RetroMagazine

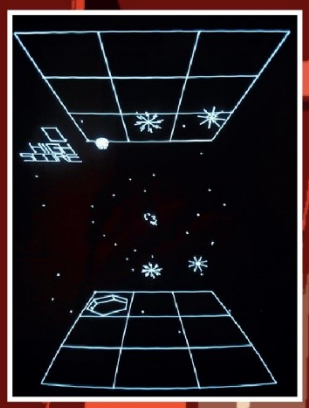
semplicemente retro

Numero 13 - Anno 3 - Febbraio 2019 - WWW.RETROMAGAZINE.NET - Pubblicazione gratuita

Sinclair QL: errori, sfortuna e molti rimpianti

Esplorando l'Amiga -parte 5

Non comprate un Vectrex!



Codice automutante in BASIC !

RetroMath: Ritorna la sfida del pi greco

C64: SHOOT'EM-UP CONSTRUCTION KIT

VAMPIRE THE MASQUERADE

Giappone: antica modernità o futuristico presente? (Prima puntata)

RetroMagazine
Anno 3 - Numero 13

Hanno collaborato a questo numero:
- Alberto Apostolo
- Roberto Lari
- Leonardo Giordani
- Robin Jubber
- David La Monaca/Cercamon
- Michele Ugolini
- Starfox Mulder
- Pinov Vox
- Fabrizio Caruso
- Daniele Brahimi
- Giorgio Balestrieri
- Querino Ialongo
- Attilio Capuozzo
- Dante Profeta
- Ass. Firenze Vintage Bit Onlus
- Marco Pistorio
- Francesco Fiorentini

Immagine di copertina:
Flavio Soldani

FEBBRAIO 2019 - WWW.RETROMAGAZINE.NET

IN EVIDENZA IN QUESTO NUMERO

Ricordi di un mito senza tempo

di Francesco Fiorentini

Non ricordo con esattezza l'anno, ma ricordo chiaramente che era il periodo natalizio. Facendo due rapidi calcoli probabilmente parliamo del Natale del 1983. Avevo quindi 10 anni ed il Natale era, come per tutti i bambini di questa eta', il periodo piu' bello dell'anno.

La Coop di Poggibonsi si trovava dopo il passaggio a livello di Largo Gramsci, in un grande stabile che adesso contiene diverse altre attivita'. All'epoca io e mio fratello passavamo alcuni dei nostri pomeriggi curiosando tra i reparti del suddetto centro commerciale, fantasticando su giochi ed altre amenita' che facevano bella mostra di se' sugli scaffali.

E fu proprio durante uno di questi pomeriggi, nei giorni antecedenti il Natale, che un misterioso oggetto nero che emetteva dei suoni simili ad esplosioni attrasse la nostra attenzione. Il 'totem scuro' aveva un monitor in bianco e nero collegato ad uno strano joystick molto piccolo ma con ben 4 pulsanti. Inutile dire che facemmo subito a gara per prendere il controllo del joystick e cominciare a giocare. Essendo piu' grande di mio fratello di 3 anni, fu facile per me essere il primo a mettere la mani sul controller ed immergermi immediatamente nel gioco.

La struttura del gioco era molto semplice, ai comandi di una navicella spaziale dovevamo distruggere tutte le astronavi nemiche che inesorabilmente ci venivano incontro e che una volta colpite si dividevano in astronavi piu' piccole, piu' veloci ed insidiose e quindi piu' difficile da colpire. Ricordo ancora benissimo le esplosioni vettoriali dal suono corposo e rimbombante ogni volta che un'astronave veniva colpita e distrutta. Nella mia mente di ragazzino quelle esplosioni in bianco e nero sembravano esplosioni dai mille colori ed il suono prodotto dall'altoparlante sembrava avere la potenza di un subwoofer super pompato. ☺

Ovviamente da quel momento in poi la mia mente comincio' a fantasticare su quel misterioso oggetto del desiderio e, considerato il Natale alle porte, sperare di riceverlo come dono natalizio... Purtroppo la vita ci insegna molto presto che non tutti i desideri vengono soddisfatti e, adesso come allora, sto ancora spettando il mio Vectrex in regalo.

Eh si', il misterioso 'totem scuro' non era altro che un Vectrex su cui girava il gioco Minestorm!

RetroMagazine in questo numero ha voluto dedicare largo spazio ad una delle console piu' iconiche della storia. La sua peculiare grafica vettoriale, il numero limitato di giochi a disposizione ed il numero limitato di console vendute ne fanno uno degli oggetti piu' esotorici da possedere; ma una collezione di retrocomputing che si rispetti non puo' dirsi completa finche' un Vectex non fa bella mostra di se' in uno degli scaffali a disposizione.

La grafica Vettoriale del Vectrex

Il Vectrex è l'unica console che utilizza una grafica basata su vettori al posto dei pixel; scopriamo insieme a **Robin Jubber** cos'altro ha di particolare e come programmarla.

Articolo a pagina 16

Sinclair QL

Continua la nostra avventura tra le macchine meno conosciute della storia. Questa volta tocca al Sinclair QL, una macchina con una storia ricca di errori e rimpianti; ripercorriamola insieme ad **Alberto Apostolo**.

Articolo a pagina 3

Sinclair QL: errori, sfortuna e molti rimpianti

di Alberto Apostolo

Con questo articolo si completa la tetralogia sulle macchine prodotte da Clive Sinclair, iniziata con l'MK14 (RM 6), proseguita con le calcolatrici (RM 9) e la serie dei computer ZX (RM 11).

Sinclair QL – immagine da Google

Nel 1982 Clive Sinclair avviò la progettazione del QL: un computer per uso gestionale più evoluto dello ZX Spectrum appena lanciato. L'hardware era costituito da un processore Motorola 68008 con 128 KB di RAM e due lettori per microdrive da 100 KB (non compatibili con lo ZX Spectrum). Sulla macchina era disponibile il SuperBasic e il sistema operativo QDOS operante in multitasking. Inoltre era offerta una ricca dotazione di programmi applicativi realizzati dalla PSION: Abacus (foglio elettronico), Archive (database), Quill (word processor), Easel (business graphic).

Clive Sinclair – immagine da [Smi14]

Per contrastare il lancio dell'Apple MacIntosh, il QL fu presentato in anticipo il 12 Gennaio 1984 a Londra, rivelandosi un clamoroso insuccesso che evidenziò le difficoltà di una azienda cresciuta troppo in fretta. La laboriosa messa a punto del computer, il rilascio incompleto avvenuto con grave ritardo, il prezzo di 399 Sterline furono fatali per il QL e la stessa Sinclair Research (esposta anche nella produzione con pochissime vendite di una TV flat-panel e del triciclo elettrico C5).

Nel 1985 chiuse la Timex (azienda americana produttrice su licenza Sinclair) ed entrò in crisi Prism (il principale distributore del Regno Unito). Ormai sull'orlo del fallimento, Clive Sinclair fu costretto a svendere i suoi prodotti.

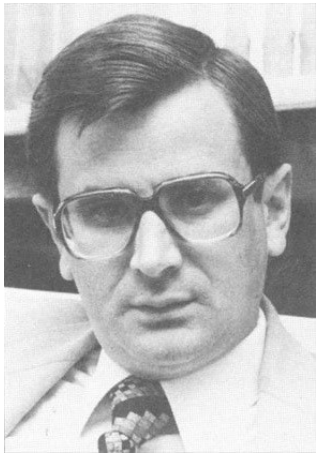
Cambridge Z88 – immagine da Google

Nel 1986 la Amstrad (Alan Michael Sugar Trading, n.d.A.) acquistò dalla Sinclair Research il ramo computer per 5 milioni di Sterline. Tale operazione decretò la fine della produzione del Sinclair QL (stimata in circa 150mila esemplari complessivi). Amstrad proseguì fino al 1990 l'evoluzione dello ZX Spectrum realizzando le serie +2 e +3, nonostante la presenza di sistemi con prestazioni superiori come MSX, Commodore Amiga e Atari 520 ST.

Clive Sinclair tornò ad occuparsi di informatica nel 1988 con il nuovo marchio Cambridge Computer e il portatile a pile Z-88 dotato di un display LCD e una CPU Z80. Pur suscitando buone impressioni, scomparve in fretta perché il mercato era definitivamente cambiato a causa dei PC-IBM compatibili e della richiesta di hardware e software con maggiore qualità.

Sogno di un computer professionale in tarda estate

Nel 1981, il 36enne Robb Wilmot venne assunto come Direttore Generale dalla I.C.L (International Computers Limited, una azienda pubblica britannica da poco privatizzata). In precedenza era stato il più giovane vice-presidente della Texas Instruments (dirigeva la Divisione Calcolatrici). Era anche amico di Clive Sinclair e credeva che la tecnologia Sinclair potesse rivitalizzare la gamma dei computer da ufficio. Inoltre la Dataskill (una azienda controllata dalla I.C.L.) offriva già programmi per il Sinclair ZX81 e aveva prodotto software gestionale per micro-computer casalinghi fin dai tempi del Nascom 1.



Robb Wilmot – immagine da [Smi14]

Con queste premesse, nel Dicembre 1981 venne annunciato un accordo stipulato tra Sinclair e Wilmot nei seguenti termini:

1) la I.C.L. avrebbe avuto la licenza del Basic Sinclair (una mossa nata in risposta alla televisione pubblica britannica B.B.C. che aveva adottato la tecnologia prodotta dalla concorrente Acorn),

2) la I.C.L. avrebbe comprato la TV a schermo piatto prodotta da Sinclair da unire a un eventuale computer da ufficio che poteva essere utilizzato anche come terminale per le comunicazioni (Sinclair avrebbe sviluppato l'hardware di tale macchina),

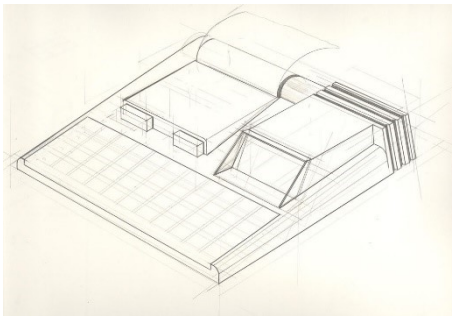
3) Sinclair avrebbe ottenuto un milione di Sterline per finanziare i costi di sviluppo e in più avrebbe ottenuto delle royalties qualora le macchine fossero state distribuite alla clientela.

Tra Maggio e Agosto 1981, Rick Dickinson (già designer dei computer ZX) aveva realizzato

alcuni bozzetti raffiguranti una macchina equipaggiata con tastiera, un paio di Microdrives (il sistema a nastro sviluppato dalla Sinclair) e un piccolo display incorporato. Altri bozzetti mostravano anche una stampante incorporata. Non erano lontanissimi dall'aspetto che avrebbe dovuto avere il One-Per-Desk, il sistema desiderato dalla I.C.L. .

Rick Dickinson – immagine da Google

Nel Maggio 1981 era apparso sul mercato il portatile Osborne 1; i bozzetti di Dickinson non rendono però l'idea se il computer da lui immaginato fosse un portatile oppure no.



Bozzetto di Dickinson – immagine da [Smi14]

Appena dopo il lancio dello ZX Spectrum, Clive Sinclair affermò nell'Aprile-Maggio del 1982 che il passo successivo sarebbe stato un nuovo computer in una opportuna fascia di prezzo superiore allo Spectrum, basato sulla stessa filosofia dell'Osborne 1, più maneggevole di un sistema IBM, dal peso di 1-1.5 Kg , con una stampante e con connessione alla TV a schermo piatto.

Ad essere incaricato di sviluppare un progetto è David Karlin, assunto dalla Sinclair nella tarda estate del 1982 in qualità di Ingegnere Capo-Progetto della divisione computer. David Karlin (all'epoca poco più che ventenne) aveva conseguito la laurea breve e la specialistica in Ingegneria Elettrotecnica a Cambridge. Poi aveva trascorso alcuni periodi di lavoro presso la Xerox di Palo Alto e la Fairchild Camera and Instrument Corporation a Singapore.

David Karlin – immagine da Google

Nel periodo alla Xerox, aveva potuto osservare lo Xerox Star e il suo sistema W.I.M.P. (Windows, Icons, Menus, Pointer) che aveva influenzato personalità come Bill Gates e Steve Jobs nella realizzazione dei loro prodotti. Quando in seguito decise di rientrare nel Regno Unito per motivi familiari e cercare un altro impiego, non era entusiasta di lavorare proprio a Cambridge. Tuttavia durante un colloquio con Sinclair, Karlin aveva esposto la sua visione di un sistema Xerox Star da 500 Sterline e Clive Sinclair lo aveva convinto ad accettare un posto di lavoro (offrendogli lo stesso stipendio ricevuto alla Fairchild, che era superiore alla media dell'epoca nel Regno Unito).

Grandi speranze per lo ZX83

David Karlin si mise subito al lavoro per scrivere le specifiche di una macchina gestionale denominata ZX83. Nelle intenzioni doveva essere un desktop computer avente quello che secondo lui era il minimo indispensabile: una tastiera decente, qualche tipo di connessione in rete, un monitor dedicato e una stampante. Intervistato, disse che:"...a grandi linee stavo tentando di progettare quello che probabilmente sarebbe diventato un Amstrad PCW 8256".

Inoltre doveva avere anche un qualche tipo di interfaccia utente a finestra, cosa che avrebbe richiesto: grafica bitmap ad alta risoluzione, abbastanza memoria e un processore veloce. Curiosamente non prese in considerazione l'idea di dotare il sistema di un mouse. Una scelta che oggi appare strana ma all'epoca era fiducioso che la gente potesse fare abbastanza usando soltanto i tasti cursore. Poi gli sforzi si concentrarono sulla circuiteria con la priorità di risparmiare il più possibile.

Dal momento che il microprocessore Z80A indirizza solo 64 KB e le tecniche di paginazione della memoria ne avrebbero limitato le prestazioni, comprese la necessità di un processore a 16 bit. Scartati lo Zilog Z8000 e l'Intel 8086, la scelta cadde sulla serie dei processori Motorola 68000 ritenuti molto buoni e una grande piattaforma per il futuro.

Si decise l'adozione del 68008 a 7.5 MHz, circa il doppio della frequenza dello Z80A usato sullo ZX Spectrum. Internamente funzionava a 32 bit con un bus dati a 8 bit e 20 bit riservati per l'indirizzamento.

Anche il 68000 (che sarà impiegato sul primo Apple MacIntosh, sugli Amiga e sull'Atari 520 ST) funzionava internamente a 32 bit ma con un bus dati a 16 bit, 24 bit per l'indirizzamento e un costo, all'epoca, 2-3 volte superiore al 68008.

Basandosi sulle raccomandazioni di Karlin, Clive Sinclair nel Dicembre 1982 approvò l'uso del 68008 giocandosi praticamente il futuro dell'azienda su quella piattaforma, credendo che in quel momento nemmeno la concorrenza si sarebbe potuta permettere di produrre un computer basato sul 68000.

Tony Tebby – immagine da Google

Sfortunatamente, nel corso del 1983, Motorola tagliò il prezzo del 68000 al di sotto di quello che Sinclair aveva contrattato per il 68008.

Rinegoziare il contratto e adottare il 68000 non sarebbe stato oneroso, ma aggiungere il 68000 nell'architettura avrebbe richiesto

banchi extra di ROM e di RAM e un chip separato per la logica di sistema (vista poi l'evoluzione del progetto, sarebbe stato meglio così ma è troppo facile parlare con il senno di poi).

L'architettura dello ZX83 prevedeva alcuni chip per la ROM, 64KB di RAM (32KB per i programmi, 32KB per la memoria video) e due chip U.L.A. (Uncommitted Logic Array) denominati ZX8301 e ZX8302.

Lo ZX8301 avrebbe connesso il 68008 con il resto del sistema; avrebbe funzionato come clock del processore, temporizzatore della memoria e mediatore tra il 68008 e il controller del display per accedere alla RAM. Il controllo del display sarebbe avvenuto tramite una connessione dedicata.

Lo ZX8302 avrebbe gestito l'Input/Output compresi i Microdrive, la tastiera , la rete, la porta della stampante. Avrebbe incorporato un modem (come da richiesta della I.C.L. per la One-Per-Desk di cui sopra) e un R.T.C. (Real Time Clock) alimentato da una batteria.

Jan Jones – immagine da Google

All'inizio del 1983, dopo avere scelto la CPU e definito uno schema di base del sistema, Karlin cominciò a valutare la macchina basata su tali componenti e a specificare il software di base richiesto. In quello stesso periodo, Tony Tebby e la neo-assunta Jan Jones furono incaricati di sviluppare il software di base: il sistema operativo e l'Interprete BASIC.

Prima di essere assunto alla Sinclair nel 1982, Tony Tebby era un laureato in Fisica che aveva imparato la programmazione per conto suo mentre lavorava alla G.E.C. sui sistemi a micro-onde. Lo aveva fatto perché riteneva poco valido il software usato da quella azienda ed era riuscito a creare qualcosa di meglio. Questo cambiò la sua carriera, prima alla Philips e poi (nel 1979) al Computer-Aided Design Centre di Cambridge, una joint-venture tra la I.C.L. Dataskill e il Ministero della Industria e del Commercio Britannico. Fu al C.A.D. che Tebby incontrò Jan Jones, una programmatrice che in precedenza si era laureata in Matematica mentre lavorava per altre aziende.

Fu Tebby stesso a convincere Jones a entrare alla Sinclair e collaborare al progretto di creare il SuperBasic, una versione del Basic Sinclair arricchita da elementi di programmazione strutturata che figuravano già nel Basic dell'Acorn BBC. Jan Jones dovette effettuare la codifica usando l'Assembler del 68008 dopo un lungo processo di analisi per definire i comandi con le loro caratteristiche. Le intenzioni erano di costruire il linguaggio "dentro" la macchina, non solo per programmare o permettere a terzi di realizzare applicativi, ma anche come un linguaggio shell per comandi operativi "elegante, facile da usare e pieno di funzioni". Con ciò la dirigenza Sinclair desiderava che il "core" dello ZX83 dovesse diventare la base delle future "incarnazioni" dello ZX Spectrum.

Nigel Searle – immagine da Google

David Karlin intendeva usare il QDOS realizzato da Tebby denominato "Domesdos". Se non avesse funzionato, avrebbe ripiegato su un sistema operativo realizzato dalla G.S.T. Computer Systems. Secondo Tebby invece era il contrario, era la G.S.T. ad essere stata contattata (senza l'approvazione della dirigenza Sinclair) ma non sarebbe riuscita a finirlo in tempo. Tuttavia diversi indizi fanno pensare che la versione di Karlin sia quella vera. Da annotare anche i tentativi di coinvolgere la Digital Research e la Microsoft, ma le loro realizzazioni non andavano bene per l'hardware dell'epoca.

David Potter – immagine da Google

Nel Dicembre 1982, Nigel Searle aveva preso contatto con possibili partner per sviluppare il software da vendere insieme al computer. Tra le aziende contattate vi era la PSION (che in seguito produrrà Symbian, un sistema operativo per cellulari). David Potter, era l'ambizioso direttore della PSION che aveva compreso l'importanza del software gestionale per far crescere l'azienda ed entrare in concorrenza con gli Americani nel mercato dei word-processor e dei fogli di calcolo. Dopo lunghe discussioni fu scelta la PSION senza però coinvolgere Karlin e Tebby (tuttavia confezionare gli applicativi con il computer faceva parte dei piani). La PSION pretese che il QL potesse supportare uno schermo in formato 80x25 caratteri e stabilì Quill come word-processor, Easel come tool grafico, Archive come data-base e Abacus come foglio elettronico.

Tali programmi avrebbero in futuro costituito la suite "xChange", una specie di standard di fatto per i computer a 16 bit (sviluppare una suite integrata comportava meno rischi che sviluppare diverse applicazioni "stand-alone").

Ciascuna applicazione fu affidata a un team-leader: Martin Brown per Easel, Martin Stamp per Quill, Charles Davies per Archive, Colly Myers per Abacus (quest'ultimo divenne Direttore Generale della PSION e in seguito C.E.O. di Symbian).

I programmatori della PSION impiegheranno 15 mesi per completare gli applicativi. Senza avere una minima idea di come funzionasse uno ZX83, essi utilizzarono un micro-computer VAX con un emulatore di 68008

finché non fu disponibile uno ZX83 con il QDOS.

Il sogno diventa presto un incubo

A Marzo furono stabiliti nove mesi come durata del progetto, per essere certi di lanciare il prodotto poco prima di Natale.

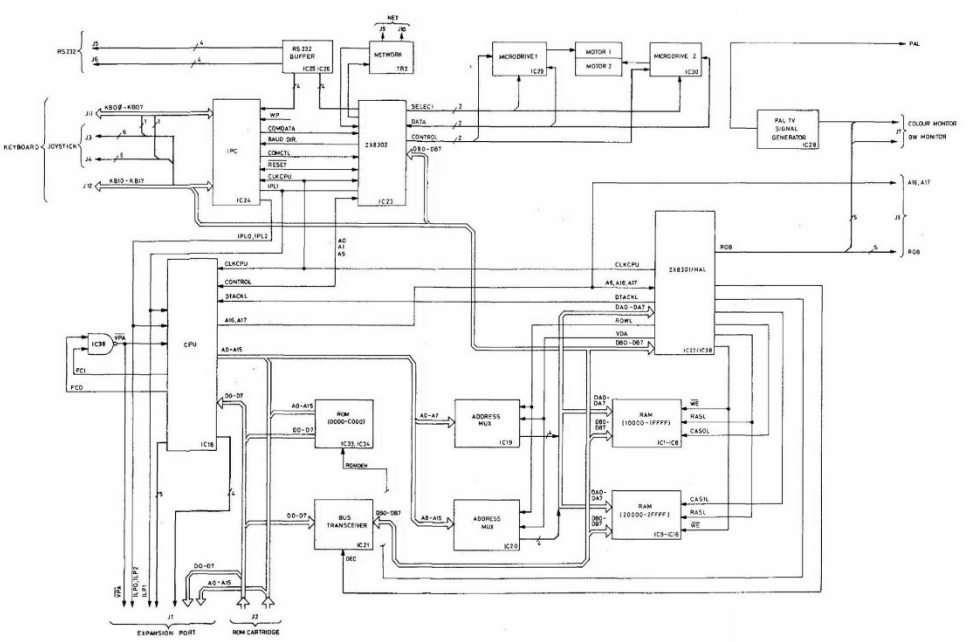
Una scadenza pazzesca: Sinclair non era mai riuscito a creare una macchina in così poco tempo e, tra i dirigenti, pare che solo Jim Westwood (Ingegnere Capo dell'Hardware) avesse evidenziato la necessità di prendere più tempo per lo sviluppo (senza tuttavia essere ascoltato).

Jim Westwood – immagine da Google

Inoltre non c'era chiarezza nella dirigenza riguardo a cosa fosse davvero lo ZX83: era un portatile oppure un desktop computer?

Nel mese di Maggio, Nigel Searle (Direttore Generale della Sinclair) convocò una conferenza stampa per annunciare pubblicamente lo sviluppo del nuovo modello ZX83, dichiarando che non sarebbe stato un clone di un PC-IBM e che la Sinclair avrebbe sviluppato un suo proprio sistema operativo.

Allontanarsi dai sistemi operativi come il CP/M della Digital Research o l'emergente MS-DOS della Microsoft (che all'epoca rappresentavano di fatto degli standard) era un vero e proprio azzardo. Di fronte a una platea entusiasta al sentire che "Zio" Clive Sinclair stava per piazzare un altro successo dopo lo ZX Spectrum, Searle stuzzicò i partecipanti con la notizia che lo ZX83 avrebbe potuto essere un portatile con un display basato sulla tecnologia della TV Sinclair a schermo piatto e memorie a stato solido. Inoltre lasciò trapelare che la macchina



Schema elettrico del Simclair QL – immagine da Google

si sarebbe probabilmente chiamata QL, una abbreviazione per "Quantum Leap" e che il lancio sarebbe stato previsto per il 12 Gennaio 1984 all'Hotel Inter-Continental presso l'Hyde Park Corner di Londra.

Le ultime speranze di avere un portatile durarono fino all'Estate del 1983, in seguito alla scoperta che le batterie prodotte per la TV tascabile Sinclair Microvision 2700 avrebbero garantito solo 30 minuti di autonomia a un portatile (10 minuti con i Microdrive collegati insieme) e che un testo visualizzato su un micro-display Sinclair (collegato per prova a uno ZX Spectrum) risultava appena leggibile.

Mettere tante funzionalità in due soli chip da 40 pin si era rivelato quasi impossibile (l'I/O seriale richiedeva 8 pin e non 4) e l'idea di creare un nuovo ZX Spectrum basato sullo ZX83 fu silenziosamente accantonata tra Agosto e i primi di Settembre del 1983.

Lo ZX8302 non era completo ed era necessario alleggerirlo dell'interfaccia per la tastiera. Questo portò alla più controversa decisione all'interno del progetto: includere un micro-processore Intel 8049 nella logica di sistema. L'Intel 8049 era soprannominato impropriamente "Intelligent Peripherals Controller". Era in anche in grado di generare suoni ed era utile per alleggerire lo ZX8302 della gestione delle porte seriali.

Ne scaturirono effetti collaterali negativi. L'8049 aveva i suoi registri di stato e Karlin non poteva più utilizzare un banco centralizzato dei registri di stato e di interrupt delle periferiche (uno schema "furbo" da lui ideato che gli consentiva di risparmiare sui costi dell'hardware e semplificare il software).

Per fare in modo che il QDOS potesse gestire anche l'8049, Tebby chiamò Aaron Turner. Nonostante fosse un lavoraccio, Turner riuscì nell'intento. Invece fu assai più difficile mettere l'8049 e le sue linee di collegamento sulla lunga e stretta scheda madre, derivata dalla forma del case scelto da tempo per lo ZX83.

Il problema delle misure della scheda madre si aggravò quando vi fu la mossa di rendere lo ZX83 più "casalingo", aggiungendo anche una uscita TV e le connessioni joystick (strani accessori per un micro gestionale). I joystick furono aggiunti grazie all'8049, mentre il modulatore UHF causò grossi grattacapi perché era stato posto a destra, vicino ai Microdrive e alle loro fragili testine di lettura/scrittura, sensibili al disturbo elettromagnetico dell'oscillatore. Inoltre fu deciso che ci voleva la "classica" partenza del Basic al momento dell'accensione.

Per realizzare il SuperBasic, Jan Jones aveva lavorato duramente e rispettato le consegne, ma il SuperBasic non era stato progettato per girare "sopra" il QDOS. Aaron Turner adattò

senza difficoltà le routine grafiche del SuperBasic (realizzate dalla G.S.T. nel tentativo di guadagnare tempo) affinché operassero attraverso il QDOS Display Manager invece di scrivere direttamente nella RAM riservata per lo schermo. Egli riscrisse l'Interprete in modo tale che potesse ottenere/rilasciare memoria attraverso il QDOS Memory Manager invece di farlo da sè.

Quattro settimane prima di Natale, Tebby e Jones sfruttarono la maggior parte della sessione di lancio pre-nataliza per testare l'interazione del software con l'hardware. Tebby decise di "smanettare" il QDOS e il SuperBasic per permettere all'Interprete di girare come fosse un monolitico task privilegiato speciale. Un grave errore che compromise l'integrità del sistema operativo.

Nello stesso tempo, c'era ancora in piedi la questione della doppia versione del SuperBasic: uno compatto e minimale cablato in ROM e un altro più esteso caricato tramite la cartuccia di un Microdrive (il Basic "minimale" si sarebbe dovuto accorgere del Basic "esteso" e caricare le estensioni di conseguenza). La "favola" dei due Basic avrebbe causato più avanti uno spiacevole equivoco.

Alla fine del 1983 fu chiaro che il QL si trovava in ritardo. Le colpe erano dovute ai dubbi su ciò che doveva essere lo ZX83 e alla dirigenza Sinclair che rifiutò di prorogare la scadenza del progetto. Decidere che la piattaforma gestionale dello ZX83 doveva essere la base per le future versioni dello ZX Spectrum, non solo aveva obbligato ad avere una nuova versione del SuperBasic basata sul 68008, ma aveva costretto Karlin a rivedere le specifiche iniziali per i due chip ULA.

Per esempio, la circuiteria del display controller nello ZX8301 aveva bisogno di essere modificata per supportare le modalità esistenti sullo ZX Spectrum. Lo ZX8302 aveva richiesto un opportuno generatore di suoni affinché la macchina producesse suoni e musica migliori di quanto facesse lo ZX Spectrum.

In separata sede fu deciso che ci voleva troppo per implementare un modem dentro lo ZX8301 e il modem fu eliminato. Allo stesso modo fu eliminata la porta di output dedicata per la stampante, sostituita da due generiche porte RS232.

Pubblicità ingannevole e il "giallo" della ROM esterna

La scadenza del progetto era stata modificata formalmente, spostandola agli inizi del 1984.

David Karlin e Jan Jones avrebbero avuto bisogno di altri sei mesi di lavoro a causa delle lungaggini dovute alle continue revisioni dei circuiti stampati e delle ULA di prova. Secondo Tony Tebby non avevano ancora a disposizione un prototipo completo funzionante.

David Karlin avrebbe dovuto avvertire Nigel Searle e Clive Sinclair, ma ciò nonostante era stata mantenuta la scadenza del Gennaio 1984 per la presentazione alla stampa. Questo stato di cose scontentò Tebby. La goccia che fece traboccare il vaso, fu l'annuncio (durante la presentazione) che i clienti avrebbero avuto la macchina entro 28 giorni dall'ordine e che gli ordini sarebbero stati accettati a partire dalla fine di Gennaio. Tebby comunicò che avrebbe lasciato la Sinclair quando il QL fosse stato in produzione, cosa che avvenne puntualmente in Aprile.

Il computer veniva offerto con 128 KB di RAM al prezzo di 399 Sterline più 7.95 Sterline per le spese di spedizione. Nessuna menzione della versione "economica" da 64 KB di RAM al prezzo di 299 Sterline, dato che gli applicativi PSION richiedevano più di 32 KB di RAM ciascuno.

Negli Stati Uniti (dove Searle aveva fondato nel 1980 una filiale della Sinclair, poi gestita stando a Cambridge a partire dal 1982) il debutto era previsto per l'Autunno 1984 al prezzo di 499 Dollari. In seguito, tale debutto sarà posticipato mettendo in difficoltà economiche la Sinclair.

Alcuni osservatori ebbero il fondato sopetto che ci sarebbero stati ritardi nelle consegne del QL. Intanto a fine Febbraio si registrarono 9000 prenotazioni, salite poi a 13000 alla fine di Aprile. La Sinclair non esitò a fare cassa con le somme di denaro versate in anticipo.

La fretta per il lancio aveva creato mancanza di comunicazione all'interno della Sinclair: il gruppo marketing sapeva che il QL era dotato di un Basic simil-Spectrum e che ci doveva essere un rilascio in due fasi. La documentazione di pre-rilascio per la stampa fu scopiazzata dal manuale dello ZX Spectrum. Ma non si confrontarono con gli sviluppatori. Il risultato fu che, avendo detto al mondo che il QL si vendeva con una versione completa del SuperBasic, l'azienda ora doveva fornirlo.

Il fatto che il QL non avesse le caratteristiche promesse e che i clienti si fossero esposti finanziariamente senza ricevere affatto il computer, suscitò le "attenzioni" della Advertising Standards Authority (equivalente britannico dell'A.G.CO.M., l'Autorità Garante della Concorrenza e del Mercato, n.d.A.).

Clive Sinclair promise che il denaro versato non sarebbe stato toccato e che sarebbe stato creato un "fondo fiduciario" finché i QL non sarebbero stati consegnati. Qualcuno contestò il fatto che Sinclair, durante il ritardo della consegna ai compratori, percepisse comunque gli interessi sulle somme di denaro versate. Allora Sinclair permise che gli ordini potessero essere cancellati, rimborsando il denaro. Ma pochi sembravano interessati a questa opportunità.

Alla metà di Marzo del 1984 (otto settimane dopo l'inizio delle prenotazioni), nessun QL era stato consegnato e cominciava a circolare il poco lusinghiero appellativo "Quite Late" ("abbastanza in ritardo", n.d.A.). Ufficialmente perché il completamento del QDOS aveva richiesto più tempo del previsto e una delle due ULA richiedeva ulteriori modifiche. Inoltre era da considerare che i Microdrive per il QL avevano richiesto più di un anno per sanare diversi problemi meccanici ed elettronici.

Per la decodifica del segnale magnetico (e tenere conto anche delle variazioni di velocità del nastro) era stato progettato un circuito digitale detto P.L.L. ("Phase Lock Loop") ma tale circuito aveva problemi a riconoscere la forma d'onda del segnale dei Microdrive. Con l'aiuto di Ben Cheese (un ingegnere della Sinclair che si occupava di Elettronica Analogica) furono raggiunti dei miglioramenti ma comunque insufficienti. L'inaffidabilità dei Microdrive si sarebbe rivelata fatale per il prodotto.

Una seconda data di scadenza per consegnare il QL nelle mani dei clienti fu fissata per la fine di Aprile del 1984. Finalmente, il 30 Aprile 1984, fu annunciato che poche dozzine di QL sarebbero state consegnate di persona da impiegati della Sinclair ai nuovi proprietari. In omaggio avrebbero avuto gratis il cavo RS232 per la stampante (che costava 15 Sterline).

Quello che il portavoce della Sinclair definiva un "gesto di buona volontà", irritò i clienti che dovevano fare i conti con un computer che non mitigava la lunga attesa sopportata. Il numero di Giugno 1984 della rivista "Your Computer" riportava che "finiture scadenti e programmi che non si caricano, sono il problema minore. ... Lo Screen Editor può mandare in crash il sistema e il promesso Real Time Clock mancava insieme ai manuali".

Il dispositivo RTC non sarà mai implementato perché lo ZX8302 doveva essere modificato per eliminare un bug riguardante l'accesso al bus del 68008 che si manifestava durante i reset. Era più facile eliminare la batteria e smettere di dire che il QL avesse un RTC.

Il QL aveva anche una ROM esterna (nota con nomignolo di "dongle"). Secondo le fonti ufficiali, Il software di base del QL (QDOS più SuperBasic) occupava 40 KB. Troppo per una sola ROM da 32 KB. Tebby ha sostenuto che in realtà QDOS e SuperBasic erano già pronti per essere cablati in ROM fin dal Marzo 1984 e che non c'era bisogno di una ROM esterna. Inoltre le specifiche del QL prevedevano 64 KB di ROM e la scheda madre incorporava 2 slot per i chip della ROM (che potevano essere da 8, 16, 32 KB). Ma allora come stavano veramente le cose?

La filosofia Sinclair era "consegnare" e poi "risolvere" i problemi piuttosto che ritardare finché fosse tutto a posto. Nel caso del QL forse era una mossa "politica" per incolpare il software invece di ammettere che l'hardware non era pronto. Il software difettoso implica tacitamente che una versione migliore è in lavorazione mentre nessuno è disposto ad accettare di comprare hardware difettoso. Ovviamente i clienti credevano a quella ingenua bugia e non avrebbero battuto ciglio qualora la Sinclair avesse di fatto sostituito le loro macchine a tempo debito.

Verso il traguardo per approssimazioni successive

Durante i primi sei mesi del 1984, molti altri inconvenienti furono risolti con aggiornamenti hardware e molti altri ricorrendo a trucchi software.

Ma non tutti i rimedi portavano al successo sperato: ad esempio, l'eliminazione di un "glitch" che si verificava sul chip ZX8301 quando si scaldava troppo, aveva minimizzato l'impatto di un bug nell'uso dei Microdrive ma aveva "distrutto" la compabilità di collegamento in rete con la Interface 1 dello ZX Spectrum.

Entro Luglio 1984, i QL non avevano più l'infame ROM esterna e quelli che uscivano dalle linee di produzione Datatech erano marcati "Do6". Alla fine del mese Sinclair disse che presto i QL con la ROM esterna sarebbero stati richiamati per una sostituzione gratuita della ROM. Le macchine vecchie sarebbero state demolite e sostituite da versioni "Do6" o più nuove. L'intenzione era di scaglionare il richiamo delle macchine e ancora non si sapeva quanto tempo i clienti sarebbero rimasti senza computer. Il "richiamo" iniziò ad Agosto e fu calcolato che sarebbe durato 10 giorni.

A questo punto si susseguirono una serie di versioni della ROM codificate: "FB", "PM", "AH". "FB "era quella più vecchia, "AH" era la versione "finale" come disse un portavoce nel Luglio 1984. Secondo Tony Tebby, la "vera" versione finale era la "JM" testata e consegnata nel Marzo 1984 e presente sui QL a partire dl Luglio 1984 (le altre erano versioni con bug risolti senza nuove aggiunte al SuperBasic). Poche settimane dopo apparve la "TB" e poi la "JS" (dopodiché Tebby lasciò la Sinclair). Nel 1985 vi fu ancora la "MG", che formava la base dei QL non inglesi.

Nell'Autunno 1984 cominciò la consegna del QL ai rivenditori al dettaglio. Con le nuove specifiche di produzione si era arrivati alla versione "D14" e la scheda madre era marcata "Issue6", nella quale un chip TTL in più sanava un difetto nella ULA ZX8301. L'hardware del QL era abbastanza stabile da considerarsi "1.0". I futuri aggiornamenti costruttivi furono fatti per migliorare cose non inserite nel progetto originale.

Il manuale completo del QL fu scritto da Roy Atherton del Bullmershe College Computer Centre, inglobando materiale prodotto dallo staff programmatori (costituito da Steve Berry della Sinclair e Dick de Grandis-Harnson della PSION).

L'oscillatore ad alta frequenza della uscita TV era stato spostato lontano dall'amplificatore della testina del Microdrive sinistro. In tal modo si impediva l'interferenza durante la fase di lettura che rendeva praticamente inservibile il drive. Sul drive stesso, un condensatore fu posto in parallelo con la testina per bloccare il disturbo indotto dal motorino del drive. I connettori stile telefono anni '80 usati per il joystick e le porte seriali, furono sostituiti da D-Sub a 9 pin, maschio per le porte joystick, femmina per le seriali.

Con il QL nei negozi, Sinclair ebbe la fortuna di effettuare un sostanziale rilancio del computer. Lo spazio pubblicitario televisivo era affollato di spot che dimostravano come il QL fosse un'alternativa economica ai suoi rivali: l'IBM PC, il Mac e il BBC Modello B (il cui prezzo partiva da 399 Sterline fino ad arrivare a 1632 se si aggiungevano un paio di floppy drive, un monitor ecc.). D'altra parte il QL proposto con il monitor a colori, costava 698 Sterline.

Ma le macchine continuavano ad avere ancora problemi. Era ancora presente un difetto relativo al generatore del segnale per il monitor RGB contenuto nello ZX8301, che avrebbe potuto bruciare il chip se si fosse staccato il cavo del monitor mentre il computer era ancora acceso.

Nel Novembre 1984, un giornalista di "Sinclair User" affermava che su 1000 QL consegnati a Dixons (una catena di vendita di elettrodomestici, n.d.A.) solo 190 funzionavano bene. Si raccontava di un negoziante che provava ripetutamente sui QL le cartucce con i programmi Psion per trovare una combinazione tale che potessero essere lette.

Malgrado la richiesta insistente di supporto fatta dalla Sinclair alle software house più famose, la scarsità di software non aiutava le vendite e nemmeno la fragilità e l'instabilità che affliggeva i primi modelli rilasciati prematuramente. Una portavoce della catena

WH Smith comunicò che le vendite alla fine del 1984 erano state molto basse. Un portavoce della catena Boots parlava di vendite deludenti. La stampa stimava che fossero stati venduti solo circa 40mila QL, una frazione del potenziale bacino di utenza.

Infine non si ancora visto almeno uno degli accessori promessi da Clive Sinclair (tra i quali l'espansione di memoria da 512 KB, un disco rigido e un modem).

La fine dei giochi

Agli inizi del 1985, David Karlin aveva l'intenzione di abbandonare la Sinclair per mettersi in proprio. Tony Tebby aveva mantenuto la sua promessa di dimettersi solo quando il QL fosse stato nelle condizioni di essere distribuito alla clientela. Jan Jones, in dolce attesa del suo primo bambino, si dimise subito dopo per dedicarsi alla famiglia e alla sua nuova carriera di affermata scrittrice di romanzi rosa.

Alan Sugar – immagine da Google

Tuttavia Nigel Searle riuscì nel convincere Karlin a restare in azienda per gestire la produzione del QL, assegnata a due aziende: la Timex and Thorn EMI-Datatech e la Samsung. Di conseguenza, Karlin per 12 mesi si occupò di ordini e fatture, prendendo il posto che era di Dave Chatten, nominato Direttore Generale nel Marzo 1985 con Bill Jeffrey (quest'ultimo proveniente dalla Mars Electronic).

Così la Sinclair (anche a causa di altri progetti fallimentari come il Wafer Scale Integration e il triciclo C5) si trovò ad avere problemi finanziari.

Nell'Estate 1985, sembrava che il tycoon Robert Maxwell fosse disposto a rilevarla versando 12 milioni di Sterline. Ma grazie a una massiccia prenotazione di macchine da parte di Dixons, Clive Sinclair riuscì a superare il momento di crisi.

ICL one per desk – immagine da Google

In Sinclair si vociferava di una nuova versione di QL e dello sviluppo di uno ZX Spectrum da 128 KB con capitale spagnolo. La nuova macchina gestionale QL 2 (nome in codice "Enigma" o forse "Tyche") avrebbe dovuto avere una intefaccia grafica Gem e supportare la suite "xChange" della PSION.

Il progetto QL 2 fu cancellato nell'Aprile del 1986 con la cessione della Sinclair Research alla Amstrad per 5 milioni di Sterline, meno della metà di quanto stimato nell'estate precedente. Il nuovo proprietario, Alan Sugar (che fra l'altro, detestava quelli della PSION) tagliò il QL a favore del PCW 8256. Molto personale proveniente dalla Sinclair si trovò in esubero (compreso David Karlin stesso, il quale passerà ad altre iniziative imprenditoriali).

In totale furono costruiti 139.454 esemplari di QL, di cui 122.793 dalla Thorn EMI Datatech per il mercato britannico e 16.661 dalla Samsung per l'Europa e gli Stati Uniti.

Nel 1985 era già arrivato, al prezzo di 1195 Sterline, l'I.C.L. One-Per-Desk (dopo una presentazione nel Novembre 1984). Questa macchina nel Gennaio 1986 vinse un Recognition of Information Technology Achievement del concorso System Innovation of the Year. Un riflesso della vittoria, assegnata al QL nel Luglio 1985, del premio Microcomputer of the Year dei British Microcomputer Awards. Segno che qualcuno aveva visto delle potenzialità nel QL, ma mai realizzate in pieno (principalmente per avere incorporato dei Microdrive difettosi).

