poi è il ribaltamento del nostro bolide dopo uno scontro violento: ancora oggi fa impressione vederlo su una console di questo tipo. La stessa cura è riposta anche nei numerosi oggetti a bordo pista, compresi alberi e cartelloni pubblicitari e all'occasionale galleria da superare. Un altro aspetto degno di nota è lo scorrere del tempo. All'inizio la luce sarà forte e diffusa poi, con l'avvicinarsi del tramonto, tutto assumerà contorni caldi ed accesi. Passeremo poi alla notte e all'alba, in maniera estremamente convincente.

Il tutto è accompagnato da uno scrolling generalmente fluido, anche se non particolarmente veloce.

Il comparto audio presenta un'ottima composizione musicale nei titoli di testa e fin da subito potremo saggiare le capacità aggiuntive del processore sonoro. Durante il gioco dovremo invece accontentarci dei soli effetti; alcuni saranno molto convincenti (il rumore delle ruote che sterzano), mentre altri decisamente più fastidiosi (il rumore del motore dopo un po' diventa pesante da sopportare).

Burnin' Rubber è un gioco di corse: nei panni di un pilota, dovremo effettuare una gara lunga un intero giorno e riuscire ad arrivare al traguardo prima dello scadere del tempo.

All'inizio potremo far pratica effettuando un giro di qualificazione, prima di passare alla sfida vera e propria. Una volta raggiunta la gara, dovremo completare un determinato numero di giri della pista entro il tempo limite. A quel punto potremo dire di aver concluso il gioco, ma prima di vedere la schermata di premiazione ci vorrà un po' di tempo.

Abituarsi al sistema di controllo richiederà una certa pazienza, in quanto la risposta della macchina non sarà istantanea. Questo influenzerà in maniera decisa le nostre prime partite, ma superato il primo scoglio (a cui contribuisce anche il pad stesso del GX4000) arrivare a buoni risultati non sarà impossibile.

La durata del titolo si attesta su livelli non particolarmente alti, ma il livello di difficoltà e lo stile stesso del gioco porteranno senza dubbio a rigiocarlo una volta ogni tanto.

Tirando le somme, Burnin' Rubber è un buon titolo di lancio per la serie Plus e sicuramente mostra parte del potenziale di questi sfortunati sistemi.

Come vi potrete immaginare, nel 1990 il grande pubblico aveva gli occhi puntati altrove, sia in ambito computer (Amiga, Atari ST, PC) che console (Megadrive, Pc Engine e l'annunciato Super Famicom), per cui le tre macchine sono cadute rapidamente nell'oblio.

Fortunatamente in tempi recenti c'è stata una parziale riscoperta della linea Plus, perciò potremo magari finalmente vedere di cosa sono davvero capaci. Nel frattempo, se vi piacciono i sistemi oscuri, potete recuperare un GX4000 con Burnin' Rubber e dargli una possibilità.

Alla prossima.

di Federico Gori

GIUDIZIO SUL GIOCO

GIOCABILITA'

70%

Il sistema di controllo richiede un po' di tempo per essere padroneggiato, ma alla fine diverte.

LONGEVITA'

70%

La durata del gioco non è particolarmente consistente, ma prima di arrivare alla schermata di premiazione dovrete davvero sudare le proverbiali sette camicie.

RetroGiochiAmo: Rick Dangerous

di Daniele Brahimi

Anno nuovo, vita nuova... Ma buone vecchie abitudini! La nostra rubrica **RetroGiochiAmo** continua imperterrita grazie all'impegno del nostro affezionato lettore **Daniele Brahimi**.

Ormai considerarlo soltanto un lettore e' riduttivo. Vista la puntualita' con cui ci fa pervenire i suoi lavori ad ogni uscita della rivista, Daniele e' a da considerarsi a tutti gli effetti un **collaboratore esterno** di RetroMagazine.

