Jurassic News

Silicon Graphics INDY

Super QL

Lo sviluppo del linguaggio C

Il mio Z80

Nosferatus

L'indecibilità

Geriatric Linux

Retrocomputer Magazine

Anno 11 - Numero 57 - Febbraio 2016

Jurassic News
Rivista aperiodica di Retrocomputer

Coordinatore editoriale:
Tullio Nicolussi [Tn]

Redazione:
redazione@jurassicnews.com

Hanno collaborato a questo numero:
Lorenzo [L2]
Salvatore Macomer [Sm]
Sonicher [Sn]
Besdelsec [Bs]
Lorenzo Paolini [Lp]
Mario Raspanti
Gizmo
Emery Fletcher
Damiano Cavicchio

Diffusione:
Lettura on-line sul sito o attraverso il servizio Issuu.com; il download è disponibile per gli utenti registrati.

Sito Web:
www.jurassicnews.com

Contatti:
info@jurassicnews.com

Jurassic News
promuove la libera circolazione delle idee

Jurassic News

E' una fanzine dedicata al retro-computing nella più ampia accezione del termine. Gli articoli trattano in generale dell'informatica a partire dai primi anni '80 e si spingono fino ...all'altro ieri.

La pubblicazione ha carattere puramente amatoriale e didattico, tutte le informazioni sono tratte da materiale originale dell'epoca o raccolte su Internet.

La redazione e gli autori degli articoli non si assumono nessuna responsabilità in merito alla correttezza delle informazioni riportate o nei confronti di eventuali danni derivanti dall'applicazione di quanto appreso sulla rivista.

Il contenuto degli articoli è frutto delle conoscenze, esperienze personali e opinioni dei singoli autori; possono pertanto essere talvolta non precise o differire da fonti "ufficiose" come Wikipedia e siti Web specializzati.

Sono gradite segnalazioni di errori, imprecisioni o errate informazioni che possono, a discrezione della redazione, essere oggetto di errata-corrige in fascicoli successivi.

Scrivere a:
redazione@jurassicnews.com
dettagliando il più possibile l'argomentazione.

Veterani cercasi.

Come è lontana l'Inghilterra!

Il museo nazionale "The National Museum of Computing (TNMOC)" con sede a Londra, nel famoso Bletchley Park, ha fra le sue iniziative dei veri e propri percorsi di retro-informatica che sono seguiti annualmente da circa 4500 studenti delle superiori.
Numerosissimi i progetti, le iniziative e importanti anche gli sponsor (l'ultimo è Fujitsu) che consentono di portare avanti iniziative di un certo spessore.
Fin qui quello che manca in Italia, anche se le iniziative private ci sono e sono degne del massimo rispetto.

La collezione ricca di reperti (uno per tutti il famosissimo Colossus) ha come massima numerosità il famoso micro BBC, uscito nel 1981 e diffusissimo come macchina da studio sia nelle scuole che nelle case inglesi.
Ora questa numerosissima collezione di micro BBC abbisogna di cure e i volontari non sono mai abbastanza.
Ecco dunque che il museo cerca veterani (proprio così li chiama) che abbiano dimestichezza con la macchina nei suoi aspetti, hardware, software e periferiche varie.
L'obiettivo è la cultura.

E le istituzioni pubbliche italiane?
Assenti.
C.v.d.

Sommario

Jurassic News - Il contenuto di questo fascicolo

OK, il prezzo è giusto!

di Tullio Nicolussi

Periodicamente se ne parla: qual è il valore di un retro computer e per conseguenza qual è il prezzo equo da sborsare per acquistarne uno?

Nonostante ci sia qualche tentativo di compilare un listino identificando per ogni oggetto una fascia di prezzo, tali iniziative sono destinate a suscitare l'ilarità di molti e l'irritazione di altrettanti conoscitori e potenziali appassionati di retro informatica.

Il problema non è l'impreparazione di chi compila i listini (svarioni a parte). Il fatto è che è ancora un mercato molto legato alla nazionalità. Cioè un sistema comune in Italia, ad esempio l'M24, potrebbe essere raro negli States e quindi spuntare un buon gap di differenziale di prezzo fra le due nazioni.

Spesso anch'io mi sono trovato davanti al dilemma se comprare o desistere, al punto che come regola generale mi sono ripromesso di non acquistare mai se il prezzo richiesto supera i 50 Euro, qualunque sia lo stato dell'apparecchio proposto.

50 Euro è una soglia psicologica più che un dato di valenza rispetto al prodotto. Tuttavia sarei disposto a più di una eccezione se la richiesta riguardasse un pezzo particolarmente significativo. Non dico un pezzo "raro", per il quale le quotazioni esulano dall'aspetto retro-computer e sfociano nel campo della stima dei reperti artistici. Un Apple I non si discute, ovviamente!

Già per un Lisa potrei anche arrivare ai 100 Euro, ben conscio di star prendendo un qualcosa che non funziona o che è molto difficile da riparare un giorno che avesse deciso di "passare a miglior vita". Non spenderei mai 100 Euro per un C64, nemmeno mancasse alla mia collezione, o per uno Spectrum...

Va bene, queste due ultime menzionate sono macchine comuni, nel senso che si può ragionevolmente rifiutare l'acquisto oggi per trovare domani lo stesso prodotto a prezzo equivalente o inferiore. Quindi una qualche idea della fascia di prezzo di un certo reperto la abbiamo: forse ce la siamo fatta mediando le aste su eBay? Può essere:

penso che chiunque ci faccia un giro su eBay prima di vendere/comprare qualsiasi cosa.

Molti commentano il valore presunto della loro collezione, appena qualche incauto venditore posta un articolo a prezzo esagerato. Queste "uscite dal seminato", che apparentemente sembrano tutte frutto dei tentativi di utenti non esperti del comparto, in realtà qualche volta mi fanno sospettare diversamente.

Ho una opinione in proposito e cioè che ci siano due specie di venditori "fuori bersaglio": gli inconsapevoli e i furbastri.

Dei primi c'è poco da dire: ci provano a vendere il loro bravo C64 a 200 Euro e il loro bravissimo Amiga 500 a peso d'oro. Ci provano e di solito sono sbeffeggiati con lazzi e sberleffi degni della migliore gogna mediatica.

I secondi, cioè i furbastri, mi preoccupano un pochino di più. Ci sono quelli (almeno un paio in Italia) che vendono le macchine a pezzi. A seguire un attimo le loro aste si capisce che hanno preso un computer, poniamo un Amiga, l'hanno smontato, tolti i chip e tentano di realizzare un totale largamente superiore a quello che ragionevolmente (e lo sanno bene) potrebbero ricavare da una macchina intonsa.

E' facile fare due conti: so che potrei vendere diciamo a 100 Euro quell'Amiga completo con una certa dotazione di software e con i suoi manuali. Ma invece ne ricavo 20 pezzi (per non esagerare), che venduti a 15 euro cadauno (cosa vuoi che siano 15 Euro quando ti serve proprio quel particolare?), ne ricavo 300 di Euro e comunque 150 se ne vendo anche solo 10.

Ottenere 20 pezzi da un Amiga non è difficile se ci si mette di impegno, basta cominciare con l'involucro, due manuali, un joystick, l'alimentatore, un disco esterno, tre giochi in confezione originale, un set completo di 50 floppy usati con giochi a go-go, qualche rivista... E poi si comincia a smontare e via le ROM di kickstart (quante ne abbiamo viste su eBay?), la RAM, i chip Paula, etc...,

Spero veramente di non stare dando strane idee a qualcuno...

Ammetto che recuperare un set di chip Amiga potrebbe risolvere il problema del sistema che abbiamo in casa e che risulta mancante proprio di ciò che viene offerto, ma dal punto di vista del conservatore non vorrei mai vedere un sistema funzionante smembrato per bramosia di qualche Euro!

Inoltre c'è un effetto collaterale spiacevole: quando tizio ha venduto i pezzi più appetibili (e più cari) è facile che non si tenga in casa un case o una tastiera semi cannibalizzata, con il risultato che la rottamazione toglie definitivamente dalla circolazione quello che poteva essere un degno partecipante ad una collezione.

Credo in ultima analisi che la lievitazione dei prezzi di qualche reperto, dichiaratamente non funzionante, sia dovuta anche a questa pratica.

Che fare?

Non ho soluzioni. Potremmo tutti non comprare pezzi singoli obbligando chi vende a quotare "vuoto per pieno" l'intera macchina. Potremmo stabilire ad esempio una quotazione massima per il sistema non funzionante, qualunque esso sia (nei limiti della possibilità di effettuare una classificazione omogenea).

Tutte cose facili a dirsi ma di difficile realizzazione.

La cosa migliore però è il comportamento etico di ciascuno di noi e che credo debba far parte del bagaglio dell'appassionato di retro computer, così come di qualsiasi altro genere di collezionismo.

(=)

Silicon Graphics - INDY

di Mario Raspanti

Incipit

E' ormai passato molto tempo dall'articolo sulla Silicon Graphics Indigo, apparso su Jurassic News No.27. Oggi vediamo la macchina che a questo modello ha fatto seguito, e che insieme alla successiva O2 ha forse rappresentato il modello più popolare di casa Silicon Graphics (o SGI).

Il contesto

Siamo nel 1993. Presidente del Consiglio è Carlo Azeglio Ciampi. Alla Casa Bianca si insedia Bill Clinton. Al cinema esce Jurassic Park, cui la stessa SGI ha pesantemente contribuito e che apparirà nella sua pubblicità dell'epoca (Building a better dinosaur). La Intel ha appena presentato il nuovo processore Pentium 60 MHz, e i PC di fascia alta già in circolazione sono dei 486DX-33 o 486DX2-66, di solito con 1 o 2 MBytes di RAM. Il sistema operativo corrente è Windows 3.1, ma i giochi non sono ancora fatti e nel dubbio la stessa Microsoft gli affianca anche la prima versione di Windows NT, anch'essa chiamata 3.1, e la versione 2.1 di OS/2, lenta e pesante (OS/2 Warp è ancora lontano). Nel campo professionale ci sono le workstation DEC 3000, la IBM RS/6000 e la SUN SparcStation 10.

In questo contesto la Silicon Graphics sostituisce la "piccola" di casa, ossia la sua prima e finora unica macchina desktop, di-

videndola in due modelli:

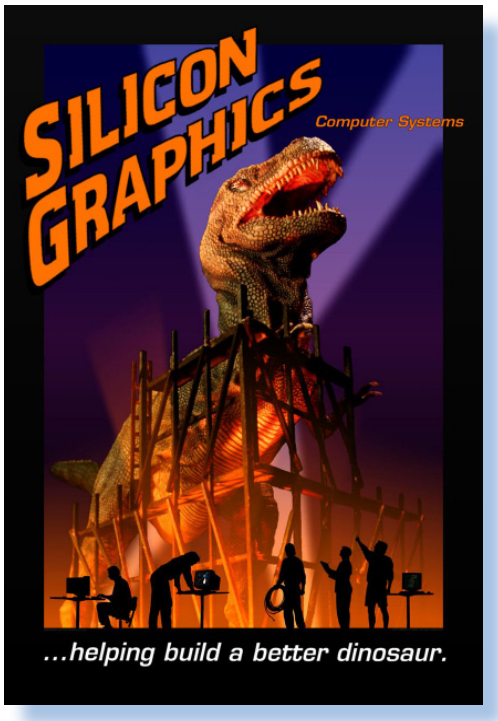
- *La Indigo2, che della Indigo eredita il nome e la destinazione professionale, è un enorme desktop/deskside (può essere montato anche in verticale con due appositi sostegni, forniti di serie) da circa 50 x 50 x 15 cm e 20 Kg di peso, di un singolare colore verde/blu. La macchina è robustissima, performante e molto espandibile; in seguito conoscerà una seconda giovinezza con una seconda serie modificata, la Indigo2 Impact, con nuovi processori e nuove schede grafiche, e con la carrozzeria colore fucsia (!).*
- *La Indy, più snella, leggera ed economica, è meno espandibile ma più versatile e già dotata di serie di una impressionante quantità di interfacce e connessioni.*

La macchina offre di serie un processore RISC ad almeno 100 MHz, grafica ad alta risoluzione con 16 milioni di colori ed accelerazione 3D, quattro canali audio, digitalizzatori audio e video incorporati, due diversi standard Ethernet, ISDN, due porte seriali, una SCSI, una parallela ed un sistema operativo Unix-based con una interfaccia grafica evoluta.

Mentre è facile a posteriori comprendere la destinazione commerciale della Indigo2, la Indy occupa una posizione più sfumata. A chi era indirizzata questa macchina, più brillante della sorella ma tutto sommato meno performante e ancor meno espandibile, ad un prezzo che (seppur basso in termini SGI) la separava comunque nettamente dai PC di fascia alta? Probabilmente la Indy è stata in qualche modo una macchina-esca, pensata per attirare verso i prodotti SGI una clientela più vasta ed eterogenea di quella già consolidata. Senza dubbio, il grande successo che ebbe (all'estero) nei computer center delle università ed affini avvicinò al marchio Silicon Graphics un vasto pubblico di studenti e futuri professionisti e, come vedremo, creò un vasto movimento di interesse intorno alla macchina.