Nonostante le avversità, molti vollero lo stesso continuare a lottare. A Stevenage, David e Vic Oliver possedevano la Cambridge System Technology che vendeva accessori per il QL. Assieme al tecnico Graham Priestley realizzarono il computer Thor con la scheda madre del QL realizzata dalla Samsung. Il QDOS era stato modificato per gestire uno o due floppy disk. Le due configurazioni costavano 599 e 699 Sterline rispettivamente. Aggiungendo un disco rigido SCSI da 20 MB, il prezzo saliva a 1399 Sterline. La stampa specializzata lo battezzò subito il "figlio" del QL.

La Amstrad ne ostacolò l'iniziativa vietando l'uso del nome QL e i componenti QL. Fu fatto un tentativo per acquisire la licenza dalla Amstrad ma ci fu un netto rifiuto. La PSION fu più accomodante e, per i suoi quattro famosi programmi applicativi, concesse la licenza alla C.S.T., al suo partner danese DanSoft e al suo distributore britannico Eidersoft.

La C.S.T. ebbe poi successo nel convincere la Samsung a produrre un kit compatibile. Riuscì anche a presentare la versione Thor 20 in alcune mostre tra il 1986 e il 1987, evitando le interferenze della Amstrad. Si credette perfino che avesse un certo numero di compratori, ma comunque erano troppo pochi per garantire la produzione che cessò nel 1988.

Thor – immagine da Google

Nel frattempo Tony Tebby stava progettando un QL di seconda generazione con Jonathan Oakley (anche lui tecnico ex-Sinclair). Fondarono la QJump, una azienda che produceva software dedicato al QL. Avevano anche trovato dei finanziatori ma, giunti al dunque, questi ultimi non furono disposti a

stanziare le 250mila Sterline richieste da Tebby per creare un prodotto finito.

Ad ogni modo Tebby resterà sempre legato al mondo QL offrendo accessori per il QDOS e il SuperBasic. Realizzò pure una versione di QDOS chiamata SMS2 per l'Atari ST (che aveva il 68000). L'SMS2 in seguito divenne SMSQ per il Miracle System QXL, un accessorio per i PC con il 68000.

Conclusioni

Errori e circostanze sfavorevoli hanno reso il Sinclair QL un insuccesso. Ma non tutto il male viene per nuocere. Tra gli utenti eccellenti del QL si annovera Linus Torvalds (l'inventore di Linux), il quale non ha mai fatto mistero di avere imparato a programmare su un QL.

Altre aziende hanno analizzato le vicissitudini del QL e ne hanno tratto vantaggio cercando di non ripetere gli stessi errori.

Per gioco, tra di noi capita qualche volta di parlare del Sinclair QL. Contando sulla vicinanza dell'Università di Cambridge, Clive Sinclair avrebbe potuto procurarsi senza perdere tempo un collaudato sistema operativo Unix e personalizzarlo. L'architettura avrebbe dovuto avere un paio di floppy-disk drive invece di quegli orrendi Microdrive. Con il Sistema Operativo e il SuperBasic su dischetti, invece di essere cablati in ROM, risolvere gli errori sarebbe stato più facile e gli acquirenti avrebbero sopportato i bug in cambio di versioni sempre più aggiornate.

Bastava montare una sicura uscita per un monitor e poi vendendo un adattatore esterno si sarebbe potuta usare anche una TV. Buona l'idea di usare 2 porte RS232 e magari fare in modo di espandere il sistema aggiungendo altri dispositivi da collegare attraverso lo slot (ad esempio una interfaccia joystick o un modem). Male l'idea di venderlo con solo 128 KB invece che fornire da subito tutti i 640 KB. Se il QL fosse costato anche 599 Sterline, l'entusiasmo nei confronti di Sinclair era tale che molti avrebbero comunque effettuato il passaggio dallo ZX Spectrum al QL.

Infine il supremo colpo di mercato, sarebbe stato dotare il QL di un emulatore per ZX Spectrum, capace di utilizzare ancora l'enorme quantità di giochi prodotti fino a quel momento. Ma, come si dice, se mio nonno avesse avuto tre palle allora era un flipper...

Appendice

In Italia il Sinclair QL fu presentato il 20 Febbraio 1984 all'Hotel Michelangelo di Milano da Charles Cotton (Business Manager Sinclair) ed era venduto al prezzo di 1.300.000 Lire. Fu accolto con curiosità ("MC Microcomputer" n.28, Marzo 1984, www.issuu.com), rivelandosi poi una delusione come testimoniano le recensioni su "MC Microcomputer" n.31 (Giungno 1984) e n.32 (Luglio-Agosto 1984).

Una situazione analoga si ritrova anche su "Sinclair Computer", la rivista "istituzionale" del mondo Sinclair in Italia (vedi nn. 2,3 8,9,11 e poi le rubriche avviate dal n.14 al n.19 prima della confluenza nella rivista "Personal Computer" avvenuta nel Gennaio 1986).

Al giorno d'oggi, cercando su Internet si trovano numerose pagine Web, siti sull'argomento e sugli eventi organizzati da club di appassionati. Per esempio il 14 Ottobre 2018, si è svolto a Modena il 15° Meeting Italiano Sinclair QL.

Links utili

1) www.sinclairql.it gestito da Davide Santachiara,

2) http://sinclairql.net/chronology.html di Urs König (con una cronologia delle vicissitudini del QL),

3) www.dilwyn.me.uk del mitico Dilwyn Jones (dove è presente la sezione "QL Emulators" dalla quale scaricare gli emulatori del QL),

4) http://sinclairql.speccy.org/manuales.htm (una pagina in spagnolo dove si possono scaricare utili manuali in formato pdf),

5) http://www.archeologiainformatica.it con l'articolo di Stefano Paganini e Carlo Santagostino del 6 Gennaio 2014,

6)https://archive.org/details/sinclaircomputeritalian, tutti i numeri di Sinclair Computer,

7) https://issuu.com/adpware, dove si trovano tutti i numeri della rivista MC MicroComputer.

Bibliografia

[AK86] I.Adamson, R. Kennedy, "The decline of Uncle Clive", New Scientist n.1512 , 12 Giugno 1986, pp.33-36.

[Lea16] T. Lean, "Electronic Dreams: How 1980s Britain Learned to Love the Computer", Bloomsbury Publishing, 2016.

[NS84] AA.VV., "Sinclair's latest computer is faulty", New Scientist n.1411, 24 Maggio 1984, p. 5.

[Smi14] T. Smith, "Sinclair's 1984 big shot at business: The QL is 30 years old", https://www.theregister.co.uk/ , consultato il 29 ottobre 2018.

[Sto18] AA.VV., "Sinclair QL: l'inizio del declino e l'arrivo di Amstrad" , https://www.storiainformatica.it , consultato il 29 ottobre 2018.

Sinclair QL in breve – fonte Wikipedia

Classe di computer	Personal computer
Paese d'origine	Regno Unito
Produttore	Sinclair Research
Inizio vendita	12 gennaio 1984
Fine vendita	1993
Prezzo di lancio	£ 399 (nel Regno Unito)
CPU	MC68008 a 7,5 MHz
FPU	non presente
MMU	non presente
Altri coprocessori	1 Intel 8049, 1 ZX8301 e 1 ZX8302
ROM	32 o 48 KiB
RAM di serie	128 KiB
RAM massima	640 KiB
SO di serie	Sinclair QDOS
Altro software di serie	SuperBASIC, Quill, Archive, Abacus e Easel

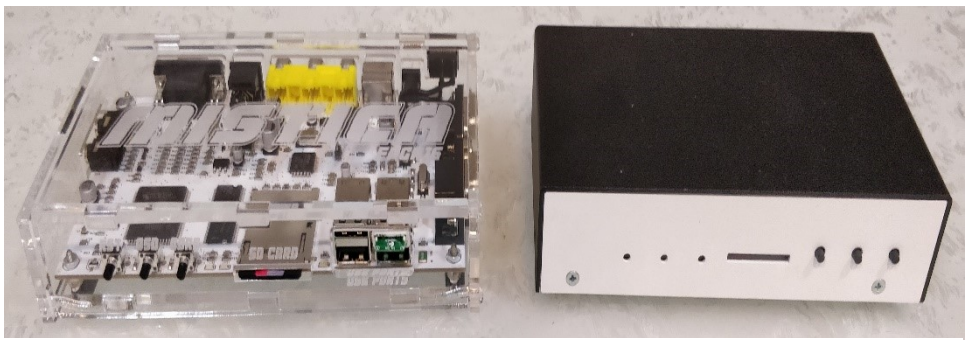
Oggetti MISTeriosi e da visione MISTICA

di Roberto Lari

In questo articolo vi voglio parlare di un paio di prodotti che sono sconosciuti ai più, ma che, secondo il mio parere, meriterebbero molta più attenzione poiché sono prodotti ideati e realizzati soprattutto per il mondo del retrocomputing e, di conseguenza, del retrogaming.

Si tratta dei sistemi MIST e MISTICA, due piccoli "computerini" (grandi all'incirca la metà di un mini PC ITX), ideati con lo scopo di riprodurre il funzionamento di computer e console a 8 e 16 bit degli anni '80 e '90. Persino diversi giochi arcade sono stati convertiti per questi apparecchi, tra cui l'ultimissimo (appena realizzato e pubblicato) e famosissimo Ghosts'n'Goblins. Nel momento in cui scrivo, è in fase di realizzazione dallo stesso autore anche il classico arcade 1942.

Vi chiederete: quali caratteristiche possiedono questi sistemi tanto da renderli così interessanti? Andiamo con ordine. Al loro interno batte un "cuore" speciale, un sistema completamente basato su FPGA (acronimo di Field Programmable Gate Array) ovvero un circuito integrato che all'accensione è sostanzialmente "vuoto" ma se opportunamente programmato, può contenere le informazioni hardware e software relative alla struttura di vari computer e console. I diversi firmware che si possono caricare sui sistemi Mist/Mistica sono definiti dal termine inglese "core" (nel senso di nucleo elaborativo). Per fare un rapido esempio, se carichiamo sul nostro sistema un *core* che contiene tutte le informazioni



Mistica a sinistra & Mist a destra

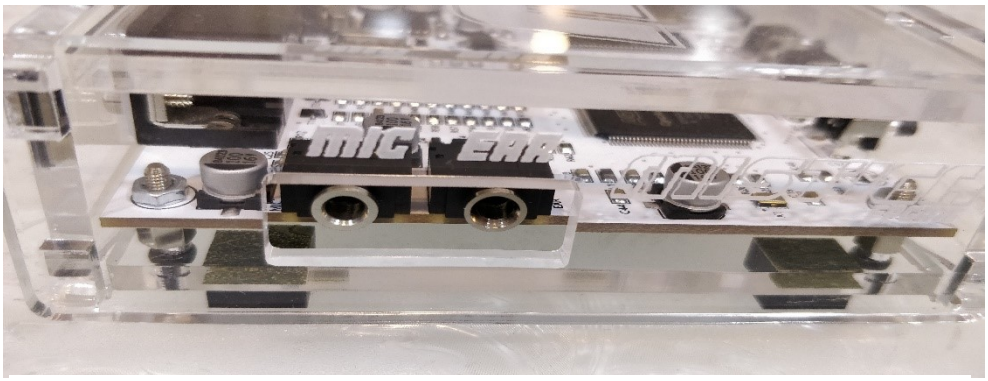
tecniche hardware relative a un Commodore 64 (e quindi CPU, SID, VIC-II, i 2 chip CIA, RAM, chip ROM, ecc.), il sistema riprodurrà fedelmente il funzionamento di un Commodore 64.

La preparazione e la programmazione dei vari *core* è opera di esperti tecnici e ingegneri che sfruttano le potenzialità di due diversi linguaggi di programmazione appositamente creati per le schede FPGA, Verilog e VHDL. Il caricamento dei file *core* nel circuito FPGA avviene attraverso una normalissima scheda SD (o MicroSD con adattatore). L'implementazione FPGA comporta che non si tratti di una mera emulazione software (come ad esempio avviene con un PC Windows/Linux su cui gira l'emulatore VICE o con un Raspberry PI e le varie distribuzioni basate su RetroPie o Lakka) ma piuttosto di una vera e propria emulazione hardware (io preferisco chiamarla "riproduzione hardware"). E allora quali sono i vantaggi di un'emulazione più a basso livello? Sono diversi, fra cui: nessun sistema operativo sottostante, nessun emulatore software che riproduca il computer o la console, nessun ciclo macchina sprecato da nessuno dei due e soprattutto nessun tempo di caricamento all'accensione o al passaggio tra un sistema riprodotto e un altro. Niente male davvero!

Uscite video del Mistica

Dal punto di vista dell'utente, poi, è sufficiente copiare su una SD card (normale o Micro) uno o più *core* dei sistemi (computer e/o console) che si vogliono riprodurre insieme ai nostri giochi e programmi preferiti per ciascun sistema emulato *et voilà!* il sistema è pronto all'uso! Naturalmente possiamo decidere di usare una sola SD card abbastanza capiente per tutti i sistemi oppure una singola SD per ogni computer/console che vogliamo riprodurre. Ovviamente sta a noi deciderlo. ☺ Tutto questo si può gestire usando il firmware di sistema del Mist, che viene aggiornato periodicamente.

Se ci limitassimo a quanto detto finora sarebbe già abbastanza, non credete? In realtà c'è molto altro: ad esempio la dotazione hardware dei sistemi Mist/Mistica è altrettanto importante. Entrambi sono dotati di 2 porte joystick DB9 ovvero quelle classiche in standard Atari. Di conseguenza possiamo facilmente collegare i nostri vecchi joystick e usarli senza problemi (oppure comprarne di nuovi come il Competition Pro che si trova in vendita ancora oggi). I sistemi Mist/Mistica sono dotati anche di ben 4 porte USB, permettendo di collegare *gamepad* moderni (tipo il clone cinese del gamepad del Super Nintendo) da usare con quei sistemi che necessitano di vari pulsanti di fuoco o di selezione (come le console NES o Megadrive). Si possono collegare senza problemi anche



ingresso ed uscita EAR & MIC del Mistica

tastiere e mouse USB (i sistemi Amiga, Atari ST, Macintosh e Archimedes lo richiedono, anche se Amiga accetta anche il mouse originale su porta DB9 ^_^).

Passando alle uscite video, i prodotti Mist/Mistica offrono uscite VGA ed RGB (tramite un cavo che termina con la classica presa SCART), ma sul Mistica, che per inciso è l'evoluzione dei sistemi Mist, hanno fatto di meglio: oltre a separare fisicamente i 2 connettori VGA ed RGB, hanno aggiunto anche l'uscita composita con i 3 cinch RCA (i classici 3 cavetti giallo, rosso e bianco) e perfino l'uscita S-Video. Per la nostra esperienza i risultati migliori si ottengono usando un classico TV a tubo catodico tra i 14 e i 21 pollici collegato via SCART.

Un'altra grande chicca di cui è dotato il Mistica sono le 2 prese jack da 3.5" EAR e MIC (purtroppo assenti nel Mist). In questo modo possiamo usare facilmente su alcuni dei retrocomputer riprodotti (come ZX Spectrum, MSX ed Amstrad CPC e chissà quanti altri) un vecchio registratore a cassette per leggere i giochi dai nostri nastri magnetici oppure uno *smartphone* con applicazioni tipo TapDancer e un file audio registrato sullo stesso (usando un banalissimo cavetto audio maschio-maschio con jack 3.5" da entrambi i lati).

L'aggiornamento dei diversi *core* avviene con una certa frequenza. Basti pensare che solo negli ultimi mesi ne sono stati aggiornati quasi una ventina. E attualmente si trovano *core* per riprodurre moltissimi dei sistemi home computer e console più famosi e diffusi: il C64, i vari modelli di ZX Spectrum, Amiga dai modelli 500 fino al 1200 e persino diverse macchine per lo più sconosciute (prodotte in Russia, Ungheria o altri paesi extra-europei). Ogni singolo sistema (*core*) presenta il suo menù con le funzioni disponibili, ad esempio il classico C64 presenta la scelta del modello di SID chip da usare, quale gioco o programma caricare (le scelte tipicamente comprendono i file normalmente utilizzati dagli emulatori software come PRG, D64, ecc.), l'opzione video PAL o NTSC e molto altro. In base al sistema che stiamo emulando i formati dei file cambiano (ADF per Amiga, DSK per Amstrad, TAP per ZX Spectrum, ecc. Dopo un aggiornamento del *core* stesso, alle funzioni già esistenti ne possono venire aggiunte delle altre (ad esempio la scelta di quale modello di SID chip da usare sul C64 è un'implementazione molto recente). Sul *core* di Amiga è persino possibile usare un hard disk virtuale da 4GB (da creare però su un altro sistema tipo WinUAE) e avviare il Workbench da lì invece che da un ADF montato come DF0:.

Installazione e primi passi

Adesso cerchiamo di capire come installare un sistema del genere partendo da zero. Ecco l'occorrente per cominciare (nel seguito mi riferisco specificatamente ad un sistema Mistica, consigliato perché si tratta dell'evoluzione del Mist, ormai fuori produzione):

- 1 sistema Mistica completo di alimentatore e cavo di alimentazione
- 1 scheda SD o MicroSD formattata in FAT32 per quasi tutti i sistemi oppure 1 scheda SD o MicroSD formattata in FAT16 per MSX, PC Engine e/o eventuali altri sistemi che non accettano la formattazione FAT32 (nota: FAT16 e FAT32 sono gli unici tipi di formattazione accettati dal sistema)
- 1 Tastiera USB
- 1 cavo video VGA o RGB-SCART o composito in base allo schermo al quale decideremo di collegare il nostro gioiellino

Attenzione anche al fatto che si possono usare anche SD grandi anche fino a 200 GB ma queste richiedono di essere formattate tramite un programma particolare (personalmente utilizzo FAT32FORMAT su Windows, gratuito e liberamente scaricabile da Internet). Un alimentatore di quelli comuni da smartphone (il cavetto è in dotazione) va più che bene, anche quelli con amperaggio limitato (ok per quelli da 1A o poco meno).

Questo è il numero minimo di componenti realmente necessario per poter iniziare, ma opzionalmente possiamo aggiungere:

- 1 o 2 joystick in standard Atari DB9
- 1 mouse USB
- 1 o più gamepad USB per le console che non accettano i joystick DB9
- 1 o più altre schede SD o MicroSD se decidiamo di riprodurre più sistemi
- 1 registratore a cassette con cavo jack e giochi su cassetta

Prima di tutto, dopo aver preparato una SD card con la giusta formattazione, dovremo copiarvi dentro uno o più *core* relativi alle macchine che abbiamo deciso di riprodurre. Se il *core* sarà uno solo (ad esempio il Commodore 64), potremo rinominarlo CORE.RBF affinché parta automaticamente quando accendiamo il Mistica. Esiste però anche la possibilità di scaricare uno speciale *core* realizzato appositamente per avere a disposizione tutti gli altri *core* presenti nella SD. Questo speciale firmware, che farà da menù di scelta per gli altri, dovremo rinominarlo anch'esso CORE.RBF, mentre tutti gli altri potranno avere il nome con cui sono distribuiti (vedi foto esplicativa).

Alcuni dei *core* che riproducono i nostri amati retrocomputer o console richiedono dei file aggiuntivi per funzionare (ad esempio il *core* di Amiga richiede almeno 1 file di Kickstart ROM memorizzato col nome KICK.ROM), mentre per la maggior parte dei sistemi il *core* base da solo sarà più che sufficiente per emulare la relativa macchina. Naturalmente occorre prestare attenzione al fatto che alcuni file aggiuntivi (proprio come il Kickstart di

Amiga) nonostante l'età sono coperti da copyright e quindi occorre possedere una licenza o il computer/console per avere diritto ad usare il firmware in versione binaria assieme al *core* per Mistica.

Tutti questi file "di sistema" vanno copiati rigorosamente nella radice (*root*) della scheda SD per permettere al Mistica di riconoscerli ed usarli (tranne qualche file delle ROM dei sistemi Atari 5200 e 800). Il passo successivo consiste nel copiare i giochi veri e propri nei formati che il *core* è in grado di riconoscere (ad esempio il *core* per C64 riconosce i file con estensione PRG, le immagini disco D64 e le cartucce CRT). Per memorizzare adeguatamente i giochi ed i programmi, soprattutto quando si utilizza una scheda SD *multi-core* e quindi multi-sistema, suggerisco di creare una gerarchia nel filesystem come ad esempio:

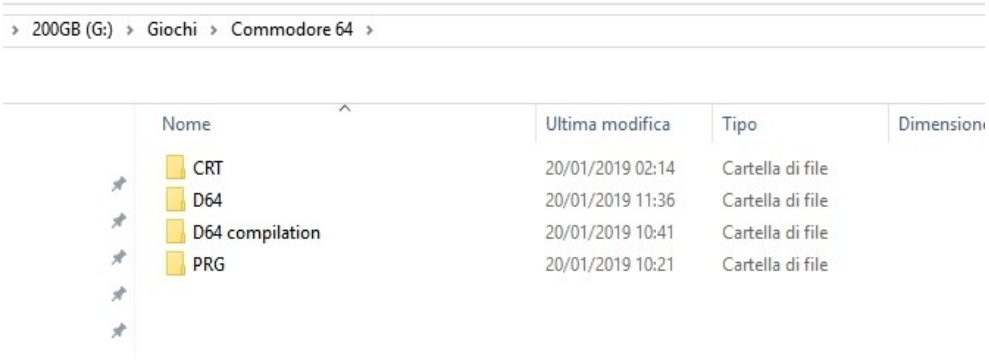
- nella radice della SD creare una cartella chiamata GIOCHI

- nella cartella GIOCHI creare una cartella C64

- nella cartella C64 creare le sottocartelle PRG, D64 e CRT

- all'interno di ogni sottocartella creare le sotto cartelle divise per ordine alfabetico o altro

Infine basta copiare all'interno di ogni singola sottocartella i giochi suddivisi nell'ordine che più ci aggrada (ad es. nel caso dell'ordine alfabetico nella cartella A inseriremo Arkanoid, nella B copieremo Boulderdash e cosi via). In questo modo avremo sempre tutto a portata di mano e potremo intervenire



Esempio di come suddividere le tipologie di file accettati

in qualsiasi momento per aggiungere o rimuovere i titoli software in base alle nostre preferenze.

Nel caso volessimo utilizzare un classico TV Color a tubo catodico (CRT), è necessario creare nella radice della scheda SD un piccolo file di configurazione chiamato "mist.ini", all'interno del quale scriveremo le seguenti 2 righe:

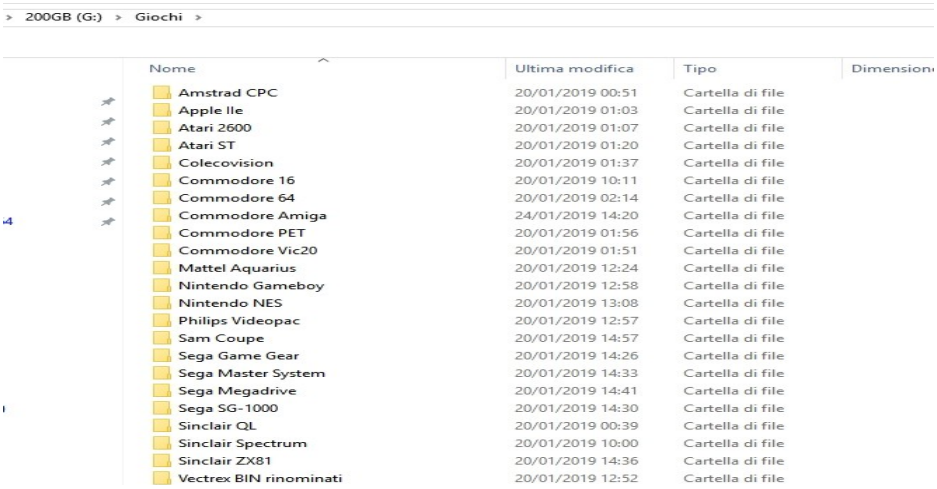
[mist]
scandoubler_disable=1

Questo file informa il Mistica/Mist che vorremo utilizzare la frequenza di aggiornamento video verticale pari a 15Khz, necessaria la piena compatibilità con i televisori CRT.

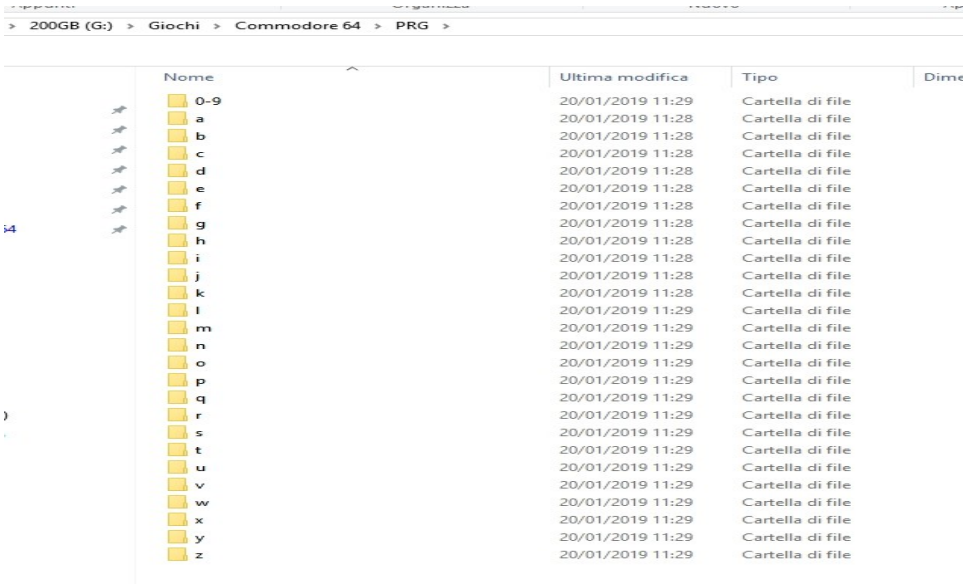
Dopo aver finito di aggiungere *core*, file di configurazione e giochi sulla SD, possiamo tranquillamente espellerla dal nostro PC (non dimenticate di effettuare la rimozione sicura!) e inserirla nello slot SD del Mistica/Mist. Se abbiamo fatto tutto correttamente, all'accensione dovremo ritrovarci direttamente con la schermata principale del computer o console prescelta, nel caso di un sistema singolo, oppure il menù con tutte le macchine disponibili nella SD nel caso avessimo deciso di optare per un multi-sistema.

Una volta che avremo il nostro computer funzionante e pronto all'uso (ad es. la prima schermata blu/azzurra del C64), premendo il tasto F12 sulla nostra tastiera USB (o il tastino centrale sull'apparecchio stesso) comparirà una piccola schermata chiamata OSD, specifica del *core* in uso in quel momento, con tutti i comandi e le opzioni a disposizione. Alcune di esse servono per caricare un gioco, altre per modificare le impostazioni previste per quello specifico *core*, altre ancora riguardano funzioni hardware (ad esempio è disponibile l'opzione per effettuare lo scambio (*swap*) delle porte joystick, utile per non spostare fisicamente il joystick da una porta DB9 all'altra).

Per caricare e lanciare l'esecuzione di un gioco dovremo sottostare alle regole di funzionamento del computer o console originale. Ad esempio nel caso dell'emulazione di un C64, in base al formato del file che vogliamo caricare, dovremo agire in modo differente: se viene selezionato un file PRG, allora basterà semplicemente scrivere RUN seguito dal tasto Return per far partire il gioco; nel caso di una cartuccia la partenza del gioco sarà automatica; se infine viene selezionato un file D64, il sistema "monterà" un dischetto nell'unità drive virtuale e quindi sarà necessario digitare i classici comandi per operare con un floppy drive (LOAD"$",8 per visualizzare la directory del disco, LOAD"*",8,1 per caricare in memoria il primo programma memorizzato nel dischetto e così via). Naturalmente, come detto, le cose cambiano da sistema a sistema



Esempio di come impostare le cartelle sulla scheda SD



Divisione in cartelle divise in ordine alfabetico

e, ovviamente, è richiesta una conoscenza della macchina reale che stiamo emulando.

Conclusioni

La mia esperienza personale con questi apparecchi è davvero soddisfacente tanto è vero che posseggo ben 2 Mist e 1 Mistica. Li uso regolarmente e in pianta stabile su un TV CRT 21" Sony Trinitron, su un monitor VGA Philips 17" CRT e su un Philips CM8833 14" CRT (sì proprio quello che si usava e usa tuttora con i computer Amiga originali ☺). Secondo la mia opinione questi sistemi (che sinceramente reputo il futuro del retrogaming a lungo termine), esattamente come le macchine reali che riproducono, non sono adatti per un utilizzo con TV moderne LED/LCD con risoluzione HD, Full HD o peggio ancora 4/8K. L'emulazione hardware fa sì che, come per tutti i sistemi a 8/16bit di un tempo, è molto meglio procurarsi un bel TV a tubo catodico di buona qualità tra i 14 e i 21 pollici compatibile, solitamente tramite SCART, con lo standard RGB. Quindi tirate fuori dal vostro armadio 1 o 2 joystick standard Atari e sarete pronti con la mente e col cuore a viaggiare indietro negli anni 80/90 per giocare di nuovo con i titoli con cui noi tutti siamo cresciuti (Hero, R-Type, Gyruss, Dig Dug, Impossible Mission e mille altri). La giocabilità ed il divertimento di allora saranno riprodotti fedelmente dai sistemi Mist/Mistica per molti anni a venire, perché noi appassionati di retrocomputing e in particolare di retrogaming, siamo stati, siamo tutt'ora e saremo sempre quegli adolescenti che hanno scoperto l'informatica fatta di passione e di amicizia (e anche di qualche litigio "amichevole" con i compagni di scuola quando si giocava uno contro l'altro). Passione che in cuor loro non hanno mai abbandonato. Quindi viva i sistemi Mist/Mistica e... viva la FPGA!

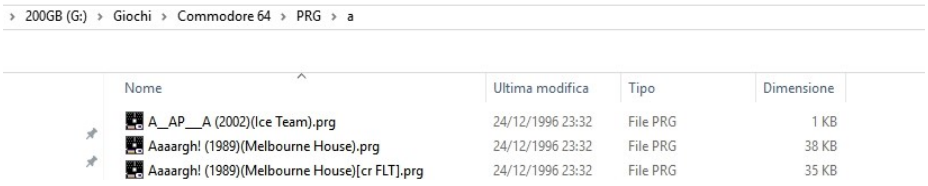
Riferimenti:

https://tinyurl.com/obzwy
(download di FAT32Format)

https://tinyurl.com/ydebfdkb
(produttore e venditore del Mistica)

https://tinyurl.com/y7h55dvj
(forum di discussione e richiesta di aiuto)

https://tinyurl.com/ybkwwp27
(sito ideato, realizzato e curato da Vincenzo Scarpa e Roberto Lari per racchiudere il più possibile in una sola pagina web tutti i link alle risorse disponibili per il Mist / Mistica, download dei firmware, core, ecc.)



Alcuni file d'esempio in una sottocartella

https://tinyurl.com/y7f2bwty
(il file .RBF che se copiato sulla SD come core.RBF ci mostra all'accensione tutti i core presenti sulla SD stessa, permettendoci di scegliere comodamente quale sistema far partire)

https://tinyurl.com/yapjuc3u
(il firmware aggiornato quando viene rilasciato. Attenzione: va salvato sulla SD col nome firmware.upg senza la data)

https://tinyurl.com/ycjapgwf
Gruppo italiano interamente dedicato ai sistemi FPGA Mist, Mistica & Mister

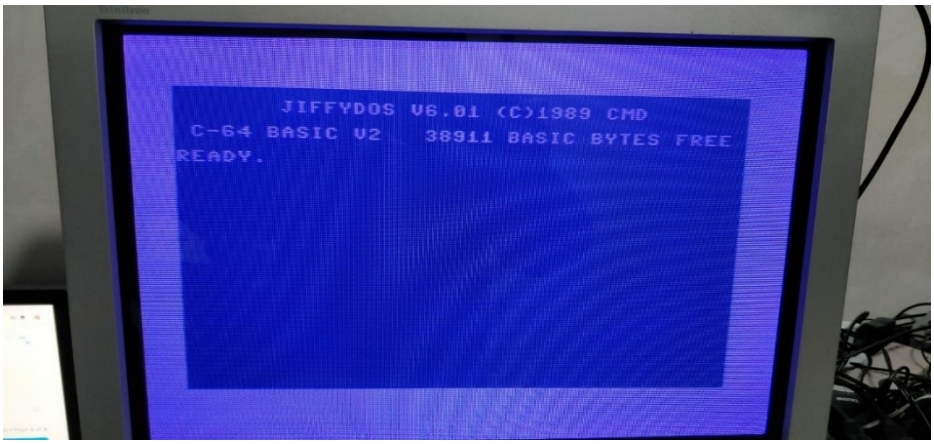
Scheda tecnica del Mistica in breve:
FPGA EP3C25E144 (Cyclone III)
256 Mbit SDRAM (32MB)
ARM AT91SAM7S256

Connessioni e porte I/O:
Slot per scheda SD o MicroSD con adattatore
4 porte USB
2 porte DB9 per joystick standard Atari
Uscita VGA
Uscita VIDEO RGB tramite connettore DIN 9
Uscita audio / video composita RCA (3 cinch, video, audio sinistra e destra)
Uscita S-Video
Jack da 3,5 mm per lettura dati da apparecchio esterno
Jack da 3,5 mm per registrare dati su apparecchio esterno (es. registratore a cassette)
Connettore femmina per cavo alimentazione (fornito)
Pulsante reset generale
Pulsante reset del sistema riprodotto
Pulsante apertura del menù o, se tenuto premuto, di passaggio tra uscita video VGA e RGB

Joystick che è possibile collegare

Mist & Mistica all'accensione (usando il core menù)

OSD del Commodore 64 con le varie opzioni (di caricamento e non)



Core del Commodore 64 appena avviato

Un gioco in funzione. In questo caso è una immagine .crt

Elenco sistemi emulati con un sistema Mistica e formato file accettato dal relativo core:

Amstrad CPC 6128 (DSK , *.E??)
Apple IIe (NIB)
Apple Macintosh+ (DSK Floppy disk, IMG Hard disk)
Atari 2600 (A26, BIN, normale e superchip)
Atari 5200 (A52)
Atari 800 (ATR, CAR, XFD, XEX)
Atari ST (ST)
BBC Micro (ROM)
ColecoVision (COL, BIN, ROM)
Commodore 64 (D64, PRG, CRT)
Commodore Amiga (ADF Floppy Disk, HDF Hard Disk)
Commodore 16 / Plus4 (PRG, D64)
Commodore Pet 2001 (TAP, PRG)
Commodore Vic 20 (PRG, CRT, D64)
Mattel Aquarius (BIN)
MSX 1 & 2 (DSK, ROM)
Nintendo Gameboy (GB, GBC, SGB)
Nintendo NES (NES)
PCEngine (BIN, PCE, SGX)
Philips Videopac (BIN)
Sam Coupè (DSK)
Sega Master System (BIN, SMS)
Sega SG-1000 (SG)
Sega Game Gear (GG)
Sega Megadrive (BIN, GEN, MD)
Sinclair QL (MDV)
Sinclair Spectrum +2 / +3 / 48K (TRD, IMG, DSK, MGT, TAP, CSW)
Sinclair Spectrum 48K only (SNA)
Sinclair ZX81 (O, P)
Vectrex (BIN)

Altro controller adoperabile

Non comprate mai un Vectrex!

di Robin Jubber (traduzione ed adattamento a cura di David La Monaca/Cercamon)

Non comprate mai un Vectrex! Certo, senza un Vectrex la vostra collezione di console è soltanto un mucchio di rifiuti di plastica e poi sì, avete ragione, si tratta della più grande console di tutti i tempi, ma, lo ripeto, non comprate mai un Vectrex! Io ne ho comprato uno tempo fa ed ora ne ho tre. Queste vitali informazioni sono naturalmente state nascoste alla mia fidanzata e dal momento che il suo italiano è terribile almeno quanto il mio, se dio vuole, potrebbe non scoprirlo mai. Alla fine mi sono ritrovato con una pila di costosi giochi homebrew, una tonnellata di periferiche stravaganti e di recente ho persino scritto un gioco – beh, in realtà 14 giochi diversi – per questa console ed ora la mia casa è piena di imballaggi, saldatori e cartucce di plastica. In pratica, sono diventato una software house con produzione domestica di cartridge per Vectrex. Lo so, avrei dovuto comprare un Neo Geo come una persona sana di mente.

Se non avete dimestichezza con il Vectrex, non sentitevi in colpa. Questo sicuramente significa che non siete un vero collezionista di console, non sapete nulla sul retrogaming e soprattutto che non dovreste leggere questa rivista, ma significa anche che il vostro saldo in banca probabilmente non è un enorme buco fumante nel terreno. Il Vectrex è una bestia rara ed insolita ed è l'unica console non portatile con lo schermo integrato. Ed è anche l'unica che utilizza una grafica basata su vettori al posto dei pixel. È una delle poche pregiate console per cui un secondo controller costa praticamente quanto la console stessa (ecco perché ho comprato il terzo esemplare, cari miei). La grafica è a colori, ma solo nel senso più bizzarro che possiate immaginare. È stata la prima console per la quale nel 1981 si poteva acquistare come accessorio addirittura degli occhiali 3D! Ogni console in genere è dotata di una grafica leggermente fuori dagli schemi, ma in questo caso il sistema grafico è davvero unico. La console emette uno strano ronzio anche con il volume abbassato a zero. Aprirlo per effettuare riparazioni può uccidervi, letteralmente. Ha una specie di maniglia per il trasporto, presumibilmente in modo che si possa portarla molto, molto attentamente a casa di un amico, ma la maniglia per il trasporto è difettosa e inclinata, quindi è molto probabile che scivoli via dalle vostre mani con conseguenze nefaste e costosissime. Il Vectrex è rimasto in produzione soltanto per circa un anno e mezzo. Ha solo circa 800 byte di memoria RAM ed è, senza alcun dubbio, la più grande console di tutti i tempi.

Figura 1 - Il Vectrex in azione!

La prima cosa che noterete è lo schermo. Essenzialmente si tratta di un vecchio monitor CRT in bianco e nero girato di lato ed il display vettoriale non ha eguali sul mercato delle console. La semplice grafica basata su linee salta fuori sullo schermo in un modo che non può essere facilmente replicata da un emulatore. La vostra scheda grafica da 8 GB non è in grado di gestire una grafica come questa. I lettori più anziani forse ricordano di aver giocato la versione vettoriale originale di Star Wars nelle sale giochi – e nel Vectrex è stata sostanzialmente integrata la stessa tecnologia, strizzata all'interno della scatola che troneggia sulla vostra scrivania. Dato che la grafica è così insolita e non usa affatto i pixel, il mio secondo errore, dopo l'acquisto, s'intende, è stato quello di provare a scrivere un po' di codice. Pensavo che avrei potuto disegnare un semplice triangolo sullo schermo. Poteva essere divertente. Poi, durante le ore dedicate al mio vero lavoro, tornavo a scrivere giochi veri e propri su macchine con milioni di kilobyte di RAM invece che su una con nemmeno 1 KB intero. Una veloce ricerca su Internet ha rivelato alcuni frammenti di informazioni, come il breve testo di Christopher Tumber risalente al 1998 che fa luce su alcuni aspetti della macchina e dell'assembly per 6809. Scoprii che avevo bisogno dell'assembler as09 per trasformare il codice in binario e di ParaJVE, un emulatore Vectrex non del tutto accurato o finito. Si può anche scrivere il codice per EEPROM e testarlo in questo modo, ma si rischia di morire di vecchiaia prima di mettere in funzione un semplicissimo programmino che mostri il classico "Hello World". Esiste anche, ed è pazzesco, un ambiente di sviluppo completo creato da un adorabile tizio di nome Malban, ma non volevo averci a che fare – in fondo, per me questo doveva soltanto essere un esperimento rapido. Ma abbiamo già stabilito che potrei essere un idiota, quindi...

La CPU 6809 che si trova nel cuore della macchina è un meraviglioso microchip, che è presente solo nei sistemi embedded: il Vectrex, il Dragon 32 e il TRS-80. Ed io penso che sia un peccato. L'architettura è big endian, il che significa che ci si può fare sopra un sacco di matematica a 16 bit, che è molto più facile da capire per gli esseri umani. L'architettura little endian, che è presente nelle CPU rivali a 8-bit, permette alle unità logiche semplici di gestire un byte alla volta e questo mantiene bassi i costi di produzione, ma non rende il codice necessario leggibile o intuitivo. Anche se è una CPU a 8-bit, il 6809 possiede anche molte funzioni a 16-bit. Ha due registri a 8 bit che si combinano per creare una word a 16 bit, insieme a una serie di altri registri e istruzioni a 16-bit. La maggior parte del codice che scriverete su una 6809 sarà a 8 bit, le solite attività di confronto fra valori bassi e cicli su piccole strutture di dati - ma quando avrete bisogno di scrivere funzioni matematiche a virgola fissa, il 6809 entra davvero in gioco. Per il mio primo esperimento, un semplice clone di Pong, questa caratteristica non era poi così importante. Con il terzo gioco, invece, un semplice clone di Spacewar, era diventata vitale per rendere fluido i movimenti. In sostanza, se non ci si vuole limitare alla matematica dei numeri interi – altrimenti, è possibile muovere solo un numero discreto di

unità per ogni fotogramma - poiché il Vectrex comprende solo 256 unità in verticale e in orizzontale, ciò significa che la velocità minima di movimento coprirebbe l'intero schermo in pochissimi secondi. allora si usano valori a 16 bit e considera il primo byte come coordinata effettiva ed il secondo byte come parte frazionaria. Tutto molto facile con un 6809.

Figura 2 - Disegnare con il Vectrex

Il problema nel programmare un semplice clone di Pong è che avevo ancora circa 25K di spazio libero sulla ROM. Ovviamente avrei potuto rilasciare il file binario e andare avanti con il lavoro di tutti i giorni, ma ormai ero "agganciato" e mi ero fissato a cercare di capire dove poteva arrivare il Vectrex ed a imparare un nuovo linguaggio. Inoltre, Joanna, la mia bambina, era appena comparsa dal nulla, quindi, tecnicamente, mi trovavo nella condizione di congedo di paternità autoimposto. Così ho scritto un altro gioco. E poi un altro. Quindi ho rimescolato un po' la memoria a disposizione ed ho notato che alcune routine venivano usate dappertutto e così ho ricavato ancora un po' di spazio ROM libero.