Dopo essere sopravvissuti ancora una volta a queste feste un po' noiose ma anche allegre, sperando che molti di voi non abbiano aumentato il girovita, augurandovi un sereno e retrogiocoso 2019, ho pensato a lungo durante la sosta natalizia ad un giochino che come i precedenti abbia lasciato il segno durante la mia infanzia; Inaugurando il primo numero di retromagazine di questo nuovo anno con... Rick Dangerous!!!

Molti di voi lo conosceranno sicuramente, era il clone di Indiana Jones e forse anche il predecessore di Tomb Raider ed uscì per diverse piattaforme non solo nei negozi di videogiochi, bensì anche in molte cassette da edicola, di fatti lo conobbi grazie ad una cassetta che mi capitò tra le mani e il nome era Dan, il mio diminutivo tra l'altro.

La Musica di accompagnamento nel primo livello ci lasciava supporre che ci aspettava un'avventura impegnativa e piena di pericoli in varie località sperdute del mondo come una caverna di indios sudamericani, una piramide egizia ecc.Ogni livello era contraddistinto da nemici da evitare o abbattere con proiettili e dinamite, tranelli come spuntoni, massi ecc.

Ed il primo livello comincia proprio con un masso rotolante che ci insegue. Si hanno a disposizione sei vite per portare a termine tutti e quattro i livelli presenti nel gioco ed una quantità limitata di munizioni ed esplosivi, recuperabili durante il tragitto.

Il gioco potrebbe sembrare a primo impatto molto difficile da portare a termine non essendoci continue ma una volta presa la mano e memorizzato i livelli, sarà un gioco (in tutti i sensi) arrivare fino in fondo. Ingannare i nemici con i loro stessi tranelli sarà la parte più divertente del gioco e vi darà molte soddisfazioni credetemi, se non ci fossero probabilmente non sarei qui a parlarne.

Il gioco ebbe anche un seguito molto simile al precedente ma con ambientazioni più futuristiche e le recensioni non furono molto generose come con il primo capitolo ma se siete degli appassionati retrogamer ed in particolare utenti Commodore 64 come il sottoscritto, vorrete rigiocarli entrambi con lo stesso entusiamo che avevate all'epoca per chi lo ha posseduto anche perché secondo me il gioco merita tutt'ora attenzioni da parte del pubblico che ama gli adventure game e ve lo dice uno abituato alle curve di Lara Croft.

Il mio consiglio per iniziare questo nuovo anno insieme è quello di giocarlo almeno una volta nella vita e magari anche finirlo, così anche il nostro amato Rick avrà il posto che si merita non solo tra le pagine della nostra rivista, bensì tra le stelle dei nostri idoli protagonisti videoludici.

Per finire, visto che il gioco uscì per vari home computer penso che valga la pena giocarlo su qualsiasi macchina possediate ancora oppure emulatore dato che le conversioni furono delle più riuscite e con similitudini tra loro a parte quella Amiga che se non erro ebbe livelli un tantino più lunghi. Al prossimo numero guys!

Che dite, lo seguiamo il consiglio di Daniele e riprendiamo in mano Rick Dangerous?

Quei pochi che non lo conoscono faranno bene a colmare questa lacuna velocemente, perche' il gioco merita, eccome se merita!

Correva l'anno 1978

a cura di Associazione Firenze Vintage Bit Onlus (Federico Gori e Leonardo Vettori)

Per molti di noi il 1977 rappresenta il momento del BIG BANG dell'era del "Personal Computer".
Il 1977 è infatti l'anno della cosiddetta "Computer Trinity" e corrisponde all'uscita sul mercato di "Commodore PET", "Radio Shack TRS-80" ed "Apple II". Questi tre computer permettono all'informatica di entrare per la prima volta nelle case delle normali famiglie americane, che iniziano a sperimentare le infinite potenzialità del mondo digitale. Oltre a loro non possiamo inoltre non ricordare la commercializzazione dell'Atari VCS, una console dall'impatto epocale, ancora oggi amatissima da appassionati di tutte le età.

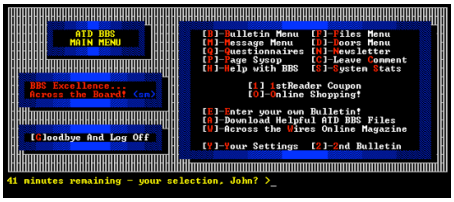
Parlare del 1978 in modo "interessante", dopo un 1977 così straordinario, non era quindi un compito facile.