La macchina

La Indy si presenta quasi come un tipico desktop dell'epoca, ossia un parallelepipedo piatto a sviluppo orizzontale, di circa 40x35 cm ed alto circa 8 cm, concepito per stare sotto il monitor (Figura 2). Non proprio tipico, perché la macchina ha diverse peculiarità di design: innanzitutto il colore è un blu acceso, movimentato da un particolare effetto granito, mentre tastiera ed accessori hanno la stessa finitura ma in colore grigio. Un gradino attraversa obliquamente la macchina, movimentandone la forma. Di una estrema pulizia il design del frontale, dove non troviamo nessuna apertura ma solo il tasto di accensione con la relativa spia (Figura_3), un minuscolo tasto di reset nascosto in una scanalatura e due tasti triangolari per la regolazione del volume (!) dell'altoparlante interno. Non è previsto un alloggiamento interno per un CD o una unità a nastro magnetico, che vanno collegati esternamente; è invece possibile montare un floppy drive interno, cui si accede dal fianco destro della macchina (Vedi la foto di apertura) e che deve avere l'espulsione sof-



tware (à la macintosh), perché non è previsto il solito pulsante. Come si vede, non è l'originalità quella che manca.

La Indy si apre sfilando verso l'avanti l'intera copertura plastica (detta skin in casa SGI).

Il lato sinistro (vedi Figura_4) è interamente occupato dall'alimentatore, montato a sbalzo e trattenuto da una sola vite e da un incastro. Sul frontale troviamo in posizione centrale la CPU, montata su una propria scheda e provvista di una impressionante (per l'epoca) alettatura di raffreddamento, e a destra i dischi, rivolti come si è detto verso il fianco destro. La parte posteriore è occupata dalla scheda madre, sulla quale è montata orizzontalmente la scheda video. Resta spazio per altre schede, da montare superiormente a quest'ultima.

L'interno è molto pulito e le parti principali sono tutte rapidamente accessibili. Avendo i pezzi di ricambio, una Indy guasta può essere fatta ripartire in pochi secondi e con un normale cacciavite.

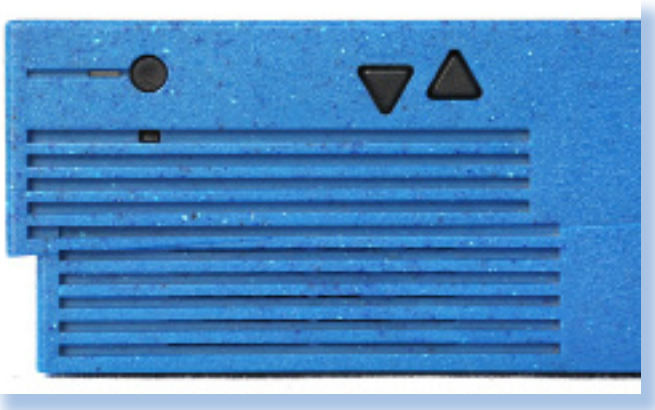
La costruzione sembra invece decisamente gracile in confronto alle altre macchine della stessa casa, tutte costruite come un carro armato. In particolare il pesante alimentatore sembra decisamente posticcio, ma devo ammettere che non si è mai staccato.

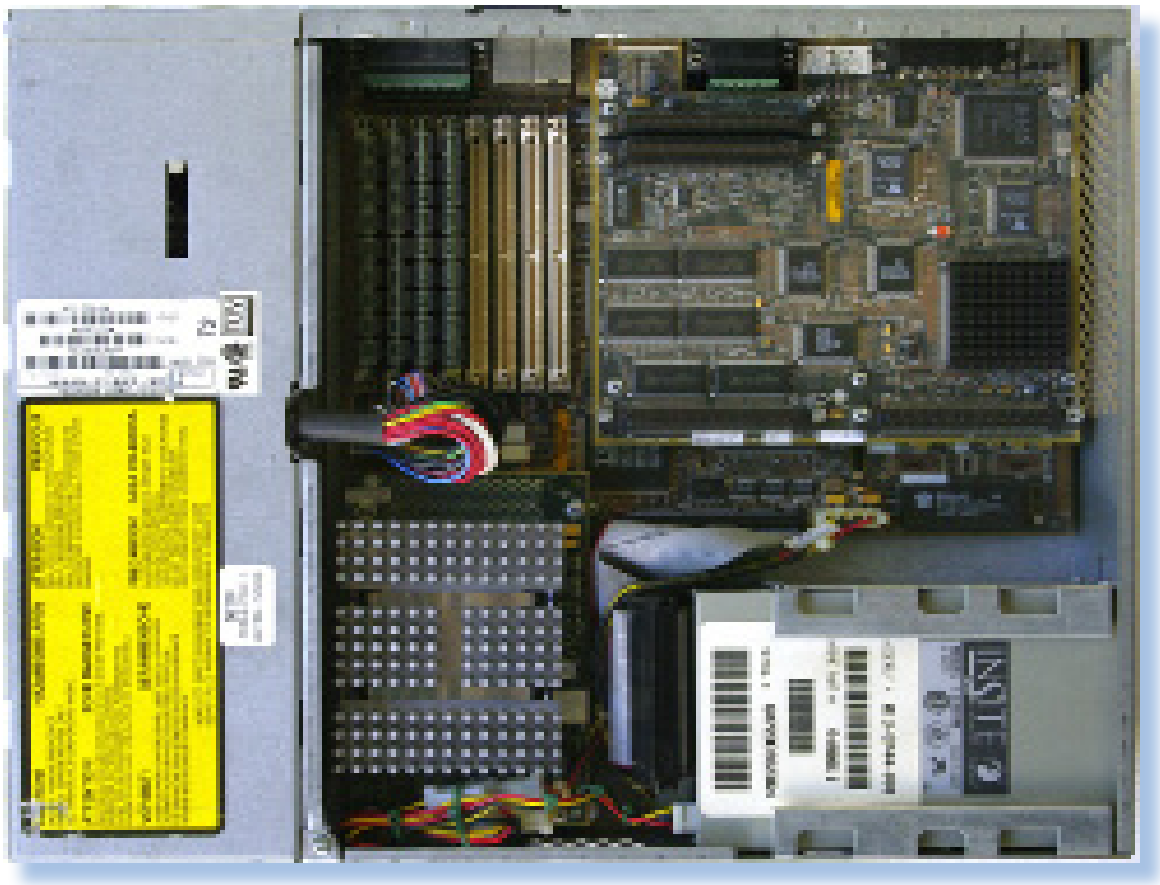
Come al solito in casa Silicon la stessa macchina era disponibile in varie soluzioni e configurazioni. Per la CPU erano disponibili non meno di dodici CPU diverse, tutti processori MIPS RISC con velocità da 100 a 200 MHz, qui elencati in ordine crescente di potenza:

R4000PC-100
R4000SC-100
R4600PC-100 e 133
R4600SC-133
R4400SC-100, 150, 175 e 200
R5000PC-150
R5000SC-150 e 180

La R4600 venne introdotta dopo la R4400, ma resta meno performante di quest'ultima. Le versioni PC sono prive di una memoria cache interna e, a parità di clock, sono sensibilmente più lente delle versioni SC. La RAM (Figura_5) era costituita da comuni moduli SIMM tipo PS/2 con parità, 36 bit su 72 pin, usati anche sulla Indigo R4000 e sulla Indigo2 e con una capacità fino a 32 MBytes per modulo. Per la particolare architettura i moduli devono essere inseriti in gruppi di quattro per volta.

Le opzioni grafiche erano invece tre e, con le loro prestazioni tutto sommato vicine a quelle della precedente Indigo, sembrano stra-



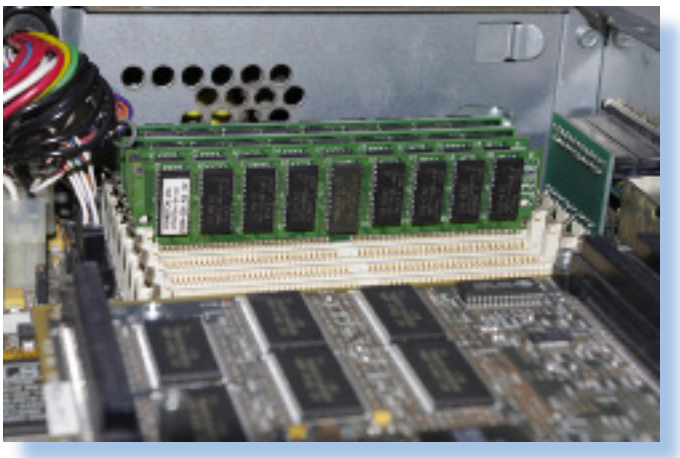


namente sotto tono per una macchina così innovativa:

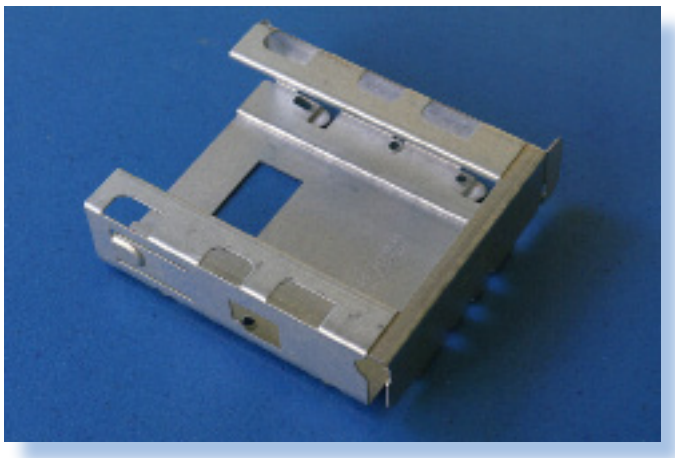
XL8 8 bpp, 1280x1024 oppure 1024x768, 256 colori
XL24 24 bpp, 1280x1024 oppure 1024x768, 16 milioni di colori
XZ 24 bpp, 1280x1024 oppure 1024x768, 16 milioni di colori e accelerazione 3D.

Nel complesso le prestazioni sono ottime in 2D, ma piuttosto anemiche in 3D. Basti dire che la scheda XZ, l'unica con accelerazione 3D hardware, veniva sconsigliata sulle Indy R5000 perché in questo caso la CPU era più veloce della GPU a gestire le trasformazioni, e la presenza di una XZ rallentava la macchina. Naturalmente chi aveva bisogno di prestazioni veramente superiori veniva indirizzato verso un sistema Indigo2.

Tutte le schede gestivano come abbiamo visto due risoluzioni fisse: 1024x768 e 1280x1024. La risoluzione poteva essere cambiata in software o automaticamente via hardware, tramite un contatto nel connettore video che, per i monitor SGI, riconosceva la periferica video in uso.



Lo spazio per i dischi (Figura_6) è molto ridotto e la Indy accetta al massimo due unità a mezza altezza, ognuna inserite in una propria slitta a scatto in metallo molto razionale (Figura_7) e montate sovrapposte. L'unità inferiore di solito contiene l'hard disk; nella superiore può essere alloggiato un floppy disk o, come nella macchina qui raffigurata, un floptical da 21 MBytes (vedi Jurassic News No. 33). Tutte le unità sono naturalmente SCSI. Quando si montavano due hard disk veniva consigliato di limitarsi ad unità da 10.000 RPM piuttosto che da 15.000 RPM, comuni sulle workstation high-end, per evitare problemi di surriscaldamento. Nel caso il bay superiore restasse inutilizzato la SGI raccomandava di lasciare comunque inserita la relativa slitta metallica, che contribuiva a sopportare il peso del monitor sovrastante.



Le interfacce

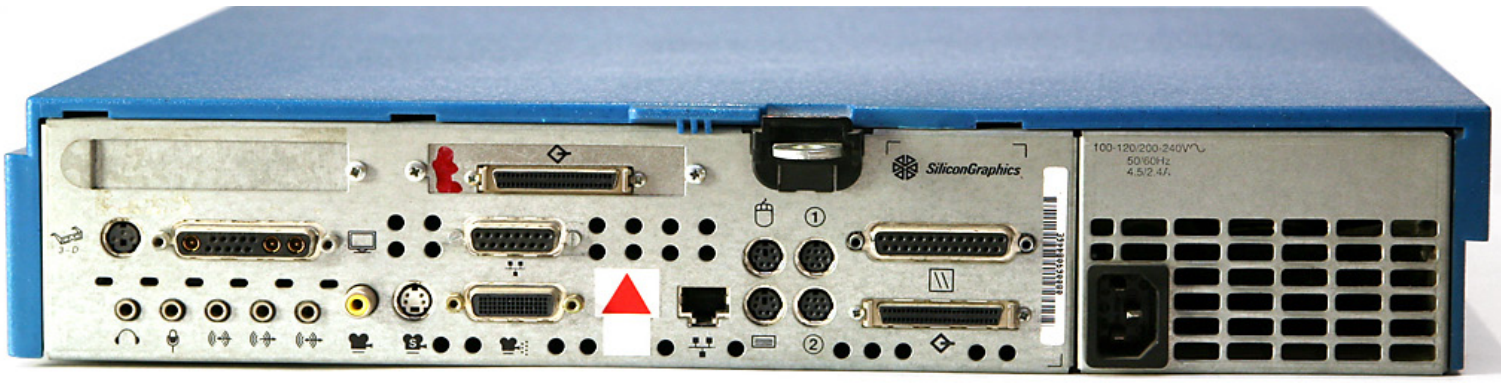
Qui la Indy si rivela veramente difficile da battere, e il pannello posteriore (Figura_8) è praticamente una distesa di connettori. Da sinistra, in basso troviamo una fila di cinque prese jack stereo da 3.5 mm: nell'ordine troviamo la presa di cuffia, l'entrata microfono, l'entrata di linea, l'uscita di linea e l'in/out digitale, riconfigurabile via software in modo da ottenere quattro canali. Sopra a questi la grande presa 13W3 per il monitor (RGB, sync-on-green) e la piccola mini-DIN per la sincronizzazione degli occhiali stereo (stiamo parlando di vent'anni fa!).

Ricordiamo che in casa SGI la presa 13W3 ha un cablaggio leggermente diverso da modello a modello (e anche leggermente diverso da SUN, IBM ecc...) e che un adattatore VGA, necessario per connettere un monitor odierno, non è detto che funzioni al primo colpo.