E così l'ho usato per scrivere un altro gioco. A questo punto l'iniziale ROM da 32K, che è il massimo indirizzabile da un Vectrex, si stava avvicinando al limite, così i miei quattro o cinque semplici giochi stavano per finire su Internet e io potevo andare oltre, dopo aver grattato quel prurito tipico che riguarda la programmazione. Avevo anche parlato delle mie avventure sul Vectrex ad altri esperti programmatori che creano giochi per questa console fin dai tempi oscuri e sapevo che erano molto gelosi. Niente tempi di compilazione. Nessun produttore col fiato sul collo. Nessun bisogno di spiegare in modo gentile ad artisti dalla mente semplice perché le loro opere grafiche sono tutte corrotte ed inutili. Solo rozzo codice assembly, scritto direttamente sul "metallo" – solo vere sessioni di programmazione di giochi come si faceva una volta, quando gli uomini vivevano ancora nelle caverne.

Giunto a questo punto le cose hanno preso una brutta piega. Ho scoperto che qualche ingegnoso coder aveva capito come fare bank-switching sul Vectrex. In questo modo si potevano avere 64K di codice e dati, a condizione di essere stati molto attenti al progetto iniziale. Se il primo gruppo di istruzioni sono le stesse per entrambi i banchi di memoria, si può inviare un segnale preciso in una locazione dell'hardware Vectrex e passare magicamente al banco 2. O tornare indietro al banco 1. Usando la stessa tecnica, si può anche scrivere su una EPROM a 32 byte, dando in pratica al Vectrex la possibilità di salvare e caricare. Beh, ovviamente dovevo sperimentare un po'. Altri 32K con cui giocare? Il mio progetto poteva diventare un compendio di giochi per due giocatori. I giochi per due giocatori comportano un paio di vantaggi ben definiti per il coder pigro e incompetente. Prima di tutto il Vectrex semplicemente non presenta molta disponibilità per giochi a due giocatori, nonostante le due porte joypad. Molti giochi permettono l'alternanza dei player, ma se volete mostrare il vostro nuovo Vectrex ad un amico e lasciarlo senza parole, avete bisogno di due giocatori simultanei. L'altro vantaggio è che con questi giochi non si deve scrivere molto codice in termini di intelligenza artificiale. E pensare che il mio insegnante di informatica temeva che non avessi la mentalità giusta per diventare un programmatore. Che sciocco! Certo, se avessi fatto le cose a modo suo, avrei scritto software gestionale per una banca e ora guiderei una Lamborghini, invece di una BMW di 13 anni fa che ama spegnersi in mezzo alla strada senza motivo apparente mentre tutti intorno mi indicano e se la ridono. Ma, d'altro canto, non saprei come muovere le astronavi su uno schermo, quindi direi che si tratta di un giusto compromesso. Almeno credo.

Ormai avevo completato 6 o 7 giochi, compreso il mio primo pezzo di codice propriamente complesso necessario per un gioco di artiglieria a schermo singolo tipo Worms. Poi è arrivato un gioco tipico di macchine con vista dall'alto e poi Tron. Il Vectrex è tutto incentrato sull'uso di linee grafiche, quindi Tron sembrava una conseguenza naturale. Questo in teoria. La particolarità del Vectrex è che non possiede "grafica persistente". Niente di ciò che si disegna sullo schermo rimane sullo schermo per più di un fotogramma - tutto deve essere ridisegnato 50 volte al secondo, completamente. Nessun altro hardware funziona allo stesso modo, anche se bisogna dire che i moderni giochi 3D devono praticamente affrontare lo stesso problema. Ci sono 30.000 cicli di CPU a disposizione per disegnare tutto sullo schermo e poi l'intero processo deve ricominciare dall'inizio, dato che le linee del fotogramma precedente svaniscono rapidamente dalla vista. Il Vectrex disegna la grafica un po' come fa un oscilloscopio - il fascio di elettroni lascia una scia ma questa non persiste a lungo. Anche lo scaling delle linee disegnate introduce ritardi e persino la stampa di testo risulta particolarmente difficile.

Figura 3 - Il Vectrex in tutto il suo splendore

Le routine del sistema operativo per la visualizzazione di stringhe di testo non sono per niente veloci e diventano sempre più distorte al crescere del numero di lettere visualizzate. I 30.000 cicli a disposizione sono costantemente sotto pressione. Nello sviluppo di Tron non riuscivo a scrivere il gioco nel solito modo, aggiungendo un pixel alla testa del "serpente" e semplicemente cancellando un pixel dalla coda. Ho dovuto invece memorizzare tutte le linee, compreso

il caso speciale delle linee che sono più lunghe di 128 unità (il massimo consentito sul Vectrex) e continuare ad estendere la linea in avanti mentre il giocatore si muove in giro per lo schermo. Questo insieme di linee doveva poi essere impacchettato e consegnato al sistema operativo come immagine vettoriale completa, in modo che potesse essere disegnata istantaneamente per entrambi i giocatori. Beh, nessun problema – ho pensato – potevo ottenere ancora qualche centinaio di byte liberi se pulivo la RAM che è usata da altri giochi ma di cui Tron non ha bisogno.

Questo significava che, con questo trucco, la gestione dei missili, l'effetto dell'esplosione, il sistema di collisione a blocchi e quello delle particelle - tutti potevano essere riproposti. Mi serviva solo un sistema di collisione che gestisse l'incrocio delle linee ed una coda gigante per gestire le estremità della linea. E anche due di tutto perché ci sono due giocatori. Oh mio Dio, questo sì che è brutto codice - ma funziona.

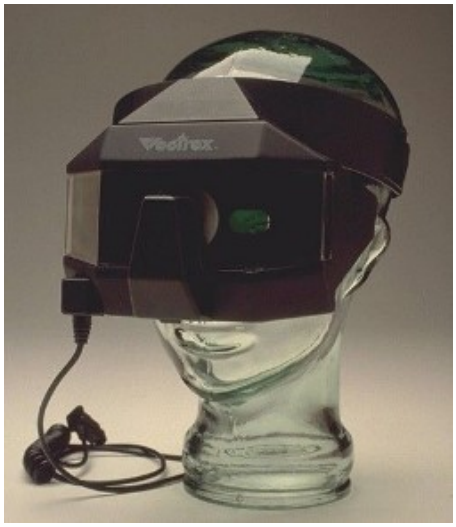


Figura 4 - Il casco Vectrex 3D Imager

Eppure... c'è ancora tanto spazio sulla ROM da riempire. E farei meglio a riempirlo velocemente o mia figlia sarà partita per l'università nel momento in cui questa cartuccia sarà completa. Così ho preso il codice di Spacewar, ho aggiunto l'effetto gravità e ho costruito una serie di caverne per far navigare le due astronavi. Ed ecco un gioco co-op (cooperativo, non coin-op!) sul Vectrex. E con 100 stanze da esplorare! Quelle 100 stanze mi hanno quasi ucciso – la progettazione dei livelli è troooooppo noiosa, ma alla fine funzionava bene. Non vi annoierò con i bug critici, con il rendersi conto che il salvataggio e il caricamento richiedono un bank switch, con le Easter eggs che ho inserito qua e là e che hanno quasi mandato in malora tutto quanto e con la quantità di dati da spostare necessaria per infilare 5 lingue di traduzione nella cartuccia. Ho anche fatto controllare a Malban il codice - "è molto buono per un primo gioco, ma probabilmente non avresti dovuto scrivere qualcosa di così esteso e qui di seguito ci sono tutte le cose sbagliate del codice" - ha passato in rassegna circa 22mila linee di assembly alla ricerca delle parti in cui ho incrementato un loop invece di diminuirlo (risparmiando così 1 o 2 byte), ho caricato due registri separatamente invece di uno (risparmiando 1 byte). Quest'uomo è un genio folle e non mi metto certo ad ignorare un aiuto come il suo - tutto quanto serve. I byte e i cicli di clock sono preziosi quando si scrive codice "nel passato". Malban mi ha anche aiutato a capire il rapporto tra la dimensione dei vettori e il loro tempo di tracciamento, il che è stato inestimabile per far funzionare il sistema di menu senza perdere fotogrammi.

Così il gioco è finito, tranne che per lo screensaver Black Vector nascosto che ho scritto all'ultimo minuto e per il messaggio segreto destinato alla mia ragazza (perché si sa, ogni ragazza sogna di ricevere delle flebili scuse attraverso una vecchia console) e gli altri Easter eggs che ho aggiunto perché ho visto alcuni byte liberi che se ne stavano in giro senza fare nulla. Ora sono arrivato a 14 giochi, 6 screensaver, 10 utility, 6 impostazioni di gioco, 5 lingue, 11 versioni alternative per i giochi principali, 3 codici stagionali pronti per sbloccare i contenuti bloccati sulla cartuccia, ma provo quella sensazione di irrequietudine in cui il progetto è formalmente finito eppure si ha voglia di scrivere ancora un po' di codice ma non c'è niente da scrivere e da nessuna parte in cui metterlo comunque!

E questo è solo l'inizio dei miei problemi. Nel momento in cui scrivo, la mia casa è diventata un centro di produzione perché per realizzare correttamente un gioco Vectrex è necessario creare le cartucce, effettuare il burn delle ROM, saldare i componenti sui circuiti, trovare un posto che produca delle scatole di cartone, realizzare la grafica necessaria, scrivere il manuale e curare la sua traduzione (grazie a tutti coloro che mi hanno aiutato in quest'operazione), disegnare poster per ricreare quella sensazione anni '80, modificare il sito web e questo prima ancora di arrivare a confezionare tutto e parlare con tutti coloro che hanno prenotato il gioco. Ho anche dovuto trovare un simpatico produttore di materie plastiche e spiegargli attentamente cos'è un Vectrex e perché avevo bisogno di un pezzo di policarbonato colorato a due strati con un bel motivo grafico sopra. Il mio primo passo verso tutto questo è stato l'acquisto di una stampante 3D per realizzare le cartucce. Quella di prova, la numero 1, assomigliava a quel pezzo nel film Judgement Day in cui T-1000 sta cercando di riformarsi dopo che Linda Hamilton lo ha colpito con un fucile da caccia. Il mio tentativo successivo assomigliava agli effetti speciali del classico fantascientifico "The Thing". Sono un programmatore - non so niente sulla plastica! Ora compro le cartucce in America, dove devo solo pagare un sacco di soldi.

Siamo arrivati alla fine di questo articolo. Spero davvero che la sua lettura vi abbia fornito un'idea chiara del **perché non dovreste comprare un Vectrex**. Dico sul serio. Non avete idea di quello in cui vi caccereste. Anche se è, senza dubbio, la più grande console mai realizzata! ;-)

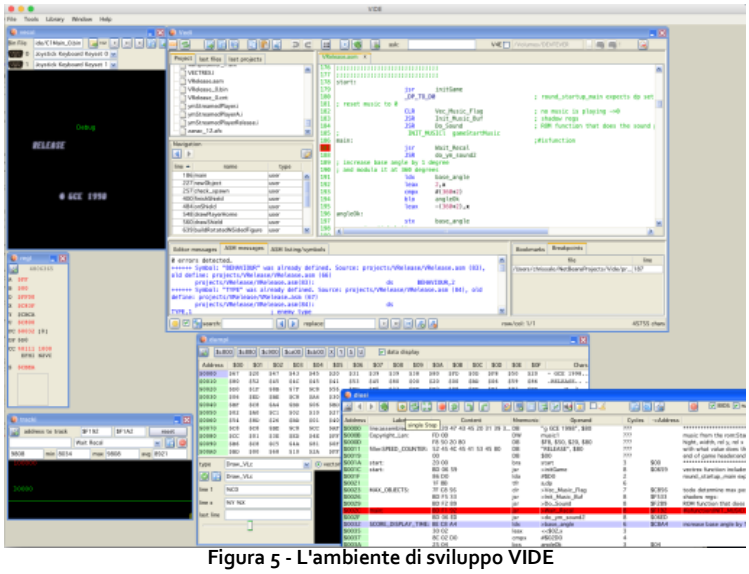


Figura 5 - L'ambiente di sviluppo VIDE

Programmare in assembly per Vectrex

Qui di seguito i passi minimi da compiere per programmare il classico "Hello World" per Vectrex usando un cross-compiler su Windows:

1. Scaricate as09.exe dall'indirizzo seguente (**link**) oppure procuratevi **VIDE** (un ambiente integrato) pubblicato da Malban, che contiene già un assembler.
2. Scegliete un IDE. All'inizio ero partito con Visual Studio ed ho creato una regola in **Tools > External Tools**. Aggiungere un nuovo tool esterno e chiamatelo Vectrex Assembler. Nel campo **Command** inserite eseguibile e percorso dell'assembler (es. c:\ovunque\as09.exe) e nel campo **Arguments** usate **–q $(ItemFilename)**. Abilitate **Use Output Window** e lasciate le altre opzioni vuote. E' comodo aggiungere anche una scorciatoia da tastiera, di solito attivabile nel menu **Options**, per eseguire l'assembler sul file corrente. Io utilizzo il tasto **F6**, ma i vostri gusti potrebbero essere diversi. Naturalmente potete saltare tutta questa parte ed usare VIDE, che include pulsanti già pronti per compilare sorgenti in codice 6809.
3. Fate girare il vostro codice. L'assembler AS09 produrrà un file .bin (eseguibile binario) che potete caricare direttamente con **ParaJVE**, un emulatore Vectrex. ParaJVE può anche gestire il bank switching ma non le funzioni di salvataggio/caricamento, anche se queste sono opzioni che la maggior parte dei giochi per Vectrex non utilizza o di cui non ha bisogno. Anche in questo caso, VIDE contiene un emulatore all'interno dell'ambiente di sviluppo ed è più accurato per controllare il framerate ed individuare i problemi di tracciamento della grafica.
4. Scrivete un po' di codice. Ogni gioco per Vectrex presenta una intestazione, che verrà mostrata dal sistema operativo della macchina al suo avvio. Il vostro codice inizia proprio la parte occupata dall'intestazione. Un esempio lo potete trovare nel Listato 1.
5. Diventate assuefatti all'assembler 6809 e osservate la vostra vita sociale disintegrarsi!

Di seguito alcuni utili suggerimenti per lo sviluppo per Vectrex, che avrei voluto conoscere prima di cominciare la mia avventura di programmazione per questa console.

- Le routine del sistema operativo per stampare stringhe di testo non funzionano senza ragione se la stringa da visualizzare è lunga un solo carattere. Non avrete alcuna idea del perché.
- Più lunga è la stringa, più grande è la sua distorsione sullo schermo. In molti casi è meglio far scorrere una stringa che visualizzarla come una pagina di testo.
- Quando si progetta la grafica, per esempio con Blender, cercate di costruire la vostra animazione vettoriale più ampia possibile – in generale anche per riempire lo schermo – e poi ridimensionatela per visualizzarla. Minore la scala di una lista di vettori, minor numero di cicli ci vorrà per la visualizzazione. Ci sono sottili aspetti di cui tener conto quando si applica questa guida, ma essenzialmente si tratta di una buona regola da seguire.
- ParaJVE disegna la grafica come se tutto girasse su un Vectrex magicamente perfetto. Nessun animale simile esiste in natura.
- Qualunque tentativo di salvare o caricare dati ha come conseguenza un bank switch. Mettete il vostro codice EEPROM, di inizializzazione e di bank switching nelle stesse locazioni all'interno dei rispettivi banchi.
- Non preoccupatevi troppo di scrivere codice super efficiente, tranne cicli interni critici. Ridisegnare la grafica vettoriale sarà ciò che ammazzerà la velocità di refresh del video, a meno che non vi mettiate a fare un sacco di matematica 3D. Nel gergo moderno, probabilmente sarete "legati alla GPU".
- Sul 6809 la maggior parte delle istruzioni, per esempio i caricamenti, influiscono i flag di stato.
- Questa pagina contiene un sommario dei comandi OS e dei registri che vengono distrutti dopo aver effettuato una chiamata alle diverse subroutine (**LINK**) – che è il motivo per cui i registri A e B non mantengono il loro valore dopo aver spostato il pennello elettronico.
- Quando si usano i joystick occorre accendere le loro rispettive caratteristiche – ad esempio il movimento in X e in Y ed il controllo analogico. Se non avete bisogno di questi servizi, lasciateli spenti e risparmierete un po' di tempo CPU.
- Prima di visualizzare del testo all'inizio del fotogramma, eseguite un MOVE, altrimenti il testo non apparirà.
- Tornare a programmare il 6502 oppure lo Z80, dopo aver programmato questo chip, vi procurerà del dolore fisico.

```
; Definizione di alcune routine dell'OS di cui avremo bisogno

OS Intensity 7F     equ   $F2A9     ; imposta intensità al massimo
OS Intensity A      equ   $F2AB     ; questa versione usa il registro A
OS_Wait_Recal       equ   $F192     ; ricalibrazione del BIOS Vectrex
OS_Print_Str_D      equ   $F37A     ; routine di stampa del BIOS
OS_Reset0ref        equ   $F354     ; chiamare spesso per evitare tremolio
OS Move             equ   $F2FC     ; muovi a loc. indicata da registri A e B
OS Draw             equ   $F3DF     ; disegna in base ai registri A e B

; direttive per l'assembler (ottimizza e fa partire il codice in memoria)
        opt
        org     0

; *** SEZIONE INTESTAZIONE
db      "g GCE 2018", $80
dw      $FF8F                   ; indirizzo di un brano (BIOS ROM)
db      $FC, $30, $20, -$58     ; altezza (negativa), larghezza, rel y, rel x
db      "JUBBERNAUT", $80       ; titolo gioco, db riserva alcuni byte in RAM
db      0                       ; fine dell'intestazione

; *** GIOCO
Main
jsr     OS Wait Recal           ; inizio del ciclo di disegno
jsr     OS Intensity 7F         ; intensità del pennello elettronico (7F=max)

Draw_A_Line
        jsr     OS_Reset0ref    ; resetta il pennello elettronico
        lda     #30
        ldb     #-64
        jsr     OS_Move         ; va alle coordinate -64,30 (A e B sono y e x)
        lda     #0
        ldb     #125
        jsr     OS_Draw         ; disegna linea orizzontale (0=y, 125=x)

Write_Some_Text
        jsr     OS Reset0ref
        lda     #0              ; OS Draw sporca i registri A e B
        ldb     #-60
        ldu     #MESSAGE        ; carica il registro U a 16-bit con la stringa
        jsr     OS_Print_Str_D  ; chiama la routine di stampa dell'OS

Draw_Another_Line
        jsr     OS_Reset0ref
        lda     #-30
        ldb     #-64
        jsr     OS_Move
        lda     #0
        ldb     #127
        jsr     OS_Draw

Main End
        bra     Main            ; e ritorna all'inizio del gioco

MESSAGE  db     "ROBIN IS SKILL!",$80
; N.B. il font è solo in maiuscolo. Solo alcuni simboli speciali sono supportati.
```

Listato 1 – Hello World in assembly Motorola 6809

Il Vectrex!

di Dante Profeta

In memoria di Nino Bonafè. Un grande uomo che mi ha sorretto e dato forza in ogni momento.

Il vantaggio di scrivere in retrospettiva è di poter rivalutare tante cose con un occhio e uno spirito differente. Per me questo è il caso del Vectrex, la console per videogiochi domestici concepita da un brillante John Ross della Smith Engineering verso la fine degli anni '70. Sì, perché agli inizi degli anni '80 il Vectrex era da me e dai miei amici e compagni di scuola media davvero tanto sottovalutato, impegnati come eravamo ai tempi della primissima console war a osannare l'Atari 2600 o l'Intellivision. Tuttavia, il Vectrex era davvero geniale nell'idea, anche se un po' fallace come realizzazione tecnica in particolar modo nel comparto audio.

L'anno era il 1982 e ai tempi era una grande fortuna possedere una console. Ricordo le battaglie combattute per convincere i miei genitori a comprarmi l'Atari 2600.

La scelta della console era dettata da diverse ragioni: in primis la diffusione della console tra i compagni di classe, con lo scopo primario di scambiarsi le cartucce gioco, ai tempi chiamate cassette, motivo anche di integrazione e di socializzazione, garantendosi così una longevità irraggiungibile altrimenti, a causa dei costi elevati dei singoli giochi; poi, il porting dei giochi da bar come Vanguard, Moon Patrol, Jungle Hunt, ecc...

Il Vectrex era per me una console considerata figlia di un dio minore vista la mancanza di grafica pixellata e di porting di giochi da sala giochi, per non parlare della totale mancanza di diffusione tra i miei compagni di scuola.

Da ragazzino poco più che decenne, entrato da poco nel fantastico mondo della televisione a colori, trovavo poi il monitor in bianco e nero del Vectrex come qualcosa di strano, come un salto indietro, e l'utilizzo degli overlay da applicare sullo schermo per ottenere zone di colore statiche, mi apparivano rasenti il ridicolo.

Insomma, avete capito che il Vectrex non destò mai il mio interesse negli anni '80.

Sono dovuti passare quasi 40 anni perché ne rivalutassi i pregi, e oggi, ala pari di un'illuminato articolo apparso sulla prestigiosa rivista Byte nel 1982, sostengo fermamente che il Vectrex è stata la console più innovativa del periodo.

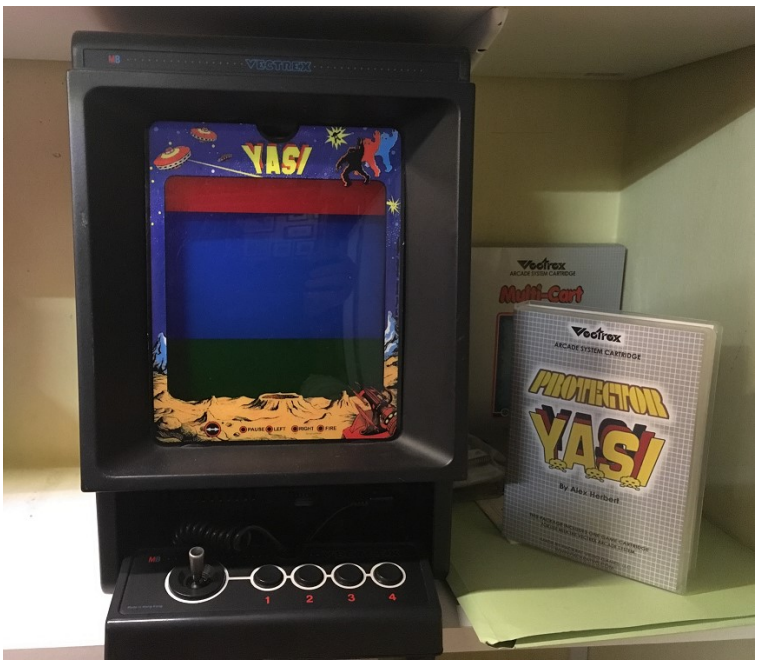
Non vi è alcun dubbio che i visionari progettisti del Vectrex avevano intuito che nel futuro i giochi sarebbero stati mossi dalla grafica vettoriale, e che pertanto, secondo la visione del periodo, fallacemente però, erano convinti che sarebbero serviti monitor in grado di disegnare poligoni invece che linee di raster.

Questo perché l'algebra lineare e gli spazi vettoriali in R3 erano invece lontani anni luce da qualunque ingegnere del periodo che si occupasse di videogiochi, e le matrici di trasformazione prospettica erano qualcosa di applicabile soltanto a conti su carta, viste le capacità di calcolo risibili delle CPU del periodo.

La renderizzazione dei poligoni veniva quindi affidata al dispositivo di rappresentazione grafica, ovvero il monitor vettoriale, e non computata dal microprocessore.

Per queste ragioni, l'hardware, inteso nella sua totalità, veniva progettato specificatamente o per la renderizzazione a pixel su CRT a deflessione elettromagnetica, ovvero il tubo catodico dei televisori, oppure per la renderizzazione su CRT a deflessione elettrostatica, ovvero il tubo catodico degli oscilloscopi.

Il Vectrex da questo punto di vista è un ibrido perché monta un classico cinescopio CRT Samsung commerciale, esattamente come quelli che si trovavano nelle TV in bianco e nero degli anni '70, e ha i deflettori di giogo del tutto simili se non identici a quelli che si trovano nelle televisioni commerciali. La differenza rispetto a un monitor/tv risiede dunque solamente nell'elettronica di controllo, che invece di far deflettere il fascio di elettroni da sinistra verso destra, guardando il cinescopio, e dall'altro verso il basso, gestisce il pennello elettronico in coordinate x, y, posizionandolo nel punto



opportuno ricavato dalla conversione delle coordinate di schermo da digitale ad analogico per mezzo di due DAC (Digital Analog Converter).

Questa scelta ha consentito di tenere bassi i costi e di produrre la console su vasta scala utilizzando solamente componentistica già presente nel mercato.

Il tubo catodico nel Vectrex è montato verticalmente, così come era abbastanza comune per tantissimi giochi arcade del periodo: Galaxian, Frogger, Amidar, Zaxxon..., dando proprio l'aspetto di una miniaturizzazione di un vero cabinet da bar.

Le dimensioni ridotte del CRT, 9" x 11", tuttavia non erano un problema, anzi erano e sono perfette, perché donano equilibrio al peso e all'area di visualizzazione, 240 mm di diagonale, permettendo la trasportabilità grazie al maniglione mimetizzato nel retro della macchina, proprio sopra il tubo catodico.

La plancia col joystick e i pulsanti, che oggi chiameremmo ingiustamente controller, si mimetizza a scomparsa elegantemente in un vano della parte frontale della console, proprio sotto il CRT.

Una console progettata per giochi vettoriali doveva per forza avere un joystick analogico, contrariamente a tutto ciò che per noi videogiocatori del momento era conosciuto: le imponenti macchine arcade, i joystick di Atari 2600 o lo scomodissimo controller a disco dell'Intellivision, erano tutti digitali, imponendo le 4 o le 8 direzioni in maniera binaria, ovvero lo switch di direzione poteva essere solamente premuto o rilasciato. Il Vectrex invece, coerentemente con la natura analogica dell'immagine video, monta un joystick analogico che permette di impartire velocità e accelerazione al moto dello "sprite" che comandiamo. Inoltre, i quattro pulsanti posizionati a destra del joystick conferiscono alla plancia uno stupendo aspetto arcade.

La plancia del joystick può essere rimossa poiché connessa al corpo macchina con un connettore Din a 9 poli, al cui fianco è presente un altro connettore identico per collegare un'altra plancia per il secondo giocatore.

Buzzzzzz. All'accensione del Vectrex la prima cosa che si percepisce è il famoso quanto famigerato disturbo audio di loop di massa, caratteristico da sempre della console, e imprescindibile per gli amanti e gli appassionati della prima ora perché rievoca comprensibilissime emozioni dovute a rimembranze di gioventù.

La ragione del buzz è dovuta al circuito di amplificazione audio posto esattamente sotto il giogo di deflessione del tubo catodico, che catturando le variazioni elettromagnetiche dell'avvolgimento provoca il ronzio, che al pari del frastuono di certe famose motociclette americane, è il marchio di fabbrica del Vectex.

In tempi recenti è possibile apportare una semplice modifica per eliminare del tutto il fruscio, ma per molti, me compreso, equivale alla violazione di un tempio. Per chi tuttavia fosse interessato o semplicemente incuriosito, rimando al box "Bye bye buzz!".

L'architettura hardware del Vectrex è estremamente semplice ma efficacissima: è basata sul Microprocessore Motorola 6809 clockato a 1.5 MHz; lo stesso Microprocessore che si trova sul TRS-80 e dei celeberrimi Defender, Robotron 2084 e Joust della Williams Electronics, e di Gyruss della Konami. La RAM è di ben 1K, tantissimo se paragonata ai 128 Bytes di Atari 2600, e una ROM di ben 8K contenente le routine di quello che potremmo assumere come il Sistema Operativo del Vectrex.

Le cartucce dei giochi possono invece avere una dimensione massima di 32K.

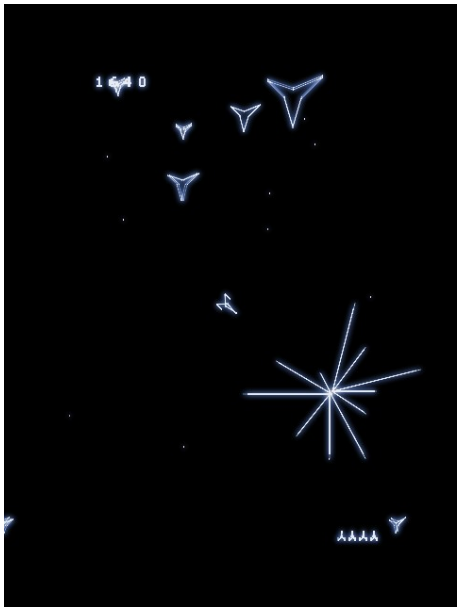
Il Motorola 6809 è coadiuvato da un 6522 VIA

(Versatile Interface Adapter), del tutto identico a quello che si trova sul primo Macintosh, e che integra al suo interno due timer responsabili per le temporizzazioni delle routine di disegno dei vettori sul monitor.

Il comparto sonoro è demandato dal famosissimo AY-3-8910, diffuso pressoché ovunque in quel periodo, dallo ZX Spectrum a Gyruss, che ne montava ben cinque.

Guardando frontalmente la console, lo slot delle cartucce si trova a destra del corpo macchina in basso, non protetto da sportello, ma sufficientemente progettato per evitare accumulo di polvere nei contatti.

Al contrario di Atari 2600 e di Intellivision, se non è presente alcuna cartuccia gioco all'accensione parte un gioco embedded su rom interna: **Mine Storm**.



Mine Storm è un palese clone di Asteroids, e da solo ben descrive le caratteristiche grafiche del sistema, come un perfetto e sobrio biglietto da visita. Ma per apprezzarlo al meglio della sua monocromia, nella confezione della console è presente un robusto foglio di plastica trasparente colorato, al secolo overlay, che semplicemente dal bianco vira di cromia al blu l'astronave, le mine e tutto ciò che viene disegnato.

Gli overlay dei giochi, pressoché delle stesse dimensioni del CRT, si posizionano uno per volta, in base al gioco, di fronte al CRT, agganciandolo agli inserti presenti nella plastica, subito sopra e subito sotto il tubo catodico.

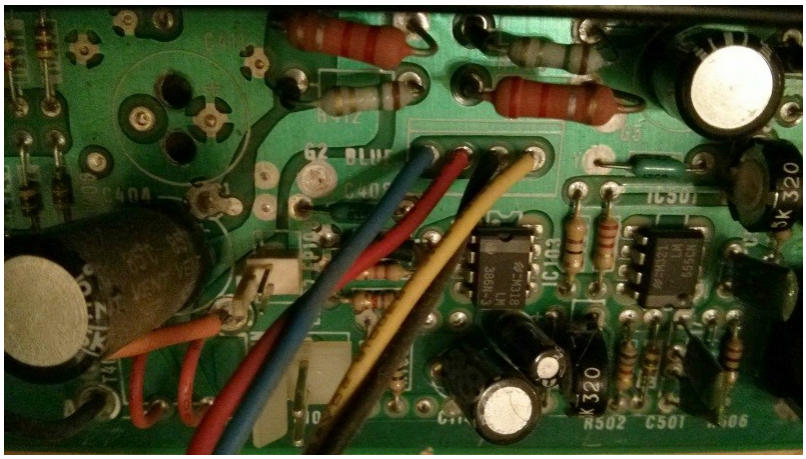
Ogni gioco veniva venduto dunque con un overlay colorato, talvolta al limite del psichedelico, al fine di dotare di zone di colore a macchie punk la nostra esperienza di gioco.

Di **Mine Storm** è famosissimo il **bug del tredicesimo livello**, che causa il crash del sistema se qualcuno osa superarlo.
L'infame bug è stato fonte di imbarazzo da parte della GCE, che offriva una cartuccia col bug corretto a chiunque ne facesse richiesta scritta per lettera... a quei tempi non esisteva ancora l'email e bisognava munirsi di carta, penna, busta, francobollo, convinzione e santa pazienza, per mostrare il proprio disappunto e rivendicare una soluzione all'orrendo misfatto. Attendendo poi la sommatoria di tutti i tempi necessari, dalla spedizione della busta, all'apertura e alla lettura della lettera che speranzosamente prima o poi sarebbe avvenuta, dato che di certo richiedeva tempi e personale di segreteria addetto, alla presa in simpatia del reclamo e infine alla spedizione della cartuccia stessa... roba di uno o due mesi di attesa al meglio delle stime, prima di poter affrontare il misterioso quattordicesimo livello... e senza salvataggi, occorreva rigiocare tutto da capo.

In tanti pensarono che non ne valesse la pena per un gioco abbastanza ripetitivo come Mine Storm, così le cartucce col bug corretto, che vennero immesse in circolazione col nome di Mine Storm II, senza scatola e senza overlay, sono diventate oggi oggetto di culto, preziose e introvabili.

Il Vectrex oggi si affranca di una comunità di sviluppatori software attivissima e brillante, che produce capolavori videoludici di impensabile impatto tecnico e scenico, e che si prodigano nella produzione di piccoli lotti di overlay custom di altissima qualità.

Su ebay da alcuni anni a questa parte circolano lotti di overlay stampati con stampanti a getto di inchiostro su fogli lucidi trasparenti, pertanto di infima qualità. Caveat emptor.



Bye bye buzz!

Il caratteristico buzz emesso con veemenza dall'altoparlantino mono del Vectrex è qualcosa che emoziona i possessori di vecchia data perché rievoca antichi ricordi.

In tempi recenti tuttavia qualcuno si è posto la questione di come e se è possibile risolvere l'annoso disturbo.

Il rumore nasce dal fatto che l'amplificatore audio, l'LM386, è posizionato sotto il giogo di deflessione del tubo catodico, e quindi oltre ad amplificare l'audio emesso dal AY-3-8910, cattura ed amplifica le onde elettromagnetiche emesse dall'avvolgimento del giogo di deflessione.

Per chi avesse il coraggio di "violentare" la console apportando una purificazione del comparto audio ai limiti del blasfemo, si tratta di un'operazione abbastanza semplice e ben descritta nel seguente blog:

http://into3.co.uk/blog/2016/02/06/vectrex-buzz-fix-enjoy-the-silence/

RetroMath: Ritorna la sfida del pi greco

di Alberto Apostolo

Quando ero ragazzo, mi capitò tra le mani il numero 54 (Ottobre 1984) della rivista italiana Bit.

A pagina 176, nella rubrica "IL RICETTARIO" si trovava un interessante articolo scritto da C. Tralamazza intitolato "La storica sfida del pi greco" con allegato un programma in BASIC per calcolarne le prime 1400 cifre tramite un PC compatibile.

Come studente di Informatica alle prime armi, ero meravigliato nel leggere che si poteva estendere la potenza di calcolo di un computer oltre le 8-16 cifre delle variabili in virgola mobile, ricorrendo alla "multipla precisione".

Per spiegarlo in parole povere, la "multipla precisione" consiste nell'effettuare calcoli aritmetici e matematici su numeri aventi moltissime cifre, memorizzandoli in strutture come i vettori di numeri interi oppure stringhe di caratteri/byte di lunghezza prefissata.

Non esistendo istruzioni dirette per effettuare calcoli matematici in multipla precisione, con il passare del tempo sono state realizzate librerie di sottoprogrammi scritte nei più svariati linguaggi.

Ma perché mai dei professionisti si sono cimentati nel mettere a punto tali librerie? Che senso ha, fare calcoli in precisione arbitraria?

Vi sono diverse ragioni e qui provo ad elencarne alcune. Esistono modelli matematici di sistemi fisici e chimici che sono assai complessi e richiedono moltissima precisione di calcolo durante la loro simulazione.

A volte in Crittografia si ha a che fare con numeri molto grandi per la creazione di codici difficili da "spezzare".

Realizzare routine di calcolo in multipla precisione si rivela un valido esercizio di programmazione ed è un banco di prova per misurare le prestazioni di un computer (con il mio ZX Spectrum, le routine più semplici fatte per "gioco" diventarono una "tesina" presentata agli Esami di Maturità).

In particolare per il pi greco, è una occasione per mettere a punto nuovi metodi calcolo ed effettuare statistiche sulla successione delle sue cifre (pi greco è un numero con parte frazionaria illimitata non periodica). Infine (perché no?) c'è gusto nello stabilire dei record di calcolo.

La storia di pi greco

Il pi greco è il rapporto tra la misura della lunghezza di una circonferenza e il suo diametro. Per calcolarlo con la maggiore precisione possibile, i matematici di ogni epoca hanno speso molto del loro ingegno. Nella Bibbia, al Libro dei Re (I,7,23), si legge: "Fece un bacino di metallo fuso di dieci cubiti da un orlo all'altro, rotondo; la sua altezza era di cinque cubiti e la sua circonferenza era di trenta cubiti". Dunque pi greco era circa 3. Per gli Egizi (papiro di Rhind, 1650 a.C.) pi greco valeva circa 256/81 = 3.16... , nonostante una misurazione del rapporto tra il perimetro e l'altezza della piramide di Cheope a Giza indichi un valore di circa 22/7 = 3.14... . Ad Archimede di Siracusa (III sec. a.C.) si attribuisce la stima

$$\frac{223}{71} < \pi < \frac{22}{7}$$

In Cina, Zu Chongzhi (noto anche come Tsu Chung Chin) nel V sec. a.C. stabiliva l'approssimazione 355/113 . Passarono diversi secoli prima di registrare il tentativo di Adrian Van Roomen (1561-1615) che calcolò pi greco con 16 cifre decimali. Nel 1596 Ludolph Van Ceulen (1540-1610?) determinava pi greco con 35 decimali.

Il risultato fu inciso sulla lapide della sua tomba (andata distrutta durante la II Guerra Mondiale e ricostruita nel 2000, mentre in figura è riportata una trascrizione dell'epitaffio come registrato negli archivi della Chiesa di S. Pietro a Leida, Paesi Bassi).

—In the archives of St. Pieter's Kerk in Leiden, Holland, this epitaph is recorded :

HIC IACET SEPULTUS MR. LUDOLFF VAN CEULEN, PROFESSOR BELGICUS, DUM VIVERET MATHEMATICARUM SCIENTIARUM IN ATHENAEO HUIUS URBIS, NATUS HILDESHIMIA ANNO 1540, DIE XXVIII IANUARII, ET DENATUS XXXI DECEMBRIS, 1610, QUI IN VITA SUA MULTO LABORE CIRCUMFERENTIAE CIRCULI PROXIMAM RATIONEM AD DIAMETRUM INVENIT SEQUENTEM. QUANDO DIAMETER EST 1, TUM CIRCULI CIRCUMFERENTIA PLUS EST QUAM

3141592653589793238462643383279502881000000000000000000000000000000000

ET MINUS QUAM

3141592653589793238462643383279502891000000000000000000000000000000000 ;

SED QUANDO DIAMETER EST

10000000000000000000000000000000000000,

TUM EST CIRCULI CIRCUMFERENTIA PLUS QUAM

3141592653589793238462643383279502881

& MINUS QUAM

3141592653589793238462643383279502889.

Con la scoperta in Occidente del calcolo infinitesimale (Newton, Leibniz) e gli sviluppi in serie di Taylor, risultò che

$$\operatorname{arctg} x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{2n+1} \qquad -1 \leq x \leq +1$$

permettendo di calcolare approssimazioni migliori (Gregory, Sharp e altri). A John Machin (1686?-1751) si deve la formula

$$\frac{\pi}{4} = 4\,\mathrm{arctg}\,\frac{1}{5} - \mathrm{arctg}\,\frac{1}{239}$$

che converge più velocemente di pi greco = 4 * arctg(1) .

Un'altra formula è attribuita a Gauss ed è quella applicata da C.Tralamazza nel suo programma:

```
PI    3             1    1             1     5             1
-- = - ARC TAN  -- + - ARC TAN  --- - ---  ARC TAN  ---
64    4            18    2            57    16           239
```

Nel 1761 (o nel 1767?) il matematico Johann Heinrich Lambert dimostrò che pi greco è irrazionale. Nel 1882 il matematico tedesco Ferdinand von Lindemann (1852-1939) dimostrò che pi greco è trascendente.

Un numero x è "trascendente" se si può dire che non esiste una equazione algebrica a coefficienti interi che abbia x come soluzione oppure non esiste un modo finito di "calcolare" x geometricamente utilizzando solo riga e compasso. Nel caso di pi greco significa anche che è impossibile fare la "quadratura del cerchio" con riga e compasso.

Tuttavia i tentativi di computazione proseguirono lo stesso (Shanks, 707 cifre calcolate nel 1875 di cui però solo 527 corrette). Nel 1947 Ferguson con una calcolatrice meccanica ne calcolò 808 applicando la formula:

$$\frac{\pi}{4} = 3\,\mathrm{arctg}\,\frac{1}{4} + \mathrm{arctg}\,\frac{1}{20} + \mathrm{arctg}\,\frac{1}{1985}$$

Nel 1949 John W. Wrench e L.B. Smith con il calcolatore ENIAC ottennero 2037 cifre di pi greco dopo 70 ore di elaborazione della formula di Machin. Negli anni successivi furono scoperte formule più avanzate (tra cui il metodo AGM usato da Brent e Salamin nel 1976) che consentirono di trovare milioni di cifre decimali. Nel 2010 Fabrice Bellard (1972-vivente) ha determinato 2699999989951 cifre mediante la formula

$$\pi = \frac{1}{2^6}\sum_{n=0}^{\infty}\frac{(-1)^n}{2^{10n}}\left(\frac{-2^5}{4n+1} - \frac{1}{4n+3} + \right.$$
$$\left. + \frac{2^8}{10n+1} - \frac{2^6}{10n+3} - \frac{2^2}{10n+5} - \frac{2^2}{10n+7} + \frac{1}{10n+9}\right)$$

Anche se non è un Santo Patrono, c'è una data dell'anno nella quale si "festeggia" il pi greco. Il 14 Marzo è il "Pi Day" (3/14 secondo la notazione americana della data).

Il pi greco non perderà mai la sua aura "mistica". Oltre alle varie filastrocche inventate da studenti universitari per memorizzare i primi decimali, esiste una variante scritta da Michael Keith della poesia "The Raven" ("Il corvo", n.d.A.) di Edgar Allan Poe dove vi sono "codificate" 740 cifre.

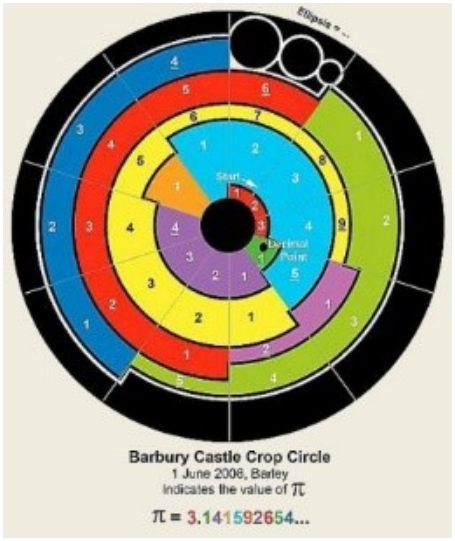
"Poe, E. Near a Raven
Midnights so dreary, tired and weary,..."