Giovedì 13 Dicembre 2018, presso la Biblioteca Comunale Boncompagno da Signa (FI), Giorgio Alduini, Massimo Belardi e Federico Gori ci hanno invece stupito, raccontandoci fatti, eventi e curiosità informatiche che hanno caratterizzato quest'anno.

Ecco quindi i fatti più significativi del 1978, così come ce li hanno presentati:

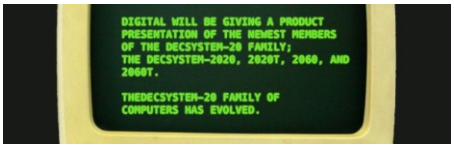
GENNAIO: al CES di Las Vegas la Apple presenta un prototipo del drive da cinque pollici e un quarto, ingegnerizzato dallo stesso Wozniak. La leggenda narra di come Steve avesse reinventato la "filosofia" della gestione del drive riducendo in maniera significativa la quantità dei componenti elettronici. Verrà messo in vendita a Giugno e contribuirà al successo dell'Apple II.

FEBBRAIO: a Chicago, Ward Chrinstensen e Randy Seuss pubblicano online la prima grande BBS (microcomputer bullettin board). Ward Chrinstensen riceverà nel 1993 la Pioneer Award da parte della Electronic Frontier Foundation.



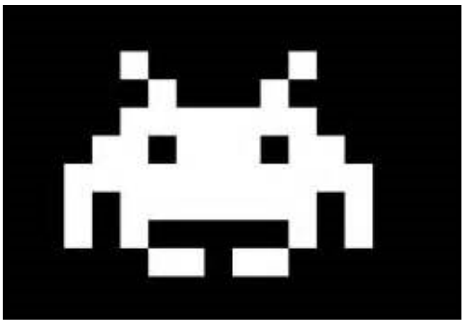
MARZO-APRILE: esce in America la console BALLY ASTROCADE con 4KB di RAM (espandibile fino a 64), 8KB di ROM, 3 voci audio ed una risoluzione grafica di 160x102. Una macchina interessante che però non riuscirà ad intaccare le vendite dell'Atari VCS.

MAGGIO: il 3 Maggio Gary Thuerk invia una pubblicità a 393 recapiti di posta elettronica utilizzando Arpanet. Nasce così il celebre SPAM. Il termine deriva da uno sketch comico dei leggendari Monty Python ed è l'abbreviazione di "Spiced Ham" (carne speziata).



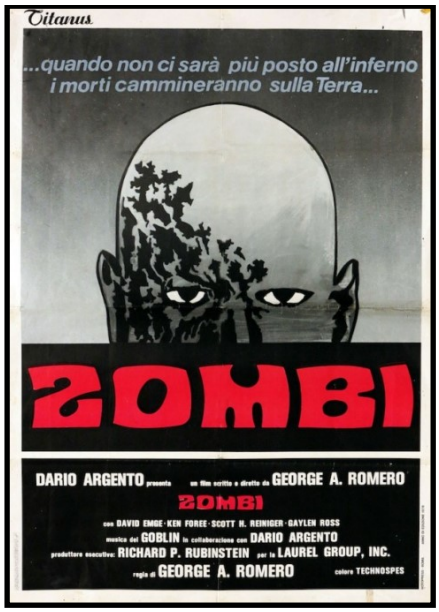
GIUGNO: Intel inizia la produzione del microprocessore 8086. Progettato in sole tre settimane da due ingegneri, dopo il fallimento del progetto iAPX432, questo processore sarà determinante per il mondo dei personal computer. Grazie ad esso nascerà infatti l'architettura x86.

GIUGNO: il Giappone viene invaso dagli alieni in formazione 11 x 5; arriva infatti SPACE INVADERS. Prima della fine dell'anno si conteranno 100.000 macchine installate in tutto il mondo. Il gioco avrà un successo talmente epocale da diventare un'icona pop.