Procedendo verso destra troviamo in successione tre ingressi video: un connettore RCA, un ingresso S-Video ed una porta video proprietaria, dedicata ad una speciale IndyCam (più oltre) e vagamente somigliante ad un DVI.

Sopra a questa abbiamo la presa D a 15 poli per una rete thick ethernet. Più in alto due tappi coprono le finestre dedicate ad eventuali schede accessorie.

Procedendo verso destra, una eti-



chetta con un triangolo rosso copre una presa ISDN, che meccanicamente è uguale alla successiva presa di rete RJ45, onde evitare costosi errori nell'inserire i cavi. L'interfaccia di rete è una sola, nel senso che il connettore thick ethernet e il connettore RJ sono in alternativa – o si usa l' uno o l' altro.

Seguono le prese tastiera e mouse, che per la prima volta in casa SGI sono unità standard PS/2, e due porte seriali su connettore mini-DIN, come allora usava anche Apple. Infine un connettore SCSI per unità esterne e, sopra a questo, una porta parallela su connettore D a 25 poli. All'estremità destra infine troviamo l'alimentatore con la sua presa per il cavo di rete.

Serve altro?

Eccome!

Sempre nel campo hardware, la Indy ha ancora un paio di ciliegine da mettere sul gelato.

La prima è la IndyCam (Figura_9), che potrebbe essere considerata la prima WebCam della storia: una piccola videocamera, color grigio-granito come tutti gli accessori Indy, destinata al teleconferencing (...nel 1993, ricordiamolo) ma anche ampiamente usata per la produzione di piccoli (e ingombranti) video home-made. Va ricordato che la IndyCam può essere collegata soltanto ad una Indy, ma il digitalizzatore audio/video (di serie!) di quest'ultima funziona perfettamente anche con gli altri ingressi, CVBS o S-Video, a cui può essere collegata qualunque sorgente video. Unico problema, produce files non compressi e quindi relativamente ingombranti, che erano una costante dannazione degli amministratori nei centri di calcolo. Se proprio si voleva andare sul pesante erano disponibili due diverse schede opzionali, da montare in piggyback sulla scheda video, per la compressione video in tempo reale.

La Indy naturalmente poteva montare anche tutta una gamma di accessori generici, come il mouse tridimensionale Spaceball (Figura_10), array di pulsanti e di manopole, tavolette grafiche ecc.

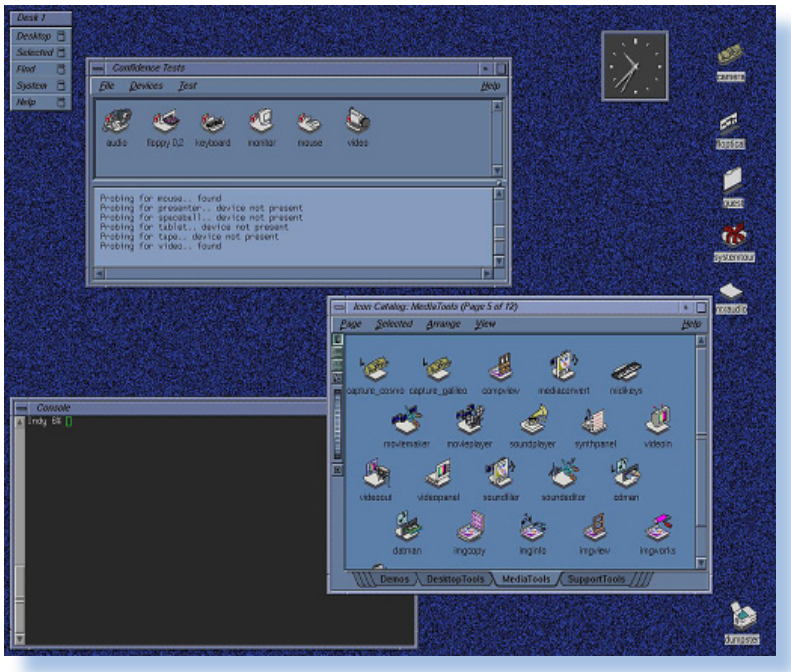
Infine il sacro Graal dei possessori di Indy, che purtroppo non siamo in grado di mostrarvi, è la scheda Ultra-64. Questa, che andava montata anch'essa sopra la scheda video, è la parte hardware del sistema di sviluppo per Nintendo 64.

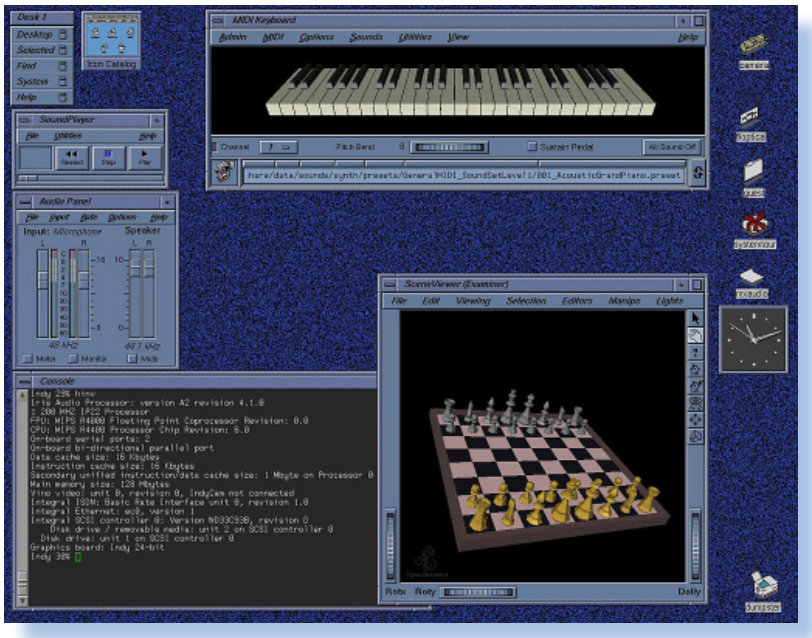
E' naturale che i videogiochi per piccole consolle venissero sviluppati su altri sistemi più potenti e più veloci; la stessa cosa accade anche adesso. Dove è nato SuperMario 64? Adesso lo sapete!

Altri rarissimi accessori (anche se non fanno parte dell'hardware in senso stretto) sono la borsa originale per il trasporto della Indy (mentre ricordiamo che non è mai esistito un portatile Silicon Graphics; se ne intravede uno fasullo nel film Twister) e una inverosimile valigetta 24 ore a forma di Indy, da cui si distingue solo per la presenza del manico, prodotta -pare- in soli 50 esemplari usando gli stampi originali.

Il software

E' ovviamente Irix, versione Silicon Graphics di Unix, che come la maggior parte dei sistemi Unix ha prestazioni elevatissime e non si pianta mai. Ne sono esistite numerose versioni, alcune molto diffuse ed alcune rimaste allo stadio di sigla; l'esemplare in oggetto monta una versione 6.2, ma la Indy è in grado di montare fino alla maggior parte delle versioni 6.5.

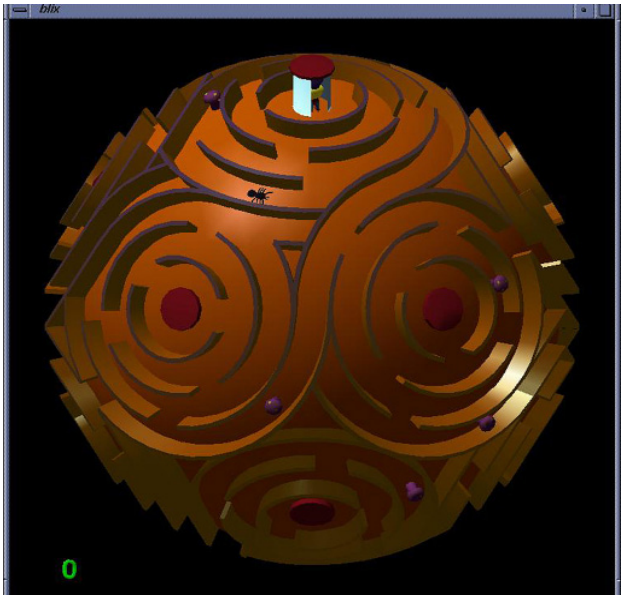




All'accensione la Indy emette un bip, seguito da un breve stacchetto musicale (una caratteristica di molte macchine Silicon Graphics, ognuna con un motivetto diverso) e dalla comparsa della schermata di welcome. Quando la macchina ha completato la diagnostica la spia di accensione da rossa diventa verde e la Indy offre, per qualche secondo, la possibilità di interromperla per richiedere una diagnostica più approfondita (una cosa veramente lunga) o effettuare operazioni di sistema, come un upgrade del sistema operativo. Dopodiché carica Irix e presenta la schermata di login.

L'estetica della GUI Motif (Figura_11_12) è sobria e persino austera in confronto alle attuali versioni Windows e Linux, ma in queste macchine ogni dettaglio, dai font ai colori, era attentamente studiato per massimizzare la produttività e per non stancare l'utente durante lunghe ore davanti al monitor, non certo per impressionare gli amici. La Indy in esame è una R4400SC-200MHz con una scheda grafica XL24, ossia una configurazione medio-alta per una Indy, ma non certo paragonabile ad una macchina attuale. Eppure la responsività della GUI è molto maggiore: al tocco del mouse le finestre si aprono subito, i programmi partono immediatamente, il feedback è istantaneo. Ci si dimentica presto della clessidra che ci fa compagnia nel nostro solito PC, e soprattutto non si deve riavviare mai. In senso letterale: la Silicon Graphics raccomandava di lasciare sempre accese le sue macchine, e del resto l'unico motivo per spegnerle era un guasto hardware.

Sotto la GUI c'è, come abbiamo detto, Unix, con tutto quel che comporta ma anche con alcune estensioni. Una esclusività Irix è un



comodo Toolchest, in alto a sinistra, con i principali comandi di sistema, tra cui una serie di approfonditi Confidence tests per tutte le periferiche presenti e un Icon catalog che raccoglie, ordinatamente per tabs, decine e decine di applicazioni. Possiamo ricordare anche alcuni utili comandi aggiuntivi da terminale, come hinv per l'inventario hardware o chkconfig per avviare/fermare individualmente i daemons. Vale per la Indy tutto quello che nel numero 27 è stato detto sul software della Indigo, a cui vi rimando.

A livello applicativo devo ricordare l'esistenza, sempre di serie, di moltissimo software multimediale, convertitori di files, generatori musicali, di visualizzatori 3D ecc. (Figura_13); ma anche di software di rete e di teleconferencing, quando questa parola non esisteva neppure. Successivamente è stato reso disponibile anche un emulatore Windows 95, di utilità pratica forse limitata ma comunque impressionante (anche perché invece un emulatore Indy sotto Windows 95 non l'ho mai visto).

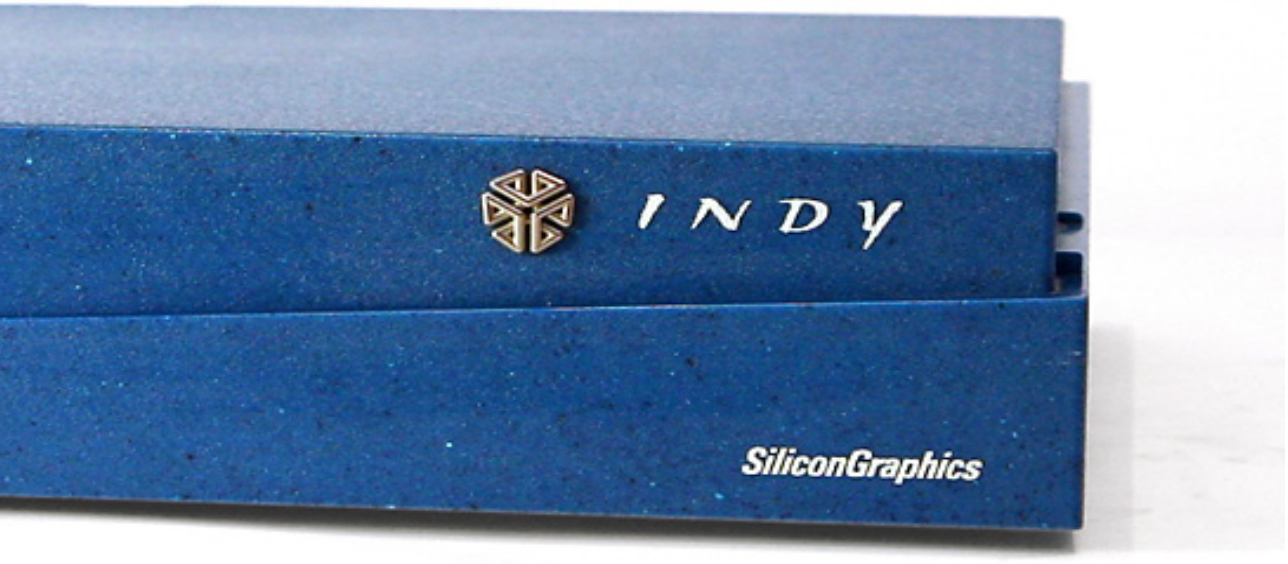
Sul fronte software bisogna tener conto, da un lato, della disponibilità di software applicativo di altissimo livello (un esempio per tutti è Wavefront, che nel 1995 verrà acquistato direttamente dalla SGI e da cui nel 1998 nascerà Maya); dall'altro della disponibilità, tipica del mondo Unix, di software gratuito. La velocità della macchina, la qualità del sof-

tware, la facilità di interfacciamento hanno infatti determinato, alla sua epoca, lo sviluppo di una attivissima comunità di utenti Silicon Graphics e di un clima culturale molto dinamico e collaborativo, con un attivo scambio di idee e di software che, seppur in chiave minore, continua ancora oggi. La stessa Silicon Graphics incoraggiava questo atteggiamento mediante un contest chiamato IndyZone articolato in cinque settori: solo player games, team games, single user tools, multiple user tools, e best Indy software. Ogni categoria prevedeva una macchina SGI in premio al vincitore e la diffusione del software mediante i CD IndyZone (pare ne siano usciti cinque). Altri CD, contenenti demos di programmi commerciali, utilities e freeware, venivano diffusi dalla Silicon Graphics tra la community degli utenti con il nome di Hot Mix. Tutti questi sono ancora attivamente ricercati.