Sempre per rimanere in ambito letterario, secoli prima della dimostrazione definitiva, il sommo Dante Alighieri nella Divina Commedia congetturava che la quadratura del cerchio con riga e compasso fosse impossibile. Così scrisse nel Canto XXIII del Paradiso (133-138) [Amo11]:

"Qual è 'l geomètra che tutto s'affige
per misurar lo cerchio, e non ritrova,
pensando, quel principio ond'elli indige,
tal era io a quella vista nova:
veder volea come si convenne
l'imago al cerchio e come vi s'indova;"

Forse l'interesse per il pi greco va anche oltre i confini del nostro pianeta Terra. Il 1° Giugno 2008 apparve un ingegnoso "crop circle" a Barston Castle in Wroughton (Regno Unito), nel quale ogni arco dello schema rappresentava una cifra in base 10 (con tanto di "punto decimale") e dove tre piccoli cerchi finali indicavano tre puntini di sospensione.

Un cerchio nel grano che lasciava adito al dubbio che fosse poco "extra" e molto "terrestre", oppure che gli alieni si fossero "allargati" un po'.



Un programma per calcolare pi greco in multipla precisione

Le prestazioni delle routine in BASIC (scritte in modo assai comprensibile dal signor Tralamazza) si possono migliorare con qualche accorgimento.

E allora, anche se non possediamo un SuperComputer oppure del software professionale, la multipla precisione ce la possiamo "fare in casa" scrivendo programmi abbastanza semplici dalle prestazioni più che decorose.

Gli algoritmi necessari al calcolo del pi greco devono eseguire la somma algebrica tra numeri in multipla precisione, moltiplicazione e divisione per una semplice variabile intera con segno, operando allo stesso modo con il quale un essere umano esegue manualmente operazioni aritmetiche scrivendo su un quaderno a quadretti e contando con le 10 dita.

Un primo "trucco" è che il computer può eseguire questi algoritmi usando anche una base 10^x. Se x=4 è come il computer avesse 10^4 (diecimila) "dita" per contare.

In questo modo si riescono a memorizzare 4 cifre base 10 in ogni elemento intero di un vettore, risparmiando memoria e aumentando la velocità di esecuzione.

La scelta di x non è a caso: dato un computer con aritmetica interna a n bit, vale la limitazione 10^(2x) < 2^(n-1) per non "sballare" i calcoli con le variabili intere. Così se il nostro computer è a 32 bit, allora deve essere 10^(2x) < 2147483648 e dunque x=4.

Un altro benefico effetto collaterale della base 10000 (che tornerà utile in seguito) è che si può ancora moltiplicare/dividere per numeri interi compresi tra -2^31/10000 e 2^31/10000 avendo la stessa velocità di elaborazione del semplice caso a una sola cifra base 10.

Il secondo "trucco" è quello di ricorrere a un linguaggio compilato come il C, senza più sottostare alla lentezza di un interprete BASIC.

Nelle mie prove su un vecchio Pentium IV del 2003, ho usato il compilatore TURBO C 2.01 della Borland (che ho trovato gratis in Rete tempo fa).

Il programma che ho allegato applica la formula di Machin vista in precedenza e l'esecuzione dura circa 2-3 secondi per ottenere 2000 cifre decimali.

Per ragioni di spazio non ho documentato il software presentato in questo articolo. Se alcune parti risulteranno poco chiare, vi chiedo scusa. Gli approfondimenti sulla "multipla precisione" sono in preparazione per essere pubblicati nei prossimi numeri di RM. Considerate questo articolo come una modesta introduzione.

```
/*****************************************************************/
/* Programma:MP_PI                                               */
/* Funzione:Calcolo pi greco con aritmetica in multipla precisione */
/*****************************************************************/
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <values.h>
#include <conio.h>    /* tastiera e video PC IBM amb. Windows MS-DOS
*/
#include <stdlib.h>   /* per avere max e min                     */
#define LMAX    503L /* ATTENZIONE! IN C   RANGE = (0:LMAX- 1)   */
#define BASE   10000L /* base aritmetica in multipla precisione  */
#define NMAX   50000L /* numero max termini di una serie         */
/* Dichiarazione dei tipi di variabile strutturati               */
typedef struct { long segno;             /* segno (-1, 0,+1)     */
                 long cifra[(LMAX+1)];    /* cifre in base beta   */

       } MultPrec;
/* Dichiarazione di funzione e subroutine                        */
void Mulz(long,long,MultPrec *p_a,long,long *p_rp);
void Divz(long,long,MultPrec *p_a,long,long *p_rp);
void Suma(long,long,MultPrec *p_s,MultPrec *p_a,long *p_rp);
void Init(long,long,long,MultPrec *p_a,long,long *p_rp);
void DisplayMultPrec(long,long,MultPrec *p_a,char *s);
/* Programma principale                                          */
main()
{
      long beta = BASE;     /* base aritmetica multipla precisione */
      long l    = LMAX;     /* lunghezza totale vettore mult.prec. */
      long lpi  = 1L;       /* lunghezza parte intera              */
      long lpf  = l-lpi;     /* lunghezza parte frazionaria        */
      long lv   = l-2L;      /* numero elementi visualizzabili     */
      MultPrec  a,b,s,p;     /* variabili multipla precisione      */
      MultPrec  *p_a =&a;    /* puntatore var. mult.prec.          */
      MultPrec  *p_b =&b;    /* puntatore var. mult.prec.          */
      MultPrec  *p_s =&s;    /* puntatore var. mult.prec.          */
      MultPrec  *p_p =&p;    /* puntatore var. mult.prec.          */
      char nomevbl[6]="";/* nome variabile m.p. (per la stampa) */
      long rp,  *p_rp=&rp;    /* riporto e puntatore riporto       */
      long rd,  *p_rd=&rd; /* resto e puntatore resto           */
/*                                                                */
      long k;                /* indice (vbl di lavoro)           */
      long z,n5,n239;         /* numeri interi                   */
/*                                                                */
      clrscr();               /* pulisci schermo                 */

      printf("Programma MP_PI : INIZIO ELABORAZIONE\n");
/*                                                                */
      Init(beta,l,lpi,p_a,1L,p_rp);
      b=a;
      Divz(beta,l,p_b,5L,p_rd);
      Init(beta,l,lpi,p_s,0L,p_rp);
      Suma(beta,l,p_s,p_b,p_rp);
      z=1L;
      for (k=3L;k<=NMAX;k=k+2L) {
          Divz(beta,l,p_b,5L,p_rd);
          Divz(beta,l,p_b,5L,p_rd);
          a=b;
          z=-z;
          Divz(beta,l,p_a,(k*z),p_rd);
          if (a.segno==0L) {
              n5=k;
              break;
          }
          Suma(beta,l,p_s,p_a,p_rp);
      }
      Mulz(beta,l,p_s,16L,p_rd);
      p=s;
/*                                                                */
      Init(beta,l,lpi,p_a,1L,p_rp);
      b=a;
      Divz(beta,l,p_b,239L,p_rd);
      Init(beta,l,lpi,p_s,0L,p_rp);
      Suma(beta,l,p_s,p_b,p_rp);
      z=1L;
      for (k=3L;k<NMAX;k=k+2L) {
          Divz(beta,l,p_b,239L,p_rd);
          Divz(beta,l,p_b,239L,p_rd);
          a=b;
          z=-z;
          Divz(beta,l,p_a,(k*z),p_rd);
          if (a.segno==0L) {
              n239=k;
```

```
            break;
         }
         Suma(beta,l,p_s,p_a,p_rp);
      }
      Mulz(beta,l,p_s,-4L,p_rd);
      Suma(beta,l,p_p,p_s,p_rp);
/*                                                                    */
      strcpy(nomevbl,"pi ");
      DisplayMultPrec(lv,lpi,p_p,nomevbl);
/*                                                                    */
      printf("\n\nTermini utilizzati 1ma arctg = %5d",n5);
      printf("\nTermini utilizzati 2da arctg = %5d",n239);
      printf("\n\nProgramma MP_PI : FINE ELABORAZIONE\n\n");
      return 0;
}
/*--------------------------------------------------------------------*/
/* Funzione: visualizza numero in multipla precisione base 10000      */
/*--------------------------------------------------------------------*/
void DisplayMultPrec(long lv,long lpi,MultPrec *p_a,char *nomevbl)
{     long i,j;
      printf("\n%-6s = ",nomevbl);
      j=lpi%10L;
      if ((*p_a).segno >  0L) {printf("\+");}
      if ((*p_a).segno == 0L) {printf(" ") ;}
      if ((*p_a).segno <  0L) {printf("\-");}
      if (j != 0L) {
         for ( i = j ; i < 10L ; i++ ) {printf("     ");} /*  5 blank
*/
      }
      for ( i = 1L ; i <= lv ; i++ ) {
         if (i == (lpi+1L)) {printf(".");} else {printf(" ");}
         printf("%4.4ld",(*p_a).cifra[i]);
         if ((i % 10L)==j) {printf("\n          ");}      /* 10 blank
*/
      }
}
/*--------------------------------------------------------------------*/
/* Funzione: moltiplicazione corta a=a*z in multipla precisione       */
/*--------------------------------------------------------------------*/
void Mulz(long beta,long l,MultPrec *p_a,long z,long *p_rp)
{     long ipro,i,izer,sz;
      sz=max(min(z,1L),-1L);           /* simula sgn(z) se z integer
*/
      izer=0L;
      *p_rp=0L;
      for (i = l;i >= 1L;i--) {
         ipro=(*p_a).cifra[i]*labs(z)+*p_rp;
         *p_rp=ipro/beta;
         (*p_a).cifra[i]=ipro % beta;
         izer=max(izer,(*p_a).cifra[i]);
      }
      (*p_a).segno=(*p_a).segno * sz * min(1L,izer);
}
/*--------------------------------------------------------------------*/
/* Funzione: divisione corta a=a/z in multipla precisione             */
/*--------------------------------------------------------------------*/
void Divz(long beta,long l,MultPrec *p_a,long z,long *p_rd)
{     long i,izer,sz;
      sz=max(min(z,1L),-1L);           /* simula sgn(z) se z integer
*/
      izer=0L;
      *p_rd=0L;
      for (i = 1L;i <= l;i++) {
         *p_rd=(*p_a).cifra[i]+*p_rd*beta;
         (*p_a).cifra[i]=*p_rd / labs(z);
         *p_rd=*p_rd % labs(z);
         izer=max(izer,(*p_a).cifra[i]);
      }
      (*p_a).segno=(*p_a).segno * sz * min(1L,izer);
}
/*--------------------------------------------------------------------*/
/* Funzione: somma algebrica s=s+a in multipla precisione             */
/*--------------------------------------------------------------------*/
void Suma(long beta,long l,MultPrec *p_s,MultPrec *p_a,long *p_rp)
{     long i,isom,s1,a1,izer,icnf;
      *p_rp=0;
      if ((*p_a).segno != 0L) {
         if ((*p_s).segno == 0L) {
            for (i = 1L;i <= l;i++) {(*p_s).cifra[i]=(*p_a).cifra[i];}
            (*p_s).segno = (*p_a).segno;
         } else {
             s1=+1L;
             a1=(*p_s).segno * (*p_a).segno;
             if ((*p_s).segno!=(*p_a).segno) {
                for (i = 1L;i <= l;i++) {
                   icnf=(*p_s).cifra[i]-(*p_a).cifra[i];
                   if (icnf != 0L) {
                      icnf = icnf / labs(icnf);
                      s1=icnf;
                      a1=-icnf;
                      (*p_s).segno = (*p_s).segno * icnf;
                      break;
                   }
                }
             }
             izer=0L;
             for (i = l;i >= 1L;i--) {
                isom=*p_rp+s1*(*p_s).cifra[i]+a1*(*p_a).cifra[i];
                *p_rp=((isom+beta)/beta)-1L;
                (*p_s).cifra[i] = isom - *p_rp * beta;
                izer=max(izer,(*p_s).cifra[i]);
             }
             (*p_s).segno=(*p_s).segno * min(1L,izer);
         }
      }
}
/*--------------------------------------------------------------------*/
/* Funzione: inizializzazione con un numero z                         */
/*--------------------------------------------------------------------*/
void Init(long beta,long l,long lpi,MultPrec *p_a,long z,long *p_rp)
{     long i,irp,izer;
      izer=0L;
      (*p_a).segno=max(min(z,1L),-1L); /* simula sgn(z) se z integer
*/
      for (i = 1L;i <= l; i++){ (*p_a).cifra[i] = 0L; }
      (*p_a).cifra[lpi] = labs(z);
      *p_rp=0L;
      for (i = lpi;i >= 1L;i--) {
         irp=(*p_a).cifra[i]+*p_rp;
         *p_rp=irp / beta;
         (*p_a).cifra[i] = irp % beta;
         izer=max(izer,(*p_a).cifra[i]);
      }
      (*p_a).segno=(*p_a).segno * min(1L,izer);
}
/*********************************************************************/
/* FINE PROGRAMMA                                                    */
/*********************************************************************/
```

```
Programma MP_PI : INIZIO ELABORAZIONE

pi     = +                                                      0003
           .1415 9265 3589 7932 3846 2643 3832 7950 2884 1971
            6939 9375 1058 2097 4944 5923 0781 6406 2862 0899
            8628 0348 2534 2117 0679 8214 8086 5132 8230 6647
            0938 4460 9550 5822 3172 5359 4081 2848 1117 4502
            8410 2701 9385 2110 5559 6446 2294 8954 9303 8196
            4428 8109 7566 5933 4461 2847 5648 2337 8678 3165
            2712 0190 9145 6485 6692 3460 3486 1045 4326 6482
            1339 3607 2602 4914 1273 7245 8700 6606 3155 8817
            4881 5209 2096 2829 2540 9171 5364 3678 9259 0360
            0113 3053 0548 8204 6652 1384 1469 5194 1511 6094
            3305 7270 3657 5959 1953 0921 8611 7381 9326 1179
            3105 1185 4807 4462 3799 6274 9567 3518 8575 2724
            8912 2793 8183 0119 4912 9833 6733 6244 0656 6430
```

```
           8602 1394 9463 9522 4737 1907 0217 9860 9437 0277
           0539 2171 7629 3176 7523 8467 4818 4676 6940 5132
           0005 6812 7145 2635 6082 7785 7713 4275 7789 6091
           7363 7178 7214 6844 0901 2249 5343 0146 5495 8537
           1050 7922 7968 9258 9235 4201 9956 1121 2902 1960
           8640 3441 8159 8136 2977 4771 3099 6051 8707 2113
           4999 9998 3729 7804 9951 0597 3173 2816 0963 1859
           5024 4594 5534 6908 3026 4252 2308 2533 4468 5035
           2619 3118 8171 0100 0313 7838 7528 8658 7533 2083
           8142 0617 1776 6914 7303 5982 5349 0428 7554 6873
           1159 5628 6388 2353 7875 9375 1957 7818 5778 0532
           1712 2680 6613 0019 2787 6611 1959 0921 6420 1989
           3809 5257 2010 6548 5863 2788 6593 6153 3818 2796
           8230 3019 5203 5301 8529 6899 5773 6225 9941 3891
           2497 2177 5283 4791 3151 5574 8572 4245 4150 6959
           5082 9533 1168 6172 7855 8890 7509 8381 7546 3746
           4939 3192 5506 0400 9277 0167 1139 0098 4882 4012
           8583 6160 3563 7076 6010 4710 1819 4295 5596 1989
           4676 7837 4494 4825 5379 7747 2684 7104 0475 3464
           6208 0466 8425 9069 4912 9331 3677 0289 8915 2104
           7521 6205 6966 0240 5803 8150 1935 1125 3382 4300
           3558 7640 2474 9647 3263 9141 9927 2604 2699 2279
           6782 3547 8163 6009 3417 2164 1219 9245 8631 5030
           2861 8297 4555 7067 4983 8505 4945 8858 6926 9956
           9092 7210 7975 0930 2955 3211 6534 4987 2027 5596
           0236 4806 6549 9119 8818 3479 7753 5663 6980 7426
           5425 2786 2551 8184 1757 4672 8909 7777 2793 8000
           8164 7060 0161 4524 9192 1732 1721 4772 3501 4144
           1973 5685 4816 1361 1573 5255 2133 4757 4184 9468
           4385 2332 3907 3941 4333 4547 7624 1686 2518 9835
           6948 5562 0992 1922 2184 2725 5025 4256 8876 7179
           0494 6016 5346 6804 9886 2723 2791 7860 8578 4383
           8279 6797 6681 4541 0095 3883 7863 6095 0680 0642
           2512 5205 1173 9298 4896 0841 2848 8626 9456 0424
           1965 2850 2221 0661 1863 0674 4278 6220 3919 4945
           0471 2371 3786 9609 5636 4371 9172 8746 7764 6575
           7396 2413 8908 6583 2645 9958 1339 0478 0275 9009


Termini utilizzati 1ma arctg =  2869
Termini utilizzati 2da arctg =   845

Programma MP_PI : FINE ELABORAZIONE
```

Calcolare il reciproco di pi greco in multipla precisione

In questo paragrafo si procederà a illustrare il calcolo del reciproco di pi greco, applicando una formula scoperta nel 1914 dal matematico indiano Srinivasa Ramanujan (22/12/1887 - 26/04/1920).

$$\frac{1}{\pi} = \sum_{k=0}^{\infty} \binom{2k}{k}^3 \frac{42k+5}{2^{12k+4}}$$

La serie converge piuttosto lentamente (1.8 cifre decimali per termine). Tuttavia è possibile con essa calcolare la seconda metà delle cifre binarie del reciproco di pi greco senza calcolare la prima metà perché i denominatori crescono il doppio più rapidamente dei numeratori. Da questo punto di vista, Ramanujan trovò una specie di precursore delle serie B.B.P. (Bailey-Borwein-Plouffe), scoperte recentemente, le quali permettono il calcolo delle cifre binarie nel "mezzo" di pi greco [AH12].

I più brillanti studiosi di Matematica e i conoscitori di Ramanujan, avranno ragione nel contestare la mia scelta perché esistono formule di Ramanujan più "prestazionali" per giungere allo stesso risultato. Tuttavia la formula da me scelta si adatta con facilità agli stessi sottoprogrammi usati per il calcolo di pi greco, riscrivendola come segue:

$$s_n = \frac{2607}{8192} + \sum_{k=2}^{n} c_k \frac{42k+5}{16}$$

$$\text{dove} \begin{cases} c_1 = \frac{1}{512} \\ c_k = \frac{c_{k-1}}{4096}\left(\frac{4k-2}{k}\right)^3, k \geq 2 \end{cases}$$

```
/*****************************************************************/
/*Programma: MP_INVPI                                            */
/*Funzione:Calcolo reciproco pi greco con aritmetica in mult.prec. */
/*****************************************************************/
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <values.h>
#include <conio.h>    /* tastiera e video PC IBM amb. Windows MS-DOS
*/
#include <stdlib.h>   /* per avere max e min                      */
#define LMAX     503L /* ATTENZIONE! IN C   RANGE = (0:LMAX- 1)   */
#define BASE   10000L /* base aritmetica in multipla precisione   */
#define NMAX   50000L /* numero max termini di una serie          */
/* Dichiarazione dei tipi di variabile strutturati                */
typedef struct { long segno;              /* segno (-1, 0,+1)     */
                 long cifra[(LMAX+1)];    /* cifre in base beta   */

       } MultPrec;
/* Dichiarazione di funzione e subroutine                         */
void Mulz(long,long,MultPrec *p_a,long,long *p_rp);
void Divz(long,long,MultPrec *p_a,long,long *p_rp);
void Suma(long,long,MultPrec *p_s,MultPrec *p_a,long *p_rp);
void Init(long,long,long,MultPrec *p_a,long,long *p_rp);
void DisplayMultPrec(long,long,MultPrec *p_a,char *s);
/* Programma principale                                           */
main()
{
      long beta = BASE;    /* base aritmetica multipla precisione */
```

```
      long l    = LMAX;        /* lunghezza totale vettore mult.prec.
*/
      long lpi  = 1L;         /* lunghezza parte intera             */
      long lpf  = l-lpi;      /* lunghezza parte frazionaria        */
      long lv   = l-2L;       /* numero elementi visualizzabili     */
      MultPrec  c,s,p;        /* variabili multipla precisione      */
      MultPrec  *p_c =&c;     /* puntatore var. mult.prec.          */
      MultPrec  *p_s =&s;     /* puntatore var. mult.prec.          */
      MultPrec  *p_p =&p;     /* puntatore var. mult.prec.          */
      char      nomevbl[6]=""; /* nome variabile m.p. (per la stampa)
*/
      long rp,  *p_rp=&rp;    /* riporto e puntatore riporto        */
      long rd,  *p_rd=&rd;    /* resto   e puntatore resto          */
/*                                                                  */
      long k,m;               /* indice (vbl di lavoro)             */
      long z,n;               /* numero intero                      */
/*                                                                  */
      clrscr();               /* pulisci schermo                    */

      printf("Programma MP_INVPI : INIZIO ELABORAZIONE\n");
/*                                                                  */
      Init(beta,l,lpi,p_s,2607L,p_rp);
      Divz(beta,l,p_s,8192L,p_rd);
      Init(beta,l,lpi,p_c,1L,p_rp);
      Divz(beta,l,p_c,512L,p_rd);
      for (k=2L;k<=NMAX;k++) {
          for (m=1L;m<=3L;m++) {
              z = 4L*k-2L;
              Mulz(beta,l,p_c,z,p_rd);
              z = k;
              Divz(beta,l,p_c,z,p_rd);
          }
          Divz(beta,l,p_c,4096,p_rd);
          p=c;
          z = 42L*k+5L;
          Mulz(beta,l,p_p,z,p_rd);
          Divz(beta,l,p_p,16L,p_rd);
          if (p.segno==0L) {
              n=k;
              break;
          }
          Suma(beta,l,p_s,p_p,p_rp);
      }
/*                                                                  */
      strcpy(nomevbl,"1/pi ");
      DisplayMultPrec(lv,lpi,p_s,nomevbl);
/*                                                                  */
      printf("\n\nTermini utilizzati = %5d",n);
      printf("\n\nProgramma MP_INVPI : FINE ELABORAZIONE\n\n");
      return 0;
}
/*------------------------------------------------------------------*/
/* Funzione : visualizza numero in multipla precisione base 10000
*/
/*------------------------------------------------------------------*/
void DisplayMultPrec(long lv,long lpi,MultPrec *p_a,char *nomevbl)
{     long i,j;
      printf("\n%-6s = ",nomevbl);
      j=lpi%10L;
      if ((*p_a).segno >  0L) {printf("\+");}
      if ((*p_a).segno == 0L) {printf(" ") ;}
      if ((*p_a).segno <  0L) {printf("\-");}
      if (j != 0L) {
         for ( i = j ; i < 10L ; i++ ) {printf("     ");} /*  5 blank
*/
      }
      for ( i = 1L ; i <= lv ; i++ ) {
         if (i == (lpi+1L)) {printf(".");} else {printf(" ");}
         printf("%4.4ld",(*p_a).cifra[i]);
         if ((i % 10L)==j) {printf("\n          ");}      /* 10 blank
*/
      }
}
/*------------------------------------------------------------------*/
```

```
/* Funzione: moltiplicazione corta a=a*z in multipla precisione     */
/*------------------------------------------------------------------*/
void Mulz(long beta,long l,MultPrec *p_a,long z,long *p_rp)
{     long ipro,i,izer,sz;
      sz=max(min(z,1L),-1L);           /* simula sgn(z) se z integer
*/
      izer=0L;
      *p_rp=0L;
      for (i = l;i >= 1L;i--) {
         ipro=(*p_a).cifra[i]*labs(z)+*p_rp;
         *p_rp=ipro/beta;
         (*p_a).cifra[i]=ipro % beta;
         izer=max(izer,(*p_a).cifra[i]);
      }
      (*p_a).segno=(*p_a).segno * sz * min(1L,izer);
}
/*------------------------------------------------------------------*/
/* Funzione: divisione corta a=a/z in multipla precisione           */
/*------------------------------------------------------------------*/
void Divz(long beta,long l,MultPrec *p_a,long z,long *p_rd)
{     long i,izer,sz;
      sz=max(min(z,1L),-1L);           /* simula sgn(z) se z integer
*/
      izer=0L;
      *p_rd=0L;
      for (i = 1L;i <= l;i++) {
         *p_rd=(*p_a).cifra[i]+*p_rd*beta;
         (*p_a).cifra[i]=*p_rd / labs(z);
         *p_rd=*p_rd % labs(z);
         izer=max(izer,(*p_a).cifra[i]);
      }
      (*p_a).segno=(*p_a).segno * sz * min(1L,izer);
}
/*------------------------------------------------------------------*/
/* Funzione: somma algebrica s=s+a in multipla precisione           */
/*------------------------------------------------------------------*/
void Suma(long beta,long l,MultPrec *p_s,MultPrec *p_a,long *p_rp)
{     long i,isom,s1,a1,izer,icnf;
      *p_rp=0;
      if ((*p_a).segno != 0L) {
         if ((*p_s).segno == 0L) {
            for (i = 1L;i <= l;i++) {(*p_s).cifra[i]=(*p_a).cifra[i];}
            (*p_s).segno = (*p_a).segno;
         } else {
             s1=+1L;
             a1=(*p_s).segno * (*p_a).segno;
             if ((*p_s).segno!=(*p_a).segno) {
                for (i = 1L;i <= l;i++) {
                   icnf=(*p_s).cifra[i]-(*p_a).cifra[i];
                   if (icnf != 0L) {
                      icnf = icnf / labs(icnf);
                      s1=icnf;
                      a1=-icnf;
                      (*p_s).segno = (*p_s).segno * icnf;
                      break;
                   }
                }
             }
             izer=0L;
             for (i = l;i >= 1L;i--) {
                isom=*p_rp+s1*(*p_s).cifra[i]+a1*(*p_a).cifra[i];
                *p_rp=((isom+beta)/beta)-1L;
                (*p_s).cifra[i] = isom - *p_rp * beta;
                izer=max(izer,(*p_s).cifra[i]);
             }
             (*p_s).segno=(*p_s).segno * min(1L,izer);
         }
      }
}
/*------------------------------------------------------------------*/
/* Funzione: inizializzazione con un numero z                       */
/*------------------------------------------------------------------*/
void Init(long beta,long l,long lpi,MultPrec *p_a,long z,long *p_rp)
{     long i,irp,izer;
```

```
      izer=0L;
      (*p_a).segno=max(min(z,1L),-1L); /* simula sgn(z) se z integer
*/
      for (i = 1L;i <= l; i++){ (*p_a).cifra[i] = 0L; }
      (*p_a).cifra[lpi] = labs(z);
      *p_rp=0L;
      for (i = lpi;i >= 1L;i--) {
         irp=(*p_a).cifra[i]+*p_rp;
         *p_rp=irp / beta;
         (*p_a).cifra[i] = irp % beta;
         izer=max(izer,(*p_a).cifra[i]);
      }
      (*p_a).segno=(*p_a).segno * min(1L,izer);
}
/*****************************************************************/
/* FINE PROGRAMMA                                                */
/*****************************************************************/
```

```
Programma MP_INVPI : INIZIO ELABORAZIONE

1/pi   = +                                              0000
          .3183 0988 6183 7906 7153 7767 5267 4502 8724 0689
           1929 1480 9128 9749 5334 6881 1779 3595 2684 5307
           0180 2276 0553 2506 1719 1214 5685 4535 1591 6073
           7858 2369 2229 1573 0575 5934 8214 6339 9678 4584
           7993 3874 8181 5514 6155 4927 9385 0615 3774 3478
           5792 4347 9532 3386 7247 8048 3447 2580 2366 4760
           2284 4539 9511 4318 8092 3780 1738 0534 7912 2409
           7882 1873 8756 8817 1057 4461 9989 2886 8004 9734
           4695 4789 1922 1796 6461 9356 6149 8123 3397 2925
           6093 9889 7304 3757 6314 9573 1339 2848 2077 9917
           4827 8697 2199 6773 6198 3999 2488 5751 1703 4235
           7716 8622 3503 7534 3210 9309 5073 9760 1947 8920
           7295 1866 7536 1186 0498 8993 2706 1065 4313 5510
           0644 0649 5556 3279 4332 0458 9349 6239 1963 3168
           1212 0336 0607 1996 2678 2397 4997 6655 7330 8870
           5595 1014 0032 4813 5512 8777 6991 4262 1760 2443
           9875 2295 3627 5552 9475 7812 6613 6092 9159 5696
           3522 6248 5462 8139 9215 5004 9000 5955 1971 4178
           1138 0559 3570 2630 5042 0032 6354 9204 1849 6232
           1248 1122 9124 0629 2968 1784 9691 8382 8704 2315
           0815 1124 0174 3053 2136 0443 4318 2815 1494 9165
           4451 9549 2570 7997 5031 0658 7816 2796 3544 8187
           1650 9594 1466 5743 8081 3999 5181 5315 4156 9869
           4078 7179 6561 7434 6851 2807 3379 0233 2509 1411
           8866 5526 2537 3000 5224 5435 9423 0642 2519 9008
           7733 5890 0752 5112 1672 6342 3390 5195 1625 6449
           8832 4666 8629 0212 2470 7375 7126 2272 7338 4334
           2841 3949 3920 2585 0115 6672 1062 3921 7189 0196
           7911 3437 4199 0949 3020 8632 4763 1035 1616 7888
           5959 9419 9901 0508 7751 3225 8891 7666 1369 2101
           5705 8303 0282 0809 7859 7701 2776 3215 5239 3986
           1468 2077 9991 5738 3781 1961 8747 5544 1237 5086
           4454 3786 0273 2510 5224 7756 0775 0777 6221 3628
           1353 0868 1656 5570 5386 6853 5991 1214 1580 7721
           2070 5477 9924 9025 1991 4985 5259 4047 1881 9116
           8602 3296 5928 2371 1554 2481 1508 8989 1404 3579
           5395 8481 8980 6545 8954 0433 2992 0713 0636 3070
           8800 7681 3797 4943 5383 1775 2638 1933 0139 2880
           9553 9413 7536 7313 5562 0955 9590 9007 0679 1516
           6037 6367 7375 8755 3224 9629 9061 1993 1160 4381
           6719 7502 0702 5425 8086 4631 6099 7439 3737 5551
           8931 3269 2442 0684 0888 1710 9957 0075 8547 7388
           5870 7323 8755 6585 7471 8756 8694 0646 0474 2916
           7584 7114 2372 7268 3858 9203 6636 4583 9283 3001
           7566 1586 6270 6995 5819 9491 7298 5805 3490 1219
           7873 7818 9176 6100 6740 6107 6109 4624 6431 6188
           6395 3520 6456 6262 8379 6194 9964 4876 6703 4871
           3979 6950 0207 9001 3677 6007 9573 4471 9921 6048
           0054 7802 1749 9097 0957 5847 1365 2227 9897 8065
           3799 4854 1669 9222 9841 6578 0755 3569 4860 7100


Termini utilizzati =  1109

Programma MP_INVPI : FINE ELABORAZIONE
```

Appendice

Per coloro che desiderassero leggere l'articolo originale pubblicato da Bit, si può accedere al link https://archive.org/details/Bit_054

Il sito del "Pi Day" è https://www.piday.org/

Bibliografia

[AH12] J. Arndt, C.Haenel, "Pi-Unleashed", Springer Science & Business Media, 2012.

[Amo11] B. D'Amore, "Dante e la Matematica", Giunti Editore, 2011.

[BBB04] L. Berggren, J.M. Borwein, P.B. Borwein, "Pi, a source book", Springer, 2004.

[BGR90] J.W. Bruce, P.J. Giblin , P.J. Rippon , "Microcomputer and Mathematics", Cambridge University Press, 1990.

[Cri07] T. Crilly, "50 grandi idee di matematica", Edizioni Dedalo, 2007.

[Tra84] C. Tralamazza, "La storica sfida del pi greco", Bit n.54 , Ed. Jackson, Ottobre 1984.

Grafica… Che passione!

di Marco Pistorio

Questo articolo nasce da una riflessione, uno "spunto" che risale a poche settimane fa, quando l'amico **Vincenzo Scarpa** mi chiese in un post all'interno del suo gruppo FB **"Quelli degli emulatori e del retrò…**" come mai oggi fosse possibile fare un maggior uso dell'alta risoluzione sul C64 rispetto al passato. Domanda intrigante, alla quale provai a rispondere così: ***"Ringrazio intanto Vincenzo per la citazione e provo a rispondere alla questione che mi ha posto, pur non riconoscendomi particolarmente "virtuoso" nell'ambito dello sviluppo di giochi/demo in hi-res. Entrando più nel merito della questione, posso affermare che oggi, a differenza del passato, esistono strumenti per la creazione e la manipolazione di immagini grafiche estremamente potenti. GIMP è certamente uno di questi. Esistono poi tools come CharPad e SpritePad che permettono di realizzare set di caratteri ridefiniti e sprites in maniera semplice, intuitiva e con un approccio moderno, tutti strumenti che permettono di lavorare comodamente sul proprio PC. Esistono delle utili librerie che trasformano un file .png in uno .ppm ed infine lo convertono in formato Koala. Per chi volesse approfondire:***

[ImageMagick]
(http://www.imagemagick.org/script/index.php)
[C64Gfx]
(http://koti.kapsi.fi/a1bert/Dev/C64Gfx/)
https://github.com/worldofchris/c64-demo

Un altro spunto interessante sull'argomento è possibile coglierlo nei samples forniti a corredo del famigerato KickAssembler. Nell'esempio 11 contenuto nella cartella Examples si può vedere in azione l'algoritmo Floyd-Steinberg , un algoritmo di dithering per la manipolazione delle immagini grafiche.
Fornisco anche qui un paio di links: https://it.wikipedia.org/wiki/Algoritmo_di_Floyd-Steinberg
http://theweb.dk/KickAssembler/KickAssembler.zip

Esistono degli interessanti tools che convertono una immagine .bmp di dimensioni ben precise in chars ridefiniti, come ad esempio questo: http://c64.cz/index.php?recenze=software_bmp2char_convert

Insomma, strumenti come questi (e diversi altri simili) permettono oggi praticamente a chiunque di ottenere immagini grafiche sul C64 di un certo effetto!"

Da qui nasce la mia riflessione: "Perché non realizzare un mio semplice tool che permetta di ottenere facilmente immagini grafiche in alta risoluzione da poter impiegare sul Commodore 64?" Ci sono già tanti strumenti in giro, è vero, ed alcuni di questi sono qualitativamente notevolissimi, tuttavia realizzare per conto proprio un apposito tool ha sempre un certo fascino! Allo scopo ho scelto di sviluppare qualcosa con **FreeBasic** onde evitare di adoperare strumenti di sviluppo proprietari (Es. Microsoft) Inoltre **Freebasic** è multipiattaforma ed infine l'eseguibile prodotto è "snello" e non richiede il .Net framework in esecuzione in sottofondo.

Il tool è estremamente semplice. Si aspetta in input un file bitmap (.bmp) monocolore la cui risoluzione deve essere 320x200, esattamente quella gestibile tramite C64. Allo scopo di rendere il codice più semplice possibile, non ho eseguito alcun controllo sulla bontà del file in ingresso o sulla sua effettiva presenza sulla stessa cartella dove viene eseguita l'applicazione. Se è tutto ok, pochi secondi dopo l'avvio del tool comparirà, all'interno di una piccola finestra, il contenuto della immagine bitmap elaborata, pronta per essere data in pasto al nostro fido "biscottone". Nel caso in cui invece la finestra comparisse vuota, molto probabilmente il file .bmp fornito non è stato trovato oppure non contiene le corrette informazioni colore (ovvero non è un file bitmap monocolore). Ho predisposto un file bitmap di prova, che potrete di seguito osservare. Tale file sarà a vostra disposizione insieme al tool ed insieme ad ulteriore materiale. E' possibile eseguire il tool facendo doppio click sull'eseguibile **tool_bitmaps_4_c64.exe**

Il file da esaminare dovrà chiamarsi **immagine.bmp** e verrà prodotto dal tool un file che si chiamerà **hires.dat,** che verrà sovrascritto di volta in volta se già presente.

Il tool è disponibile in apposita cartella su GITHUB a questo indirizzo: https://github.com/marcus73/retromagazine_03

comprensivo del relativo codice sorgente. Perché fornirne il codice sorgente? Perché realizzare simili tools da tenere chiusi nel proprio cassetto è, a mio avviso, del tutto insensato.
Solo condividendo e diffondendo il più possibile le conoscenze riguardo al Commodore64 e più in generale riguardo a tutto il mondo del retrocomputing potremo sperare di vedere questo ambito sempre vivo, con un numero di estimatori sempre più ampio, vecchi e soprattutto nuovi.

Funzionamento del tool

Proverò a descrivere il funzionamento del tool nella maniera più semplice possibile. Viene elaborata l'immagine punto per punto, dall'alto verso il basso e da sinistra verso destra, generando per ogni gruppo di 8 pixel il relativo valore corrispondente alla rappresentazione in codice binario di ciascuno di questi gruppi di 8 pixel (bit "0" per un punto spento, bit "1" per un punto acceso). Ciascuno di questi "segmenti" lunghi 8 pixel in senso orizzontale produrrà un byte in **hires.dat**
L' immagine però è larga 320 pixels, quindi i segmenti saranno 40 (320/8) per ciascuna linea di schermo. L'immagine è lunga 200 pixel, quindi avremo infine 40x200 caratteri memorizzati sul file **hires.dat,** ovvero 8000 bytes che corrispondono al numero di bytes che servono per rappresentare una intera immagine in alta risoluzione, monocolore, su Commodore64.

In testa a tale file aggiungo due bytes che servono per poter caricare il file hires.dat a partire da una certa locazione di memoria specificata. Da programma ho impostato tali bytes **0** e **32**, che corrispondono alla locazione di memoria di partenza **8192**, espressa in notazione byte alto/byte basso (**32***256+**0**)

LISTATO del tool con sintassi FREEBASIC:

```
dim shared vettore(7999+2) as ubyte
dim shared indice as integer

function converti(dato as string) as integer
    dim valore as integer
    dim i as integer
```

```
    dim el as string
    valore=0
     for i=8 to 1 step-1
       el=mid$(dato,i,1)
       if el="1" then
          valore=valore+2^(i-1)
       end if
    next
   return valore
end function

 sub elabora(x as integer,y as integer)
    dim xx as integer
    dim yy as integer
    dim xj as integer
    dim yk as integer
    dim dato as string
    dim c as integer

    xx=x*8
    yy=y*8

    for yk=yy to yy+7
       dato=""
       for xj=xx to xx+7
          c=Point(xj,yk)
          if c=1 then
               dato="1"+dato
            else
               dato="0"+dato
          end if
       next xj
       vettore(indice)=converti(dato)
       indice=indice+1
    next yk
 end sub

 sub salva_f
 dim ciclo as integer

 Open "hires.dat" For Binary As #1
    for ciclo=0 to 7999+2-1
       Put #1, ciclo+1, chr(vettore(ciclo))
    next
 Close #1
 end sub
 Screen 13, 32
 Cls
 BLoad "immagine.bmp"
 dim x as integer
 dim y as integer
 vettore(0)=0
vettore(1)=32
indice=2
for y=0 to 24
    for x=0 to 39
       elabora(x,y)
    next x
 next y
salva_f

 sleep
```

Utilizzo del file generato dal tool

Come adoperare il file "**hires.dat**"? E' possibile inserirlo all'interno di un file .d64 e gestirlo come preferite, magari con un paio di semplici istruzioni basic. Allo scopo è presente un file **graph.d64** che contiene sia hires.dat che un semplice programma BASIC per visualizzare l'immagine grafica ivi contenuta. Per fare funzionare il tutto agganciate il file immagine **graph.d64** al vostro emulatore VICE, quindi caricate da BASIC l'immagine digitando così:

LOAD "hires.dat",8,1 (e battete invio)

quindi, al termine del caricamento, proseguite con:

LOAD "PGM",8 (e battete invio)

ed al termine del caricamento battete:

RUN (ed invio)

Vi apparirà una immagine simile a questa:

Programma BASIC per la visualizzazione della immagine grafica generata dal tool:

```
10 poke 53272, peek(53272) or 8 :rem bitmap at 8192
20 poke 53265, peek(53265) or 32: rem bitmap on
30 :
40 for i=1024 to 2023: poke i,1:next
50 :
60 get a$: if a$=""then60
70 poke 53272,21: poke53265,27
80 print chr$(147);
90 end
```

Per chi predilige invece l'assembly... abbiamo pensato anche a voi! All' interno della cartella su github troverete il codice assembly necessario per la visualizzazione della immagine, ed un file batch go.bat che potrete adoperare per lanciane la compilazione mediante KickAssembler. Tuttavia, tenendo conto della semplicità e della brevità del codice, dovrebbe essere possibile modificarlo agevolmente per adattarlo a qualsiasi altro compilatore.

Listato Assembly per visualizzare l'immagine grafica generata dal tool:

```
*= $1000
:BasicUpstart2(start)
start:
            lda $d018
            ora #$08
            sta $d018
            lda $d011
            ora #$20
            sta $d011
  // riempie la memoria schermo
  // (locazioni 1024-2023)
  // con il valore 1.
            ldx #$00
loop:
            lda #$01
            sta $0400,x
            sta $0500,x
            sta $0600,x
            sta $06E8,x
            inx
            bne loop

            jmp *
.pc=$2000
.import c64 "hires.dat"
```

Riassumendo, il tool presentato è un tool estremamente semplice che permette di convertire una immagine bitmap (monocolore, 320x200 pixels) dal PC in un file immediatamente fruibile dal nostro C64, emulato oppure reale. Di questi tools, come già detto, ne esistono diversi e certamente anche migliori, ma questo ci permetterà (grazie anche al fatto di poterne modificare le caratteristiche a nostro piacimento dal momento che ne possediamo il relativo codice sorgente), di manipolare le immagini per ottenere risultati davvero interessanti. Nella seconda parte di questo articolo, che troverete nel prossimo numero di RetroMagazine, vedremo come generare una immagine grafica **FLI,** con molti più colori a disposizione rispetto ai quattro colori per cella che abbiamo di norma adoperando il modo multicolor sul C64, effettuando alcune semplici modifiche al tool.

Chi volesse compilare da sé il tool, di cui forniamo comunque il relativo eseguibile, dovrà scaricare il compilatore FreeBasic partendo da questo link fornito di seguito: https://www.freebasic.net/

A questo link invece troverete un editor che permette di lavorare con Freebasic in maniera sicuramente più agevole ed intuitiva: http://fbide.freebasic.net/

Per il momento è tutto, amici lettori di "RetroMagazine".

Alla prossima!

Codice automutante in BASIC – parte 1

di Francesco Fiorentini

Con questo articolo vorrei riprendere in mano la programmazione in BASIC del C64, che ho abbandonato dal primo numero di RM, provando a fare qualcosa di diverso dal solito ed allo stesso tempo lanciare un guanto di sfida ai nostri lettori ed agli utenti del gruppo **8 Bit RetroProgramming Italia** con il quale RM ha stretto un proficuo sodalizio.