GIUGNO: la Texas Instruments costruisce Lo Speak and Spell. In Italia sarà conosciuto come "Il Grillo Parlante"

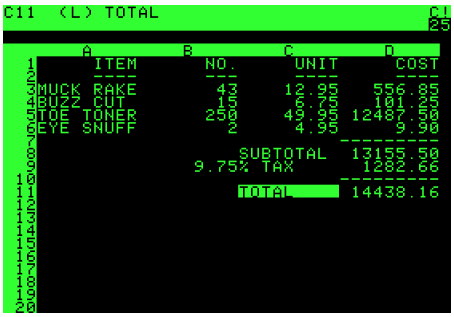
SETTEMBRE: George Romero e Dario Argento presentano a Torino, in anteprima mondiale, ZOMBI. Il film è una rappresentazione metaforica del consumismo ed è inoltre accompagnato da un alto tasso di violenza. Il successo è travolgente in tutto il mondo, tanto da farlo diventare un'icona del genere horror.



SETTEMBRE: Rob Barnaby apre la MicroPro International, che sviluppa il celebre WORDSTAR. Inizialmente scritto per i sistemi CP/M, successivamente viene rilasciato per MS-DOS. Wordstar è uno dei primi programmi di videoscrittura commerciali ed ha una grande diffusione fino alla metà degli anni '80.

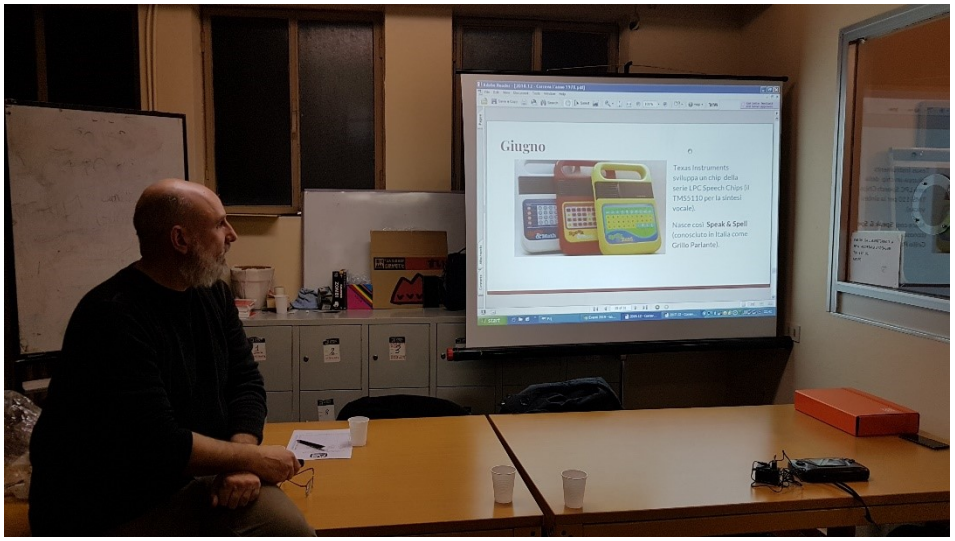
Nello stesso periodo viene presentato anche il prototipo del VisiCalc, il primo foglio elettronico della storia. Sviluppato da Dan Bricklin, inizialmente prevede una matrice di cinque colonne e venti righe. Verrà migliorato nel 1979 e diventerà la vera "Killer Application" dell'Apple II.

Da Visicalc nasceranno tantissimi altri fogli elettronici tra cui: Supercalc, Lotus 1-2-3 ed infine Excel.



DICEMBRE: ad Atlanta appare nei negozi il LASERDISC (allora chiamato DiscoVision), un formato rivoluzionario che per la prima volta presenta contenuti registrati su di un supporto ottico. Anche se il video è analogico, la resa visiva e sonora è un grande balzo in avanti rispetto alla concorrenza su nastro magnetico. Nonostante non sia mai riuscito ad intaccare il dominio del VHS, il laserdisc si è ritagliato una fetta di mercato significativa in America e Giappone. All'inizio degli anni '80 diviene molto celebre per l'uscita dei lasergames, titoli da sala giochi con video preregistrato (Dragon's Lair, Space Ace, Badlands etc.). Ancora oggi riscuote un buon successo tra gli appassionati ed i collezionisti.