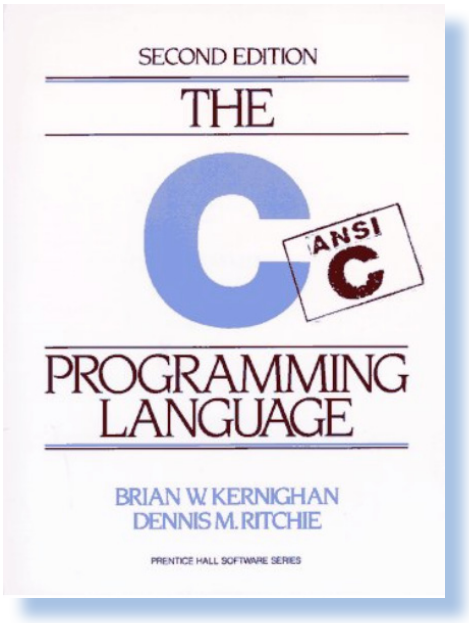
Naturalmente sarebbe inutile cercare Assassin Creed tra il software home-made di vent'anni fa, ma il livello è comunque decisamente brillante. Un esempio è visibile nell'ultima immagine (Figura_14), che raffigura il vincitore di IndyZone 1994 nella categoria best solo player game: Blix, una specie di PacMan mappato su una sfera, dove oltre a guidare l'avatar dovete, man mano che questo si muove, girare anche la sfera in modo da non perderlo di vista.

Sembra semplice, vero? Provate a giocarci...

(=)



Lo sviluppo del linguaggio C



di Salvatore Macomer

Introduzione.

Il linguaggio C è diffuso come forse nessun altro linguaggio di programmazione. Gli elementi principali della sua nascita sono noti: progettato presso i Bell's Laboratories da B. W. Kernighan e Dennis M. Ritchie come evoluzione del linguaggio B, a sua volta derivato dal BCPL, per lo sviluppo del sistema operativo Unix, si impose ben presto sia per le sue caratteristiche che per la sua portabilità.

Questo articolo rivisita i passi essenziali della sua nascita e soprattutto della sua evoluzione ed è basato in particolare su un articolo scritto da Ritchie in persona nel 2003 per la Lucent Technology, che nel frattempo ha rilevato i Bell's Laboratories.

Uno sguardo di insieme.

Il C nasce negli anni 1969-1970 in parallelo all'evoluzione del sistema operativo Unix. Il suo riconoscimento ufficiale si ebbe nel 1977 e nel 1979 quando la portabilità di Unix fu definitivamente dimostrata. Finalmente alla metà degli anni '80 il linguaggio venne standardizzato dal comitato ANSI X3J11, quando ormai esistevano compilatori C per quasi tutte le piattaforme, home computer compresi.

Nel 2005 è stato eletto al vertice della classifica dei linguaggi di programmazione più diffusi al mondo, scettro che probabilmente detiene ancora, mentre di fatto tutti o quasi i linguaggi "moderni", si basano sulle sue idee, pur evolvendo l'ambiente di produzione del software verso una concezione più consona alle idee della programmazione ad

oggetti.

Il testo "principe" che ne descrive le caratteristiche è il famoso "libro bianco" scritto dai due autori del linguaggio e che ha come titolo "The C Programming Language". Viene chiamato "libro bianco" perché nella prima edizione, cosa rimasta invariata nelle successive, la copertina è semplicemente bianca con una grande C azzurra che spicca nel titolo.

Premesse storiche.

Alla fine degli anni '60 i Bell Telephon Laboratories decisero di uscire dalla join venture che vedeva come altri attori il MIT e la General Electric. Il consorzio aveva come obiettivo la realizzazione di un sistema operativo multitasking denominato Multics, progettato per il mainframe di General Electric GE 645. La realizzazione si stava però rivelando irta di ostacoli e la prospettiva di un rilascio definitivo si allontanava sempre di più, oltre che presentarsi sempre più costosa. I vertici dei Bell Labs decisero di esplorare un'altra possibilità e così venne incaricato Ken Thomson di lavorare sullo stesso sistema per vedere se era possibile una strada alternativa più semplice e ovviamente meno costosa da utilizzare su hardware diverso (il PDP in particolare). Thomson inserì nel nuovo progetto le idee migliori che erano state pensate per Multics: il file system ad albero, la struttura di descrizione dei task, l'accesso generalizzato ai device e la presenza di un ambiente di comando a livello user (la shell) potenzialmente interoperabile. Vennero abbandonate altre caratteristiche di Multics la cui realizzazione era complicata, come ad esempio l'accesso alla memoria e ai file con le medesime funzionalità. Inoltre venne scartata la possibilità di scrivere parte del codice con il PL/I scegliendo altri linguaggi come il BCPL e l'assembler. Si perse però la possibilità di disporre di un sorgente strutturato e leggibile, cosa che PL/I permetteva e questo ha influenzato nel primo periodo la questione della portabilità del sistema operativo in altre piattaforme, questione che venne deciso di accantonare e riprendere successivamente.

Il DEC PDP-7 nel 1968 era una macchina con architettura a 18 bit con 8K di memoria e nessun programma ad alto livello. La programmazione di Unix su questo sistema doveva avvalersi dell'uso di un cross-assembler chiamato GEMAP sul GE-635 che generava un nastro di carta eseguibile poi su un PDP-7. Questo metodo venne usato

Fig. 1 -

Ken Thompson (a sinistra) e Dennis Ritchie (a destra). [Immagine da Wikipedia]

da Thompson per creare il primi nucleo del sistema e una shell minimale con pochi comandi come cp, rm cat, etc... Messa a punto questa parte lo sviluppo poté continuare in autonomia sullo stesso PDP-7 senza coinvolgere la macchina della General Electric.

I predecessori.

Thompson cominciò sviluppando un macro-assembler per il PDP-7, minimale e senza alcuna possibilità di utilizzare librerie, linker, loader, etc... Il file eseguibile, frutto della compilazione aveva un nome fisso: "a.out" e questo spiega perché a tutt'oggi si sia conservata questa convenzione sul nome dell'output per i compilatori Unix.

Nel 1969 un certo Doug Mcllroy aveva scritto un linguaggio ad alto livello per Multics, il linguaggio TransMoGrifiers (TMG), specificatamente adatto a scrivere compilatori. Con esso si era sviluppato il PL/I che girava sul GE 645. Da qui Thompson maturò l'idea che era necessaria una cosa analoga anche per il PDP-7. Provò a implementare una versione del FORTRAN, che abbandonò quasi subito, per dedicarsi al linguaggio che chiamò semplicemente B. Il B è semplicemente una versione del BCPL "strizzata" per farla digerire ai soli 8K di memoria della macchina Digital. Rispetto al successore (il C), manca della tipizzazione. La scelta del nome "B" non è certa: alcuni affermano che sia una contrazione di BCPL, altri dicono derivi dal nome BON, un'altro linguaggio scritto da Thompson per il GE, a sua volta probabilmente derivato dal nome della moglie di Thompson che si chiamava Bonnie; altri ancora non hanno dubbi nell'affermare che BON sia una sorta di formula magica della tradizione mormone, comunità alla quale Thompson apparteneva.

Tutti questi linguaggi, compreso il loro successore C, appartengono alla classe dei linguaggi cosi detti "procedurali", i cui capostipiti sono FORTRAN e ALGOL 60, ma rispetto a questi ultimi sono molto più "machine oriented", cioè si avvicinano di più alla struttura hardware dei moderni sistemi e sono abbastanza semplici da permetterne la portabilità su architetture diverse con relativo poco sforzo.

Pur differendo abbastanza significativamente nelle sintassi, la loro macro struttura è comune: una parte dichiarativa è seguita dall'implementazione di funzioni (o procedure). Il BCPL permette la dichiarazione ricorsiva delle procedure (una procedura può contenerne altre), particolarità che è stata rimossa nel B e C rendendo il compilatore più semplice. L'aspetto della sintassi è in fondo minimale: le vere innovazioni riguardano le strutture dati interne ad ogni singolo linguaggio.

Da notare che a volte certe sintassi vengono abbandonate ma si ripresentano dopo una o due generazioni. Ad esempio il BCPL accetta il commento a fine linea con i caratteri //, il B vuole l'inclusione dei commenti nelle coppie / ... */, mentre la coppia // è stata reintrodotta con il C++...*

BCPL e B non hanno il concetto di tipo di dato: ragionano con un unico tipo che viene chiamato "cella" ed è rappresentata da una sequenza fissa di bit. Sommare due variabili significa sommare il contenuto delle due celle, Ogni cella può rappresentare l'indirizzo di un'altra cella e il BCPL usa la notazione !p per indicare il valore della cella puntata da p, mentre il B introduce la notazione

**p, che poi sarà la stessa adottata dal C.*

Per l'accesso agli elementi di un array V, il BCPL usa V!i, mentre il B la più famigliare (almeno per noi) V[i].

Anche nella gestione delle stringhe il BCPL e il B differiscono: nel BCPL il primo carattere deve contenere la lunghezza della stringa (da notare che questa convenzione sarà usata anche dal Pascal), mentre il B e i suoi derivati non prevedono nulla di simile: deve essere il contesto che stabilisce la tipologia di dato.

Durante la scrittura del B, Thompson era assillato dalla limitazione di memoria e fece di tutto per rendere il linguaggio più compatto possibile. Ad esempio a lui di deve la reintroduzione, che prese dall'Algol 68, dell'operatore di assegnazione con incremento += e l'invenzione nuova dell'operatore di auto-incremento ++ e di auto-decremento —, compresa l'idea che la posizione di questo operatore prima o dopo il nome della variabile dovesse corrispondere alla sequenza temporale dell'operazione stessa (incremento prima o dopo l'assegnazione). Alcune fonti dichiarano che l'idea degli operatori di auto-incremento derivi dal fatto che il PDP-11 aveva una funzione nativa per queste operazioni e che Thompson semplicemente le abbia tradotte nel suo B. Questo è impossibile: quando Thompson scrisse il linguaggio B, il PDP-11 non era ancora in commercio! E' invece probabile che Thompson si sia ispirato ad una analoga funzionalità (ma non esattamente un auto-incremento), presente nel PDP-7.

Il linguaggio B è pensato non per generare codice macchina direttamente eseguibile, ma quello che viene chiamato "threaded code" che sarebbe una sequenza di indirizzi di routine (rum-time) che puntano ad istruzioni elementari eseguiti da una B-Machine che in pratica è una macchina a stack. Thomson continuò nel suo lavoro di sviluppo del B ma il PDP-7 cui disponeva era una macchina troppo limitata e gli 8K celle di memoria riuscivano a malapena a contenere l'interprete. Il risultato fu che programmi scritti in B furono pochissimi, se non il B stesso e l'idea di scrivere parte del sistema operativo UNIX con questo linguaggio si rivelò una strada non percorribile. Stessa sorte per le altre idee di sviluppo di compilatori per i maggiori linguaggi dell'epoca: FORTRAN, PL/I e Algol 68. Una delle prime routine messe a punto fu il programma "dc", che è un calcolatore a precisione variabile che fa parte ormai della dotazione standard di Unix e dei suoi derivati.

La situazione si sbloccò nel 1979 quando ai

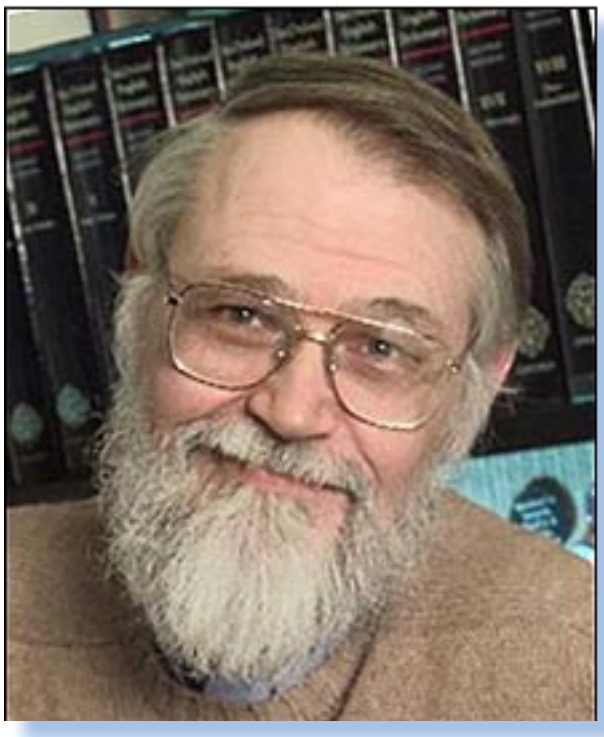


Fig. 2 - Brian Kernighan [Immagine da Old-computers.com]

Fig. 3 -

Dennis M. Ritchie. [Immagine da Old-computers.com]

Bell Lab arrivò il primo PDP-11 della Digital. Per quanto non fosse ancora dotato di memoria di massa, il nuovo sistema disponeva di 24K di memoria e fu ragionevolmente facile per il team di Thomson, fare un porting del codice B e sviluppare a partire da esso le prime routine di utilità per lo Unix. Nel 1971 Steve Johnson scrisse la prima versione del suo YACC (Yet Another Compiler Compiler) usando l'assembler del PDP-11 e molte routine nel frattempo sviluppate in codice B.