Piccola premessa, il seguente articolo non ha la pretesa di generare complesso codice automutante con il BASIC V2, cosa peraltro ottenibile solo ricorrendo a codice Assembly da programmatori veramente skillati, quanto piuttosto gettare le basi e far riflettere su come, giocando opportunamente con alcune semplici PEEK e POKE e ben conosciuti indirizzi di memoria, sia possibile ottenere effetti divertenti anche con il BASIC. Ma partiamo dall'inizio che e' sempre la cosa migliore da fare.

Come sicuramente molti dei nostri lettori sapranno, il C64 memorizza i listati BASIC a partire da un indirizzo di memoria configurato dai puntatori **43** e **44**:

Hex	Dec	Commento
$002B	43	Low Byte, start of BASIC code default: 2049 / $0801
$002C	44	High Byte, start of BASIC code default: 2049 / $0801

Tale dato puo' essere facilmente recuperato anche da un semplice programmino in BASIC che legga i valori dei puntatori 43 e 44 e stampi, dopo un opportuno calcolo, il risultato a video:

```
10 print peek(43)
20 print peek(44)
30 print peek(43)+256*peek(44)
```

Una volta eseguito questo piccolo programmino avremo come risultato:

```
1
8
2049
```

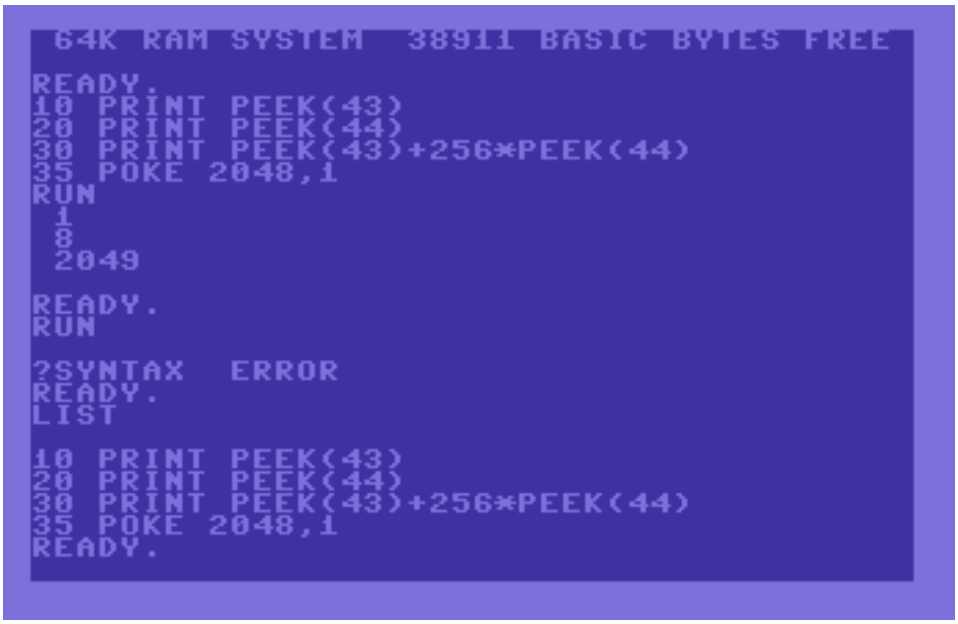


Figura 1 – Un programma BASIC che puo' essere eseguito 1 sola volta...

Come riportato anche nella tabella dei puntatori, **2049** e' l'indirizzo di partenza del nostro codice BASIC. Fin qui niente di strano, volendo potremmo cambiare questo indirizzo utilizzando la formula in modo inverso (piu' un piccolo accorgimento che vedremo di seguito*), ma basterebbe leggere di nuovo i puntatori 43 e 44 per scoprire dove il nostro codice BASIC e' stato spostato.
La cosa si fa leggermente piu' interessante se andiamo a stampare il contenuto della cella di memoria immediatamente precedente a quella che contiene l'indirizzo di partenza del codice BASIC, nel nostro caso la cella 2048.
Proviamo quindi ad aggiungere questa linea al nostro codice precedente:

```
40 print peek(2048)
```

Il risultato sara': 0 (zero)
Fin qui niente di eccezionale... ma cosa succede se proviamo a scrivere un valore qualsiasi (diverso da zero) in quella stessa cella di memoria? Proviamoci:

```
10 print peek(43)
20 print peek(44)
30 print peek(43)+256*peek(44)
35 poke 2048,1
```

Dopo essere stato eseguito la prima volta, il nostro programmino non ne vorra' piu' sapere di essere eseguito, ritornando un laconico ?SINTAX ERROR. Potremo listare il codice ed appurare che e' corretto sintatticamente, ma lo stesso non ne vorra' sapere di funzionare.
Semplicemente il C64 si aspetta di trovare nella cella di memoria immediatamente precedente a quella che contiene l'indirizzo di partenza del codice BASIC il valore 0(zero). Se cosi' non fosse, il codice BASIC, nonostante la sua correttezza, non verrebbe eseguito. Pensate un po' se questo valore fosse cambiato all'interno di un programma di migliaia di righe pieno di PEEK e POKE; a quanti verrebbe in mente di controllare una cosa simile? :-D

*Proprio a questo mi riferivo nel mio commento tra parentesi, se vi venisse la voglia di spostare l'area dedicata al codice BASIC, oltre ad aggiornare i puntatori, ricordatevi di azzerare l'indirizzo di memoria immediatamente precedente.

Adesso che sappiamo dove risiede il nostro codice BASIC, proviamo a giocare con il codice e fare in modo che lo stesso muti a seconda delle nostre esigenze...

Provate a dare un'occhiata alla tabella dei codici PETSCII pubblicata all'indirizzo https://www.c64-wiki.com/wiki/PETSCII, vi viene in mente come potremmo usarla al nostro scopo?

Proviamo a scrivere un programmino che stampi a video del semplice testo:

```
10 print "prova1"
20 print "prova2"
30 print "prova3"
```

Conoscendo l'area dedicata al codice BASIC e' facile ricavare i seguenti valori come gli indirizzi di memoria che puntano alle righe:

```
2051 (10)
2066 (20)
2081 (30)
```

Allo stesso modo e' semplice ricavare le seguenti informazioni:

```
2061 (49) - valore 1 di "prova1"
2076 (50) - valore 2 di "prova2"
2091 (51) - valore 3 di "prova3"
```

Proviamo quindi a modificare il nostro programma nel seguente modo:

```
10 print "prova1"
20 print "prova2"
30 print "prova3"
40 poke 2061,51
50 poke 2076,49
60 poke 2091,50
```

Ed eseguiamolo con un run:

```
prova1
prova2
prova3
```

Se proviamo a listare di nuovo il nostro programma pero' ci accorgeremo che qualcosa e' cambiato...:

```
10 print "prova3"
20 print "prova1"
30 print "prova2"
40 poke 2061,51
50 poke 2076,49
60 poke 2091,50
```

Ed una volta eseguito di nuovo, ovviamente il risultato sara' tutt'altro:

```
prova3
prova1
prova2
```

Interessante vero? Vi viene in mente niente che potrebbe beneficiare del codice che venga mutato ogni volta che lo stesso viene eseguito?

Ma continuiamo pure il nostro articolo introducendo il concetto dei **BASIC TOKEN**. Un BASIC TOKEN e' la rappresentazione tramite un singolo byte di un'istruzione BASIC.
Si veda la tabella pubblicata alla pagina https://www.c64-wiki.com/wiki/BASIC_token per una lista esaustiva di tutti i token basic del Commodore 64.

Senza ombra di dubbio interessante, ma come possiamo usare questa informazione a nostro vantaggio? In realta' ci potremmo fare millemila cose, tipo ampliare i comandi del BASIC V2, ma questa e' un'altra storia, al momento limitiamoci a ricavare le seguenti informazioni:

```
2053 (153) - token di PRINT della riga 10
2068 (153) - token di PRINT della riga 20
2083 (153) - token di PRINT della riga 30
```

Provate un po' ad indovinare cosa succederebbe se aggiungessi al nostro programmino queste 3 righe di codice:

```
40 poke 2053,143
50 poke 2068,143
60 poke 2083,143
```

Trasformandolo in:

```
10 print "prova3"
20 print "prova1"
30 print "prova2"
40 poke 2053,143
50 poke 2068,143
60 poke 2083,143
```

A questo punto, leggendo la tabella dei TOKEN e con le conoscenze di cui sopra, vi dovrebbe essere abbastanza facile unire i puntini e capire che una volta eseguito il programma ci ritroveremmo con un codice del tutto differente:

```
10 rem "prova3"
20 rem "prova1"
30 rem "prova2"
40 poke 2053,143
50 poke 2068,143
60 poke 2083,143
```

Che, se fosse eseguito nuovamente, non stamperebbe niente perche' il comando **PRINT** (token 153) e' stato sostituito da un comando **REM** (token 143)...

Ovviamente il tutto puo' essere condensato in un'unica riga di codice:

```
10 print "prova1":poke 2053,143
```

Che una volta eseguita dara' come risultato:

```
10 rem "prova1":poke 2053,143
```

Divertente, vero? ;-)

Ma spingiamoci un pochino oltre; secondo voi e' possibile realizzare un codice BASIC che si autocancelli?

Date un'occhiata al seguente programmino e provate a capire che cosa faccia:
`10 print "prova1":poke 631,49:poke 632,48:poke 633,13:poke 198,3`

Esatto, simula la scrittura di una riga 10 vuota seguita da un RETURN che di fatto cancella il codice stesso...

Bene, come detto all'inizio dell'articolo questo e' soltanto un assaggio di cosa sia possibile fare con il BASIC V2 e qualche minima conoscenza dell'architettura del Commodore 64. Avete appurato come sia semplice imbrogliare l'interprete BASIC ma anche come sia facile capire dove sta l'imbroglio.

Questo articolo verra' ripreso all'interno del gruppo **8 Bit RetroProgramming Italia**, di cui sia io che **Marco Pistorio** facciamo parte del team degli amminstratori. Il nostro intento e' quello di lanciare la sfida ai lettori ed agli utenti del gruppo a chi riesca a creare un codice automodificante in BASIC che sia al tempo stesso efficace e non semplice da individuarsi, oppure che suggerisca altri trucchetti da aggiungere alla lista (volutamente limitata e niente affatto esaustiva).

I codici ed i trucchi piu' meritevoli, con tanto di spiegazione dell'autore e test di efficacia, verranno pubblicati sul prossimo numero di RetroMagazine. Ah, per non limitare il tutto al Commodore 64, si accettano codice e trucchi anche per altre piattaforme 8 bit (Spectrum, MSX, Amstrad...), anzi, saranno parte integrante delle prossime puntate!

Che aspettate quindi, il guanto di sfida e' stato scagliato, chi si chinera' a raccoglierlo?

ADDENDUM MANUALE "INTRODUZIONE ALL'ARCHITETTURA DEL Commodore 64"

di Attilio Capuozzo - Admin Fondatore Gruppo FB "8 Bit Retroprogramming Italia"

Salve a tutti, a seguito della pubblicazione, **sul N.11 di RetroMagazine (Pag.62), del mio Manuale "Introduzione all'Architettura del C64",** ho ricevuto diverse mail di richiesta di approfondimento di alcuni aspetti a cui cercherò di dare risposta nel presente Addendum.

1. Approfondimento POKE 53272

Vediamo innanzitutto di chiarire meglio l'istruzione **POKE 53272,21** riportata nel suddetto Manuale a cui vi rimando per ogni dettaglio:

nel Manuale ho spiegato che impostando il **Registro 24 del Chip Video VIC-II** (**53272/$D018**) al **valore decimale 21** andiamo a ripristinare il valore di default della **Character Memory Standard del C64** e in particolare puntiamo al **SET 1 Maiuscolo/Grafico** (o Standard CBM) mentre con il **valore decimale 23** (impostazione a 1 dei bit di posizione 1 e 2 del suddetto Registro come descritto nel Manuale) puntiamo al **SET 2 Minuscolo/Maiuscolo** (o Commerciale).

Vorrei chiarire, inoltre, che con il valore **21 l'Offset** (o Indirizzo Relativo) del CharSet è **4096** che nel **Banco 2**, dove è presente l'altra **ROM Image** (v. Manuale), diventa **36864**, Indirizzo Assoluto ricavato dalla somma tra l'Offset e il Bank Address (N. Banco*16384):

Offset + N. Banco*16384 = 4096 + 2*16384 = 36864

Nel Banco 0 è intuitivo che Offset e Indirizzo Assoluto si equivalgono.

Allo stesso modo è facile calcolare che con il **valore decimale 23** si punterà **all'Indirizzo Relativo/Offset 6144** (corrispondente al **SET 2**) sia nel Banco 0 che nel Banco 2:

4096 (Starting Address SET 1) + 2048 (=2K ossia Dimensione di ogni singolo SET) = 6144

Infine, vorrei ribadire che è piuttosto semplice selezionare il blocco da 2K da noi scelto per la Character Memory RAM, basta infatti scrivere nel Nibble meno significativo del Registro 53272/$D018 un numero pari compreso tra 0 e 14 che rappresenta l'Offset (espresso in K) dallo Starting Address del corrente Banco da 16K.

L'istruzione da usare è la seguente:

POKE53272,(PEEK(53272)AND240)ORCM

dove l'AND logico lascia inalterato il Nibble Alto del Registro e azzera i 4 bit più bassi (Nibble meno significativo) per poi settarli correttamente con la successiva operazione di OR in cui CM è appunto l'Indirizzo di inizio della Character Pattern Memory espresso in Kbytes.

2. Approfondimento 16K MEMORY BANK

Ora vediamo di approfondire la scelta dell'opportuno **Banco di Memoria da 16K (Video Bank),** del Chip Video VIC-II, per la memorizzazione dei blocchi di dati grafici come gli Sprite Data, i Character Data, ossia la Character Memory RAM per i Programmable Character (o **UDG** – **User Defined Graphics Characters** -), la Text Screen Memory e la Bitmap (o Pagina Grafica).

Nel Manuale tutti gli esempi si riferiscono al **Banco N.0** che è quello di default e che va dall'indirizzo di memoria **0/$00** alla locazione **16383/$3FFF**.
In questo Banco abbiamo un'area compresa tra **828/$033C** e **1019/$03FB, il Cassette I/O Buffer** (**o Tape Buffer**) usato per le operazioni di LOAD/SAVE/PRINT#/INPUT# del Datasette, che possiamo utilizzare per blocchi di memoria grafici come ad esempio 3 definizioni di Sprite (Sprite Shapes) da 64 byte ciascuno.

In particolare potremo utilizzare i seguenti **numeri** di **Sprite Data Block** (64 Byte) come valori da passare agli **Sprite Data Pointer** (**da 2040/$07F8 a 2047/$07FF**):

13 (13*64=832), 14 (14*64=892) e 15 (15*64=960)

E' da notare che l'ultimo Sprite Data Block Number, il **15**, fa uso anche di ulteriori **4 free Byte** allocati subito dopo il Tape Buffer (**1020/$03FC – 2023/$03FF**)

Esiste anche un'altra area di RAM libera che va dalle locazioni **704/$02C0 a 767/$02FF** e che corrisponde allo **Sprite Data Block Number 11**.

In questo caso non abbiamo bisogno di modificare i Puntatori al **Top della Memoria RAM** riservata al **Programma BASIC** (**Puntatori TOM – Top Of Memory -55/$37 e 56/$38**).
Peraltro la **Screen Memory** di default occupa le locazioni **1024-2047 ($0400-$07FF)** e dunque non entra in conflitto con l'Area riservata al testo BASIC (**BASIC Program Text Area**) a meno di opportune rilocazioni da parte nostra che in tal caso la potrebbero fa rientrare nel Workspace della BASIC RAM.

Per memorizzare, invece, molti più blocchi di Sprite Pattern, così come blocchi da 2K dei Character Pattern, ci possiamo affidare all'area di memoria che inizia dall'indirizzo **12288/$3000 in poi**

.
In tal caso, poiché ci troviamo nello spazio di memorizzazione riservato alla BASIC RAM compreso tra **2048/$0800 e 40959/$9FFF,** dobbiamo intervenire sul **Puntatore 56/$38** per riservarci una cosiddetta "**Safe Area**" dedicata ai dati grafici inserendo la seguente riga all'inizio del nostro programma:

POKE 56,48:CLR

Vi ricordo che l'istruzione CLR è necessaria per aggiornare i Puntatori e per assicurare che la **Storage Area delle Stringhe Dinamiche** (**Puntatori BOS – Bottom Of Strings - 51/$33 e 52/$34**) inizi al di sotto della nostra "**Safe Area**".

In questo modo lasciamo al Programma BASIC, nonché alle Tabelle delle Variabili, degli Array (Vettori e Matrici) e all'Area delle Stringhe Dinamiche **uno spazio di memoria di 10K**.

E' opportuno a questo punto descrivere meglio l'Area di Memoria riservata alle Stringhe che nell'ambito della BASIC RAM è allocata al di sopra della BASIC Program Text Area:

le Variabili Stringhe occupano **7 Byte** allo stesso modo degli altri 2 tipi di Variabili del BASIC V2, ossia le Variabili Intere e le Float (Floating Point o a Virgola Mobile) e sono memorizzate dall'interprete BASIC suddividendole in 2 parti: avremo dunque una entry della String Variable memorizzata nella **Tabella delle Variabili Semplici** (**BASIC Variable Storage Area**) e questa entry, a sua volta, conterrà i puntatori alla Stringa salvata in un'area separata di memoria.

Se la Variabile Stringa è "**dinamica**" (**Dynamic String**) ossia deriva da un'operazione di concatenazione (**A$="Hello"+B$**) oppure di manipolazione (**LEFT$, RIGHT$, MID$, STR$**) o ancora da un'input da tastiera (**INPUT e GET**) allora la Stringa contenente i caratteri sarà memorizzata nella **String Text Storage Area** (vista in precedenza) **altrimenti** sarà interna alla **BASIC Program Text Area** (e.g. A$="8 BIT RETROPROGRAMMING ITALIA").

I puntatori alla **BASIC Variable Storage Area** sono i cosiddetti **Puntatori SOV – Start Of Variables** – **45/$2D** e **46/$2E.**

Tornando al discorso dei Banchi di Memoria da 16K, se invece volessimo lasciare alla **BASIC RAM** un maggior spazio di memorizzazione allora dovremmo cambiare Banco e in particolare selezionare il **Banco 2** che va dagli indirizzi **32768/$8000** a **49151/$BFFF**.
Se effettuassimo dunque uno switching al Banco 2, potremmo far coincidere la **Starting Location della nostra Safe Area con il Base Address del Banco 2,** ossia 32768/$8000.

In tal caso potremmo organizzare la nostra Safe Area per i seguenti scopi:

- l'Area di Memoria che va da **32768/$8000 a 33791/$83FF** è uno spazio sufficiente per **16 blocchi da 64 Bytes** di dati per la definizione degli **Sprite** (**Sprite Data o Sprite Pattern**)

- **da 33792/$8400 a 34815/$87FF** per la **Video Screen Memory da 1K** (la rilocazione della Screen Memory ai predetti Indirizzi avviene in **automatico** a seguito dello Switch al Banco 2)

- **da 34816/$8800 a 36864/$9000** per **la Character Memory RAM** (ossia un New Character Set/Caratteri Ridefiniti nell'ambito della User Graphics RAM)

Vi ricordo che nei seguenti Indirizzi, **36864/$9000-40959/$9FFF,** è presente la **Character ROM Image per il Chip VIC-II** (v. Manuale).

In tal modo lasceremmo alla **BASIC RAM, uno spazio di memoria di ben 30K su 38911 BASIC BYTES FREE** (come recita la schermata iniziale del C64) corrispondenti a circa **38K**.

In un nostro Programma BASIC, per switchare correttamente al **Memory Video Bank N.2,** andremo a scrivere le segg. linee:

```
10 POKE56,128:CLR
20 POKE648,132
30 POKE56576,(PEEK(56576)AND252)OR1
```

La riga 20 è necessaria nei Programmi BASIC in quanto la locazione 648 serve ad aggiornare la Routine KERNAL della PRINT affinché punti correttamente all'attuale Base Address della Screen Memory rilocata a seguito del cambio di Banco, altrimenti continuerebbe a puntare all'Indirizzo 1024/$0400 (v. Manuale). Solo nel BITMAP MODE non è necessario aggiornare la locazione 648 poiché la Screen Memory assume un diverso significato per la Bitmap (ossia diventa una Color Map) e dunque la normale PRINT dei Caratteri non può essere effettuata sulla Pagina Grafica.

Prima di uscire dal Programma dovremmo ripristinare le condizioni di memoria iniziali inserendo le righe seguenti (la numerazione, ovviamente, è a titolo esemplificativo):

```
500 POKE55,0:POKE56,160:CLR
510 POKE56576,(PEEK(56576)AND252)OR3
520 POKE648,4
```

3. CODICE ASCII, PETSCII e SCREEN CODE

E' opportuno precisare che nell'ambito della bibliografia tecnica dedicata al C64 (come anche degli altri 8 bit Commodore), il termine **ASCII** viene spesso utilizzato come sinonimo (e dunque del tutto equivalente) del termine **PETSCII** (o **PETASCII**).

Il **PETSCII**, infatti, è l'implementazione Commodore, estesa e personalizzata (256 Caratteri), dell'**ASCII** (128 Caratteri) che è l'acronimo di **American Standard Code for Informations Interchange**.

Dunque ASCII = PETSCII (o PETASCII) = Commodore ASCII Code.

Nel BASIC se si utilizza la PRINT, i Codici Numerici dei caratteri corrispondono ai Codici ASCII e quindi PETSCII; se invece utilizziamo la POKE, per scrivere il carattere direttamente nella SCREEN MEMORY, allora ci dobbiamo riferire agli **SCREEN CODE** (indicati anche come **DISPLAY CODE** o **CHARACTER CODE**).
In definitiva, a seconda di COME vogliamo visualizzare un Carattere ci riferiremo o alla **TABELLA dei CODICI ASCII/PETSCII** (**caso PRINT**) oppure alla **TABELLA degli SCREEN CODE** (**caso POKE nella CHARACTER o TEXT SCREEN MEMORY**) tenendo anche presente che entrambe le Tabelle contemplano i codici per tutti e 2 i SET di Caratteri Standard del C64: **il SET 1 MAIUSCOLO/GRAFICO** e il **SET 2 MINUSCOLO/MAIUSCOLO**.

4. TERMINOLOGIA GRAPHICS MODE

Infine vorrei parlare della questione della terminologia del C64 che purtroppo non ha mai trovato uno standard di fatto, a partire dall'autorevole **C64 Programmer's Reference Guide**, lasciando quindi campo libero, ai numerosissimi testi tecnici cronologicamente successivi alla manualistica Commodore, di adottare i termini più disparati ereditati successivamente nelle varie forme anche dalle più recenti fonti Internet (C64 Wiki e similari) **con la conseguenza di un'attuale marasma eterogeneo di termini tecnici che ancora oggi non riesce a trovare una giusta e doverosa forma standardizzata**.

L'intento che ho cercato di perseguire fortemente nel Manuale così come nel presente Addendum, è quello di adottare i termini, inglesi e italiani, che fossero più appropriati nonché più diffusi cercando di spiegare gli argomenti secondo un'ottica che fosse la più "accreditata" possibile secondo la mia personale esperienza filtrata anche e soprattutto dallo studio di decine di testi considerati tra i più autorevoli mai scritti sul C64.

Veniamo ora all'argomento che dà il titolo a quest'ultimo Paragrafo del presente Addendum:

il **BITMAPPED DISPLAY** è spesso indicato anche come **HIGH RESOLUTION SCREEN** in contrapposizione al **LOW RESOLUTION SCREEN** del **TEXT DISPLAY**.

La motivazione va ricercata essenzialmente nella differenza tra il numero di elementi base dello Schermo singolarmente controllabili (o indirizzabili).

Nella modalità BitMap, la Pagina Grafica è larga 320 pixel (o dots o punti) e alta 200 per un totale di 64.000 pixel controllabili individualmente.
Nel Character Mode, invece, lo Schermo è costituito da 40*25 Character Blocks (o Character Space o ancora Character Cells) singolarmente controllabili: dunque potremo riempire il nostro Text Screen di 1000 caratteri presumibilmente grafici (i cosiddetti caratteri PETSCII) o di User Defined Characters (i Caratteri Ridefiniti o UDG o ancora Programmable Characters).

Va correttamente aggiunto che anche nel Character Mode ci troviamo virtualmente in uno Schermo di 320*200 pixel in quanto ogni singolo Character Block rappresenta una matrice di 8*8 pixel dove a ogni singolo pixel corrisponde un singolo bit in memoria (per codificare un Character Pattern sono necessari 8 Byte=64 bit o pixel).

Non a caso, infatti, la letteratura Commodore usa la definizione **High Resolution Character Mode** come sinonimo di **Standard Character Mode**.
Per quanto riguarda la terminologia relativa ai punti di immagine, va detto che la letteratura Commodore utilizza prevalentemente il termine "**dot**" e che solo nella bibliografia tecnica cronologicamente successiva alla manualistica di riferimento Commodore ci si inizia a riferire in maniera quasi esclusiva al termine "**pixel**" come sinonimo più esplicativo di dot in quanto derivante dalla contrazione di "**picture element**".

Vorrei solo aggiungere che ancorché nell'accezione attuale è ormai largamente consolidato l'uso del termine pixel come equivalente di dot, ci torna comunque utile considerare il dot come l'elemento di base di una immagine sullo Schermo Video (Video Screen) ai fini di una migliore comprensione di argomenti quali il **MULTICOLOR MODE** (**MCM**) attivabile sia nel BitMap Mode che nel Character Mode.

Senza entrare nei dettagli tecnici che esulano dallo scopo del presente Paragrafo, nel Multicolor Mode viene utilizzata in memoria una coppia di bit (**bit pair**) per codificare una corrispondente coppia di dot (**dot pair**) in modo da incrementare il numero di colori utilizzabili (basta pensare che un pattern di 2 bit permette 4 possibili combinazioni). **La dot pair diventa quindi un double-wide pixel**.
Per contro avremo un dimezzamento della Risoluzione Orizzontale che in un Multicolor Bitmapped Display diverrà di 160 pixel di larghezza mentre nel Multicolor Text il Character Block passerà da una matrice di 8*8 pixel a una di 4*8 in cui i pixel hanno un cosiddetto Aspect Ratio (PAR) di 2:1 ossia pixel di larghezza doppia (double-wide pixel).

Virtualmente il numero di dot sullo Schermo della Risoluzione Orizzontale non cambia in quanto i punti risultano solo accoppiati (dot pair-->bit pair->double-wide pixel) e dunque in realtà sia la modalità Standard (o High Resolution) che Multicolor conservano lo stesso numero di dots.

È infine interessante notare come il blocco di base di 8*8 pixel (Character Block) del Text Display è in realtà presente anche nel Bitmapped Display e viene generalmente denominato **ATTRIBUTE CELL** (Cella Attributo), o più semplicemente **CELL** (Cella), e corrisponde a un gruppo di 8 Byte consecutivi che formano un'area quadrata avente la stessa dimensione di un singolo carattere.

Le Cells, in maniera simile al Text Display, sono disposte in 40 Colonne e 25 Righe. Nell'ambito di ciascuna di questa Area di 8 Byte di altezza, in uno STANDARD (HIRES) GRAPHICS MODE (TEXT o BITMAPPED DISPLAY) si possono utilizzare solo 2 colori - Background e Foreground Color- mentre nel caso di un MULTICOLOR GRAPHICS MODE i colori salgono a 4.

In definitiva abbiamo:

- **l'HIGH RESOLUTION SCREEN** corrispondente a un **BITMAPPED DISPLAY** che può trovarsi in una delle 2 seguenti modalità:
    - **STANDARD BITMAP MODE o HIRES BITMAP MODE** (che potremmo anche definire **Hires Monocolor Mode**) con Schermo (Bitmapped Display) di 320*200 pixel e Pixel Aspect Ratio (PAR) 1:1
    - **MULTICOLOR BITMAP MODE** (che potremmo anche indicare con **Hires Multicolor Mode**), con Schermo (Bitmapped Display) di 160*200 pixel e PAR 2:1
- **il LOW RESOLUTION SCREEN** corrispondente al **TEXT DISPLAY** a cui sono associati i seguenti Graphics Mode:
    - **STANDARD CHARACTER MODE o HIRES CHARACTER MODE** (talvolta indicato anche come **REGULAR TEXT MODE**) con Schermo (Text Display) di 40*25 CHARACTER BLOCKS di 8*8 pixel ognuno e PAR 1:1; le stesse suddette caratteriche tecniche appartengono anche a un altro Text Mode, **l'EXTENDED BACKGROUND COLOR MODE (ECM)**, o più semplicemente **EXTENDED COLOR MODE**, in cui si possono scegliere tra 4 diversi colori per il Background Color del Carattere
    - **MULTICOLOR CHARACTER MODE** (o **MULTICOLOR TEXT MODE**) con Schermo (Text Display) di 40*25 CHARACTER BLOCKS della dimensione ognuno di 4*8 pixel e PAR 2:1.

E' bene, infine, notare come il Multicolor Text Mode può essere agevolmente utilizzato in combinazione con l'Hires Character Mode in quanto il Multicolor può essere attivato o disattivato per ogni **SINGOLO** Character Block del Text Display.

IMMAGINE TRATTA DA COMMONS.WIKIMEDIA.ORG

Esplorando l'Amiga - parte 5

di Leonardo Giordani

NdE: da un paio di numeri a questa parte per garantire una maggiore leggibilita' del codice, abbiamo preferito formattare alcuni articoli su 2 colonne al posto delle canoniche 3.

Le gestione della memoria

La gestione della memoria è sempre una delle parti più ricche e complesse di un'architettura, principalmente perché la memoria disponibile è sempre meno di quanta ne vorremmo avere a disposizione. (https://users.ece.cmu.edu/~koopman/titan/rules.html)

Questo è ad ogni modo un argomento molto interessante in quanto generalmente la memoria è gestita sia dall'hardware che dal software. Il microprocessore fornisce un'unità di gestione della memoria (memory management unit, MMU) che obbliga a seguire un determinato schema e quindi il software deve implementare i propri algoritmi di gestione partendo da questo.

L'Amiga originariamente fu costruita attorno ad un Motorola 68000, che non fornisce alcuna gestione della memoria via hardware. Questo significa che il processore non può in alcun modo fermare un tentativo di lettura della memoria da parte di un processo, caratteristica che mancava anche ai primi processori Intel x86. Intel "risolse" il problema introducendo la modalità protetta (protected mode) nella famiglia 386 (anche se una versione iniziale di tale tecnologia era già presente nei 286). Motorola aggiunse una MMU esterna al processore con i modelli 68010 e 68020, mentre il 68030 e i processori seguenti possedevano una MMU interna.

Il Motorola 68k è un processore a 32 bit, pertanto i registri e il bus di indirizzamento hanno quella dimensione. La memoria, invece, è connessa solamente a 24 delle 32 linee del bus, che riduce lo spazio di indirizzamento a disposizione del microprocessore a 24 bit, ovvero un totale di 16 mebibyte invece dei possibili 4 gibibyte. Tale quantitativo di memoria era ad ogni modo sufficiente per l'epoca in cui l'Amiga fu progettata. Si consideri che il modello di Amiga più affermato, l'Amiga 500, aveva 500 KiB di memoria, che alcuni dei più fortunati aumentavano a 1 MiB tramite un'espansione hardware.

La mancanza di uno schema imposto dal processore significa che all'accensione la memoria è semplicemente un'area piatta a cui il microprocessore può accedere direttamente. Come abbiamo visto precedentemente Exec crea la propria struttura in memoria, generando i nodi di libreria e creando la jump table. Tutto questo avviene durante l'inizializzazione della macchina, un processo notevolmente complesso che prevede per l'appunto la creazione delle strutture di gestione della memoria.

L'indirizzo base di Exec

Ogni programmatore Amiga sa che l'indirizzo `0x4` contiene l'indirizzo base della libreria Exec, e ho mostrato precedentemente come questo sia usato in congiunzione con la jump table per chiamare le funzioni esposte da quest'ultima.

La ragione per cui l'indirizzo base di Exec è salvato in quella zona di

memoria viene spiegata raramente. La scelta di quell'indirizzo, però, è tutt'altro che casuale.
La famiglia di microprocessori 68000 riserva il primo kibibyte di memoria per i vettori delle eccezioni (exception vectors), ovvero il codice che viene eseguito in caso di errore. "Errore" qui significa qualcosa di sbagliato a livello hardware, da una divisione per zero ad un errore del bus. Questo è imposto dall'architettura di questi processori e quindi è una caratteristica che l'Amiga ha in comune con altri computer e console basati su di essi.

I primi due vettori sono utilizzati quando il processore viene acceso (o in generale quando viene eseguito un reset), e il vettore numero 1 (il secondo) all'indirizzo `0x4` è chiamato Reset Initial Program Counter.

Dopo un reset il processore inizializza il Program Counter (l'indirizzo della successiva istruzione che verrà eseguita) con il valore salvato all'indirizzo `0x4` nella memoria. Quando la CPU viene accesa, però, la ROM di Kickstart viene copiata in memoria, pertanto gli indirizzi 0 e 4 (i primi due vettori) sono gli indirizzi che compaiono nella ROM stessa.

I primi 8 kibibyte della ROM di Kickstart sono

```
00000000: 1111
00000002: 4ef9
00000004: 00fc
00000006: 00d2
```

e si nota chiaramente che la long word all'indirizzo `0x4` è `00fc 00d2`. A tale indirizzo possiamo pertanto trovare il codice iniziale della ROM stessa

```
000000d2: 4ff9 0004 0000            lea     0x40000.l,sp
```

che imposta lo stack pointer, e di cui parlerò in un articolo futuro.

L'indirizzo `0x4` viene quindi utilizzato durante un reset, e può essere usato per altri scopi durante il funzionamento normale. In caso di reset, inoltre, esso viene automaticamente sovrascritto dal codice della ROM,

pertanto risulta un luogo estremamente sicuro dove salvare l'indirizzo base di Exec.

List headers

Exec gestisce la memoria e le risorse utilizzando (https://en.wikipedia.org/wiki/Linked_list). Come è noto, per gestire una lista concatenata abbiamo bisogno dell'indirizzo della testa della lista, della coda, e del penultimo elemento, per permettere alla coda di venir scollegata e sostituita. È evidente che l'unico indirizzo realmente necessario è quello della testa, in quanto gli altri indirizzi possono essere trovati percorrendo la lista (la coda punta per convenzione all'indirizzo 0). I due indirizzi addizionali, ad ogni modo, possono semplificare in modo notevole il codice che gestisce la lista e soprattutto incrementare le prestazioni del codice, in quanto non è necessario effettuare un attraversamento completo della lista per trovare l'ultimo elemento ogni volta che vogliamo operare su questa.

La libreria Exec fornisce una buona struttura per la gestione delle liste, chiamata `LH`, definita in `include_i/exec/lists.i`

```
* Full featured list header
*
   STRUCTURE    LH,0
    APTR    LH_HEAD
    APTR    LH_TAIL
    APTR    LH_TAILPRED
    UBYTE   LH_TYPE
    UBYTE   LH_pad
    LABEL   LH_SIZE ;word aligned
```

Gli unici due campi che questa struttura aggiunge rispetto a quando abbiamo detto precedentemente sono `LH_TYPE`, che contiene prevedibilmente il tipo di dato che viene gestito dalla lista, e `LH_pad` che è esattamente quello che il nome suggerisce, un padding che permette alla struttura di essere allineata ad una word (come riportato nel commento).

Dobbiamo ora scoprire dove Exec salva l'intestazione (header) della lista che gestisce la memoria. Analizzando la struttura `ExecBase` contenuta in `include_i/exec/execbase.i` troviamo la seguente definizione

```
******* System List Headers (private!)
*******************************

    STRUCT  MemList,LH_SIZE
    STRUCT  ResourceList,LH_SIZE
    STRUCT  DeviceList,LH_SIZE
    STRUCT  IntrList,LH_SIZE
    STRUCT  LibList,LH_SIZE
    STRUCT  PortList,LH_SIZE
    STRUCT  TaskReady,LH_SIZE
    STRUCT  TaskWait,LH_SIZE
    STRUCT  SoftInts,SH_SIZE*5
```

che è esattamente quello che vogliamo. Quando Exec si installa nella prima parte della memoria inizializza anche questi header per tenere traccia delle risorse corrispondenti.

È interessante notare, comunque, che la struttura `ExecBase` descritta nel file include non viene usata direttamente, ma è più una descrizione di quanto il codice andrà a creare. Questo è differente da quanto accade in un linguaggio ad alto livello come il C. In questo linguaggio il programmatore dichiara una struttura, riserva della memoria per essa e poi accede ai suoi campi, ma in Assembly, fondamentalmente, non c'è alcun concetto di struttura e campo. Tutto ciò che abbiamo è della memoria (piatta) e degli indirizzi.

Visto che `ExecBase` è una descrizione della struttura di Exec una volta che questa viene installata in memoria è interessante analizzare i suoi campi e annotarsi gli indirizzi relativi di ciascuno di essi.

Ho preso il codice contenuto in `include_i/exec/execbase.i` e ho calcolato l'indirizzo di ciascun campo. La prima colonna contiene l'indirizzo relativo nella struttura (cominciando pertanto da `0x0`), mentre la seconda colonna contiene l'indirizzo relativo all'indirizzo di base di Exec. Come ho mostrato nel quarto articolo di questa serie le strutture `LN` e `LIB` occupano i primi 34 byte, pertanto i successivi indirizzi partono da `0x22`

```
0000 0022    UWORD   SoftVer         ; kickstart release
number (obs.)
0002 0024    WORD    LowMemChkSum    ; checksum of 68000 trap
vectors
0004 0026    ULONG   ChkBase         ; system base pointer
complement
0008 002a    APTR    ColdCapture     ; coldstart soft capture
vector
000c 002e    APTR    CoolCapture     ; coolstart soft capture
vector
0010 0032    APTR    WarmCapture     ; warmstart soft capture
vector
0014 0036    APTR    SysStkUpper     ; system stack base
(upper bound)
0018 003a    APTR    SysStkLower     ; top of system stack
(lower bound)
001c 003e    ULONG   MaxLocMem       ; top of chip memory
0020 0042    APTR    DebugEntry      ; global debugger entry
point
0024 0046    APTR    DebugData       ; global debugger data
segment
0028 004a    APTR    AlertData       ; alert data segment
002c 004e    APTR    MaxExtMem       ; top of extended mem,
or null if none

0030 0052    WORD    ChkSum          ; for all of the above
(minus 2)


******* Interrupt Related
*****************************************

    LABEL   IntVects
0032 0054    STRUCT  IVTBE,IV_SIZE
003e 0060    STRUCT  IVDSKBLK,IV_SIZE
004a 006c    STRUCT  IVSOFTINT,IV_SIZE
0056 0078    STRUCT  IVPORTS,IV_SIZE
0062 0084    STRUCT  IVCOPER,IV_SIZE
006e 0090    STRUCT  IVVERTB,IV_SIZE
007a 009c    STRUCT  IVBLIT,IV_SIZE
0086 00a8    STRUCT  IVAUD0,IV_SIZE
0092 00b4    STRUCT  IVAUD1,IV_SIZE
009e 00c0    STRUCT  IVAUD2,IV_SIZE
00aa 00cc    STRUCT  IVAUD3,IV_SIZE
00b6 00d8    STRUCT  IVRBF,IV_SIZE
00c2 00e4    STRUCT  IVDSKSYNC,IV_SIZE
00ce 00f0    STRUCT  IVEXTER,IV_SIZE
00da 00fc    STRUCT  IVINTEN,IV_SIZE
00e6 0108    STRUCT  IVNMI,IV_SIZE


******* Dynamic System Variables
************************************

00f2 0114    APTR    ThisTask        ; pointer to current
task (readable)

00f6 0118    ULONG   IdleCount       ; idle counter
00fa 011c    ULONG   DispCount       ; dispatch counter
00fe 0120    UWORD   Quantum         ; time slice quantum
0100 0122    UWORD   Elapsed         ; current quantum ticks
```

```
0102 0124    UWORD   SysFlags       ; misc internal system
flags
0104 0126    BYTE    IDNestCnt      ; interrupt disable
nesting count
0105 0127    BYTE    TDNestCnt      ; task disable nesting
count

0106 0128    UWORD   AttnFlags      ; special attention
flags (readable)

0108 012a    UWORD   AttnResched    ; rescheduling attention
010a 012c    APTR    ResModules     ; pointer to resident
module array
010e 0130    APTR    TaskTrapCode   ; default task trap
routine
0112 0134    APTR    TaskExceptCode ; default task exception
code
0116 0138    APTR    TaskExitCode   ; default task exit code
011a 013c    ULONG   TaskSigAlloc   ; preallocated signal
mask
011e 0140    UWORD   TaskTrapAlloc  ; preallocated trap mask


******* System List Headers (private!)
*******************************

0120 0142    STRUCT  MemList,LH_SIZE
012e 0150    STRUCT  ResourceList,LH_SIZE
013c 015e    STRUCT  DeviceList,LH_SIZE
014a 016c    STRUCT  IntrList,LH_SIZE
0158 017a    STRUCT  LibList,LH_SIZE
0166 0188    STRUCT  PortList,LH_SIZE
0174 0196    STRUCT  TaskReady,LH_SIZE
0182 01a4    STRUCT  TaskWait,LH_SIZE
0190 01b2    STRUCT  SoftInts,SH_SIZE*5

01e0 0202    STRUCT  LastAlert,4*4

01f0 0212    UBYTE   VBlankFrequency        ;(readable)
01f1 0213    UBYTE   PowerSupplyFrequency   ;(readable)

01f2 0214    STRUCT  SemaphoreList,LH_SIZE

0200 0222    APTR    KickMemPtr     ; ptr to queue of mem
lists
0204 0226    APTR    KickTagPtr     ; ptr to rom tag queue
0208 022a    APTR    KickCheckSum   ; checksum for mem and
tags

020c 022e    UBYTE ExecBaseReserved[10];
0216 0238    UBYTE ExecBaseNewReserved[20];
022a 024c    LABEL   SYSBASESIZE
```

Secondo questa struttura ci aspettiamo di trovare l'intestazione della lista che gestisce la memoria 322 byte dopo l'indirizzo base (`0x142`), che significa che questo numero deve essere citato da qualche parte nel codice di Kickstart.

Non è quindi sorprendente che la funzione `AllocMem` utilizzi tale numero. La funzione è parte dell'API di Exec che abbiamo esplorato nel terzo e quarto articolo, e seguendo lo stesso metodo descritto in essi ho trovato la funzione all'indirizzo `0x17d0` del codice di Kickstart

```
; memoryBlock = AllocMem(byteSize, attributes)
; d0                    d0        d1

000017d0: 522e 0127                addq.b  #0x1,0x127(a6)
000017d4: 48e7 3020                movem.l d2-d3/a2,-(sp)
000017d8: 2600                     move.l  d0,d3
000017da: 2401                     move.l  d1,d2
000017dc: 45ee 0142                lea     0x142(a6),a2
000017e0: 2452                     movea.l (a2),a2
[...]
```

In questa parte iniziale della funzione possiamo chiaramente vedere che il codice carica l'indirizzo effettivo di `0x142(a6)`. Si ricordi che si suppone che il registro `a6` contenga sempre l'indirizzo base di Exec.