Ovviamente il 1978, a livello tecnologico, non finisce certo qui. Vi invitiamo ad approfondire l'argomento. Se **interessati ai video della conferenza, li potete trovare nel gruppo Facebook di Firenze Vintage Bit**. Alla prossima!



Due novita' in arrivo...

di Francesco Fiorentini

La collaborazione e' sempre stata il punto cardine di RetroMagazine. Molti articoli sono nati dalla collaborazione diretta ed indiretta con i nostri lettori ed i membri dei gruppi Facebook o semplicemente dalla volonta' di singoli di veder pubblicato il loro lavoro su una rivista, seppur in solo formato elettronico. Ebbene il 2019 non poteva che continuare questa tradizione. Non mi stanchero' mai di ripeterlo, RetroMagazine e' la vostra rivista, consideratela alla stregua di un gruppo Facebook piu' complesso dove poter vedere pubblicati i vostri lavori.

A questo proposito, il sodalizio con il gruppo **8bit RetroProgramming Italia** e' un esempio perfetto di quanto appena affermato. La presenza mia e di Marco Pistorio tra gli amminstratori del gruppo ci da' la possibilita' di selezionare alcuni dei lavori prodotti dai redattori e riproporli in un formato differente all'interno della rivista. Attenzione non e' una ripetizione di contenuto, quanto piuttosto dare la possibilita' ad una platea piu' ampia (non tutti usano FB) di accedere a contenuti interessanti ed allo stesso tempo fare in modo che questi non vadano smarriti tra i meandri della rete.

Ma veniamo alle novita' del titolo. Vi ho detto che ci sono due novita' in arrivo, ma tanto per non smentirmi ve ne svelero' soltanto una. ☺

Alzi la mano chi si ricorda il mitico **S.E.U.C.K.?** Si', proprio lui, lo **Shoot'Em-Up Construction Kit** che tanto successo ha riscosso negli anni ottanta sul Commodore 64. Per chi non lo conoscesse, il SEUCK e' un tool di sviluppo che, senza necessita' di scrivere codice, permette a chiunque di creare un videogioco.

Come dice il nome stesso, il software e' fortemente orientato allo sviluppo di giochi di tipo sparatutto (con scorrimento verticale) ma, usando un po' di fantasia e' possibile aggirare questa limitazione per creare interessanti variazioni sul tema.

RetroMagazine ha deciso di chiedere l'aiuto di un super esperto creatore di giochi SEUCK e rilasciare un corso a puntate che speriamo soddisfera' la sete di conoscenza dei nostri lettori.

Pinov Vox in arte VG Vox sara' il nostro mentore che nei prossimi numeri ci illustera' dapprima il software e poi ci guidera' passo passo alla creazione di un videogioco.

Le credenziali di Pinov sono di tutto rispetto. Il suo gioco **Battle in the Woods** si e' piazzato terzo nell'agguerrita competizione **SEUCK Compo 2018**: http://tnd64.unikat.sk/Seuck_Compo_2018.html#BattleOnTheWoods

I presupposti per un corso affascinante ci sono tutti e personalmente non vedo l'ora di cominciare; e voi?

Ringraziamenti

Chiudo, come mio solito, con i **ringraziamenti** a tutti **i gruppi Facebook**, ai siti **OldGamesItalia** ed **IlVideoGioco.com** che ci aiutano a condividere la rivista ad ogni uscita e con un ringraziamento particolare a **Vincenzo Scarpa** che sta riservando nel suo ottimo sito **EmuWiki** uno spazio dedicato a RetroMagazine.

Arrivederci al prossimo numero!

RetroMagazine
Anno 3 - Numero 12

Direttore Responsabile
Francesco Fiorentini

Vice Direttore
Marco Pistorio

Gennaio 2019