I problemi del linguaggio B.

Il PDP-11 introduceva una variazione importante nell'architettura: la macchina era byte-oriented rispetto alla precedente che era word-oriented. Ne seguiva una perdita di ottimizzazione volendo trasportare il codice B senza interventi significativi sulla struttura del run-time, senza contare che queste modifiche annullavano per buona parte tutto il lavoro di ottimizzazione in termini di occupazione di memoria portate avanti da Thompson.

Il secondo problema era che l'architettura PDP non prevedeva il supporto nativo per l'aritmetica floating-point. Se per il GE sotto BCPL e il PDP-7 sotto B era stato possibile intervenire con una simulazione ad-hoc, la cosa era molto meno performante se adattata ad una macchina byte-oriented. Infatti il vantaggio dell'architettura word-oriented era quello di poter "accomodare" una variabile in virgola mobile in una word, cosa impossibile in un byte! Coinvolgere più byte nella definizione di una variabile FP è possibile ovviamente, ma le prestazioni di calcolo scendono. L'ultimo problema era per il B una progettazione della gestione dei pointer che non poteva trasferirsi paro-paro sull'architettura del PDP-11.

Nasce il linguaggio C.

Considerati i problemi descritti, nel 1971 Ritchie cominciò a riscrivere il B introducendo la gestione a caratteri delle stringhe e la modifica delle altre strutture dati per permettere la realizzazione di un vero compilatore che generasse dal sorgente direttamente codice macchina per il PDP-11 e anche per il GE465. Questa nuova versione

del B venne chiamata NB (che starebbe per New B). Questo NewB durò pochissimo ma fu un passaggio importante perché introdusse i tipi di dati int e char e gli array degli stessi secondo la sintassi mutuata dal BCPL e dal B:

```
int i, j;
char c, d;
int iarray[10];
int ipointer[];
char carray[10];
char cpointer[];
```

Inoltre nella dichiarazione degli array di puntatori la memoria non veniva allocata, altra innovazione concettuale che poi il C renderà sua con una modifica semantica di convertire a puntatore l'indirizzo del primo elemento di un array quando viene dichiarato all'interno di una struttura dati.

Ad esempio la struttura di una directory di Unix può essere rappresentata da un record del tipo:

```
struct {
int inumber;
char name[14];
};
```

Il compilatore trasformerà la dichiarazione "char name[14]" nel puntatore al primo carattere del nome tutte le volte che il nome dell'array viene incontrato nel codice.

Questa idea, messa alla prova con la riscrittura di codice sviluppato in B, si rivelò promettente, incoraggiando lo sviluppo del nuovo linguaggio, evoluzione dei due precedenti, che venne chiamato C, come logica conseguenza del suo predecessore principale.

Nel 1973 la definizione del C era completa e si decise di dedicare l'estate di quell'anno alla riscrittura del kernel dello UNIX. Lo stesso esperimento lo aveva tentato Thompson l'anno prima ma mancava ancora la definizione delle strutture nel nuovo linguaggio e la sfida era troppo ardua.

Intanto ai Bell Lab erano arrivati nuovi calcolatori: un Honeywell 635 e un IBM 360/370 sui quali si decise di realizzare la portabilità del sistema operativo con completo successo. Un altro sistema, leggermente più ostico per via della sua architettura, fu l'Interdata 8/32. Il progetto fu portato a termine ma servì anche per inserire alcune minime ma essenziali modifiche alla definizione del linguaggio.

La strada era tracciata! Dal 1973 al 1980 il linguaggio subì dei piccoli ritocchi nella definizione dei tipi e delle strutture che divennero molto simili a quelle che oggi chiamiamo "classi". In particolare furono introdotti i tipi "unsigned" che resero possibile la definizione delle operazioni su numeri privi di segno distinguendo la loro aritmetica da quella dei puntatori. Nel 1978 Kerninghan scrisse la prima versione del famoso libro "bianco" e la documentazione delle routine di servizio al sistema operativo che fanno parte ora del famoso "man", il manuale online di riferimento.

Una sfida che dovette vincere il linguaggio fu anche quella di convincere i programmatori all'utilizzo corretto degli statement che per la loro somiglianza con quelli del B e BCPL, tendevano a far scrivere codice "obsoleto" o comunque senza quelle caratteristiche innovative che il C permetteva raggiungendo la pulizia del sorgente e l'ottimizzazione del codice.

All'inizio degli anni '80 il linguaggio C varcò i confini accademici per rendersi disponibile sulle piattaforme home e personal e il suo uso si diffuse rapidamente. Nel 1983 si cominciò l'iter per la standardizzazione che arrivò con l'inclusione delle specifiche ANSI nel 1989 e ISO/IEC nel 1990. Il più significativo cambiamento apportato alla sintassi del linguaggio dal comitato X3J11 fu la specifica di come doveva essere dichiarata una funzione esterna. Prima della standardizzazione la sintassi era ad esempio:

```
double sin();
```

Mentre dopo divenne:

```
double sin(double);
```

Ciò rese più rigido il controllo degli argomenti per la funzione. Vennero aggiunti anche le specifiche "const" e "volatile".

I discendenti.

*Dopo la standardizzazione del linguaggio non fu più possibile chiamare "C" un fork dello stesso con aggiunte varie, per cui i tentativi meglio riusciti in questa direzione dovettero dichiarare un nuovo nome e quindi formalmente un linguaggio diverso da quello cui si ispiravano. Ad esempio il Cuncurrent C, l'Objective C, lo StarC (*C) e ovviamente l'arcinoto C++, definito da Stroustrup nel 1986, dal quale poi sono derivati ulteriori "figli". Fra i discendenti non dimentichiamo soluzioni più esoteriche come Modula 3 (1991) e Eiffel (1988).*

Fig. 4 - Dennis M. Ritchie (in piedi) al lavoro con un programmatore sul PDP-11.

Non sono mancate critiche alle idee e soprattutto a come esse sono state messe in pratica nella definizione del linguaggio. La più evidente e giustificata è quella che attribuisce al C la colpa della scrittura di un codice non proprio "leggibile", al punto che lo stesso compilatore abbisogna a volte di un "aiutino" per risolvere i casi dubbi. Ad esempio nella sequenza di statement:

```
int *fp();
int (*pf)();
```

che può essere tradotta in:

```
int *(*pfp)();
```

con perdita netta di leggibilità.

Dopo le critiche il perché di tanto (e per buona parte insperato) successo?

Sicuramente alla diffusione del linguaggio ha contribuito la diffusione del sistema operativo Unix e derivati (Linux compreso). Questa caratteristica da sola non spiega tutto! Un fattore sicuramente di successo è la sua relativa compattezza e facilità nello scrivere compilatori, dovuta alla vicinanza dei suoi tipi di dato e funzionalità con le strutture dei calcolatori moderni (una CPU con registri e accesso diretto alla memoria).

La sua stabilità nel tempo è un altro fattore. Il linguaggio è sufficientemente completo da non richiedere estensioni se non proprietarie e limitate a certi computer. La sua definizione può rimanere quindi quella iniziale e questo contribuisce a dare l'idea della stabilità del linguaggio. Nel tempo si sono consolidate le librerie di supporto che sono state raffinate passo dopo passo e hanno raggiunto ora, anche nel calcolo scientifico, una efficienza difficilmente superabile anche da idiomi specializzati come il FORTRAN.

Conclusione.

La storia del C menziona una serie di nomi che hanno contribuito a gettare le basi della sua definizione o ne hanno migliorato l'implementazione. Essi sono:

Ken Thompson che nel 1969 ha creato il linguaggio B;

Martin Richards che ha creato il BCPL;

Dennis Ritchie che ha trasformato il B nel NB prima e nel C dopo (1973);

Alan Snyder, Steve Johnson e Michael Lesk che nel periodo 1972-1977 hanno contribuito al miglioramento del linguaggio e lo hanno usato in progetti di notevole portata;

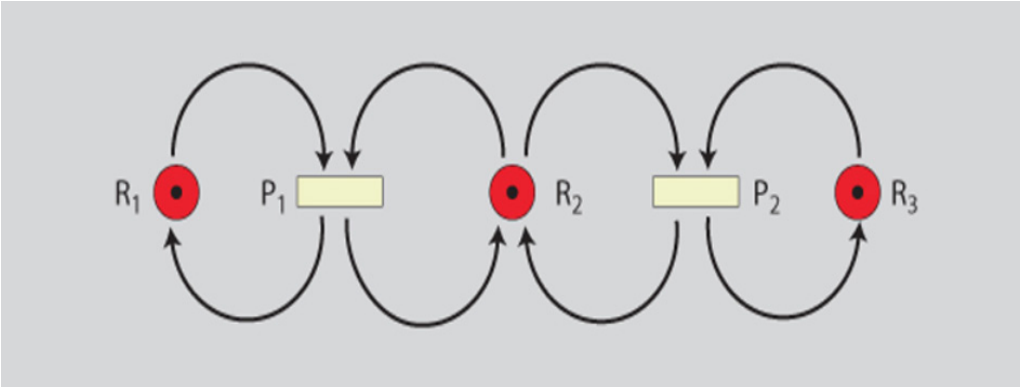
Brian Kerninghan che ha scritto assieme a Ritchie la prima versione del "libro bianco", vero standard del linguaggio;

I componenti del comitato X3J11 che hanno reso definitiva la versione dello standard ancora oggi in uso: Jim Brodie, Tom Plum, P.J. Plauger, Larry Rosler e Dave Prosser.

Reference

Dennis M. Ritchie "The Development of C Language" - Lucent Technologies 2003.

L'indecidibilità
(parte 2)



di Salvatore Macomer

La scorsa puntata abbiamo introdotto il concetto di indecidibilità dei sistemi formali come risultato del teorema di Gödel, ed esteso il discorso ai sistemi che riguardano i linguaggi per calcolatore dai quali si generano gli insiemi di tutti i programmi.

Ci aspettiamo che l'insieme di tutti i programmi scritti in un certo linguaggio, ma in generale qualsiasi linguaggio abbastanza generale va bene, contenga dei teoremi non dimostrabili. Abbiamo anche nella scorsa puntata indicata una via: il paradosso.

La domanda allora è: -"E' possibile scrivere un programma paradossale?" Sembra improbabile ma invece è proprio così!

Sgombriamo subito il campo dall'amletico dubbio del programmatore: i programmi che scrivi per calcolare il pi-greco piuttosto che il bilancio aziendale, non sono paradossali, sempre ammettendo che siano formalmente e logicamente corretti. Essi sono quei programmi "buoni" cui l'insieme di tutti i programmi è pieno. Tuttavia una categoria di programmatori è, diciamo, "a rischio" e sono coloro che scrivono compilatori. La scrittura di un compilatore per un certo linguaggio coinvolge, se ci pensate, un aspetto particolare: si tratta di scrivere un programma che interpreta un altro programma e ne produce un output (il codice oggetto compilato). Non ha importanza se il sorgente è, diciamo in BASIC e il compilatore è scritto in C: abbiamo detto nella scorsa puntata che la sintassi non è importante. Entrambi, cioè il codice sorgente e il programma del compilatore (e anche il codice oggetto e codice macchina lo sono) appartengono allo stesso insieme di oggetti: i programmi per computer.

Uno dei problemi della Computer Science è il problema dell'arresto. Di cosa si tratta?

E' possibile stabilire se dato un input un certo programma si ferma dopo un numero finito di passi oppure non si ferma mai?

Prendiamo ad esempio un classico ciclo for:

```
FOR N:=1 TO 10
        PRINT N
END
```

Va bene, questo si ferma. E' facile capirlo "al volo".

Quest'altro invece non lo fa:

```
FOR N:=1 TO 10
        PRINT N
        N := 1
END
```

Sono due programmi deducibili dal punto di vista della proprietà di arrestarsi o meno.

Se il codice risulta più complesso è necessario farsi aiutare dalla macchina stessa per avere la risposta. Perché non ci costruiamo un esaminatore di arresto? Daremmo in input a questo programma il nostro codice e questo esaminatore ci stampa il responso che potrebbe essere: "il programma si ferma dopo X passi", oppure "Il programma non si ferma mai".

Questo nostro esaminatore dovrebbe essere universale, cioè funzionare per qualsiasi codice in input e per qualsiasi condizione di partenza.

Prima di precipitarci a scrivere codice o semplicemente a cercare il come si potrebbe fare, ricordiamoci di Gödel: un simile programma non si può scrivere.

Panico...

Come possiamo affermare una cosa del genere?

Lo dimostreremo con una tecnica che viene chiamata "diagonalizzazione", che poi è la stessa tecnica che ha permesso a George Cantor di dimostrare che l'insieme dei numeri reali non è numerabile e quindi non esaurisce lo spazio dei numeri.

La dimostrazione formale necessiterebbe di una serie di passaggi piuttosto noiosi perché implica la dimostrazione dell'equivalenza fra un programma in un linguaggio qualsiasi e il corrispondente programma scritto però su una macchina a registri.

Questa affermazione è nota in logica come "Tesi di Church" ed afferma che:

"Fino a prova contraria si assume vera l'affermazione che una funzione computabile sia R-computabile"

Il significato terra-terra è il seguente: "data una funzione per la quale sia possibile calcolare il valore (ad esempio la radice quadrata, il coseno, etc...), allora è possibile calcolare il valore usando un sistema automatico codificato come programma di una macchia a registri".