Il dislocamento (offset) `0x142` è anche citato in una tabella verso la fine del codice, ed è questa parte quella a cui siamo più interessati in questo momento

```
; Initialise list headers

000002b0: 43fa 0020                lea     0x2d2(pc),a1
000002b4: 3019                     move.w  (a1)+,d0
000002b6: 6700 0086                beq.w   0x33e
000002ba: 41f6 0000                lea     (0,a6,d0.w),a0
000002be: 2088                     move.l  a0,(a0)
000002c0: 5890                     addq.l  #0x4,(a0)
000002c2: 42a8 0004                clr.l   0x4(a0)
000002c6: 2148 0008                move.l  a0,0x8(a0)
000002ca: 3019                     move.w  (a1)+,d0
000002cc: 1140 000c                move.b  d0,0xc(a0)
000002d0: 60e2                     bra.b   0x2b4

; List headers

000002d2: 0142 000a
000002d6: 0150 0008
000002da: 015e 0003
000002de: 017a 0009
000002e2: 0188 0004
000002e6: 0196 0001
000002ea: 01a4 0001
000002ee: 016c 0002
000002f2: 01b2 000b
000002f6: 01c2 000b
000002fa: 01d2 000b
000002fe: 01e2 000b
00000302: 01f2 000b
00000306: 0214 000f
0000030a: 0000
```

Ho formattato il codice in modo da mostrare che i valori dopo `02d2` sono dati e non codice. Il disassembler ovviamente mostrerà delle istruzioni a quegli indirizzi, ma sono solamente delle interpretazioni forzate del codice binario.

Questa tabella è immediatamente seguita dalla tabella che abbiamo trovato nel terzo articolo, quando abbiamo cercato i valori della struttura `LIB`.

Analizziamo il codice a `02b0` riga per riga

```
; Initialise list headers

000002b0: 43fa 0020                lea     0x2d2(pc),a1
000002b4: 3019                     move.w  (a1)+,d0
000002b6: 6700 0086                beq.w   0x33e
000002ba: 41f6 0000                lea     (0,a6,d0.w),a0
000002be: 2088                     move.l  a0,(a0)
000002c0: 5890                     addq.l  #0x4,(a0)
000002c2: 42a8 0004                clr.l   0x4(a0)
000002c6: 2148 0008                move.l  a0,0x8(a0)
000002ca: 3019                     move.w  (a1)+,d0
000002cc: 1140 000c                move.b  d0,0xc(a0)
000002d0: 60e2                     bra.b   0x2b4

000002b0: 43fa 0020                lea     0x2d2(pc),a1
```

Prima di tutto il codice carica l'indirizzo assoluto di `0x2d2(pc)` nel registro `a1`. Questo è esattamente l'indirizzo iniziale della tabella, come mostrato precedentemente.

```
000002b4: 3019                    move.w  (a1)+,d0
000002b6: 6700 0086               beq.w   0x33e
```

Il codice poi carica il primo valore della tabella (`0142`) in `do` e incrementa il registro utilizzato (`a1`). Questo suggerisce che stiamo eseguendo un ciclo. Le successive istruzioni sono in effetti un confronto che salta a `0x33e` se il valore è 0. Si può vedere facilmente che la tabella in questione è terminata da `0000`.

```
000002ba: 41f6 0000               lea     (0,a6,d0.w),a0
```

Il registro `ao` viene poi utilizzato per salvare l'indirizzo base di Exec + `do`. Questo significa che utilizziamo il valore che abbiamo appena letto come un puntatore. Nel primo passaggio del ciclo, quindi, stiamo puntando `0x142` byte dopo l'inizio della libreria Exec, esattamente dove ci aspettavamo di trovare l'intestazione della lista che gestisce la memoria.

```
000002be: 2088                    move.l  a0,(a0)
000002c0: 5890                    addq.l  #0x4,(a0)
```

Una lista concatenata vuota ha la testa che punta alla coda e la coda che punta a zero. Per fare questo impostiamo il contenuto dell'indirizzo `(ao)` all'indirizzo stesso, e poi lo incrementiamo di 4. Si ricordi che `ao` contiene l'indirizzo effettivo (assoluto) del campo `LH_HEAD` della struttura `MemList` (di tipo `LH_SIZE`), pertanto con `move.l ao,(ao)` noi salviamo in quel campo l'indirizzo della struttura `MemList` stessa. Aumentandolo di 4 non facciamo altro che cambiare il valore di quel campo in modo che punti al campo successivo, 4 byte più avanti, ovvero `LH_TAIL`.

```
000002c2: 42a8 0004               clr.l   0x4(a0)
000002c6: 2148 0008               move.l  a0,0x8(a0)
```

Il contenuto della coda (campo `LH_TAIL`) viene settato a 0 e il campo `LH_TAILPRED`, il predecessore della coda, viene riempito con il valore della testa della lista.

```
000002ca: 3019                    move.w  (a1)+,d0
000002cc: 1140 000c               move.b  d0,0xc(a0)
```

La successiva word nella tabella (`000a`) viene poi salvata in un campo 12 bytes (`0xc`) dopo l'inizio della struttura, ovvero in `LH_TYPE`. I valori possibili di questo byte possono essere trovati nel file `include_i/exec/nodes.i`, in cui scopriamo che il valore `0xa` corrisponde a `NT_MEMORY`.

```
;------ Node Types for LN_TYPE

NT_UNKNOWN      EQU 0
NT_TASK         EQU 1   ; Exec task
NT_INTERRUPT    EQU 2
NT_DEVICE       EQU 3
NT_MSGPORT      EQU 4
NT_MESSAGE      EQU 5   ; Indicates message currently
pending
NT_FREEMSG      EQU 6
NT_REPLYMSG     EQU 7   ; Message has been replied
NT_RESOURCE     EQU 8
NT_LIBRARY      EQU 9
NT_MEMORY       EQU 10
NT_SOFTINT      EQU 11  ; Internal flag used by SoftInts
NT_FONT         EQU 12
NT_PROCESS      EQU 13  ; AmigaDOS Process
NT_SEMAPHORE    EQU 14
NT_SIGNALSEM    EQU 15  ; signal semaphores
NT_BOOTNODE     EQU 16
NT_KICKMEM      EQU 17
NT_GRAPHICS     EQU 18
NT_DEATHMESSAGE EQU 19

NT_USER         EQU 254 ; User node types work down from
here
NT_EXTENDED     EQU 255
```

A questo punto rimane solamente un'istruzione

```
000002d0: 60e2                    bra.b   0x2b4
```

che salta indietro all'inizio del ciclo. Quest'ultimo continua leggendo l'intera tabella fino a raggiungere il terminatore di lista `0000`.

Il contenuto finale della memoria a `0x142` sarà quindi (i commenti mostrano i campi della struttura)

```
00000142: 0000 0146 ; LH_HEAD
00000146: 0000 0000 ; LH_TAIL
0000014a: 0000 0146 ; LH_TAILPRED
0000014d: 0a        ; LH_TYPE
0000014e: [...]
```

Questo avviene anche per tutte le rimanenti 7 liste, da `ResourceList` a `TaskWait`

Resources

Microprocessor-based system design by Ricardo Gutierrez-Osuna http://courses.cs.tamu.edu/rgutier/ceg411_f01/ , in particular [Lesson 9 - Exception processing]: http://courses.cs.tamu.edu/rgutier/ceg411_f01/l9.pdf

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

Note for the English readers

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/2018/06/25/exploring-the-amiga-5/

C portabile e ottimizzato per gli 8-bit - parte 2

Seconda parte: Tecniche per ottimizzare il codice C per 8-bit

di Fabrizio Caruso

Questa è la seconda 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

Consigliamo la lettura del primo articolo in cui abbiamo presentato i vari cross compilatori C per architetture 8 bit e abbiamo dato alcune indicazioni su come scrivere codice C portabile su tutte le architetture 8 bit.

Ottimizzare il codice in generale

Ci sono alcune regole generali per scrivere codice migliore indipendentemente dal fatto che l'architettura sia 8-bit o meno.

Non tutte le buone pratiche di programmazione saranno ottimali per gli 8 bit. In questa sezione diamo esempi di pratiche generali che rimangono valide anche su architetture 8 bit vintage.

Riutilizziamo le stesse funzioni

In generale, in qualunque linguaggio di programmazione si voglia programmare, è importante evitare la duplicazione del codice o la scrittura di codice superfluo.

Programmazione strutturata

Spesso guardando bene le funzioni che abbiamo scritto scopriremo che condividono delle parti comuni e che quindi potremo rifattorizzare il codice introducendo delle sotto-funzioni che le nostre funzioni chiameranno.

Dobbiamo però tenere conto che, oltre un certo limite, una eccessiva granularità del codice ha effetti deleteri perché una chiamata ad una funzione ha un costo computazionale e di memoria.

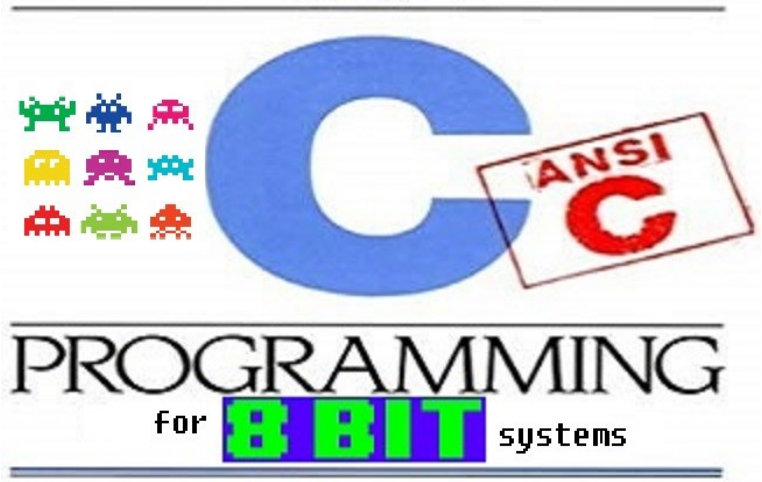
Generalizzare il codice parametrizzandolo

In alcuni casi è possibile generalizzare il codice passando un parametro per evitare di scrivere due funzioni diverse molto simili.

Passiamo delle variabili

Se due funzioni fanno la stessa cosa su oggetti diversi, sarebbe meglio avere una unica funzione a cui si passi l'oggetto su cui agire.

Passiamo delle funzioni



In altri casi avremo porzioni di codice simili la cui unica differenza è l'applicazione di una funzione diversa. In questo caso possiamo scrivere un'unica funzione a cui si passa un puntatore a funzione.

Non tutti sono familiari con la sintassi dei puntatori a funzione e quindi ne diamo un esempio in cui definiamo la funzione sumOfSomething(range, something) che somma il valore di something(i) per i che va da zero a i-1 :

```
unsigned short sumOfSomething(unsigned char range, unsigned short
(* something) (unsigned char))
{
    unsigned char i;
    unsigned short res =0;
    for(i=0;i<range;++i)
    {
            res+=something(i);
    }
    return res;
}
```

Quindi date due funzioni:

```
unsigned short square(unsigned char val)
{
    return val*val;
}

unsigned short next(unsigned char val)
{
    return ++val;
}
```

potremo usare sumOfSomething con l'una o l'altra funzione evitando di scrivere il codice che fa la somma due volte:

```
printf("%d\n",sumOfSomething(4,square));
```

mostrerà 14, cioè la somma dei quadrati di 0,1,2,3.

```
printf("%d\n",sumOfSomething(4,next));
```

mostrerà 10, cioè la somma di 1,2,3,4.

Passiamo degli offset di struct

In altri casi possiamo avere due funzioni quasi identiche la cui unica differenza è il campo di uno struct che si modifica. In questo caso possiamo scrivere un'unica funzione a cui passiamo l'offset dello struct .

Un esempio avanzato si trova in https://github.com/Fabrizio-Caruso/CROSS-CHASE/blob/master/src/chase/character.h dove, dato uno struct con due campi _x e _y , vogliamo potere agire sul valore di uno o dell'altro in situazioni diverse:

```
struct CharacterStruct
{
    unsigned char _x;
    unsigned char _y;
    ...
};
typedef struct CharacterStruct Character;
```

Possiamo evitare di scrivere due diverse funzioni per agire su _x e su _y creando una unica funzione a cui si passa un offset che faccia da selettore:

```
unsigned char moveCharacter(Character* hunterPtr, unsigned char
offset)
{
    if((unsigned char) * ((unsigned char*)hunterPtr+offset) < ... )
    {
            ++(*((unsigned char *) hunterPtr+offset));
    }
    else if((unsigned char) *((unsigned char *) hunterPtr+offset) > ... )
    {
            --(*((unsigned char *) hunterPtr+offset));
    }
...
}
```

Stiamo sfruttando il fatto che il secondo campo _y si trova esattamente un byte dopo il primo campo _x . Quindi con offset=0 accediamo al campo _x e con offset=1 accediamo al campo _y . I vari cast a unsigned char * servono per accedere a byte in posizioni diverse all'interno dello struct .

Avvertenze: Dobbiamo però considerare sempre che aggiungere un parametro ha un costo e quindi dovremo verificare (anche guardando la taglia del binario ottenuto) se nel nostro caso ha un costo inferiore al costo di una funzione aggiuntiva.

Stesso codice su *oggetti* simili

Si può anche fare di più e usare lo stesso codice su oggetti che non sono esattamente dello stesso tipo ma che condividono solo alcuni aspetti comuni per esempio sfruttando gli offset dei campi negli struct , puntatori a funzioni, etc. Questo è possibile in generale tramite la programmazione ad oggetti di cui descriveremo una implementazione light per gli 8-bit nel prossimo articolo.

Pre-incremento/decremento vs Post-incremento/decremento

Bisogna evitare operatori di post-incremento/decremento (i++ , i--) quando non servono (cioè quando non serve il valore prima dell'incremento) e sostituirli con (++i , --i). Il motivo è che l'operatore di post-incremento richiede almeno una operazione in più dovendo conservare il valore originario.

Nota: E' totalmente inutile usare un operatore di post-incremento in un ciclo for .

Costanti vs Variabili

Una qualunque architettura potrà ottimizzare meglio del codice in cui delle variabili sono sostituite con delle costanti. Quindi se una data variabile ha un valore noto al momento della compilazione, è importante che sia rimpiazzata con una costante.

Se il suo valore, pur essendo noto al momento della compilazione, dovesse dipendere da una opzione di compilazione (per esempio il tipo di target), allora la sostituiremo con una macro da settare attraverso una opzione di compilazione, in maniera tale che sia trattata come una costante dal compilatore.

Ottimizzare il codice per gli 8-bit

Il C è un linguaggio che presenta sia costrutti ad alto livello (come struct, le funzioni come parametri, etc.) sia costruiti a basso livello (come i puntatori e la loro manipolazione). Questo non basta per farne un linguaggio direttamente adatto alla programmazione su macchine 8-bit.

Aiutiamo il compilatore a ottimizzare le costanti

Per compilatori single pass (come la maggioranza dei cross-compilatori 8-bit come per esempio CC65), può essere importante aiutare il compilatore a capire che una data espressione sia una costante.

Esempio (preso da https://www.cc65.org/doc/coding.html):

Un compilatore single pass valuterà la seguente espressione da sinistra a destra non capendo che OFFS+3 è una costante:

```
#define OFFS 4
int i;
i = i + OFFS + 3;
```

Quindi sarebbe meglio riscrivere i = i + OFFS+3 come i = OFFS+3+i oppure i = i + (OFFS+3) :

```
#define OFFS 4
int i;
i = OFFS+3+i;
```

Implementare peek e poke in C

Quasi sicuramente su una architettura 8-bit avremo bisogno di scrivere e leggere dei singoli byte su alcune specifiche locazioni di memoria. Il modo per fare questo in BASIC sarebbe stato attraverso i comandi **peek** e **poke**.

In C dobbiamo farlo attraverso dei puntatori la cui sintassi non è leggibilissima. Potremo però costruirci delle utili macro che useremo nel nostro codice:

```
#define POKE(addr,val) (*(unsigned char*) (addr) = (val))
#define PEEK(addr) (*(unsigned char*) (addr))
```

Nota: I compilatori scriveranno codice ottimale nel caso in cui si passino delle costanti come parametri.

Per maggiori dettagli facciamo riferimento a: https://github.com/cc65/wiki/wiki/PEEK-and-POKE

I "tipi migliori" per gli 8-bit

Una premessa importante per la scelta dei tipi da preferire per architettura è data dal fatto che in generale abbiamo questa situazione:

- tutte le operazioni aritmetiche sono solo a 8 bit
- la maggior parte delle operazioni sono ad 8 bit, alcune sono a 16-bit e nessuna operazione è a 32 bit
- le operazioni signed (cioè con segno) sono più lente di quelle unsigned
- l'hardware non supporta operazioni in virgola mobile

Evitiamo conversioni inutili

Le conversioni tra tipi e soprattutto le conversioni tra tipi signed e unsigned sono costose.

Tipi interi vs tipi a virgola mobile

Il C prevede tipi numerici interi con segno (char , short , int , long , long long e loro equivalenti in versione unsigned). Molti compilatori (ma non CC65) prevedono il tipo float (numeri a virgola mobile) che qui non tratteremo. Bisogna considerare che i float delle architetture 8-bit sono tutti software float ed hanno quindi un costo computazionale notevole. Sarebbero quindi da usare solo se strettamente necessari.

Il nostro amico *unsigned*

Siccome le architetture 8-bit che stiamo considerandno NON gestiscono ottimalmente tipi signed , dobbiamo evitare il più possibile l'uso di tipi numerici signed .

"Size matters!"

La dimensione dei tipi numeri standard dipende dal compilatore e dall'architettura e non dal linguaggio.

Più recentemente sono stati introdotti dei tipi che fissano la dimensione in modo univoco (come per esempio uint8_t per l'intero unsigned a 8 bit). Per includere questi tipi a taglia fissata dallo standard usiamo:

```
#include <stdint.h>
```

Non tutti i compilatori 8-bit dispongono di questo header.

Fortunatamente per la stragrande maggioranza dei compilatori 8-bit abbiamo la seguente situazione:

tipo	nr. bit	nome in stdint.h	nome alternativo
unsigned char	8	uint8_t	byte
unsigned short	16	uint16_t	word
unsigned int	16	uint16_t	word
unsigned long	32	uint32_t	dword

Quindi dovremo:

- usare il più possibile unsigned char (o uint8_t) per le operazioni aritmetiche;
- usare unsigned char (o uint8_t) e unsigned short (o uint16_t) per tutte le altre operazioni, evitando se possibile qualunque operazione a 32 bit.

Nota: In assenza dell'header stdint.h , sarebbe bene creare dei typedef opportuni:

```
typedef unsigned char uint8_t;
typedef unsigned short uint16_t;
typedef unsigned long uint32_t;
```

Scelta delle operazioni

Quando scriviamo codice per una architettura 8-bit dobbiamo evitare se possibile codice con operazioni inefficienti o che ci obblighino a usare tipi non adatti (come i tipi signed o tipi a 16 o peggio 32 bit).

Non produciamo signed

In particolare, se possibile, spesso si può riscrivere il codice in maniera da evitare sottrazioni e quando questo non è possibile, si può almeno fare in modo che il risultato della sottrazione sia sempre non-negativo.

Evitiamo i prodotti espliciti

Tutte le architetture che abbiamo preso in considerazione, con la sola esclusione di Motorola 6809, non dispongono di una operazione per effettuare il prodotto di due valori a 8 bit. Quindi, se possibile dobbiamo evitare i prodotti adattando il nostro codice, oppure limitarci a prodotti e divisioni che siano potenze di 2 e implementandoli con operazioni di shift con gli operatori << e >>:

```
unsigned char foo, bar;
...
foo << 2; // moltiplicazione per 2^2=4
bar >> 1; // divisione per 2^1=2
```

Riscrivere certe operazioni

Molte operazioni come il modulo possono essere riscritte in maniera più efficiente per gli 8 bit usando operatori bit a bit perché non sempre il compilatore ottimizza nel modo migliore. Quindi dobbiamo dargli una mano noi:

```
unsigned char foo;
...
if(foo&1) // equivalente a foo%2
{
    ...
}
```

Variabili e parametri

Uno dei più grossi limiti dell'architettura MOS 6502 non è la penuria di registri come si potrebbe pensare ma è la dimensione limitata del suo stack hardware (in pagina uno: $0100-01FF) che lo rende inutilizzabile in C per la gestioni dello scope delle variabili e i parametri delle funzioni.

Quindi un compilatore ANSI C per 6502 sarà quasi sicuramente costretto a usare uno stack software per

- gestire lo scope delle variabili,
- gestire il passaggio dei parametri.

Le altre architetture 8-bit che stiamo considerando soffrono meno di questo problema ma la gestione delle scope delle variabili e dei parametri ha un costo anche quando si usa uno stack hardware.

Un antipattern può aiutarci

Un modo per ridurre il problema è limitare l'uso delle variabili locali e dei parametri passati alle funzioni. Questo è chiaramente un antipattern e se lo applicassimo a tutto il nostro codice otterremo il classico spaghetti code. Dobbiamo quindi scegliere sapientemente quali variabili sono necessariamente locali e quali possono essere usate come globali. Avremo codice meno generico di quello che avremmo voluto ma sarà più efficiente. NON sto suggerendo di rendere tutte le variabili globali e di non passare mai parametri alle funzioni.

[6502] Usare funzioni non re-entrant

Il compilatore CC65 per l'architettura MOS 6502 mette a disposizione l'opzione -Cl che rende tutte le variabili locali come se fossero static, quindi globali. Questo ha l'effetto di evitare l'uso dello stack software per il loro scope. Ha però l'effetto di rendere tutte le nostre funzioni non re-entrant. In pratica questo ci impedisce di usare funzioni ricorsive. Questa non è un grave perdita perché la ricorsione sarebbe comunque una operazione troppo costosa per una architettura 8-bit.

[6502] Usare la pagina zero

Il C standard prevede la keyword register per suggerire al compilatore di mettere una variabile in un registro.

In genere i compilatori moderni ignorano questa keyword perché lasciano questa scelta ai loro ottimizzatori. Questo è vero per I compilatori in questione ad eccezione di CC65 che la usa come suggerimento al compilatore per mettere una variabile in pagina zero. Il MOS 6502 accede in maniera più efficiente a tale pagina di memoria. Si può guadagnare memoria e velocità.

Per quanto riguarda l'architettura MOS 6502, il sistema operativo di queste macchine usa una parte della pagina zero. Resta comunque una manciata di byte a disposizione del programmatore. CC65 per default lascia 6 byte della pagina zero a disposizione delle variabili dichiarate con keyword register. Potrebbe sembrare quindi ovvio dichiarare molte variabili come register ma NON è così semplice perché tutto ha un costo. Per mettere una variabile sulla pagina zero sono necessarie diverse operazioni. Quindi se ne avrà un vantaggio quando le variabili sono molto usate.

In pratica i due scenari in cui è conveniente sono:

1. parametri di tipo puntatore a struct usati almeno 3 volte all'interno di una funzione,
2. variabile in un loop che si ripete almeno un centinaio di volte.

Un riferimento più preciso è dato da: https://www.cc65.org/doc/cc65-8.html

Il mio consiglio è quello di compilare e vedere se il binario è divenuto più breve o, ancora meglio, ispezionare il codice Assembly generato.

Evitare l'allocazione dinamica della memoria

I compilatori che stiamo considerando consentono di allocare e deallocare la memoria dinamicamente (con comandi come malloc e free) ma questo ha un ovvio costo computazionale. Se possibile è preferibile allocare tutta la memoria staticamente.

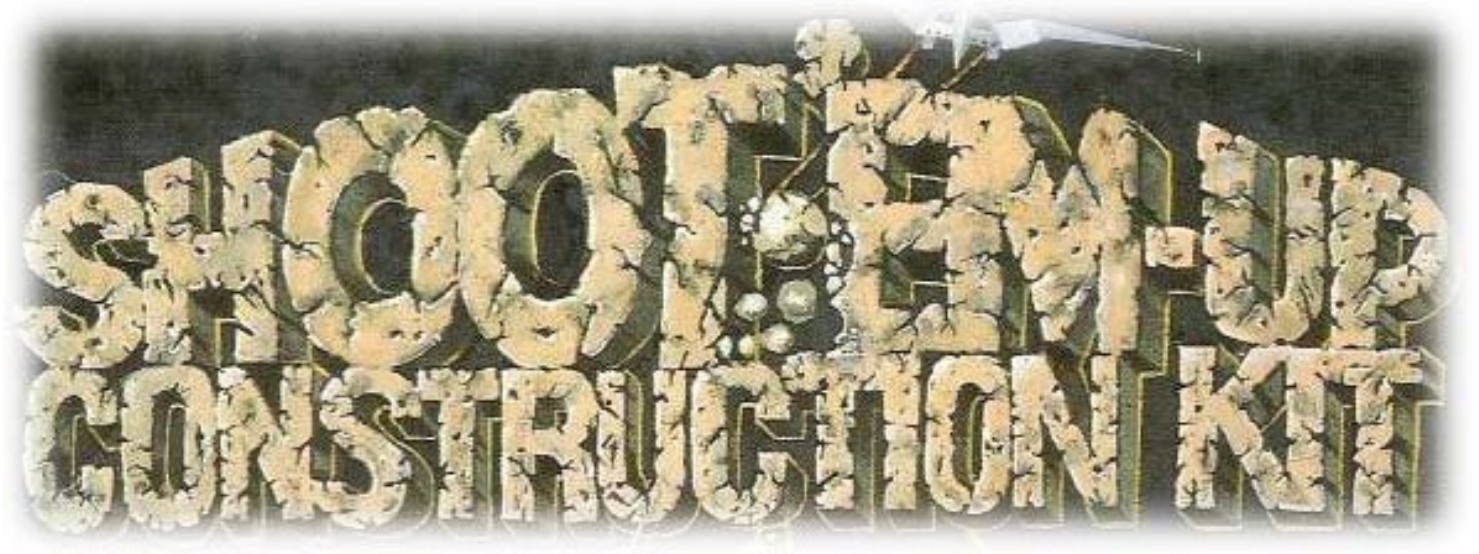
Codice su file diversi?

In generale è bene separare in più file il proprio codice se il progetto è di grosse dimensioni.

Questa buona pratica può però avere degli effetti deleteri per gli ottimizzatori dei compilatori 8-bit perché in generale non eseguono link-time optimization, cioè non ottimizzeranno codice tra più file ma si limitano ad ottimizzare ogni file singolarmente.

Quindi se per esempio abbiamo una funzione che chiamiamo una sola volta e la funzione è definita nello stesso file in cui viene usata, l'ottimizzatore potrebbe metterla in line ma non lo farà se la funzione è definita in un altro file.

Il mio consiglio **NON** è quello di creare file enormi con tutto ma è quello di tenere comunque conto di questo aspetto quando si decide di separare il codice su più file e di non abusare di questa buona pratica.



Il mio intento sarà quello di fare un manuale basato sull'originale in lingua inglese e poi di farvi capire, spero in maniera semplice ed intuitiva, i meccanismi del tool "S.e.u.c.k." per il nostro computer Commodore 64 o emulatore (com ad es. il Vice).

Svilupperemo anche un livello di video-game fatto con seuck partendo dalle basi e qualche breve cenno storico sul tool in questione.
Cominciamo con la nostra guida.

SHOOT'EM-UP CONSTRUCTION KIT

Nel 1987 i programmatori di PARALLAX e WIZBALL, decisero di rivelare in parte i loro segreti commerciali.
Era qualcosa di speciale, si trattava del "Shoot'Em Up Construction Kit" che ha rivelato i segreti dei programmatori di videogiochi professionali dandoci il potere sviluppare video-giochi per competere con i più grandi programmatori e sviluppatori di video-game dell'epoca.
Non è stato necessario alcun addestramento o esperienza di programmazione per utilizzare la selezione di programmi di utilità progettati da Sensible Software. Usando il tuo joystick puoi costruire il tuo veicolo spaziale e le armi, progettare i tuoi paesaggi, effetti sonori, livelli e onde d'attacco. Puoi iniziare da zero o riprogettare uno dei 4 giochi GRATUITI inclusi nel pacchetto. SHOOT 'EM UP CONSTRUCTION KIT era una parte essenziale di ogni collezione di proprietari di Commodore 64, con le versioni Amiga e Atari ST che seguivano poco dopo.
Bene, dopo dei brevi cenni storici inziamo il nostro tutorial.

E' indispensabile conoscere il **MENU** del **S.e.u.c.k. C64**.

STRUTTURA DEL MENU

Procediamo a caricare il nostro tool S.e.u.c.k. su di un *floppy disk (o disco immagine " Seucko1.D64 "se usate un emulatore VICE) con il comando classico del BASIC del Commodore 64 con il **LOAD" * ",8** ...

Dopo il *LOAD " * ",8* comparirà sul vostro monitor schermata Seuck...

Come abbiamo detto, la struttura del **MENU** dopo il caricamento in C64 si presenterà in quest'ordine:
Il nostro esempio si baserà sul caricamento del tool Seuck su disco ma se avete la versione Tape potrete usare il comando Basic **LOAD per il nastro (per comodità vi consiglio la versione disco).*

Bene, inizieremo la nostra guida con l'argomento "sprite".

Non vorrei dilungarmi molto su cos'è uno sprite, ma è necessario sapere per chi non ha mai affrontato il problema dello sviluppo e la programmazione di un video-game sul C64 e di cos'è uno "sprite" farò un breve cenno.

SPRITE: Con sprite, in informatica, si indica un'immagine in *grafica raster, generalmente bidimensionale (2D), che fa parte di una scena più grande (sfondo o background) e che può essere spostata in maniera indipendente rispetto ad essa può essere sia statica che dinamica.

Esempi di sprite sono gli oggetti 2D inclusi nei giochi, le icone delle interfacce grafiche ecc..

Nella computer grafica, la grafica raster (detta anche g. bitmap) è una tecnica usata per descrivere un'immagine in formato digitale.

Il termine raster (= trama, reticolo, griglia, rasta) ha origine nella tecnologia televisiva analogica, ovvero dal termine che indica le righe orizzontali (dette anche scan line) dei televisori o dei monitor). In computer grafica, indica la griglia ortogonale di punti che costituisce un'immagine raster. Nella grafica raster l'immagine viene vista come una scacchiera e ad ogni elemento della scacchiera, chiamato pixel, viene associato uno specifico colore. Il colore può essere definito con varie tecniche...
Non voglio dilungarmi molto sull'argomento

colori e tecnica raster a colori 8,16,32 bit...; troverete tutte le informazioni facendo una ricerca con il nostro caro amato *Google*, passiamo al nostro MENU SEUCK C64 e al suo primo comando per editare i *sprite "EDIT SPRITE - S"*.
*Su questo è bene notare che per editare in Seuck C64 si fa uso sia del **joystick** che della **tastiera** con dei tasti dedicati. (Esempio per gli sprite useremo il (Tasto - S).

EDIT SPRITES - S

SPRITES sono le piccole bestie che vedi correre in tutti i giochi arcade. L'eroe, la nave, gli alieni, i proiettili, le esplosioni - questi sono tutti SPRITES. Tutto ciò che si muove è uno SPRITE. Tutto ciò che è uno SPRITE si sposterà.

EDIT SPRITES Tasto [**S**] ti permetterà di colorare, ruotare, girare e generalmente manipolare tutti i **127 sprite** che puoi usare nel tuo gioco, come preferisci.

In questo **MENU** useremo il *JOYSTICK* per spostarci in su e giù ed il Tasto *FIRE* per selezionare oppure la tastiera con: [**S**] (per editare i sprite), [**F1,F2,F3,F5,F7,S,M,C,E**] ed il Tasto [**X**] per uscire dal MENU.

*Con il VICE si può usare sia il Joystick che la Tastiera del pc (setterete in Vice nel "Menu Settaggi/Joystick" (Nord, Sud, Est ed Ovest e Fire) il [**SU,GIU,SINISTRA e DESTRA** con i tasti direzionali del PC e per Fire (Selezione Menu) il Tasto **CTRL** destro o sinistro come vi è comodo].*

Per tornare indietro dovrete usare sempre il Tasto **[X]** che sarà il nostro punto di riferimento molto importante per quando andremo a modificare qualcosa, permettendoci di tornare indietro nei Menu e nelle schermate del Seuck C64.

SELECT SPRITE - F1
Puoi usare il joystick o F1 per selezionare lo Sprite da modificare.

EDIT SPRITE - F3
Permette di spostare il quadratino lampeggiante su tutta la griglia e di riempire i PIXEL con il colore attuale premendo il FIRE. *(Creazione Sprite Pixel per Pixel)

EDIT COLOUR - F5
Modifica il colore delle palette con il tasto F5 andate in basso nelle palette colore e con *Joystick Tastiera (Tasti cursore)* per muovervi da un colore all'altro e Fire per selzione.

SELECT COLOUR - F7
Scelta COLORI per le PALETTE C64. Voi Avete a disposizione **3 colours PIU' il Background** (Sfondo con ColoreTrasparente *che non vedrete nel gioco) . You'll be lumbered with two of them for all of your sprites so make sure you really want them. The third can be different on every sprite you design.

SLIDE SPRITE
Questo permette di far scivolare il vostro sprite tra un casella (*pixel) e l'altra.

MIRROR SPRITE
Vi permette di riflettere lo sprite nella direzione opposta (specchio) o in verticale oppure in direzione orizzontale.
La direzione cambia muovendo il Joystick o Tastiera (tasti cursore) per poi selezionarla con il FIRE o *CTRL*
**per il Vice se avete selezionato nell'emulatore l'apposito tasto che sostituisce il Fire nei settagi di simulazione Joystick A o B...*

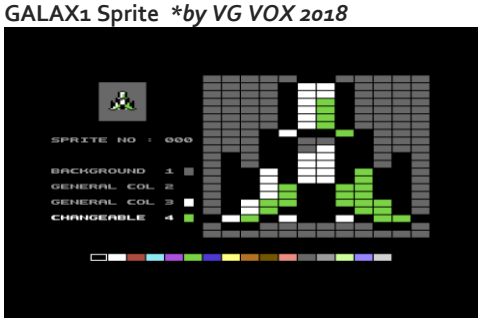
COPY SPRITE
Usate questa opzione per COPIARE lo SPRITE da un posto all'altro. Dalla posizione dello Sprite Origine a quello di Destinazione (classico copia incolla). Questa opzione vi sarà molto utile per fare le vostre animazioni.
Es. Vogliamo copiare lo Sprite da posizione "000" a posizione "001"; procederemo come segue: F1 selezioniamo il nostro **F1 - Select Sprite** ci spostiamo sullo Sprite desiderato che in questo caso sarà la "000" e premiamo il **Tasto C** poi **FIRE** *COPIA* TO "**000**" ci spostiamo (*Joy o Tastiera) da ***Sprite Origine*** che vogliamo COPIARE in ***Sprite Destinazione*** es. TO "**001**", spostandoci sempre con il Joystick o Tastiera (tasti cursore) e **FIRE** per Selezionare e terminare l'operazione.
Adesso il nostro Sprite è stato copiato da posizione **000** a posizione **001**.

ERASE SPRITE
Con il Tasto **E** *Erase Sprite* e rispondete **Y** alla domanda "ARE YOU SURE ?"; quindi premete il tato **Y** se volete cancellare tutto lo sprite completamente e **N** per rinunciare e conservare lo Sprite esistente in griglia.

Esci dal MENU premendo il *Tasto **X** vi farà tornare indietro da tutti i Menu del Seuck compreso quello di Test Gioco, Tiles set, Oggetti, Livelli ecc... Importante e anche comodo nei momenti di difficoltà. *Tasto Salvavita per i vostri giochi :D

Adesso vi farò un esempio con un mio gioco editato qualche mese fa; il gioco in questione è Galax1.
Vedremo come creare navicelle, omini, mostri, proiettili e tutte le schifezze che volete... ;)

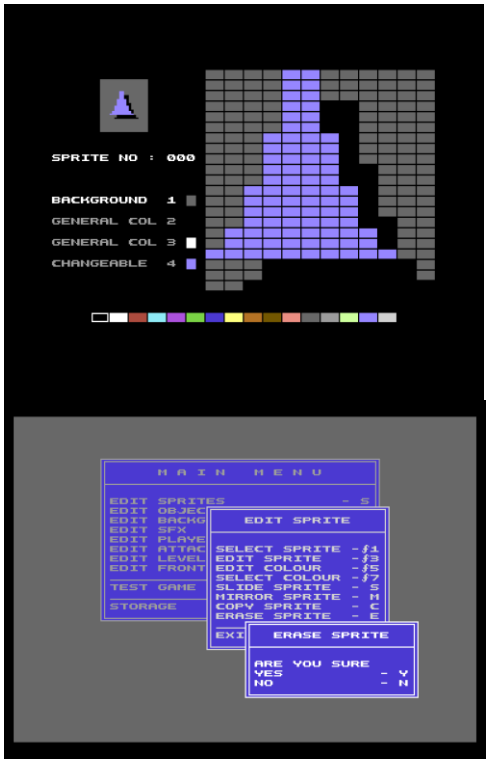
GALAX1 Sprite **by VG VOX 2018*



Lo **sprite** in questione, di mia creazione, ricorda una classica navicella spaziale dei videogame *Arcade* (per chi non lo sapesse i giochi Arcade sono tutti quei giochi da sala giochi es. Pac Man, Space Invaders, Street Fighter... Ormai famosissimi per la diffusione nelle Rom per M.A.M.E. (Multi-emulatore di macchine arcade).

EDITARE UNO SPRITE

Di default il S.e.u.c.k. C64 ha uno sprite già disegnato. Se volete, e se siete grafici esperti, potete modificare questo oppure crearne un altro Cancellando lo Sprite con il Tasto **E** Erase Sprite e rispondete **Y** alla domanda "ARE YOU SURE ?"; quindi premete il tato **Y** se volete cancellare tutto lo sprite completamente e **N** per rinunciare e conservare lo Sprite.



Per la prima puntata e' tutto, appuntamento al prossimo numero.

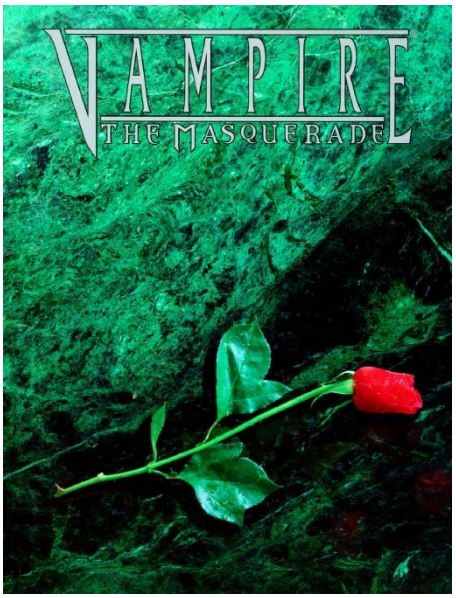
di Pinov Vox

VAMPIRE THE MASQUERADE (White Wolf, 1991, Mark Rein Hagen)

- I Vampiri non brillano al sole, tramano nell'ombra. -

di Starfox Mulder

Quanto non se ne può più dei Vampiri? Creature orrorifiche che hanno sempre affascinato l'immaginario comune con il loro aspetto curato ed al contempo mostruoso. Dandy d'altri tempi, malvagie sanguisughe pronte a togliere una vita con la stessa facilità con cui noi ci divoriamo una brioche e nonostante tutto...mi hanno saturato gli zebedei.
Facile dare la colpa alla serie di libri per casalinghe annoiate denominate Twilight o The Vampire Diaries, così come opere subito seguite come True Blood, in cui i celebri non morti venivano mostrati come dei fighetti bellissimi calati in parti degne di Brendon e Brenda di Beverly Hills 90210, ma se questo poteva valere per la noia popolare, con me le cose cominciarono prima, esattamente con questo gioco di ruolo.



"Fermi tutti: stai per recensire un gioco che ti ha portato a disprezzare il tema di cui tratta?"

Messa così pare una sconfitta annunciata, se nonché un inutile perdita di tempo, ma di fatto non ho altro che lodi per il lavoro che fece la White Wolf sulla sua serie di GDR, tutt'altro discorso sono i fans della stessa. Andiamo per gradi.

E' un mondo di tenebra.

Mentre gli umani se ne dormono tranquilli nei loro letti, al calare del sole ogni sorta di creatura mostruosa inizia la sua routine quotidiana. I vampiri esistono da sempre, da quando Caino uccise Abele, ed infatti proprio del fratricida rimasto sono gli eredi. Nel 1480 la stretta dell'inquisizione cattolica si fece così minacciosa da costringere i Vampiri ad organizzarsi in una nuova istituzione chiamata Camarilla e con essa ad istituire la Masquerade. Questa parola d'origine spagnola servì ad identificare una serie di norme da seguire strettamente all'interno dell'organizzazione per non svelare mai agli umani l'esistenza dei Vampiri, rendendoli (attraverso i secoli) un semplice mito.

Nel mondo di tenebre però non tutti i tredici clan in cui sono suddivisi i vampiri accettarono di prendere parte alla Camarilla, qualcuno restò indipendente e qualcun altro infine le si oppose in modo diretto, fondando quello che prese il nome di Sabbat.

Questa diffusa divisione in organizzazioni, clan e intenti è il vero motivo di traino del gioco di ruolo. Come qualcuno potrebbe erroneamente immaginare, non interpreteremo dei cacciatori di vampiri che si muovono all'interno del mondo di tenebre infatti, ma bensì dei veri e propri Non Morti con una passione particolare per il sangue.

Essere un vampiro è un lavoraccio. Ogni giorno dovremo procurarci del cibo senza lasciare cadaveri sospetti. Dovremo guardarci da cacciatori di vampiri, licantropi, avversari politici all'interno dell'organizzazione ed ogni sorta di altra minaccia.

Le nostre avventure saranno di genere spesso urbano e con connotazioni politiche ma attenzione: nulla di troppo verboso o noioso. A seconda del personaggio che sceglieremo di interpretare potremmo trovarci a fare gli arrampicatori sociali così come a finir coinvolte in guerre intestine decise da altri. La società interna ai vampiri è corrotta e sporca (quasi) quanto la nostra politica ai vertici più alti ed il semplice fatto di essere un cainita ci rende come minimo delle pedine da muovere nella grande scacchiera della guerra diplomatica.

I personaggi interpretabili differenzieranno molto tra loro in base a come i giocatori sceglieranno di distribuire i propri punteggi nelle caratteristiche e nelle abilità ma il tratto di maggior differenziazione, quello che per altro caratterizza a mio avviso più d'ogni altro elemento questo gdr, sono le discipline.

Ogni clan ha le sue ed ognuna di esse influisce in maniera marcatissima nel modo in cui i vampiri possono essere minacce per gli altri loro simili. Esistono discipline come "Potenza" o "Velocità" che aumentano la forza dei colpi portati in corpo a corpo così come il numero di attacchi eseguibili, nulla di troppo originale ammettiamolo, ma quando si va a scavare più nello specifico troviamo

"Dominazione", l'arte di piegare la volontà di chi ci circonda manipolandolo come un burattino, o "Vicissitudine" con cui plasmare ogni parte del proprio corpo. C'è "Oscurazione" con cui certi cainiti riescono a nascondersi nelle ombre stesse per risultare sempre invisibili o "Animalità" con cui parlare agli animali e renderli nostri alleati, e tante altre ancora. Le discipline sono tantissime ed aprono a prospettive mai immaginate in precedenti gdr.

Bellissimo essere un vampiro quindi, nonostante le minacce esterne? Non proprio.

A parte l'ovvia necessità di nutrirsi di sangue quotidianamente (il solo svegliarsi fa consumare al vampiro un punto sangue ed altri può spenderli utilizzando alcune discipline) il vampiro è soggetto alla Frenesia, una sorta di furia berserk in cui cade quando umiliato, spaventato o sotto forte pressione psicologica. I casi di Frenesia sono più comuni nei membri del clan Brujah, affetto da una facile irritabilità, mentre devastanti sui membri del clan Gangrel, che in seguito ad una crisi di Frenesia acquisiscono sul proprio corpo una mutazione animale. In termini pratici, un giocatore che vedrà il suo personaggio cadere in Frenesia si ritroverà in balia del Master per un tempo variabile entro il quale potrebbe facilmente morire in seguito alle sue azioni incontrollabili.

Una vita di sotterfugi quindi, di complotti e azioni ponderate. Essere una creatura potentissima non preserva dal morire bruciati, impalati o massacrati dai propri simili.

Quel che fa la forza del gioco è de facto l'ambientazione, molto curata e approfondita. Nel manuale base ci sono riferimenti alla storia dei vampiri, alla loro organizzazione e una breve descrizione di ogni clan. Questi ultimi hanno poi avuto ognuno dei manuali dedicati in cui viene approfondito dettagliatamente ogni singolo aspetto storico e caratteriale. Il punto debole? A mio avviso il sistema di gestione delle azioni in gioco.

A differenza di altri gdr, Vampire utilizza solo dadi a 10 facce ma se ne tirano tantissimi. Ogni tiro di dadi è composto solitamente dalla somma di alcuni punteggi separati, come spesso potrebbe essere una caratteristica ed un abilità. Contato quanti punti si possiede per ognuno degli elementi coinvolti si passa a tirare un numero di dadi equivalente (se ad esempio volessimo colpire qualcuno con un colpo di spada dovremo tirare un numero di dadi pari al nostro punteggio di Forza sommato a quello di Armi da Mischia) e contare i successi. Quanto deve fare il dado per essere considerato un successo? Dipende dai casi (di base 6) ma a prescindere ogni risultato di 1 sul dado annulla un successo ottenuto su altri.

Già avete capito: la frustrazione la farà da padrona. Avere personaggi molto abili nel fare una cosa significa tirare più dadi e più dadi significano più fallimenti. Ad onor del vero va detto che i 10 valgono come successi doppi ma vi garantisco che nel momento in cui la difficoltà del tiro passerà dal mediano 6 al già più difficile 7, essere più competenti significherà statisticamente riuscire meno spesso.

Non ha alcun senso ed infatti molte regole homebrew sono state applicate da tanti master sapienti per compensare a questa scelta ridicola e nelle successive edizioni del medesimo gioco è stata del tutto rimossa ma questo dice il manuale ed a volerlo giudicare per come si presenta questo dovremmo giocare.

Un regolamento con delle falle non preclude però un gioco divertente ed emozionante, tant'è che Vampire ha generato una community di appassionati seconda solo ai fans di D&D. Giochi di ruolo dal vivo (nel bel paese gestiti soprattutto da Camarilla Italia), videogames, giochi da tavolo e tanto altro.

Subito dopo la pubblicazione del primo manuale ambientato nel World of Darkness la White Wolf espanse l'offerta presentando manuali per giocare come Licantropi, Maghi e altre creature fantastiche appartenenti alla medesima ambientazione, crescendo di anno in anno per notorietà e seguito, ma proprio questo seguito mi portò ad allontanarmi da questo ottimo gdr.

La premessa era che i vampiri mi hanno stancato ed è vero perché non ne posso più di vedere nerd brufolosi atteggiarsi ad oscuri signori della notte con tre dita di eyeliner e un mantello sulle spalle. L'ambiente ruolistico attira persone di ogni sorta, da quelle più lucide (la maggior parte) che sanno sempre distinguere il gioco dalla realtà a quelle più esaltate e sconnesse. Il sottogruppo dei fans di Vampire the masquerade ha purtroppo la tendenza ad accorpare un numero più alto di altre realtà tra i fanatici del dark a tutti i costi. Sia chiaro: amo l'ambiente dark e soprattutto la musica che lo caratterizza (Joy Division e Bauhaus sono tra le mie band preferite di sempre), ma ho sempre pensato che i giochi debbano rimanere tali e non un pretesto per chiudersi in un mondo immaginario.

La stragrande maggioranza dei giocatori di ruolo, compresi gli appassionati del World of Darkness, sono persone sveglie ed ricche di autoironia che sanno divertirsi con un ottimo prodotto capace di calarci in un atmosfera horror in cui la minaccia siamo (anche) noi senza per questo scordarsi che la vita vera è tutt'altra cosa. Quei pochi che invece non hanno mai accettato la differenza tra realtà e fantasia li ritengo più dannosi di cento Edward Cullen coi brillantini addosso.

RetroGiochiAmo: Montezuma's Revenge

di Daniele Brahimi

La nostra rubrica **RetroGiochiAmo** continua imperterrita grazie all'impegno del nostro collaboratore/lettore **Daniele Brahimi**.

In questo numero forse potrei sembrare ripetitivo, oppure potrei essere preso per un fanatico dei giochi di avventura ma non ho saputo resistere dal ripescare un altro titolo adventure del Commodore 64 che sicuramente meritava e merita tuttora tutte le nostre attenzioni: Montezuma's'revenge!

Avevo cominciato a scrivere per RM dopo aver completato a distanza di decenni mission impossible, condividendo con voi la mia esperienza durante le lunghissime ore di gioco e dandovi i migliori consigli per poterlo terminare, adesso invece ho deciso non solo di dare consigli sullo svolgimento del gioco sperando che possano portarvi all'epilogo confidando nella vostra bravura e passione che da sempre ci mettete, bensì anche il compito di parlare di videogiochi all'epoca sottovalutati e messi presto nel nostro angolo personale dimenticato di casa, se non riportato al negoziante con l'intramontabile scusa " non carica" sperando che la beva.

Ma adesso parliamo della vendetta di Montezuma, molti di voi sapranno di cosa si tratta oltre al titolo di questo videogioco, altrimenti informatevi presto! Bene, siamo di nuovo in compagnia del nostro avventuriero dal cappello che visita un tempio azteco pieno di pericoli e insidie alla ricerca del tanto agognato tesoro finale.

Lo svolgimento del gioco non si presenta lineare come in Rick Dangerous, recensito dal sottoscritto nel numero scorso, questa volta ci troviamo in un vero e proprio labirinto in cui dovremo trovare oggetti indispensabili per proseguire nel gioco: chiavi colorate che apriranno le relative porte colorate, spade monouso utili nel caso si dovesse venire a contatto con i nemici, torce per illuminare alcune stanze totalmente al buio, ecc...

Personalmente ho trovato il gioco veramente realistico per il fatto che si possono perdere le vite cadendo da una certa altezza, oppure che possiamo trovarci al buio quando la torcia si esaurisce... Il piccolo attacco di panico quando si capita in una delle stanze buie dove non si sa dove mettere i piedi, e' un po' come quando da piccoli dormivamo con le luci spente e quando il legno dei mobili si dilatava correvamo in ogni dove fino a quando si accendeva la luce del comodino.

Ad un certo punto del gioco si potrebbe notare che le stanze si ripetono, l'uscita sembra impossibile ma come ogni labirinto che si rispetti, un uscita c'è sempre, basta solo avere un minimo di memoria e procurarsi gli oggetti adatti che ci faranno andare avanti. Siccome è stato uno dei primi giochi del Nostro Commodore 64, una volta terminato non aveva un finale vero e proprio ma ricominciava tutto dall'inizio ad un livello superiore.

Una volta capite tutte le meccaniche, oltre che semplice, sarà un attimo portarlo a termine e vi assicuro che vale tutto il divertimento se non lo avete mai giocato. Gli effetti sonori li ho trovati abbastanza simpatici e mi ricordano quelli degli scacciapensieri; la musica è totalmente assente durante il gioco.

Sono sicuro che prima o dopo averlo giocato non vedrete l'ora di informarvi sul leggendario popolo azteco e sull'imperatore Montezuma, almeno così io ho fatto e forse è per questo motivo che nutro parecchia simpatia per il suddetto titolo.

Ringraziando tutto lo staff per avermi dato ancora una volta l'opportunità di partecipare alla rivista, auguro a tutti voi buon divertimento con il gioco, buono studio sulla cultura azteca e soprattutto, che la vendetta di Montezuma non si abbatta su di voi!

Il piacere e' sempre nostro Daniele. Siamo veramente felici che continui a farci scoprire i giochi che hanno caratterizzato la tua infanzia, e ci auguriamo che altri possano presto seguire il tuo esempio. ☺

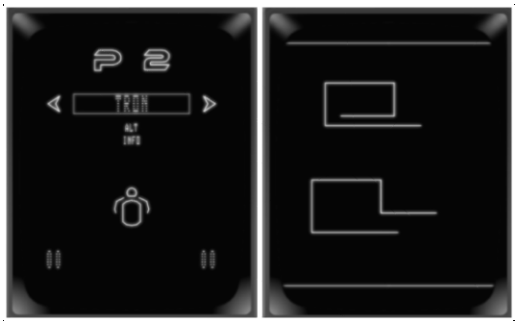
PLAYER 2

Player 2

Jubbernaut – Anno 2018 – Piattaforma: Vectrex

Packaging in stile Vectrex

La confezione in perfetto stile anni '80 comprende ogni sorta di sorpresa, libretto di istruzioni in 5 lingue, adesivi, un bel poster e persino la mascherina colorata trasparente da apporre sullo schermo della console.



Tanti giochi per due sfidanti

Con ben 10 titoli "classici" pronti all'uso non c'è davvero il rischio di annoiarsi. Il menu di scelta è quanto di più comodo si possa desiderare. A completare la *cartridge* molti extra, un *bonus game*, uno *screensaver*, vari *easter egg* da scoprire e persino un hardware test.

GIUDIZIO SUL GIOCO

GIOCABILITÀ

90%

La giocabilità media dei vari giochi è il punto di forza principale di Player 2. Il gioco a due di per sé aumenta il divertimento e per ciascun titolo la tecnica di gioco è stata curata fin nei minimi dettagli. Un'esperienza ludica piena d'atmosfera e in un certo senso "magica", un po' come la console su cui gira la cartuccia.

LONGEVITÀ

95%

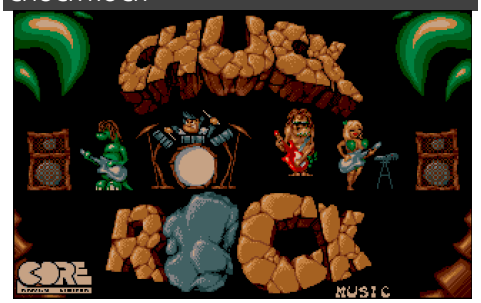
La quantità e la varietà di giochi disponibili disseminati nello spazio ROM della cartuccia garantiscono a Player 2 un posto speciale nella biblioteca software di qualsiasi utente Vectrex che si rispetti. Oltre a battere il vostro avversario di turno, ad ogni partita dovrete cercare di farlo con lo *score* più alto di sempre. Come dire: una sfida nella sfida che vi terrà incollati allo schermo!

È stato circa un anno e mezzo fa quando Robin Jubber mi ha parlato per la prima volta del suo progetto di mettersi a programmare qualche piccolo gioco in assembly 6809 per Vectrex, che per inciso è senza ombra di dubbio la console più "enigmatica" della storia dei videogames per come è stata progettata e per il fascino che, ancor oggi, riesce a trasmettere a qualsiasi appassionato di *retrogaming*. Sul momento, lo confesso, ho pensato che il suo piano fosse un po' folle o decisamente fuori dalle righe. Ma Robin è inglese quindi niente di cui stupirsi! In più, la virtù dell'eccentricità e della lucida follia è indubbiamente uno dei tratti caratteristici dei programmatori, soprattutto di quelli creativi che si dedicano al design e alla creazione di videogiochi, perciò in fondo nulla di nuovo sotto il sole dell'*hard-coding*, la solenne e meritevole pratica (ormai praticamente in via di estinzione) nella quale è d'uso spremere un sistema fino all'ultimo ciclo-macchina. Robin è infatti un programmatore professionista di videogiochi nella vita reale, mentre nel suo tempo libero fa tutt'altro: ahem... cioè, non proprio, perché quando non è al lavoro, dove solitamente resta seppellito sotto una tonnellata di codice C/C++ o sostanzialmente disperso nella piattaforma di sviluppo Unity, beh, Robin... scrive altro codice per i suoi amati computer a 8/16 bit, BBC Micro, Archimedes ed ora Vectrex!. E anche se in fondo lo conosco poco, fin dai suoi primi messaggi e-mail mi sono reso conto che egli appartiene alla Sacra Tavola Rotonda dei Cavalieri Programmatori Senza Macchia e Senza Paura che davanti a nuovi ostacoli e a sfide impossibili si lanciano in avanti temerari in barba a qualsiasi regola di buon senso. E non mi sbagliavo. Circa sei mesi dopo dal suo primo messaggio che cominciava con "Ehi, sai che sto dando un'occhiata all'assembly 6809 e a come scrivere un gioco per Vectrex?", infatti, mi ha annunciato di essere prossimo al completamento di un nuovo gioco per Vectrex. Anzi, per la verità, mi ha confessato che si trattava di 9 giochi diversi e che era persino riuscito a farli stare tutti dentro un'unica cartuccia! Ma delle sue peripezie alla scoperta di questa misteriosa console potete leggere in questo stesso numero (vedi il suo articolo "Non comprate mai un Vectrex!"), mentre io vi parlerò del suo "multi-gioco", Player 2, uscito lo scorso ottobre per Jubbernaut, la soft-house che Robin stesso ha messo in piedi per distribuire il prodotto delle sue fatiche in modo efficiente e alla vecchia maniera, packaging, libretto di istruzioni e accessori inclusi. In effetti, se lo chiedete a lui, vi dirà che la parte davvero complicata della produzione del gioco è stata proprio quella relativa alla confezione o al reperimento di plastiche e mascherine colorate trasparenti da incollare allo schermo, non certo la fase di *coding*! E partiva essenzialmente da zero con l'assembly del 6809 e con l'OS della console! Che vi avevo detto? Un folle! ☺ Ma la passione per la programmazione e per la progettazione di giochi, unite al desiderio nostalgico di scrivere codice e di giungere alla pubblicazione di giochi impacchettati ad arte, hanno fatto "il miracolo". Robin è davvero riuscito a produrre un videogioco straordinario per Vectrex e certamente unico nel suo genere. E allora

partiamo proprio dal nome del gioco, ovviamente non scelto a caso. La cartuccia contiene, come detto, **ben 10 giochi completi,** tutti ispirati ai classici per console/arcade dei primi anni 80. I titoli di Player 2 sono da giocare rigorosamente in 2, in una sfida interminabile per i (purtroppo) rari possessori di due gamepad. La lista è molto nutrita: si spazia da **Pong/Tennis** a **Tron**, da un clone di Asteroids (**Rotatazor**) ad uno di Pit Stop (**Racers**). Non mancano le battaglie di astronavi fra le stelle (**Space War**) o gli shooting game (**SpiderFish**, **Balloons** e **Artillery**). Nel più esteso **Incoming**, una vera novità che chiude la lista dei giochi a disposizione, i due giocatori sono chiamati a collaborare in un'avventura a enigmi attraverso 100 schermi e 7 livelli differenti pieni di insidie e pericoli, porte e trappole, alieni pronti a farci la guerra, magneti, pulsanti, ventilatori e numerosi altri ostacoli. Il nostro obbiettivo è quello di condurre una cella di energia critica fino al termine del labirinto di stanze e *location*. **Incoming** è l'unico gioco disponibile anche per giocatore singolo. Fino ad ora pochi titoli per Vectrex consentivano a 2 giocatori di sfidarsi sullo schermo. Con Player 2 questa lacuna è stata colmata. Alcuni giochi sono abbordabili, altri difficili e altri ancora impossibili. Almeno per me. Ma lo spettacolo è assicurato: nella sua apparente semplicità l'hardware del Vectrex viene sfruttato a dovere dal codice di Robin, che ha utilizzato, a mano a mano che le scopriva, tecniche avanzate per superare le molte limitazioni grafiche, elaborative, spazio disponibile sulla cartuccia e assurdi bug del sistema operativo di base. Per maggiori informazioni e per acquistare il gioco: http://www.jubbernaut.com

di David La Monaca/Cercamon

CHUCK ROCK



Schermata introduttiva

La schermata introduttiva con tanto di musichetta rock suonata live e' veramente un must!

GIUDIZIO SUL GIOCO

GIOCABILITA'

95%

La giocabilità è galattica, difficilmente vi staccherete da Chuck Rock prima di averlo terminato.

LONGEVITA'

80%

Colorato e divertente, lo rigiocherete spesso.

Chuck Rock

CORE - Anno 1991 - Piattaforma Amiga

Età della pietra, stiamo comodamente seduti sulla nostra poltrona di pietra, guardando il nostro programma preferito sul nostro televisore di pietra quando, ciuffo selvaggio Gary, decide di rapire la nostra cara mogliettina Ophelia per portarla alla sua caverna e farne la sua domestica. Ovviamente questo potrebbe essere anche una cosa positiva, una bocca in meno da sfamare, ma perdiana, ci sono un sacco di panni nella lavatrice di pietra da finire di lavare e da tendere e una donna in casa è indispensabile per finire il lavoro... E' deciso, andremo a recuperare le nostra mogliettina. (NdR: Speriamo non lo legga mia moglie... ☺).

La strada per arrivare alla caverna di Gary è lunga ed irta di pericoli, ed in più sembra che tutti i dinosauri del pianeta si siano alleati con il rapitore di fanciulle per renderci la vita difficile ed impedirci di recuperare la nostra utile e laboriosa mogliettina.

Tecnicamente il gioco è ben realizzato, la grafica è colorata, fumettosa e decisamente accattivante. Lo scrolling dello schermo risulta sempre fluido, anche quando ci sono molti sprite a video ed i 2 livelli di parallasse contribuiscono a rendere più verosimile il movimento all'interno del mondo virtuale.

Il gioco è essenzialmente un platform che si sviluppa su 5 livelli differenti, a sua volta suddivisi in altri 5 sottolivelli (soltanto l'ultimo livello è composto da solo 3 sottolivelli). Il primo livello è ambientato all'aperto, mentre il secondo livello si trova all'interno di una caverna, il terzo in un mondo semisommerso, mentre il quarto ci vede alle prese con un mondo ghiacciato ed innevato. Il quinto livello merita un discorso a parte, in quanto ambientato a sua volta nel cimitero dei dinosauri ed all'interno dello stomaco di uno di loro (sicuramente il livello meno curato graficamente, gli altri sono spettacolari!). I nemici ovviamente sono di volta in volta in sintonia con l'ambiente che ci circonda, quindi troveremo enormi dinosauri ed insetti nel primo livello, pesci e balene nel mondo semisommerso, dinosauri ghiacciati ed infreddoliti (anche con la tosse!) nel livello ghiacciato, sino ad arrivare agli scheletri di dinosauro nel cimitero ed ai villi intestinali ed i globuli rossi all'interno dello stomaco... Semplicemente esilarante!!!

I tocchi di classe e di humour si sprecano e devo ammettere che è veramente un piacere giocare a Chuck Rock, non fosse altro per scoprire quali nuove trovate hanno escogitato i programmatori per divertirci... La difficoltà è ben calibrata e si passa dai primi semplici schermi agli schermi finali, decisamente più difficili, attraverso un percorso ben costruito. Decisamente troppo semplici risultano i mostri di fine livello, una volta scoperto il trucchetto per batterli, risulta veramente un gioco da ragazzi sbarazzarsi di loro senza nemmeno farsi colpire.

Il menù iniziale è simpaticissimo, ma permette soltanto di scegliere se ascoltare la musica o gli effetti sonori durante il gioco; anche se, i codici per barare devono essere digitati proprio qui. La musica roccheggiante che si sente come sottofondo del menù è semplicemente stupenda.

Il sonoro è ben realizzato, anche se la musica durante il gioco, alla lunga, può diventare monotona e ripetitiva. Meglio scegliere gli effetti sonori, sono meno invadenti (questo difetto comunque è comune a molti giochi, e la possibilità di scelta permette di non pregiudicare il valore di questo titolo).

La giocabilità è galattica, difficilmente vi staccherete da Chuck Rock prima di averlo terminato... Soltanto per vedere come va a finire e quanto è bello graficamente, vale la pena di passare qualche ora in compagnia di Chuck.

Lo consiglio caldamente a tutti, un must!

di Francesco Fiorentini

WONDERBOY

Wonderboy

Sega - Anno 1986 - Piattaforma Arcade

Il mitico primo livello

Il primo stage di gioco fa subito innamorare per i colori sgargianti e la grafica curatissima.

Nel tempio del Boss

Alla fine di ogni mondo ci aspetta un Boss che dobbiamo colpire sempre in testa.

A volte ritornano

Nel 2016 è stato rilasciato dalla Sega il remake del gioco originale e distribuito per PS4 e PC.

GIUDIZIO SUL GIOCO

GIOCABILITA'

95%

LONGEVITA'

95%

Nella semioscurità della sala giochi una musica avvolgente mi attira da un nuovo cabinato. Mi avvicino e il suo protagonista mi conquista subito con quella sua capigliatura dorata e il corpo paffutello. Lo vedo correre, saltare per mangiare frutta e dessert vari e rompere le uova. No ma dai, in una trova un ascia e in un'altra... noooo ma è davvero uno skateboard quello? Non ci credo...

Comincia più o meno così la mia conoscenza di quello che reputo uno dei migliori platform arcade mai comparsi nelle sale giochi degli anni ottanta...signori e signore ecco a voi lo stupefacente *Wonderboy*.

Siamo nel 1986 quando mamma Sega decide di contrastare lo strapotere di *Super Mario Bros.* della sua eterna rivale Nintendo portando nelle sale giochi il ragazzo magnifico creato dalla semisconosciuta Escape. E subito siamo tutti colpiti da questo magnifico titolo che sembra avere nel suo motore non uno, ma innumerevoli punti di forza. La storia di fondo è la stessa della maggior parte dei titoli del periodo, ovvero salvare la nostra amata attraversando lande ostili piene di trabocchetti e nemici vari, ma è il sistema di gioco di Wonderboy che conquista subito per le sue meccaniche e per la sua fluidità. Possiamo comandare il nostro eroe, che ha le sembianze di un divertente cavernicolo, usando due tasti: uno per sparare asce primitive e un altro per saltare sulle diverse piattaforme con la possibilità di combinarli per effettuare salti più lunghi. Deve come sempre muoversi da sinistra a destra saltando e correndo ma durante il suo percorso si possono trovare delle uova che una volta distrutte rilasciano degli elementi bonus. Il più importante è senza dubbio lo skateboard che ci permette di muoverci con più velocità e con una potenza di salto maggiore e in pratica ci offre anche una vita bonus contro gli attacchi dei nemici.

In alto a sinistra dello schermo abbiamo una health bar che scorre inesorabilmente verso lo zero ma che possiamo ricaricare di continuo prendendo frutta, verdura e dolci che appaiono improvvisamente sullo schermo. Il gioco è diviso in 10 aree e ogni area ha diversi mondi che vedono alternarsi sempre tre tipi di template: un bosco colorato, una spiaggia esotica che dobbiamo esplorare saltando da una nuvoletta all'altra e una caverna sotterranea piena di buche e massi rotolanti. Alla fine dell'ultimo mondo ci aspetta nella sua casa il solito boss che dobbiamo colpire sempre alla testa che alla fine cadrà per terra sostituita da quella del nuovo mostro che ci aspetterà sempre più agguerrito nella prossima casa.

Ad arricchire questo gameplay ci sono tanti bonus che possiamo raccogliere, come angeli che ci proteggono per un periodo limitato, o vite e score aggiuntivi che gli sviluppatori si sono divertiti a nascondere in vari punti dei livelli. La grafica è coloratissima e curata nei minimi dettagli ed il sonoro, pur essendo costituito da soli due temi musicali, ha una melodia piacevole che ci rimane in testa e che ancora oggi è riconoscibilissima. Ma il punto di forza di Wonderboy è senza dubbio la sua giocabilità poichè è davvero divertente muoverci nei vari scenari guidando il nostro cavernicolo.

Come accade per tutti i titoli ben riusciti Wonderboy ha avuto diverse conversioni per le console casalinghe tra le quali dobbiamo ricordare quelle per commodore 64 e per il sega master system, davvero ben fatte, e diversi cloni tra i quali non possiamo non citare la serie *Adventure Island* che ha avuto un grande successo sul super nintendo .

Wonderboy ha avuto due seguiti: *Wonderboy in Monster Land* che a dire il vero meriterebbe una recensione a parte poiché siamo davanti ad uno dei primi riuscitissimi incroci tra platform e rpg, e *Wonderboy III Monster Lair* molto più vicino al primo capitolo.

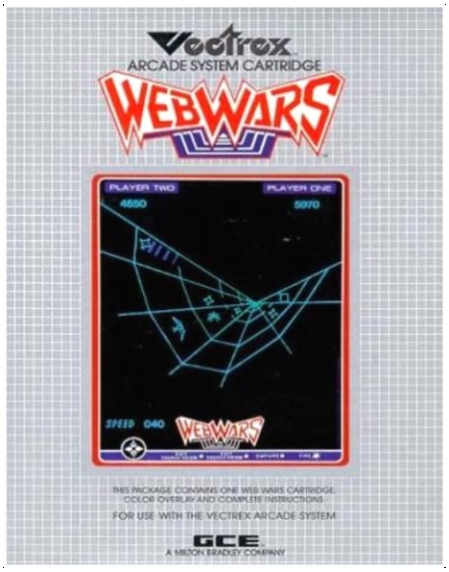
Ma la magia del ragazzo magnifico non è mai tramontata così nel 2016 la Sega ha pubblicato *Wonderboy Returns* che è un remake dell'originale con grafica maggiormente dettagliata... Ma il fascino del primo capitolo resta e resterà inimitabile.

Querino Ialongo

WEB WARS

Web Wars

GCE - Anno 1982 – Piattaforma: Milton Bradley Vectrex



La copertina "americana"

Della confezione furono realizzate più versioni, di cui le più note furono quelle per il mercato americano e quello europeo.

Uno sguardo attraverso l'overlay

Per ovviare in parte all'assenza di colori, i giochi del Vectrex venivano forniti di schermi trasparenti da applicare sul monitor.

GIOCABILITA'

95%

l'ottima manovrabilità del Re Falco vi lascia liberi di concentrarvi solo sulla meccanica e sulle dinamiche di questo ottimo titolo.

LONGEVITA'

95%

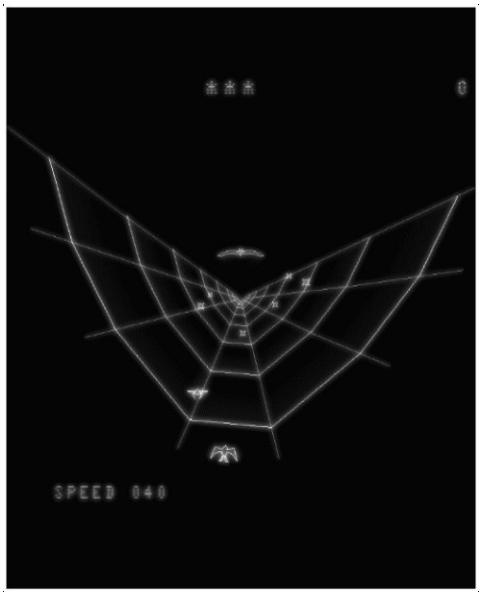
Venti livelli con una curva di difficolta ben bilanciata vi terranno incollati al monitor per un bel po'.

"Entra in una galassia piena di avventura e pericolo - una galassia diversa da tutte quelle che hai immaginato prima. Tu incarni il Re Falco, una creatura capace di incredibile velocità e potenza, nel tentativo di catturare le straordinarie creature che abitano il veloce Web of Fantasy. Le 20 creature sono protette da guardiani e dal temibile Dragone Cosmico, catturarli sarà progressivamente sempre più difficile. Nessuno ha mai catturato l'elusiva ventesima creatura. Sarai tu a farlo?"

Quando i giochi erano contenuti in una manciata di K, le meccaniche estremamente semplici e lo storytelling praticamente optional, poche righe come quelle su riportate erano sufficienti ad imbastire una storia (spesso senza né capo né coda) che facesse da introduzione ad uno sparatutto come quello di qui presentato. Creato per la mitica console Vectrex, Web Wars, noto anche come Web Warp, è un divertente e frenetico shoot'em up con un chiaro richiamo a Tempest della Atari.

L'azione si svolge in un unico campo di gioco, un corridoio tridimensionale approssimativamente a forma di "V" con le fattezze di una ragnatela la cui inquadratura cambia continuamente in maniera indipendente dai nostri spostamenti. Nella parte bassa dello schermo, nel punto più vicino all'osservatore, è posizionato il nostro veicolo, il Re Falco, che può muoversi su un'ampia zona, sia in senso laterale che in profondità, azione che aumenta anche la velocità con cui si percorre il corridoio ed il punteggio ottenuto per ogni avversario abbattuto. Dal fondo, nel senso della profondità, sciamano frotte di nemici in veste di stelloidi a quattro punte, che percorrono il corridoio fino a noi per poi tornare indietro, casomai fallissero nel loro intento di eliminarci al primo passaggio. Gli stelloidi possono essere distrutti grazie ai colpi del mitragliatore laser di cui il Re Falco è dotato e che, per nostra fortuna, richiede la sola pressione del tasto 4 del controller per essere azionato (i.e. si preme il tasto e spara a raffica). Oltre agli avversari, il campo da gioco viene attraversato anche da una creatura mitologica, diversa per ogni livello, che dovremo catturare per poter passare al round successivo. Una volta catturata la creatura grazie ad un bastone di recupero montato sul nostro veicolo, apparirà un portale che ci permetterà l'uscita dal livello e l'ingresso nella sala dei trofei, dove potremo ammirare tutte le creature catturate sino a quel momento. Fallire la cattura di una creatura o mancare il portale per due volte provoca la comparsa del Dragone Cosmico, un invincibile avversario che volerà verso di noi sparandoci contro delle insidiose palle infuocate.

Il gioco gira tutto intorno a questi pochi, semplici, concetti ma catturare tutte e venti le creature sarà un compito niente affatto banale. Superare i primi 4 o 5 livelli non sarà particolarmente difficile, anche se già dal terzo la pressione sarà sensibile, ma raggiungere il ventesimo sarà davvero dura. Con il progredire del gioco, gli avversari diventeranno più scaltri, veloci ed efficienti nella loro opera di spazzarci via dal web e la precisione del dragone aumenterà fino a farlo diventare un vero cecchino. Inoltre, se le prime creature saranno moderatamente lente e percorreranno il corridoio con traiettorie prevedibili, quelle dei livelli più avanzati



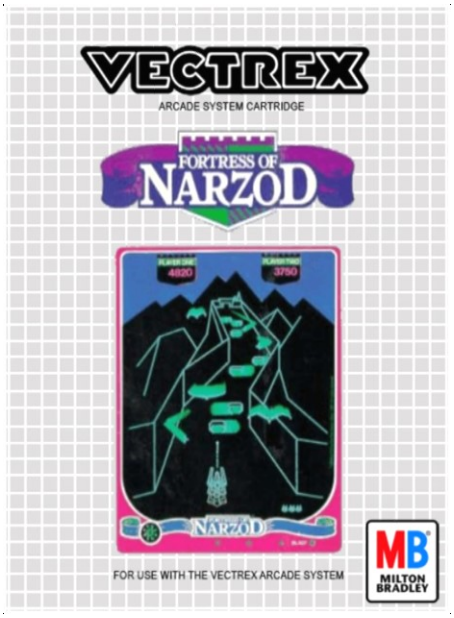
esibiranno comportamenti estremamente elusivi e vi costringeranno a faticare non poco per prenderle. Nonostante un gameplay molto semplice, Web Wars è un titolo incredibilmente giocabile, capace di regalarvi ore di divertimento sia per una partitina veloce come pausa dagli impegni quotidiani che per una impegnativa sessione di cattura di tutte le 20 creature.

di Giorgio Balestrieri

FORTRESS OF NARZOD

Fortress of Narzod

GCE - Anno 1982 – Piattaforma: Milton Bradley Vectrex



La copertina della confezione

Essenziale come tutte quelle della MB per il Vectrex, fanno comunque la loro figura una volta sistemate nella nostra ludoteca.

Il supporto del gioco

Come tutte le console dell'epoca, anche il Vectrex utilizzava cartucce per i giochi, da inserirsi sul fianco destro della macchina.

GIOCABILITA'

95%

L'hovercraft si manovra in maniera eccellente e la meccanica del gioco si racchiude tutta in "schiva e spara". Più giocabile di così...

LONGEVITA'

95%

Tre livelli, più il boss finale sembrano pochi, ma vi assicuro che superare anche solo il primo ciclo di livelli non sarà affatto facile. Il livello di sfida e la gratificazione che se ne trae sono tali da stimolare sempre la voglia rigiocarlo.

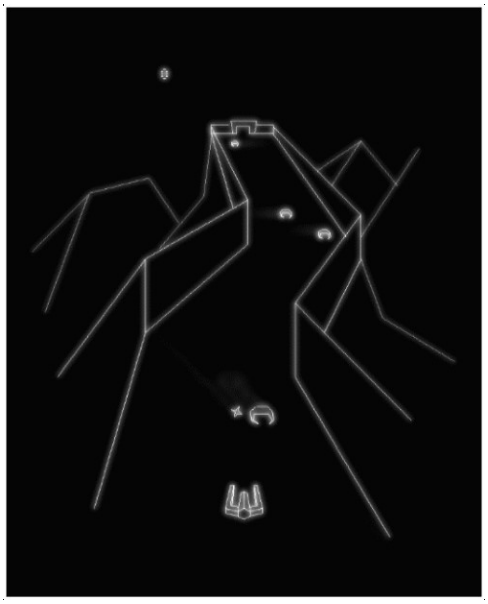
"Molto tempo fa, prima dell'inizio della nostra storia, maghi - sia buoni che malvagi - hanno combattuto per la supremazia della Terra. Era un tempo di magia e avventura quando tutti gli uomini possedevano poteri speciali. Il mago più malvagio, Narzod, acquisì il potere di schiavizzare l'intera umanità. Tu sei l'unico mago buono rimasto con la consapevolezza di dover impiegare tutte le tue forze per sconfiggere Narzod e distruggere la sua sinistra fortezza. Ma Narzod è preparato all'assalto e ha posto creature mortali, incluso il Mystic Hurler, a difesa del suo regno."

Con questa la breve storiella, il manuale di gioco introduce uno dei titoli più noti e divertenti prodotti per il Vectrex.

Fortress of Narzod è uno sparatutto a schermata fissa modellato su un ciclo ripetitivo di tre ondate più un boss finale; l'azione si svolge in un'unica schermata, rappresentante una porzione del sentiero roccioso che conduce alla fortezza, con una porta posta nella parte alta dello schermo da cui fuoriescono tonnellate di nemici messi a guardia del livello in cui ci troviamo. In basso ci siamo noi a bordo di un hovercraft (è noto, i maghi girano sempre in hovercraft) dotato di un'inesauribile riserva di proiettili che può sparare a raffica contro gli avversari. Per distruggere la fortezza, dovremo attraversare tre livelli, corrisponenti da altrettante sezioni del sentiero, uccidendo tutti gli accoliti di Narzod ed abbattere il boss guardiano del quarto livello, il Mystic Hurler, che richiederà ben sei centri prima di crollare. Raggiunto l'obiettivo, il gioco ricomincerà con un livello di difficoltà maggiore, fino all'esaurimento delle vite a disposizione.

Semplice, vero? Niente affatto, ovviamente. Gli avversari, oltre che numerosi, sono anche piuttosto scaltri e lo diventeranno sempre più con l'avanzare dei livelli, arrivando a nascondersi dietro le pareti rocciose per tenderci trappole o per ripararsi dai nostri proiettili. Dovremo batterci contro diversi tipi di nemici, ognuno con le sue peculiari caratteristiche e capaci di lanciarci contro proiettili di generose dimensioni in grado di rimbalzare sulle pareti fino al fondo dello schermo. Dal secondo livello, insieme alle truppe "appiedate", faranno la loro comparsa anche creature alate, i Warbirds, particolarmente insidiose e che lasceranno il loro cadavere sul percorso una volta abbattute, a fare da scudo ai nostri proiettili (ma per nostra fortuna, anche a quelli dei nemici). Come quelle degli avversari, anche le nostre pallottole sono in grado di rimbalzare sulle pareti, il che permette di implementare strategie di "sparo da copertura" ma per contro possono rivelarsi un problema quando nei loro rimbalzi tornano verso di noi, poiché sono capaci di distruggere il nostro mezzo tanto quanto i seguaci di Narzod. Ai livelli più alti, i proiettili dei nostri avversari tenderanno a dividersi in due, come quelli del Mystic Hurler, aumentando il rischio per la salute del nostro hovercraft...

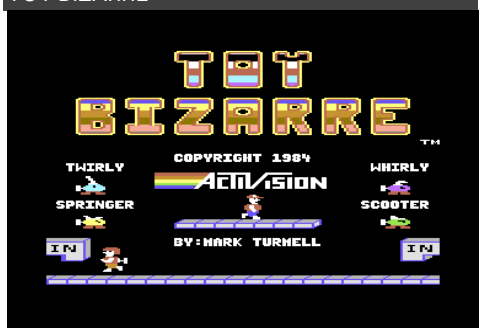
Tutto ciò rende l'azione decisamente frenetica e vi serviranno riflessi fulminei per riuscire a restare in vita e completare quanti più round di gioco possibile.



Fortress of Nazrod è decisamente un gioco impegnativo e divertente, assolutamente da avere in ogni collezione che si rispetti. La sua notorietà è tale che i membri di una band australiana ne hanno scelto il titolo come nome del gruppo e nel 2009 ne è stata effettuata una conversione per Commodore 64 di eccellente livello, seppur con meno atmosfera rispetto al titolo originale a causa dell'enorme diversità dell'hardware tra i sistemi. Se possedete un Vectrex, non potete non avere una copia di questo gioco, ergo aprite il vostro retrosito preferito e cercatene una , possibilmente ad un prezzo onesto.

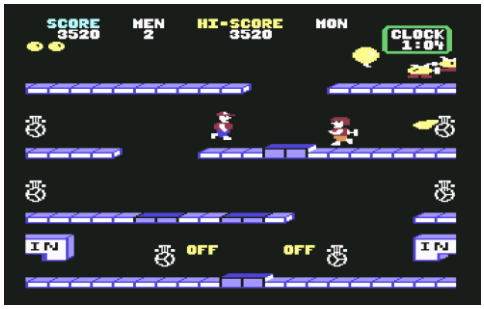
di Giorgio Balestrieri

TOY BIZARRE



Schermata introduttiva

Ecco la schermata introduttiva, dove vengono presentati i vari personaggi, insieme ai "tremendi" giocattoli!



Un gioco movimentato...

Durante il gioco il nostro personaggio sarà sempre in movimento, e per terminare il gioco occorreranno riflessi pronti ed un controllo del joystick non da poco!

GIUDIZIO SUL GIOCO

GIOCABILITA'

95%

Veloce, coloratissimo, divertente! E' un gioco da giocare, sia per chi non lo conosce sia per chi lo ha già giocato in passato.

LONGEVITA'

65%

La mancanza di un vero e proprio finale è probabilmente il tallone di Achille di questo gran bel gioco. Per chi continuerà a giocarlo dopo aver superato il livello dell'ultima ora del Venerdì notte, si tratterà di una sfida ad ottenere un punteggio maggiore.

Toy Bizarre

Activision - Anno 1984 - Piattaforma Commodore 64

Toy Bizarre è un "platform" 2D pubblicato nel 1984 per Commodore 64 e nell'anno successivo per ZX Spectrum dalla Activision. Il codice del gioco originale su Commodore 64 è stato scritto da Mark Turmell, mentre la musica da Russell Lieblich. Conobbi questo gioco grazie ad un amico. Lui ne era entusiasta ed io iniziai a giocarlo sul mio Commodore 64 e pian piano, lo diventai ancor di più!

In *Toy Bizarre* il giocatore controlla Merton, il guardiano notturno di una fabbrica di giocattoli, che si trova a fronteggiare la ribellione dei giocattoli, che improvvisamente hanno preso vita. I livelli sono a schermata fissa e composti da piattaforme, con delle coppie di blocchi che si sollevano alternativamente saltando sull'uno o sull'altro, e sei valvole. I controlli sono solo correre e saltare, comprese deviazioni al volo durante il salto.

Merton deve scoppiare tutti i palloncini generati dalle valvole toccandoli, prima che scoppino da soli e liberino un nuovo giocattolo. I giocattoli, che normalmente sono letali al contatto, si possono "stordire" se si riesce ad alzare il blocco sul quale si trovano poggiati, e quindi eliminare andandogli addosso. L'unico giocattolo che non è possibile sconfiggere in questo modo è la bambola Hilda, di cui ci si può liberare soltanto sfruttando il meccanismo dei "blocchi" che si alzano e si abbassano alternativamente, che possono farla schizzare via, ma solo temporaneamente. Merton può anche chiudere le valvole camminando in prossimità delle stesse, ma Hilda le riapre.

Ogni volta che un livello viene completamente ripulito da ogni giocattolo e dal numero di palloncini richiesto, il giocatore passa a un livello successivo, che deve essere sempre risolto entro un tempo limite. I livelli sono 8 raggruppati per giorno di lavoro, uno per ciascuna ora lavorativa.