Dal momento che i programmi per computer sono delle funzioni, la cosa calza a pennello. Non dimostreremo nulla di ciò: il nostro è un articolo divulgativo, qualche passaggio rigoroso si può saltare in favore di una più facile comprensione dell'aspetto generale.

Abbiamo anche capito, e se non vi è chiaro lo ripetiamo qui definitivamente, che quando parliamo di "programma" non intendiamo il mero codice sorgente scritto in un qualche linguaggio, ma piuttosto una coppia di oggetti: $P = (C, V_o)$, cioè il codice vero e proprio e l'input del programma V_o.

Tutte le possibili combinazioni di queste coppie, cioè qualsiasi programma (ovviamente sintatticamente corretto), corredato da uno dei suoi input, sono elementi di un insieme che chiameremo D.

Quando diciamo che vogliamo dimostrare l'indecibilità di D, significa che vogliamo trovare almeno un suo elemento per il quale non sia possibile affermare o confutare la computabilità, cioè la determinazione di una sua proprietà, ad esempio sapere se si ferma oppure no.

All'interno dell'insieme D esistono dei sotto-insiemi, che sono i modi con le quali possiamo raggruppare certi elementi in base a particolari caratteristiche.

Un sotto-insieme di D è ad esempio l'insieme dei programmi che si fermano: chiameremo D-HALT questo sotto-insieme.

Analogamente esisterà un'altro sotto-insieme, disgiunto dal precedente: D-NOHALT e sarà composto da quei programmi che non si fermano.

Se dimostriamo che un elemento di D-HALT o di D-NOHALT è indecidibile, allora abbiamo dimostrato l'indecidibilità dell'intero insieme D.

Questa è una tecnica comune in matematica: si cerca di restringere il campo d'azione e di confutare una certa affermazione su di esso per dedurre che l'intero insieme globale è viziato dalla stessa tara.

Nella riduzione a macchina a registri la coppia P=(C, V_o) diventa una sequenza ordinata di istruzioni che fanno evolvere lo stato della macchina stessa. Lo "stato" di una macchina a registri è l'insieme dei valori di tutti i suoi registri in quel particolare punto dell'elaborazione.

P avrà una rappresentazione, si chiama in gergo "gödeliano" del programma P con input V_o, del tipo:

0 - A_o, S_o
1 - A_1, S_1
2 - A_2, S_2
....
n - A_n, S_n

intendendo con questa notazione che ad ogni passo "i" del programma verrà eseguita l'istruzione A_i e conseguentemente lo stato della macchina diverrà S_i.

Ovviamente qualsiasi programma che stia in D-HALT si ferma per definizione, quindi avrà un gödeliano di questo genere:

0 - A_o, S_o
1 - A_1, S_1
2 - A_2, S_2
....
n - HALT, S_n

L'ultima istruzione non può essere che quella di stop dell'esecuzione. Anche l'ipotetico programma in grado di dirci se un altro programma qualsiasi si fermerà dopo un certo numero di passi, si deve per forza fermare.

Vi ricordo che stiamo ipotizzando che questo programma esista e andremo a dimostrare che non può esistere.

Questo ipotetico P_o prenderà come input il programma P con il suo stato S e dopo k step si fermerà fornendo la sua risposta in uno dei registri della macchina, diciamo R_o che conterrà il valore nullo se il programma si ferma, altrimenti R_o avrà un valore diverso da nullo e significa che il programma P in esame non si ferma mai (è una convenzione ovviamente).

Il gödeliano del nostro P_o sarà:

$P_o(P)$:
0 - A_o, S_o
1 - A_1, S_1
2 - A_2, S_2
....
k-1 - A(k-1), $S_k(R_o)$
k - HALT

Allo step k-1 il registro R_o ci dice qual è il risultato del calcolo in base, come abbiamo stabilito, al valore del registro R_o.

Abbiamo quindi la nostra "macchinetta universale".

Bene, partendo da questo costruiamo un altro programma P così fatto:*

$P^*(P_o(P))$:
0 - A_o, S_o
1 - A_1, S_1
2 - A_2, S_2
....
k-1 - A(k-1), $S_k(R_o)$
k - IF R_o = null THEN k else k+1
k + 1 - HALT

Sono esattamente le stesse istruzioni del programma P_o fino allo step k-1, poi lo step k è stato cambiato e si è aggiunto lo step k

+ 1.

P non appartiene necessariamente a D-HALT, vi apparterrà solo se il programma che va ad esaminare è in D-HALT.*

Quindi diciamo che P si ferma se il suo input è un programma in D-HALT e non si ferma per niente se il programma è fuori da questo sotto-insieme.*

Tutti d'accordo?

Bene, adesso applichiamo P al programma P_o, che sappiamo essere in D-HALT:*

Arrivati allo step k-1 avremo R_o = null e quindi dovremmo dedurre che P_o si ferma, ma il passo successivo manda P in loop e pertanto P* non si ferma mai per P_o.*

Siamo arrivati alla contraddizione diagonale: "abbiamo costruito un programma P_o che appartiene a D-HALT ma non è possibile dimostrarlo".

L'insieme D è indecidibile.

Attenzione: abbiamo volutamente semplificato la dimostrazione lasciandola su un piano più intuitivo che rigoroso; non me ne vogliano i logici matematici "accademici".

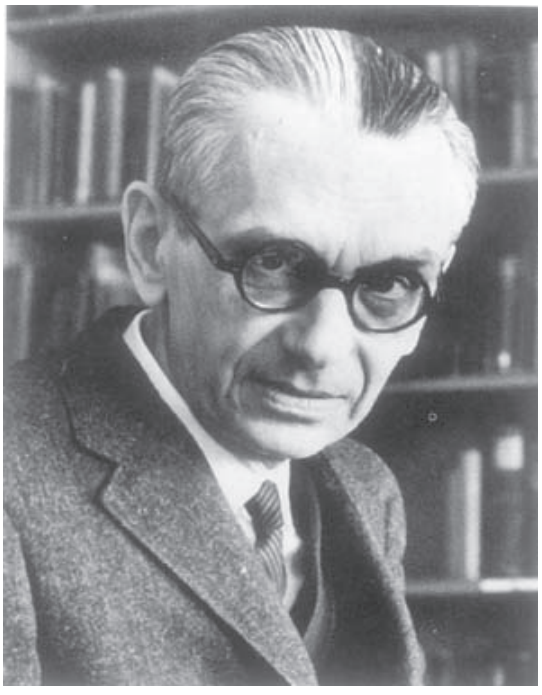
L'obiezione più immediata potrebbe essere fatta al nostro ragionamento affermando che le istruzioni A_o, A_1 e così via del programma P_o, potrebbero essere diverse dalle corrispondenti di P. In realtà il processo di gödelizzazione assicura che sono le stesse.*

Mi auguro che qualcuno, magari non pretendo in tantissimi, sia riuscito a seguirmi in questo ragionamento che è difficile, me ne rendo conto, ma che è una delle basi della moderna scienza informatica.

Concludo con una citazione che un professore di informatica era uso declinare ai sui studenti:

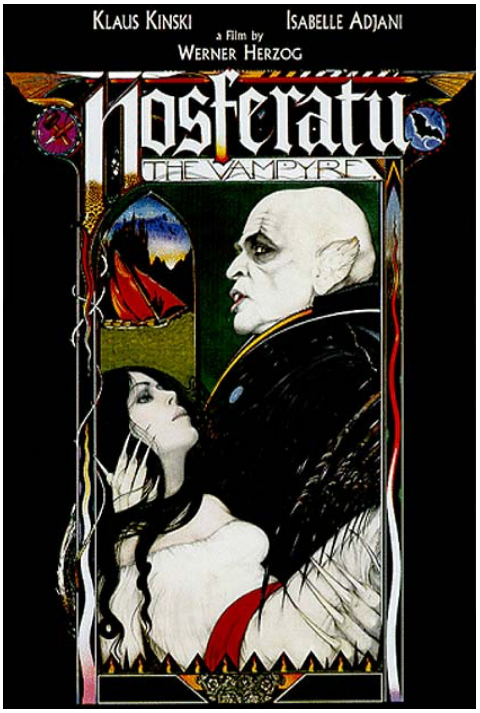
"Fatti non foste solo per scrivere codice, ma per tradurre la vostra intelligenza nel linguaggio della macchina".

(=)



Una foto di Kurt Gödel verso la fine della sua vita (morì suicida nel 1978 all'età di 71 anni). E' considerato il più grande logico moderno e secondo solo ad Aristotele.

Automatik (27)
Nosferatu



Di Lorenzo Paolini

Dove vi faccio conoscere Patrizio, da noi soprannominato Nosferatu il vampiro.

Dunque accadde che Romano, il titolare della Automatik Noleggio Giochi, decidesse di affiancare a Daniele, lo storico operaio tuttofare della ditta, un ragazzo che potesse presto imparare e quindi raddoppiare le capacità logistiche dell'azienda.

In pratica i lavori da fare erano semplici e ripetitivi: si trattava di scorrazzare in lungo e largo nel territorio di competenza della ditta (un raggio più o meno di cinquanta chilometri), per sostituire i giochi nei bar, ritirare gli incassi e fare al limite delle riparazioni o delle manutenzioni ordinarie. Quali erano queste "manutenzioni ordinarie"? Più che altro pulire il vetro che copriva lo schermo, che con il tempo si offuscava per il deposito interno di polvere elettrizzata dallo schermo CRT, cambiare i gommini ai flipper, controllare che le gettoniere funzionassero bene, etc... Come si vede niente che non si potesse imparare nel giro di tre mesi o poco più.

Con questa idea Romano assunse, Dio solo sa perché, un certo Patrizio: un ragazzo più o meno di vent'anni, alto e dinoccolato con aria abbastanza assente e taciturno per natura. La carnagione chiarissima di questo tizio, i suoi capelli neri portati lisci e lunghi fin quasi alle spalle, unita alla sua indole solitaria e ad un modo di vestire che si potrebbe definire "emo", ma all'epoca non c'era questo movimento dark, gli guadagnò il soprannome di "Nosferatu, il vampiro" (per gli amici "il vamp").

Io e Daniele eravamo spietati in questo senso e quando arrivò in ditta Patrizio e fu ben presto chiaro che non se ne sarebbe cavato nulla di eccezionale, le celie sul suo soprannome e sulla sua indole non si contarono più!

Peraltro lui non dimostrò mai di aversene a male anche quando talvolta ci si spazientiva della sua indolenza e lo si incitava decisamente.

Patrizio cominciò seguendo Daniele nei suoi giri giornalieri. Il piano era di renderlo autonomo per le cose più semplici il più presto possibile, poi avrebbe dovuto imparare qualche facile riparazione dal sottoscritto e così via.

Bene, non si uscì mai dalla fase 1.

Il perché non ce lo spiegammo all'epoca, cioè Daniele e il sottoscritto, perché Romano, che doveva saperne qualcosa di più, tagliava corto ad ogni accenno da parte nostra. Cosa diavolo ci fosse fra i due nessuno lo seppe mai. Sospettammo che fosse figlio di un amico di famiglia ma i tentativi che portò avanti Daniele per farlo sbottonare e saperne qualche cosa della sua vita, non sortirono alcun effetto. Il sottoscritto poi non era appunto considerato da Nosferatu, proprio come non esistessi. Arrivava la mattina in laboratorio, rispondeva al saluto solo se qualcuno lo salutava per primo e poi si metteva con calma a proseguire con quello che stava facendo la sera prima, cioè pulire un gioco o il vetro di un flipper o sostituire i gommini o far girare fino allo sfinimento il programma di diagnostica dei flipper Bally. Questa era proprio una cosa insoffribile perché il nostro amico non si limitava a farlo eseguire una/due volte ma lo rilanciava all'infinito anche quando era tutto a posto e bisognava che qualcuno ci andasse a forza a interromperlo quel maledetto programma che faceva saltare le bobine in sequenza con un fracasso alla lunga insopportabile!

La conversazione del "vamp" era limitata a pochissime frasi di servizio, quasi mai iniziava una conversazione e men che meno partecipava. Uno di quei tipi che quando ci sono pesano sull'ambiente come dei macigni: non sai cosa pensano, non sai cosa a loro interessi, non sai se fai bene a parlare con loro... insomma una sofferenza relazionale vera e propria.

Da un certo punto di vista io stavo meglio di Daniele che se lo doveva sopportare tutto santo il giorno. Se ne lamentò più volte con il titolare e una mattina, evidentemente aveva superato il limite della sua pazienza, saltò sul furgone e lo lasciò lì in mezzo al piazzale come una statua. Peraltro Patrizio non diede segno di esserne rimasto colpito; probabilmente pensava che Daniele si fosse dimenticato di lui o che avesse avuto qualche cosa di urgente da fare e che sarebbe ritornato presto. Romano si arrabbio con Daniele e lo riprese ma ebbe come ritorno una reazione che non si aspettava da quel suo mite collaboratore e quindi, come spesso gli capitava quando non aveva argomenti da contrapporre in una disputa, girò i tacchi e se ne andò senza ripresentarsi per tutto il giorno.

Parlandone anni dopo con una ragazza che avevo conosciuto, mi spiegò che secondo i suoi studi di psicologia, Patrizio era autistico, un disordine mentale che è molto più diffuso di quanto si pensi e che da non pochi anni era stato inquadrato con precisione, risultando fino ad allora catalogato in maniera generica come un ritardo mentale.