Ci saranno quindi 8 livelli per il Lunedì notte, 8 per il Martedì notte, 8 per il Mercoledì notte etc, e la difficoltà media crescerà via via fino ad arrivare al Venerdì notte. Ogni due livelli di gioco standard ce ne sarà uno di bonus denominato "safety check" ("controllo di sicurezza"), dove lo scopo sarà quello di riuscire a chiudere tutte le valvole insieme entro il tempo massimo stabilito, mentre Hilda cerca di riaprirle.

E' molto utile anche sfruttare il "coffe break", se si riesce ad ottenerlo, perchè consente di eliminare tutti i giocattoli andando loro incontro, ma solo per breve tempo. Può risultare davvero determinante per terminare un livello difficile.

Si parte con 4 vite iniziali. E' possibile guadagnarne altre raccogliendo le "teste" durante il gioco, p rima che scompaiano. Inoltre, quando il punteggio raggiunto supera i 10.000 punti, si otterrà sempre una vita in più, e così sarà anche superando i 20.000 punti, i 30.000 punti e così via. Qualcuno noterà, nell'uso dei colori, nel disegno e nell'abbigliamento di "Merton", ma anche nel gameplay, qualche analogia con il ben più noto e blasonato "Mario Bros". Probabile che all'epoca, in Activision, ci fosse qualche appassionato del famoso idraulico! "Toy Bizarre" è comunque, di suo, un ottimo gioco.

E' un gioco solo all'apparenza semplice, che riesce ad essere molto coinvolgente e mai monotono.
Per sfruttare proficuamente il sistema dei "blocchi", sempre disposti in maniera diversa di livello in livello, è necessaria una certa abilità che si matura soprattutto giocando. Gioco scorrevole, coloratissimo, ipnotico a volte, è dotato di un meccanismo di rilevazione delle collisioni degli sprites dannatamente accurato, tanto efficiente quanto frustrante, a volte.

Il gioco richiede pertanto buoni riflessi ed una perfetta padronanza dell'uso del joystick. Non ricordo "glitches" e/o rallentamenti particolari dovuti a bugs di programmazione. Se proprio dovessi trovare un difetto in questo gioco, l'unico che menzionerei è la mancanza, purtroppo, di un vero e proprio finale. Raggiungendo e superando l'ottavo livello della notte di Venerdì infatti verrà presentato nuovamente il livello relativo alla prima ora di Venerdì notte.Che dite? Sono riuscito ad incuriosire anche voi e volete provarlo? Mi auguro di si! Non ve ne pentirete ☺

Ciao a tutti!

di Marco Pistorio

Classico Retrò made "by" Japan?

Classica foto "da" Arakurayama?

Classico Cosplay made "in" Japan?

... Credo sia meglio precisare!

Giappone: antica modernità o futuristico presente?

Prima Puntata

Giappone, una parola che desta in noi l'idea di un regno mistico, dove nasce tutto ciò che è elettronico.

Viaggio in questa terra ormai da tanti anni, è nuovamente ora di rinnovare il passaporto!

Vorrei raccontare qualcosa di questo meraviglioso arcipelago, poiché purtroppo i racconti che giungono all'Occidente, subiscono spesso edulcorazione e contraffazione. Questi racconti, nella maggior parte dei casi, nascono dall'esperienza di un unico viaggio.

Tale esperienza è spesso così tanto coinvolgente che il narratore, lasciandosi trasportare dal vento delle intense emozioni vissute, espone in maniera incompleta o inesatta, sia i dettagli dei fatti osservati, che le dinamiche che li hanno generati.

In questo racconto, mi impegnerò ad eludere tali dosi mistificanti e come un lavaggio a freddo, mi impegnerò a raccontare obiettivamente il backstage di questa emozionante terra, così diversa dalla nostra, dove nascono quotidianamente migliaia di prodigi elettronici.

Siamo da poco entrati nel 2019, la passione per il Retrogaming è giunta ad uno stadio di maturazione tale che, finalmente, sono venute alla luce numerose mini console per soddisfare (e spesso scandalizzare) la nostalgia insita in noi, popolazione di trentenni, quarantenni, cinquantenni, che abbiamo vissuto la magnifica esplosione degli 8 bit, nel momento della massima deflagrazione.

L'effetto nostalgia è stato soddisfatto appieno dopo le ultime mini console? La Playstation mini ha definitivamente spezzato il ciclo di questo fabbisogno? Come ne siamo usciti da queste reminiscenze? Il retrogusto è amaro o dolce?

E' giunto il momento delle riflessioni, a mente lucida e mano ferma, di questa fase relativamente breve dei vari remake.

Stiamo parlando del Nintendo Mini Nes, mini Famicom, mini Snes, mini Super Famicom, Super Famicom Mini edizione Dragon Ball, Atari Flashback, Intellivision, Colecovision, Mega Drive, Neo Geo, Game & Watch, Playstation, TheC64... e la lista potrebbe proseguire a lungo.

Anche le mie vetrine non hanno più posto, eppure, ogniqualvolta collego queste mini console al mio televisore, penso sempre che i pollici dello schermo siano troppi per giocare ad un semplice Pac Man. Sembra che anche i pixel siano diventati giganteschi ed antiestetici. Una vocina dentro di me sussurra che sono stati ".. soldi sprecati .." e purtroppo rimane un senso di vuoto in me.

Una visione pessimistica o una semplice constatazione?

Chi ha tratto profitto da questa breve moda lanciata (anche) dal Giappone, soprattutto grazie al famoso Nintendo Mini Nes? I vari produttori delle mini console hanno tratto il massimo profitto? In realtà il fenomeno è ben più articolato.

Dietro questo nostalgico vuoto da colmare e soprattutto dietro questo retrogusto amaro c'è un meccanismo, perlopiù nipponico, dotato di solidi ingranaggi, oliati da decenni di esperienza nel settore videolutico, inerente al negativo ramo sociologico delle "dipendenze dell'umanità".

La storia, nel proprio ciclo di eventi, è reiterante, come una ruota senza fine, poiché racconta sempre i soliti pregi, difetti, virtù, della solita razza umana. Il Giappone non è il paese dei balocchi.

Si, certamente, siamo tutti impressionati da questo popolo capace di fondere tradizioni

secolari e sviluppo della tecnologia, il rispetto per gli anziani e la propensione verso il futuro, però dietro le quinte, il loro sistema è talmente evoluto che nulla viene lasciato al caso , oppure alla mera furbizia, al sesto senso o all'intuizione: è tutto programmato, è tutto codice, come in un film di Matrix.

Questa visione può meravigliarci, ma per loro è tutto assodato, digerito: veste il quotidiano comun denominatore della loro esistenza.

Il loro sistema organizzativo si plasma in una fitta rete di regole interdisciplinari appositamente attuate ad espellere qualsivoglia forma grottesca di approccio furbesco. Fidatevi, in Giappone non esistono scorciatoie e non si salta la fila. Non sto bluffando. L'unico approccio realmente valido è la conoscenza di alcune regole, settoriali al campo che vogliamo indagare, ahimè, senza poterci agevolmente entrare: semmai ci sarà permesso costeggiare.

In tutti questi anni mi sono realmente accorto che una regola molto valida per poter interagire con il loro mondo è la legge di John Nash: "il guadagno di ciascuno, in generale, dipende dalla propria strategia e dalle strategie scelte dagli avversari".

Allora analizziamo matematicamente i vari passaggi per entrare in un piccolo angolo del cuore giapponese, più precisamente osserviamo il distretto dell'elettronica giapponese, raggiungendo fisicamente tale distretto in Tokyo.

Scopriremo rapidamente che le dinamiche dominanti di John Nash (Premio Nobel per l'economia, 1994) risiedono proprio in queste vie, spesso prive di indicazioni stradali comprensibili.

Tokyo è una di quelle megalopoli classiche dove la vita notturna supera di gran lunga quella diurna, sia per la quantità di persone che affollano i vari distretti, sia per la quantità di servizi di qualsiasi tipo, sia per l'esagerata luminosità ambientale irradiata dalle insegne pubblicitarie lungo i grattacieli che metterà in crisi le varie correzioni riguardo la corretta esposizione fotografica.

Tokyo(東京): Shinjuku (新宿区)

Tra i quartieri di Tokyo più adatti al mondo dell'elettronica si può elencare: Roppongi, Shibuya, Shinjuku e soprattutto Akihabara chiamata anche "Electric Town".

Akihabara (秋葉原)

Se vorrete nei prossimi numeri vi parlerò di come raggiungere tali distretti, come spostarsi con precisione e quali grattacieli visitare per trovare ciò che desiderate: dalle più introvabili valvole di amplificatori, fino alle collezioni più impensabili di statuette, Figure, Anime, merce Hentai e così via. Per ora , in questo numero, parlerò delle domande tipiche riguardanti la curiosità di noi italiani.

Partiamo dalla prima domanda. Mi è stata posta frequentemente , mi sembra molto attinente per il mondo del retrogaming, credo che sia maturata autonomamente nella fase del tramonto delle mini console e soprattutto, la reputo sincronizzata al numero 13 di Retromagazine.

La domanda è: "il prezzo di tutte le mini console di questi ultimi anni, è stato sempre coerente?".

Questa risposta è semplice. In Giappone qualsivoglia servizio viene venduto al minimo prezzo giudicato possibile dalla ditta.

Il prezzo viene sempre programmato e studiato a puntino per offrire il miglior prodotto possibile al prezzo più accattivante possibile. Faccio un esempio classico: in Giappone la mancia non è contemplata proprio per questo motivo, offrire una mancia al cameriere porterebbe a disorientamento, un bug nel sistema, questo povero cameriere non saprà dove mettere quel pericoloso denaro vagante nelle proprie tasche. Lo ha prelevato dalla cassa? Si domanderà se al tavolo c'è stato qualche disservizio? Maneggiare denaro mentre serve un piatto potrebbe portare al suo licenziamento per motivi igienici?

Lo stesso discorso è valido per l'uscita del fantomatico Nintendo Nes Mini, al suo debutto mondiale. Il prezzo era destinato ad essere proprio quello, né più né meno, offrendo plastiche superiori, giochi con licenza, saturando la vendita sui mercati mondiali ad una soglia prestabilita, per trasformarlo a breve in "oggetto dei desideri" ed in futuro "oggetto da collezione". Tutto programmato. Anche la notizia che la prima distribuzione sarebbe terminata in brevissimo tempo fu programmata a puntino. La seconda distribuzione infatti lo aveva già elevato allo stadio di "oggetto dei desideri". Non sapremo mai se nei magazzini Nintendo vi fossero stipate milioni di unità non vendute. Sappiamo però che il marketing stava costruendo la genealogia del Mini Nes, stilando un pedigree che avrebbe influenzato il proprio destino. Tutto programmato: prezzo, tempi, distribuzione, fughe di notizie, fake news.

Un'altra domanda che mi viene spesso posta è: "ci sono differenze tra il loro e il nostro mercato?".

Anche questa risposta è semplice. Il Giappone è un altro mondo e per loro è il primo e l'unico, complice un isolamento geografico che ha portato ad un isolamento psicoattitudinale. Il resto del mondo, per loro, è un mercato secondario, relativamente importante, non indispensabile, il problema dei soldi e della quantità di unità vendute, non appartiene alla loro mentalità, o almeno non con valori a noi logicamente comprensibili.

Nella mentalità del marketing giapponese ho potuto analizzare almeno tre obiettivi del loro poliedrico scenario consumistico.

Il loro principale obiettivo è soddisfare la golosità consumistica interna ed alimentare il fervido panorama collezionistico interno.

L'esportazione nel mondo Occidentale è una congettura, vi propongo l'esempio della desiderata Ps4 Pro Million Edition, con tiratura estremamente limitata, la quale ha dimostrato una vantaggiosa distribuzione giapponese e volutamente modesta al di fuori della loro nazione, lanciata in Agosto 2018.

https://www.playstation.com/en-us/explore/ps4-pro/systems/500-million-limited-edition-ps4-pro-console/

Un ultimo esempio è l'altrettanto desiderata Ps4 Pro 2 Tb (CUH-7200CB01) lanciata a fine Novembre 2018 nel mercato nipponico e, chissà per quali accordi e patti a noi sconosciuti, poco dopo sul mercato malesiano (CUH-7218CB01). Quante di queste ultime due console sono sbarcate nel famelico mondo occidentale? Zero.

Vi lascio degli articoli per l'approfondimento: https://www.everyeye.it/notizie/playstation-4-pro-2tb-in-arrivo-in-giappone-fine-novembre-349921.html

https://pur.store.sony.jp/ps4/products/ps4/CUH-7200CB01_product/

Il loro secondo obiettivo è fidelizzare la massa per attirarla al mondo dell'elettronica e sfornare nuovi "Creativi" che alimenteranno software ed hardware del gaming.

In questa fase nascono gli Otaku. Non facciamoci ingannare dai traduttori automatici, questo aggettivo è carico di caratteristiche intrinseche alla loro cultura.

Otaku non è assolutamente sinonimo di nerd: da loro le sottocartelle inerenti a queste tipologie di persone sono ben più numerose e ben distinte rispetto alle nostre conoscenze.

In Giappone per esempio ci sono persone che non escono di casa, ma non sono definibili agorafobici.

Altresì esistono persone che non parlano al microfono durante una Battle Royale, ma non sono autistici.

Esistono anche persone che parlano di Retrogaming da mattina a notte senza mai dormire, ma non sono nerd.

Otaku in realtà significa "profondo conoscitore" con una sottile provocazione al lato "Yabai" cioè un confine pericoloso dell'ovvio lato oscuro definibile in "dipendenza patologica".

In realtà questa situazione sociale spaventa molto noi occidentali, invece da loro, è una prassi comune riunirsi in serate Cosplay, in una piazzetta semi deserta della propria città, con costumi curati all'inverosimile e videoregistrare un'intera puntata dell' Anime in questione. Fedelmente. Con accurata minuziosità. E' una visione agghiacciante ai nostri occhi: quando mi imbatto in queste situazioni vivo un conflitto interiore di sorpresa e non accettazione di ciò che sto osservando, ma col trascorrere delle scene "tagliate" e le scene "buone" rimango coinvolto dalla loro sfrenata energia. Non riesco a giudicare "strana" questa passione, sebbene la nostra astrazione culturale venga messa a dura prova, dato che all'ultimo Nerd Show di Bologna di Febbraio 2019 c'erano Cosplay semplicemente magnifici, tra i quali Loki, la Regina di Cuori, Tiger Man, mentre la stragrande maggioranza portava da casa un costume che agli occhi nipponici avrebbe sicuramente destato un irrefrenabile spirito fraterno di upgrade. Nessuna critica amici miei, anzi, mi sorprendo della maniacale ricerca del dettaglio giapponese piuttosto che dei genuini Cosplay italiani.

Ritornando al discorso videogiochi moderni o passati, non fa alcuna differenza, il mondo Cosplay e quello videolutico sono entrambi fusi ed indissolubili: potremmo vedere passeggiare un Cosplay del primo Goku di Dragon Ball del lontano 1984.

Oppure un giapponese potrebbe ottenere una distribuzione demo di un videogioco che uscirà a breve, in fase beta, destinata ai migliaia di collaboratori silenti (gli Otaku del gaming) che venderebbero l'anima pur di ottenere in anteprima una versione ancora instabile, correggerla, inviare gratuitamente le correzioni alla casa madre del software/hardware ed entrare nell'Olimpo personale del Web fregiando un Nickname che diverrà eterno nella loro Web Comunity.

Follia? Assolutamente no, è semplice prassi!

Il loro terzo obiettivo conclude questa "catena della dipendenza". Facilmente intuibile: nasce una domanda spontanea in noi, desideriamo capire come sia stato creato questo meccanismo così oliato nella fidelizzazione e oserei dire... schiavizzazione di questi Otaku dell'elettronica.

Il meccanismo, ripeto, è stato studiato a tavolino nel marketing. L'elettronica sin dagli albori della loro rivoluzione, dopo la seconda guerra mondiale, ha fatto leva sulle caratteristiche intrinseche della popolazione. Il denaro per loro ha un interesse relativo, la perfezione e l'integrazione nei meccanismi interni invece deve essere assolutamente perfetta, altrimenti il "Sistema giapponese" li espellerà.

Questo terzo obiettivo, ai nostri occhi si può definire come un "censimento degli elementi interni al Sistema". Da queste regole di base, il passaggio successivo è storicamente comprovato: il controllo. Un controllo che funziona perfettamente in Giappone è, appunto, la fidelizzazione: al contrario di noi italiani che possiamo cambiare compagnia telefonica pressoché mensilmente!

Loro, invece, sono fidelizzati all'inverosimile, spenderanno sforzi ed energie per sostenere e migliorare l'ingranaggio al quale appartengono. Se in una famiglia si nasce fidelizzati, per esempio alla Sony, la propria esistenza sarà destinata alla grandiosità della Sony ed ognuno sarà onorato ed orgoglioso di appartenere alla maturazione ed evoluzione dei processi interni della ditta.

L'ultimo passaggio, dopo la fidelizzazione, porta alla degenerazione del controllo, cioè la creazione di una malsana dipendenza al meccanismo.

Faccio un esempio semplicissimo.

In Italia durante l'uscita del Nintendo Nes Mini, la modalità di prenotazione più funzionale era attraverso il web. Purtroppo nel 90% delle volte che si ricercava la disponibilità dell'articolo dentro internet, ahimè, risultava non disponibile. Improvvisamente, alcune volte, magari alle due di notte, poteva tornare brevemente disponibile. Si prenotava all'istante, magari con mesi di anticipo. Si pagava al momento della spedizione. Arrivava alla data stabilita. Massima comodità!

In Giappone invece?

Mentre in Occidente usciva il Nes Mini, da loro usciva il Fami-Com Mini, alcuni giochi interni differivano dalla nostra versione e la scocca era simile al loro Family-Computer del 1986.

Il numero di pezzi venduti era drasticamente inferiore alla domanda interna del Giappone.

Il web nulla poteva per adempiere alla prenotazione.

Se un giapponese avesse desiderato comprarlo, come prima cosa doveva obbligatoriamente selezionare la catena alla quale rivolgersi, Bic Camera era una delle più approvvigionate. Ci si armava di santa pazienza e il giorno fatidico in piena notte (o addirittura giorni e giorni prima), ci si metteva in fila davanti alla catena di distribuzione selezionata, ovviamente tutto avveniva in maniera composta ed ordinata.

Perchè questo sadismo sociale? Semplice: il disegno di creazione dell'"oggetto del desiderio" iniziava a prendere forma attraverso la lunghezza della fila. La lunga fila infatti quantificava il desiderio di possesso dell'oggetto e ne stabiliva il grado di elevazione riguardo al concetto di collezione.

Finalmente giunta la mattina dell'apertura, dopo la divulgazione di foto, video ed interviste ad alcune persone della fila attraverso web, giornali, telegiornali... si poteva procedere alla prenotazione.

Ho detto prenotazione? Si, ma a modo loro! Non è una prenotazione reale dell'oggetto in questione. Semplicemente si procedeva ad un sorteggio. Arrivato il proprio turno al banco delle prenotazioni, finalmente sarebbe arrivato il verdetto: la persona dopo la lunga attesa poteva mettere la mano in un recipiente contenente biglietti, prenderne uno e partecipare al sorteggio.

Sorteggio positivo? Prenotazione garantita. Sorteggio negativo? Prenotazione negata! Qual è il motivo di questo sorteggio? Semplicemente i pezzi dell'articolo venduto non dovevano soddisfare il numero della domanda.

Si, oltre il Nes Mini, io possiedo anche il Famicom Mini e il Mini Famicom Dragon Ball edition. Grazie di cuore a chi mi ha aiutato ad espletare questa cinica, oliata, consolidata ed ormai nazionalmente accettata procedura giapponese!

Passiamo alla prossima domanda che spesso mi viene posta.

Sovente mi viene chiesto come gestiscono il tuning, modding, hacking in Giappone.

Purtroppo, questa volta, la risposta è particolarmente triste. Ricordo una puntata dei Simpsons dove il conduttore di un Quiz Show giapponese spiegava ai Simpsons che in Occidente veniva premiata la conoscenza, differentemente in Giappone veniva punita l'ignoranza. Si è tutto vero. Se sarai un fedele suddito del Sistema, tutto ok. In caso di sgarro, sarai punito. Duramente.

Ricordate le cause legali dell'anno scorso, intentate da Nintendo e Sony, per debellare il recupero illegale delle ROM dal Web? In Occidente la cosa fu molto combattuta.

Nella nostra coscienza sappiamo benissimo che possediamo alcuni cimeli rari dei giochi di tanto tempo fa. Il supporto magnetico è destinato a dissolversi col tempo. L'usura del supporto stesso, durante le nostre partite, non fa altro che accelerare il processo di deterioramento magnetico.

Pertanto, detenere "anche" una copia digitale del gioco fisicamente (e legalmente) in nostro possesso, potrebbe preservare o almeno allungare la vita del nostro cimelio custodito in vetrina. Allo stesso tempo il discorso dell'Abandonware rischia di farci perdere milioni di opere dell'intelletto umano, esattamente come è accaduto con molte pellicole del cinema. Preservare per ricordare, significa forse rubare? Infine salvaguardare la memoria trova concreto sodalizio con il tema di libertà di informazione globale, come coscienza e scibile collettivo, al fine di migliorare il Sapere. Queste, secondo me, sono cause nobili, fraterne, sacrosante. Non stiamo parlando di giocare ad una perfida copia pirata del primo Super Mario, nella penombra, con un caschetto di alluminio come copricapo (esempio di Fry in Futurama) senza aver pagato i diritti alla Nintendo. Oggigiorno i capolavori videolutici del 2019 (per esempio Apex Legends o Horizon Zero Dawn) sono dei titani di meraviglia grafica rispetto al retrogaming. Nel retrogaming sappiamo che non c'è un reale discorso monetario. Piuttosto si parla di preservare, collezionare, studiare e mostrare al mondo le origini. Un tema simile ai delicatissimi "Giorni della Memoria": dobbiamo semplicemente ricordare di non dimenticare.

Come funziona in Giappone? No amici miei, come avrete già intuito, in Giappone questi temi non scuotono la coscienza collettiva.

Ultimamente Sony e Nintendo hanno creato una dilagante censura. Hanno fatto chiudere l'attività di distribuzione di numerosi siti in questi ultimi due, tre anni.

Per noi occidentali, questa fase, è un semplice, rapidissimo momento di riorganizzazione del materiale... ma in Giappone?

Ecco a voi il loro punto di vista: http://www2.accsjp.or.jp/activities/2018/pr6.php

L'articolo spiega che da Novembre 2018 è vietato l'hacking e il modding delle console e retroconsole. La legge punisce sia chi rivende le console modificate, sia chi informa sul come modificarle, sia chi aggiunge giochi alle console modificate. Saranno puniti anche gli individui che ospitano tali informazioni e soprattutto chi rivende codici di attivazione e chiavi dei giochi digitali. La multa, che sarà pagata alla ditta informatica lesa, potrà ammontare alla mostruosa cifra di cinque milioni di yen (circa 46.000 dollari) e penalmente si rischierà la detenzione fino a cinque anni. Tutto è partito dall'Unfair Competition Prevention Act che da alcuni mesi riconosce il software come un elemento che deve essere legalmente protetto.

Ecco a voi il primo arresto e multa avvenuti a Novembre 2018, buona lettura (previo servizio di traduzione automatica online):

https://www.asahi.com/articles/ASLCM6D81LCMPTIB00G.html

Se avete dubbi sulla legalità o giustizia di tale legge, non preoccupatevi, il Sistema giapponese, nel Gennaio 2019 l'ha prontamente rivista ed **inasprita**.

Ricordate le discusse PandoraBox V3, V4, V5? Bene, per via di questo clima punitivo, anche le odierne Pandora Box cinesi (V6s etc...) vengono vendute senza ROM interne, la cui reperibilità sarà meticolosamente spiegata di volta in volta dai rispettivi venditori della Pandora medesima.

Cosa ne pensate? Opprimere o divulgare? Ha senso parlare di lucro nel mondo del retrogaming? E' meglio perdere bobine di nastri magnetici e floppy da cinque pollici o custodire e preservare nel tempo la memoria, rischiando di oltrepassare la barriera della legalità? Dopo aver appreso l'esistenza di questa legge nipponica, col rischio che si

amplierà a macchia d'olio nel resto del mondo... legalità e giustizia sono diventati sinonimi oppure si sono trasformati in un ossimoro?

"Ai posteri l'ardua sentenza". Sicuramente i temi dell'oppressione nella storia hanno portato a rapida riorganizzazione degli ingredienti. La storia si ripete: il passato è costellato di virtuosi che si sono sacrificati per un'attività in cui hanno creduto profondamente.

Un'ultima frequente domanda che vorrei discutere in questa recensione insieme a voi è questa: "cosa riserva il futuro dei videogiochi in Giappone"?

Questa risposta è la più articolata in assoluto. Il Giappone crea, modifica, amplia, distrugge, degenera le tendenze ad una rapidità che ai nostri occhi risulta incomprensibile. Un esempio semplice: oggi potremmo aprire un ristorante italiano nel centro di Tokyo e vivere un mese di assedio da parte di migliaia e migliaia di clienti curiosi. Tra due mesi potremmo rischiare di chiudere poiché la loro curiosità è stata soddisfatta e non rientriamo nel loro Sistema per via di una sola variabile del loro poliedrico cosmo di complicate variabili interne.

Le vie principali delle grandi città cambiano fisionomia di insegne e locali, ad una frequenza praticamente mensile. Tutto è vorticosamente rapido e mutevole. Ai nostri occhi è qualcosa di incomprensibile, per loro è normalità assoluta.

Quindi, cosa ci riservano i colossi Nintendo, Sony, Konami, Sega, Naomi, Taito, Snk etc..? Sicuramente la cosa che dobbiamo tenere a mente è la quantità di titoli rilasciati internamente al Giappone che mai e poi mai usciranno in formato compatibile a noi occidentali.

I motivi sono numerosi: problemi di traduzione di testi-linguaggi, problemi di compatibilità del preistorico divario riguardo al formato audiovisivo pal-ntsc e non solo, infine problemi di esportazione nel mondo per via della severa regolamentazione del Copyright. La stragrande maggioranza delle loro sterminate saghe Manga, Anime, videogames sono completamente sconosciute a noi occidentali. Non arriveranno mai a noi. Molti titoli geniali e spettacolari rimarranno in pasto al loro Sistema interno. Il Giappone è una nazione molto evoluta in quasi tutti i temi, specialmente quelli sociali.

Quando una nazione, extra nipponica, raggiungerà tali livelli di evoluzione relativamente a determinate tematiche socioeconomiche, allora le ditte nipponiche di marketing, specificatamente per una determinata saga, decideranno se divulgare i propri titoli in tale nazione. In parole semplici: se quella determinata saga di Anime non sbarca da noi, significa che non abbiamo ancora maturato alcuni elementi sociali o economici che ci permetterebbero di accettare e comprendere tale saga.

Con questi dati riusciamo a comprendere meglio perchè Dragon Ball abbia attecchito in Italia e ad oggi sia trasmesso ogni anno, ciclicamente, con brevi interruzioni di qualche mese.

I temi di Akira Toriyama nascono nel lontano 1984, appartengono al 1984 sebbene il recente remake Dragon Ball Super sia stato un successo mondiale, in Giappone è già storia passata. In Italia invece il marketing è ancora in fermento. Adesso cari lettori, con questi dati, fate una valutazione. Quanti aspetti evolutivi del 2019 e del 1984 appartengono all'Italia? Quanti aspetti di queste due epoche, così lontane, appartengono invece al Giappone odierno?

Come abbiamo visto prima, è un sistema discutibile ai nostri occhi, ma vi assicuro che lì funziona tutto in maniera perfetta. Concludendo, ad oggi posso aggiornarvi su due temi prossimi nello sviluppo software. Nintendo probabilmente abbandonerà l'ambiente delle console. No tranquilli, Nintendo non fallirà: avevo letto un interessante articolo redatto da un economista americano. Questo studioso aveva dimostrato un calcolo riguardo l'eventuale possibile futura dichiarazione di fallimento della Nintendo. La nostra amata Nintendo, dopo l'enorme successo dello Switch, se dovesse chiudere la ditta e vivere di rendita con diritti e Copyright, vivrebbe un declino di durata semplicemente plurisecolare.

Il destino della Nintendo è stato già deciso da qualche mese: entrerà nella quotidianità con le proprie icone ormai divenute immortali (Mario, Donkey Kong, etc.), pertanto Android e altri sistemi proprietari e non, potrebbero presto ospitare licenze Nintendo.

https://www.money.it/Nintendo-addio-console-futuro-mobile-smartphone

L'altro aggiornamento riguarda la Konami, nata ad Osaka nel 1969. Sebbene l'anno scorso abbia fatturato un +18% ha già programmato di abbandonare l'ambiente videoludico. Nel 2019 molto probabilmente sfornerà solo un altro titolo per l'Occidente: Pro evolution soccer.

Quasi sicuramente sarà l'ultimo, o tra gli ultimi, quindi da collezione, trascinando inevitabilmente al rialzo le valutazioni di tutti i PES precedenti. Konami si dedicherà allo sviluppo software e hardware delle slot machine giapponesi nelle famose sale di Pachinko.
https://www.senzalinea.it/giornale/konami-sta-abbandonando-il-settore-videogiochi/

Sono passato davanti queste sale migliaia di volte, non mi hanno mai attirato. Non sono fidelizzato da questo tipo di scommesse né tantomeno la mia cultura occidentale è riuscita a far germogliare in me il seme della curiosità per queste palline argentate che scorrono in questa sorta di flipper perpendicolare.

In Giappone invece le sale sono sempre piene. Giorno e notte. A qualsiasi orario. Come funzionano realmente le slot del Pachinko? Siete desiderosi di sapere la verità, anzi i vari possibili scenari attuali? Quali informazioni sono giunte a noi e soprattutto, è arrivata la pura verità o la mistificazione di queste realtà dallo sfondo poco cristallino?

Fatemi sapere cari lettori ciò che siete desiderosi di apprendere. In base alle richieste si potrà parlare di Tokyo, Akihabara, Shinjuku, di come raggiungere i distretti del collezionismo oppure di come reperire oggetti rari nei negozi dell'usato, sia nelle grandi che soprattutto nelle piccole città meno agguerrite nel collezionismo.

A voi la scelta, a presto!

di Michele Ugolini

I-N-C-O-M-P-R-E-S-I

a cura di Associazione Firenze Vintage Bit Onlus (Massimo Belardi)

Giovedì 24 gennaio 2019, presso la biblioteca comunale di Lastra a Signa, una delle sedi dei ragazzi del Firenze Vintage Bit, Massimo Belardi, socio dell'Associazione Firenze Vintage Bit Onlus e collaboratore del Museo del Calcolatore "Laura Tellini" di Prato, ha tenuto una conferenza su alcuni (famosi) "Incompresi" nella storia del retrocomputer.

Per Massimo è stata la quarta presentazione, si può considerarlo "uno di casa" qui a Lastra a Signa.

Per questa conferenza ha selezionato cinque piattaforme che, per motivi diversi, vedi un design troppo avveniristico, uscita prematura sul mercato o scelte di marketing errate, sono state incomprese e bocciate dal grande pubblico.

Già nella locandina (disegnata da Leonardo Vettori di AFVBO) si annunciavano le cinque piattaforme sotto una velata criptazione...

Vediamole più da vicine

1 – NeXTSTEP. Ripartire da zero.

La conferenza è iniziata con una introduzione di NeXTSTEP.

Massimo ha condensato, nel limite dei pochi minuti, la storia e l'importanza di NeXTSTEP.

Abbiamo avuto modo di fare un salto indietro di circa trent'anni, al 18 settembre 1990, giorno in cui si tenne una key note al Symphony Hall di San Francisco.

Un giovanissimo Steve Jobs, più formale dell'abituale, presentava la seconda generazione di casa NeXTSTEP.

NeXTSTEP nasceva nel 1985, per la forte determinazione di Jobs di ricostruire, partendo da zero, la sua "Apple", dopo che la stessa l'aveva defenestrato poco tempo prima.

L'azienda ebbe una breve vita, poco meno di dieci anni, non riuscendo mai, compiutamente, a produrre una piattaforma stabile e credibile (e piuttosto cara).

Nonostante ciò, sia l'hardware utilizzato (architettura 68000 avanzatissima, uso di coprocessori dedicati alla multimedialità) che il software sviluppato (sistema operativo preemptive orientato agli oggetti, pieno supporto software del Postscript) pongono NeXTSTEP come uno dei passaggi tecnologici fondamentali nella storia dell'informatica.

Tra gli altri, proprio su una stazione NeXTSTEP Tim Barners Lee sviluppò il primo sito WWW.

Da aggiungere che proprio nel momento della bancarotta, dopo aver chiuso il laboratorio di sviluppo hardware, arrivò l'offerta d'acquisto Apple, interessata al sistema operativo come sostituto dell'oramai vetusto MacOS

2- Mattel Aquarius, la miglior piattaforma degli anni '70.

Dopo il NeXTSTEP, la narrazione ci ha condotto sulle tracce del Mattel Aquarius, la cui genesi è piuttosto originale.

All'inizio degli anni '80, Mattel decise di contrastare lo strapotere dell'Atari 2600.

Gli ingegneri delle casa di Ken e Barbie svilupparono a detta di molti, la miglior console di gioco del periodo. L'Intellivision.

Al lancio pubblicitario, Mattel fece intendere che, tra le future funzionalità della console, ci sarebbe stato un kit composto da tastiera, ram di espansione e processore aggiuntivo, che avrebbe trasformato Intellivision in un home computer completo.

Passato più di un anno e mezzo, non si avevano aggiornamenti sull'uscita prevista del kit; e molti clienti (specie in terra USA), iniziarono ad innervosirsi e lamentarsi.

Lamentele prese in seria considerazione dall'ente governativo FTC (US Federal Trade Commission), che accusò la Mattel di pratiche commerciali scorrette (in realtà Mattel stava incontrando enormi problemi nello sviluppo della periferica, tanto che tramite il silenzio, stava perseguendo il sogno di porlo nel "dimenticatoio" della storia).

Mattel fu costretta a fare qualcosa... Ed uscì Aquarius... E l'Aquarius fu un fallimento completo.

La piattaforma era troppo scadente in ogni sua parte, con pochissima RAM (4K), una grafica inferiore allo ZX e C64, con un Basic povero di comandi (si narra che un ingegnere di casa Mattel lo definì "la miglior piattaforma degli anni '70"). Pochi mesi ed uscì di produzione.

3. Sinclair QL e le origini di Linux.

Potrà apparire assurdo oggi, eppure a metà del 1984, il settore degli home computer del Regno Unito stava raggiungendo la saturazione. In soli due anni, si passò da 20.000 a più di 500.000 home computer venduti.

Conscio del successo, che lui stesso aveva determinato, Clive Sinclair studiò nuove soluzioni. In quei mesi, pensò ad un modo per anticipare e sconfiggere il più volte annunciato Apple. Non solo volle battere la concorrenza, volle proporre una macchina

mai vista. E non si può negare che il nucleo del Sinclair QL è assolutamente innovativo

Il processore , della famiglia Motorola 68000, è il primo 32 bit montato a bordo di un home computer; anche il sistema operativo, (il primo preemptive) aveva delle potenzialità per il tempo infinite.

Ma, in perfetto stile Clive Sinclair, la scelta di ottimizzare i costi al minimo possibile (cosi che i bus di trasmissione furono ad 8 bit, penalizzando le potenzialità del processore) unito alla fretta di uscire (che determinò lo sviluppo di un sistema operativo tanto potente quanto buggato), decretò il fallimento commerciale del QL.

La macchina non incontrò il favore del pubblico (che, incredibilmente, si lamentò perfino dell'assenza di un'unità nastro dedicata); le software house non vi investirò più, e senza software, il QL divenne uno strumento di nicchia.

Sinclair dovette cedere il tutto all'Amstrad, che non avendoci mai creduto alle potenzialità del QL, interruppe la produzione di lì a poco. Il QL divenne una macchina di nicchia, uno dei più clamorosi "WhatIf" tecnologici.

Una nicchia prestigiosa, visto che uno dei suoi più utilizzatori fu un giovanissimo Linus Torvalds, che proprio dalla necessità di crearsi software per proprio conto, porrà le basi per la creazione di Minix che diverrà Linux.

NdR – Del Sinclair QL trovate un esauriente articolo in questo stesso numero a cura di Alberto Apostolo.

4. Grundy NewBrain e il governo dell'Angola.

Il Grundy NewBrain rappresenta un caso unico nella storia dell'informatica.

Pensato come primo home computer di casa Sinclair (il progetto era già in fase avanzata quando Sinclair passò al futuro ZX80), in seconda battuta fu corteggiato dalla BBC come prodotto di punta (ed anche in questo caso, sempre in fase avanzata di progetto, si cambiò idea scegliendo il prototipo di casa Acorn).

Dopo due defezioni così di prestigio, NewBrain uscì per la neonata Grundy, trasformando quello che doveva essere un computer "portatile" in una macchina business espandibile a 2MB di RAM , che faceva girare CP/M, per gestire la filiera della farmacie di Sua Maestà britannica. Non male.

La versatilità del NewBrain durerà negli anni, anche dopo il fallimento della Grundy; le rimanenze di magazzino furono smistate verso due destinazioni. Una parte finì nell'Europa del Nord, in particolare negli istituti scolatici danesi ed olandese; una parte in Angola, dove ancora nel 1994 fu utilizzato come piattaforma governativa dei propri piani quinquennali.

5. Commodore Plus 4, la pecora nera della Commodore.

L'ultima piattaforma è stato un dovuto omaggio all'essenza stessa dell'incompreso/incomprensibile retrò.

Per motivi mai completamente chiariti, Commodore progettò di sostituire il C64 con una serie di home computer, nati dal prototipo Commodore 216, con caratteristiche insensate.

Il Plus 4 montava onboard un BASIC evoluto e potente, 4 programmi precaricati nella ROM (una sorta di pacchetto Office ad 8 bit), che lo ponevano come un candidato credibile nel settore business; il suo coprocessore TED semplificava l'architettura hardware (una sola scheda per gestire audio e grafica); la presenza di un assembler/disassembler semplificava lo sviluppo in codice ASSEMBLER.

Ma... Sul Plus 4 non si potevano attaccare le già costose periferiche del C64;

La potenza del chip grafico non presentava l'uso degli sprite (limite assoluto per i videogiochi post C64), il sintetizzatore audio era inferiore allo stesso SID.

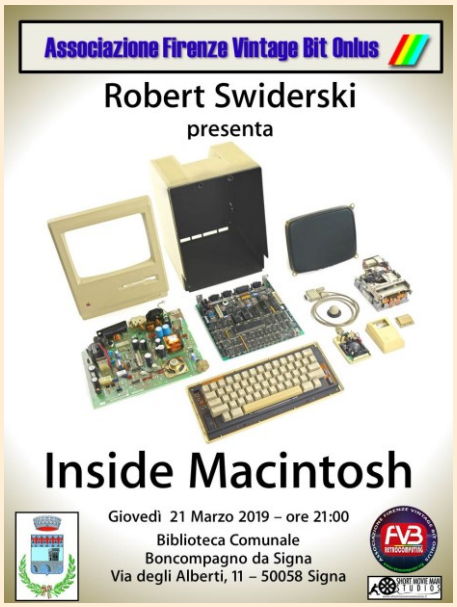
Risultato. Non essendo una vera macchina "professionale" e tantomeno una macchina per il gaming; i possibili clienti, costretti anche ad un oneroso esborso per le periferiche, non lo acquistarono, decretando per il Plus 4 un fallimento commerciale.

Il dibattito

La serata si è conclusa nel più classico dei modi, con il dibattito.

Chi aveva il Plus 4 ha raccontato pregi e difetti del il Plus 4, chi aveva il QL ha raccontato pregi e difetti del QL, chi aveva il NEXT ha raccontato pregi e difetti del NEXT, ma del NewBrain e il povero Aquarius... Beh... Di quelli non ne abbiamo proprio discusso.

Potete rivedere la conferenza sul canale Facebook di Firenze Vintage Bit.



Inoltre l'**Associazione Firenze Vintage Bit Onlus** vi ricorda il prossimo appuntamento "**Inside Macintosh**" con **Robert Swiderski**.

Vi aspettiamo a Signa il **21 Marzo** presso la Biblioteca Comunale Boncompagno da Signa.

Svelato l'arcano

di Francesco Fiorentini

Nella chiosa finale del numero scorso avevo anticipato due novita', svelandone pero' soltanto una, quella relativa al corso SEUCK.

Quello che non volevo anticipare, per non rovinare la sorpresa, era lo speciale sul **Vectrex**, con un articolo scritto da **Robin Jubber**, uno che di Vectrex se ne intende eccome. Tanto e' vero che ci ha fatto anche la cortesia di allegare un Hello World per il 6809. E con un articolo scritto da **Dante**, che mi ha fatto tornare indietro nel tempo e scoprire il bug del tredicesimo livello di Mine Storm, che non conoscevo. Senza contare le recensioni dei giochi fatte da **Giorgio** e **David** (che ha curato anche la traduzione dell'articolo di Robin).

Vorrei anche sottolineare come questo numero in particolare sia ricco di collaborazioni esterne, segno evidente che sempre piu' lettori stanno raccogliendo l'invito a trasformarsi in redattori per renderci tutti partecipi delle loro conoscenze. Una pubblicita' di un noto pneumatico diceva: la potenza e' nulla senza controllo. Vorrei fare mia questa massima e trasformarla in: **la conoscenza e' nulla se non e' condivisa**.

A costo di ripetermi quindi, rinnovo l'invito a tutti per collaborare alla nostra realta' comunicandoci articoli e/o materiale da pubblicare.

Facce da RM

I lettori piu' navigati avranno colto subito la citazione. ☺ Esisteva, ed esiste tuttora, una rubrica su TGM che si intitolava: facce da TGM. Il titolo ben si addice ad introdurre il primo raduno della redazione di RM che si terra' nel mese di Marzo a Roma.

Purtroppo non tutti i redattori riusciranno a partecipare al nostro primo incontro, ma sara' l'occasione per tanti di noi per scoprire che faccia abbiano gli altri.

Come detto sin dai primi numeri, la redazione di RM e' una redazione virtuale; una manciata di persone che collaborano tutti i mesi per garantire l'uscita di questa fanzine, senza pero' avere ancora avuto modo di conoscersi personalmente. Probabilmente io sono uno dei piu' fortunati perche' ho gia' incontrato per una ragione o per un'altra circa la meta' dei collaboratori, ma qualcuno di noi non ha mai visto nessuno degli altri dal vivo... Inutile dirvi che la curiosita', almeno da parte mia, e' alle stelle.

L'idea di fare una bella foto di gruppo e pubblicarla in uno dei prossimi numeri per rendere giustizia alla bellezza esteriore dei redattori c'e tutta. Speriamo che non ci siano redattori timidi. ☺

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!

Disclaimer

RetroMagazine
Anno 3 - Numero 13

Direttore Responsabile
Francesco Fiorentini

Vice Direttore
Marco Pistorio

Febbraio 2019