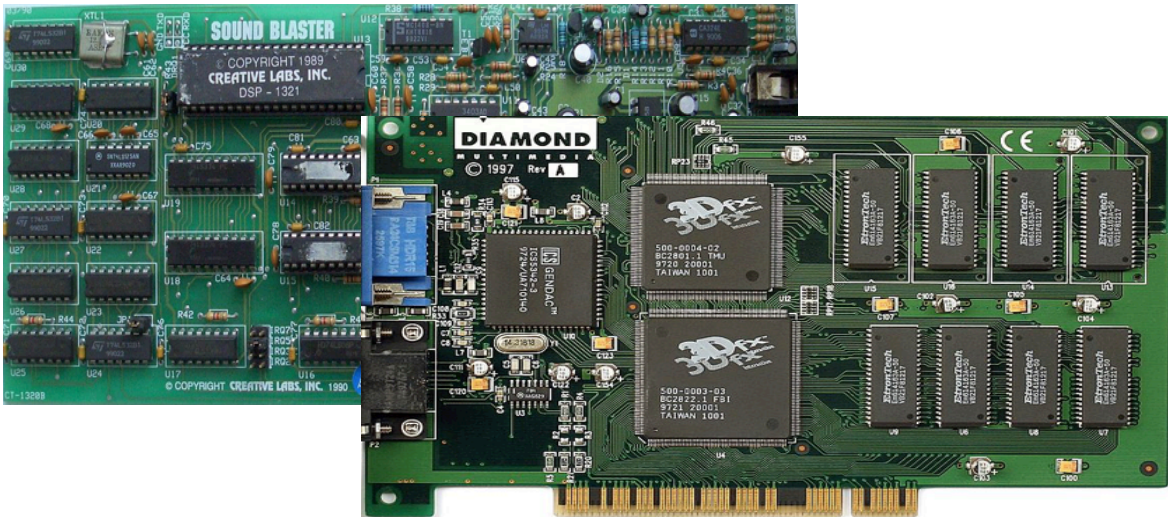
Penso che questa dottoressa (o futura tale) avesse proprio ragione: stetti più attento a certi segnali nel prossimo e, veri o presunti, ne individuai un sacco di persone con problemi simili!

Mi pento un po' di aver considerato Patrizio un "demente" quando lavorava alla Automatik. Certamente con migliore conoscenza e una adeguata informazione avremmo potuto aiutarlo e certo anche lui ci avrebbe insegnato qualcosa. Invece andò in tutt'altro modo e dopo qualche tempo Patrizio non si presentò più, peraltro senza aver salutato nessuno. Romano ci spiegò in due parole che "era andato via". Di più non ci fu dato di sapere e Daniele e il sottoscritto tirarono un sospiro di sollievo riprendendo la nostra convivenza goliardica fatta di giornate passate assieme a scorrazzare giochi su e giù e a inventare scuse per il titolare alle nostre marachelle quotidiane.

Ma questa è un'altra storia...

(=)

Due nomi quasi dimenticati



di Lorenzo/2

La mia esperienza come video giocatore non nasce con gli home computer come ad esempio il C64 o lo Spectrum o le varie console Atari, Nintendo, etc... Pur essendo un informatico "di prima generazione", in realtà praticavo le macchine di calcolo della facoltà e gli Apple (a cominciare dal //e).

Fu solo con l'avvento del PC IBM e in particolare con la generazione dei 386 che la passione videoludica mi coinvolse. Era un periodo quello della fine del 1980 che si viveva una stagione eccitante, forse più di quella che l'aveva preceduta, cioè quella degli home fino all'Amiga. Senza nulla togliere ai meriti degli anni "ruggenti" dell'informatica-quasi-elettronica, bisogna riconoscere che fu l'avvento del PC clone a costo abbordabile a dare una svolta decisiva all'affermazione dell'informatica come fenomeno di massa.

In particolare la trasmigrazione di utenti skillati dal Peek-Poke sugli home verso il PC dotato di strumenti software di adeguato livello, ha interessato una intera generazione di programmatori/smanettoni, quelli che si definiscono in una parola "hacker".

La disponibilità di titoli ludici per PC esplose di conseguenza e come spesso capita, creò occasioni di aggregazione. Ci si scambiavano giochi con i colleghi/amici, si frequentavano i club che, convertitosi al PC partendo dalle specializzazioni precedenti (Sinclair Club, Commodore Club, Amiga Club,....), vissero una stagione brillante alla luce della nuova mania del videogioco. Non c'era negozio di elettronica/informatica che non fosse fornitissimo di scatole coloratissime (e praticamente vuote, visto che contenevano quattro floppy e un manualetto di venti pagine...), scrigno delle più esaltanti avventure notturne in compagnia del fido calcolatore domestico.

I copiatori andavano a mille! Praticamente era la prima cosa che veniva fatta quando si riceveva in prestito il gioco da un amico o dal club: si copiavano i supporti e ci si barcamenava a scopiazzare il manuale, spesso chiave indispensabile per superare le protezioni anti-copia.

Mi sono sempre chiesto due cose in propo-

sito: quale era la convenienza nel produrre programmi di copia, quando i primi ad essere copiati erano proprio loro stessi? E seconda domanda: veramente chi vendeva il gioco si illudeva che la protezione costituita dalla necessità di possedere il manuale poteva essere un deterrente alla pirateria?

Ricordo soluzioni del tutto fantasiose a questo proposito, con dime grafiche da sovrapporre a certe pagine, regoli circolari con chiavi di accesso da ricavare ruotando il cursore,... Invece di essere un deterrente secondo me stimolavano l'ingegno di quelle persone che non mancavano mai di inventarsi una qualche soluzione, fantasiosa come la protezione o magari anche ridicolmente semplice: un affronto alle menti dei progettisti!

Non ho mai avuto risposte ai miei dubbi amletici.

Tornando al "PC per giocare", assieme ai giochi fiorirono le periferiche e non parlo solo dei joystick di varia foggia in grado di farci sentire al volante di una monoposto o davanti al cocktpit di un caccia, ma delle soluzioni che ovviavano alle mancanza progettuali più eclatanti del sistema di IBM: la grafica e il suono.

In fondo il possessore di PC era abituato ad aprire il cabinet per infilarci una scheda; non era come il possessore di Amiga che si trovava equipaggiato di tutto quello che poteva servire (o quasi)!

Ci sono quindi dei nomi che oggi sono sconosciuti ma che una ventina di anni fa erano linguaggio corrente. Ne vogliamo risentire due?

Creative Sound Blaster

La sezione audio era inesistente nel progetto del PC IBM. In fondo, si pensava, in ufficio al di là di qualche bip, di quale suono c'è bisogno? Però se il PC diventa una console da gioco (e che console, vista la sua crescente potenza elaborativa!) non è possibile prescindere da un audio a livello degli ultimi home dell'epoca: Amiga, Atari e perfino Apple con il suo GS potevano vantare anni luce di distanza dal minuscolo speaker interno del PC. In fondo costruire una scheda sonora non era poi così difficile: il chip Yamaha AYxx era disponibile da tempo. Il difficile era diventare uno standard di riferimento. Ci riuscì una piccola ditta nel 1989, la Creative, con il suo modello Sound Blaster. La Creative peraltro esiste ancora e produce accessori audio professionali o semi-professionali.

Per far "digerire" la scheda al PC bisognava istruire il sistema operativo (il DOS) con una linea di comando all'interno del file AUTOEXEC.BAT. Cominciarono così ad apparire scritte come:

```
SET BLASTER=A220 I5 D1
```

Dove A220 è l'indirizzo della scheda (settabile via dip-switch sulla scheda stessa); I5 è il numero di Interrupt; D1 significa primo canale DMA.

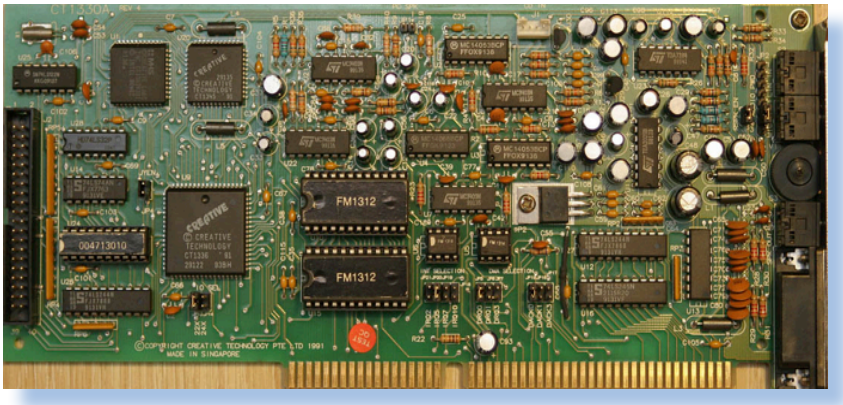
Come si vede, anche il giocatore meno tecnicamente preparato una o due cosucce sul lato tecnico del PC le doveva sapere...

Ovviamente dal primo modello se ne evolverono altri seguendo la fame di prestazioni (si sa che l'appetito vien mangiando...) degli utilizzatori, le esigenze sonore dei giochi e infine l'evoluzione della tecnologia del PC che come minimo cambiava interfaccia del BUS ad ogni generazione.

Utilizzando un chip Yamaha YM3812, la SB era in grado di operare su 11 canali audio ed era anche una periferica di I/O per la conversione digitale-analogica nelle due direzioni. Sì, la SB già all'epoca poteva fungere da scheda di acquisizione audio e campionare un segnale a 12 KHz (23 KHz erano le prestazioni dell'audio in uscita).

Nel 1989 costava 129$ (in Italia probabilmente eravamo sulle 250.000 Lire o poco meno). Già l'anno successivo il prezzo era sceso e i modelli si diversificavano per seguire le esigenze dei professionisti con prestazioni di tutto rispetto sull'interfaccia MIDI, e dei giochi per PC che praticamente da subito hanno supportato la scheda e il suo standard di interfacciamento. Nei giochi dei primi anni '90 era richiesta spesso la configurazione a menù della scheda audio e fra le scelte possibili non mancava mai la voce Sound Blaster.

Un ulteriore salto epocale fu l'introduzione del modello "pro" che aveva a bordo l'interfaccia per cd-rom. Anche per questa periferica era necessario smanettare con i file di configurazione e anche per il cd-rom la Creative impose un suo standard per l'interfaccia, anche se ormai il successo delle soluzioni standard-di-fatto stavano mostrando i loro limiti e ben presto vennero tutte assorbite nelle versioni aggiornate dei sistemi operativi.



3dfx

Sul fronte della grafica il PC non poteva rivaleggiare con macchine che nascevano ben equipaggiate, come gli Amiga ad esempio. Grazie alla sua architettura aperta però il PC poteva essere dotato di schede specializzate con qualsivoglia livello di prestazioni, bastava avere i soldi per comprarsele 'ste schede grafiche da paura!

Il problema però, dal punto di vista dei produttori di videogiochi era irrisolto perché mancava uno standard. Infatti se un produttore di programmi grafici professionali poteva permettersi di progettare e commercializzare una scheda grafica proprietaria funzionante solo con il proprio software, la stessa cosa non potevano farla i produttori di videogiochi.

Per la verità lo standard Super-VGA (SVGA) non era malaccio: 1024x768 con profondità di 8-bit per pixel, ma le esigenze si stavano evolvendo con la necessità di avere un processore grafico per calcolare poligoni, texture e quant'altro. Si stava andando nella stagione degli f/s (frame al secondo): un gioco era bello se i movimenti erano fluidi, quasi come un film!

In fondo la SVGA era uno standard abbastanza vecchio: definito poco prima del 1990 e mostrava chiari segni di vecchiaia alla metà degli anni '90.

Nacque quindi uno spiraglio di mercato dove si infilò prontamente una azienda che si chiamava 3dfs Interactive e che offriva in pratica un co-processore grafico.

Il funzionamento era abbastanza strano perché per rispettare gli standard del PC il segnale video che usciva dalla VGA usata dai programmi "normali", veniva

deviato nella 3dfx e da questa al monitor. La scheda, il cui nome di prodotto era Voodo, commercializzata dal 1996, diventava operativa con il caricamento in memoria di una libreria grafica chiamata GLIDE alle cui funzioni potevano appoggiarsi i programmi. Il primo gioco che ne fece uso fu Quake, poi DukeNukem, poi Doom al quale seguirono altri titoli dello stesso genere (sparatutto in prima persona) e via via gli altri.

Anche le schede 3dfx sono state interessate ad una evoluzione continua fino ad arrivare alla tecnologia di interlacciamento: due schede che funzionano in parallelo occupandosi ognuna di una riga di scansione. Forse lo sbaglio della 3dfs Interactive fu quello di pensare che il modello tecnologico da loro definito potesse continuare indefinitivamente. Alla fine, dopo il formale fallimento, la 3dfx Interactive fu acquistata dalla Nvidia.

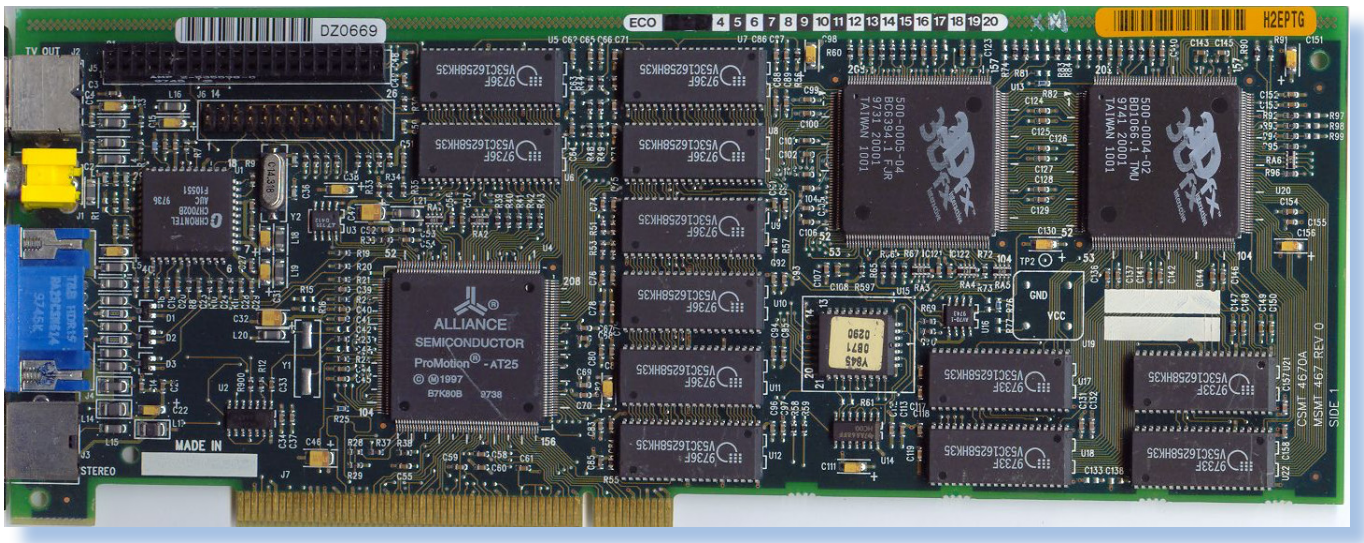
3dfx ha senza dubbio contribuito a cambiare per sempre il settore del gioco su PC. Cominciò con le schede grafiche Voodoo, nel 1996. Queste schede si potevano trovare su molti computer da gioco all'epoca, quasi tutti. Era un periodo in cui le sale giochi avevano ancora un relativo successo, prima che arrivassero le console.

Quando i prezzi della RAM si abbassarono, verso la fine dell'anno, 3dfx creò le sue schede Voodoo PCI, ormai un classico. Nello stesso tempo id Software pubblicava Quake, che introduceva un motore grafico rivoluzionario, per gli standard del momento, perché sfruttava i poligoni vettoriali invece dei pixel come base. Il nuovo Quake (GLQuake) sfruttava i vantaggi dei driver OpenGl di 3dfx, e divenne l'avanguardia di una rivoluzione che avrebbe contagiato il videogioco domestico. Da quel giorno in poi avremmo visto miglioramenti continui nella qualità delle immagini, con superfici uniformi, texture strabilianti e velocità da record. Una tendenza che continua ancora oggi.

Nel 1998 poi 3dfx fece un ulteriore passo evolutivo con lo Scan-Line Interleave (SLI), che permetteva d'installare due Voodoo-2 sullo stesso computer, e sommarne la potenza (vi ricorda qualcosa?). Negli anni seguenti però l'azienda non trovò la strategia giusta per imporsi come prodotto nativo nel PC, così nel 2000 finì in bancarotta e venne venduta alla Nvidia.

(=)

[Reference: immagini tratte da wikipedia]



Super QL

di Sonicher

Dite la verità: un Sinclair QL così non l'avete mai visto. E infatti non potete esservi imbattuti in esso, semplicemente perché questa macchina non esiste! Si tratta infatti di quello che oggi viene chiamato "concept disign" e che una volta si chiamava semplicemente "progetto".

Torniamo indietro e andiamo al 1985 in Inghilterra e precisamente al quartier generale della Sinclair Research Limited dove si lavorava ad una excalation dell'informatica personale, così come Clive Sinclair, fondatore di quella che fino a quel momento era una azienda di successo, concepiva: macchina innovativa, potente ma con rapporto prezzo/prestazioni particolarmente favorevole.

Già dal 1983 era partito il progetto ZX83 che doveva produrre il successore dello Spectrum (il cui nome in codice era ZX82). Il progetto si era però arenato perché la sfida andava oltre la semplice evoluzione dei prodotti finora commercializzati dall'azienda.

Le linee guida potevano essere (ci sostituiamo nel cervello di Clive): processore potente, molta memoria, sistema operativo avanzato, software di produttività potente e facile da usare, memoria di massa capiente e poco costosa.

Le CPU c'erano, ad esempio il Motorola 68000, un core a 32 bit potenzialmente capace di lasciare al palo gli ormai obsoleti microprocessori a 8 bit. I chip di memoria venivano prodotti in quantità crescente e i prezzi stavano scendendo. Sistema operativo e software potevano essere sviluppati con un certo sforzo, senza peraltro inventarsi nulla: video gestito a finestre, multitasking, un BASIC evoluto,...

Mancava la memoria di massa, nel senso che i floppy potevano essere adatti ma costavano ancora troppo e il drive era ingombrante e pesante anche nella versione da 2,5". Sir Clive Sinclair voleva una macchina "portatile", perché riteneva che la sua azienda dovesse concentrarsi su un fattore di forma agile. Il PC IBM era un macigno, lasciatecelo dire, buono per gli uffici contabili, lui aveva in mente ben altro.

La caccia era aperta, anche se si decise ad

un certo punto di far uscire un sistema "con quello che era disponibile" e cioè il Quantum Leap con il Motorola 68001 (core a 32 bit ma bus a 8), il minimo di memoria necessaria ((128 Kb) per un sistema operativo ancora acerbo e un Super-BASIC promettente ma non ancora consolidato al 100% e infine i famigerati micro drive come storage.

Sono note le vicende legate agli annunci prematuri, alla raccolta dei pre-ordini e ai ritardi nelle consegne. Comunque il QL era una bella macchina per l'epoca!

Nella ricerca di una periferica di massa degna di questo nome, gli ingegneri della Sinclair Research si imbatterono in una nuova tecnologia che prometteva parecchio: la Wafer Scale Integration. Ne acquisirono i brevetti e cercarono, nella tradizione della ditta, di tirarne fuori il meglio con il minimo di spesa.

Ci prendiamo un po' di tempo per descrivere in breve questa tecnologia, rimandano per chi volesse approfondire alla copiosa documentazione esistente in rete, a cominciare dalla Wikipedia.

L'idea originale dei wafer da usare come super-chip, pare sia del britannico Ivor Catt nel 1975, al quale si fanno risalire i primi brevetti. Costruire il super-chip è una cosa, poterlo utilizzare è un'altra! E infatti si sono susseguiti moltissimi brevetti che intendono risolvere il problema principale di questo prodotto: l'individuazione automatica dei componenti difettosi e il loro isolamento dalla griglia di memorizzazione.

Tutti sanno che i chip elettronici sono prodotti su un substrato di silicio attraverso un processo industriale che assomiglia alla stampa 3D: strati successivi vengono "incisi" sulla superficie per ottenere transistor e collegamenti. Il wafer, cioè lo strato sottile che costituisce la base del processo ha forma circolare di diametro variabile (sono tre le misure standard industriali) e sono fette tagliate da un cilindro di silicio mono-cristallino ottenuto da un processo di crescita di un cristallo alimentata dal gas di-metil-silano. Il processo fisico e industriale è molto complesso seppure interessante, ma esula dallo scopo di questo articolo per cui non lo approfondiamo. Sta di fatto comunque che queste fette di silicio vengano "stampate" con chip di varie dimensioni per ottimizzare l'utilizzo della superficie e poi tagliati con un laser ed assemblati nei contenitori che costituiscono i chip veri e propri da utilizzare nei circuiti elettronici.

Tutta la catena del processo produce un certo numero di scarti: la superficie del wafer non è perfetta al 100% e se un chip cade su un "buco", per quanto microscopico, tutto

Il progetto ZX83 realmente commercializzato. E' il QL che conosciamo...

Dalla sabbia il computer. In realtà non si parte dalla sabbia, ma è lo stesso.

il chip è da buttare. Il processo di assemblaggio è anch'esso foriero di qualche errore ed è la parte industrialmente più costosa.

E' logico quindi che ci sia chiesto se fosse possibile utilizzare l'intero wafer come componente elettronico invece di ricavarne chip singoli. La cosa è interessante per la memoria RAM: i chip sono minuscoli, tutti uguali e ne servono sempre di più in un computer. Se fosse possibile inserire l'intero wafer come memoria di un elaboratore il costo per bit scenderebbe di molto. Il wafer di silicio è occupato da un "chippone" di memoria e i problemi sono quelli di indirizzare una così grande quantità di celle ma soprattutto quello di tollerare una certa percentuale di chip guasti. Su questo problema lavorarono molti ricercatori, superando sia teoricamente che industrialmente le difficoltà, almeno sulla carta.

Ovviamente i tempi di accesso non sono quelli di un singolo chip individualmente indicizzato, ma come memoria di massa è molto più performante di un floppy. E' vero che c'è il problema dell'alimentazione ma con una batteria tampone e uno store alimentato, in qualche modo si affronta.

I ricercatori della Sinclair Research si imbattono in questa tecnologia già nel corso del progetto ZX83 ma il capo, sir Clive Sinclair, preme per uscire e devono accontentarsi dei famigerati micro drive in modo da lanciare nel 1984 il QL. Sappiamo che l'annuncio gode della fama dei uno dei più grandi vaporware della storia informatica: in pratica la Sinclair annuncia un prodotto che non esiste industrialmente e da questo seguiranno le delusioni dei clienti, le proteste, i bug delle prime release.

Tutti gli storici dell'industria sono concordi nell'affermare che la Sinclair, con l'affare QL, pose le basi per la propria fine prematura.

A difesa di Sinclair va comunque detto che praticamente nessuno, se non la Apple, capì che l'era del personal-gioco era finita.

Una soluzione alternativa, più in linea con l'espansione di memoria RAM già disponibile per il QL.

Il Personal Computer QL era un prodotto all'avanguardia, seppure con i problemi citati e avrebbe potuto fare da apripista per i successivi progetti di Sinclair. L'adozione del Wafer poteva far fare quel salto di qualità che Clive auspicava, cioè lasciare al palo la concorrenza. Mentre la messa a punto proseguiva dal punto di vista elettronico, la Sinclair incaricò un designer per il progetto della nuova macchina: Rick Dickinson che aveva già collaborato con lo staff della Sin-

clair in precedenti occasioni.

Il risultato del lavoro del designer appare (sulla carta) quanto mai accattivante con l'evidenziazione della periferica Wafer e l'inserto di colore rosso alla base della tastiera. Accanto alla base-tastiera Rick progetto un dock, alimentato a parte che avrebbe costituito il "magazzino" della memoria. Questo elemento viene indicato nel lavoro di Dickinson come "mini-stack" che doveva costituire una sorta di hard-disk realizzato con una serie di wafer in un contenitore autonomo.

Il QL+ non riuscì ad arrivare alla fase di realizzazione industriale, così come rimase sulla carta il suo successore che avrebbe dovuto chiamarsi Super QL, dal design avveniristico con il suo arrangiamento verticale e la rottura totale con gli schemi precedenti(vedi foto in questa pagina).

Tutto rimase sulla carta perché nel 1985 la Sinclair vendette ad Amstrad e ogni progetto venne abbandonato. Al nuovo proprietario del marchio non interessava sperimentare ma solo stare sul mercato già consolidato dell'home svecchiando i prodotti esistenti per quanto più tempo fosse possibile.

Conclusione.

Chissà come sarebbe stato con il QL+ e ancora di più con il Super QL! Certo non sarebbero stati certo loro a fermare l'inarrestabile diffusione del PC IBM: se non c'è riuscito l'Amiga... Ma ora avremmo due macchine in più da collezionare e studiare.

Peccato!

Reference.

"Delete: A Design History of Computer Vapourware" by Paul Atkinson
ISBN-13: 978-0857853479
ISBN-10: 0857853473

Foto da Rick Dickinson: https://www.flickr.com/photos/9574086@N02/sets/72157600854938578/

Aubusson, R.C.; Catt, Ivor, "Wafer-scale integration-a fault-tolerant procedure," in Solid-State Circuits, IEEE Journal of , vol.13, no.3, pp.339-344, June 1978
doi: 10.1109/JSSC.1978.1051050

Il "Super QL", il sogno finale di Sir Clive Sinclair, mai realizzato.

Il mio Z80

di Gizmo

Premessa

Costruire oggi un microcomputer ispirandosi ai progetti dei primi anni '80, con tutta la documentazione reperibile e con la grande disponibilità di componenti sia originali che moderni adattabili alla bisogna e poco costosi, non è più appannaggio di pochissimi "geak". La sfida non è quella degli anni '80 ma la passione rimane quella dei "bei giorni" e si possono ottenere grandi risultati non tanto per un utilizzo reale di questi "fai da te", quanto per il bagaglio tecnico che inevitabilmente si acquisisce nel progressivo superamento dei piccoli e grandi problemi che questa sfida propone.

Il componente più importante, indispensabile addirittura, è appunto la voglia di farlo. È la molla che ha spinto un nostro affezionato lettore e appassionato di retro-informatica che ci ha inviato il suo progetto.

Lasciamo quindi la parola a Stefano Bianchini, in arte "Gizmo".

Perché

Ho voluto costruire queste schede da perché io sono dell'epoca del commodore 64, lettore di riviste di elettronica nei primi anni 90, dove vedevo che realizzavano schede da connettere al C64 per fare varie cose, tipo accendere luci, lavatrici, aprire cancelli ed ero affascinato da queste cose. Non ero in grado allora, perché troppo giovane e inesperto, di cimentarmi in tali realizzazioni. I miei primi 15 anni di vita lavorativa li ho passati assemblando e riparando PC, ma non c'era quel gusto che ricordavo dalla mia adolescenza; realizzando questa scheda con lo Z80 ho soddisfatto il desiderio che avevo da ragazzino.

L'idea

Ci sono moltissimi siti on-line che descrivono progetti simili a quello che avevo in mente. Molti offrono anche kit più o meno completi, ad esempio lo stampato, ma io volevo fare tutto da solo documentandomi e prendendo da ciascun progetto le parti che facevano al mio caso.

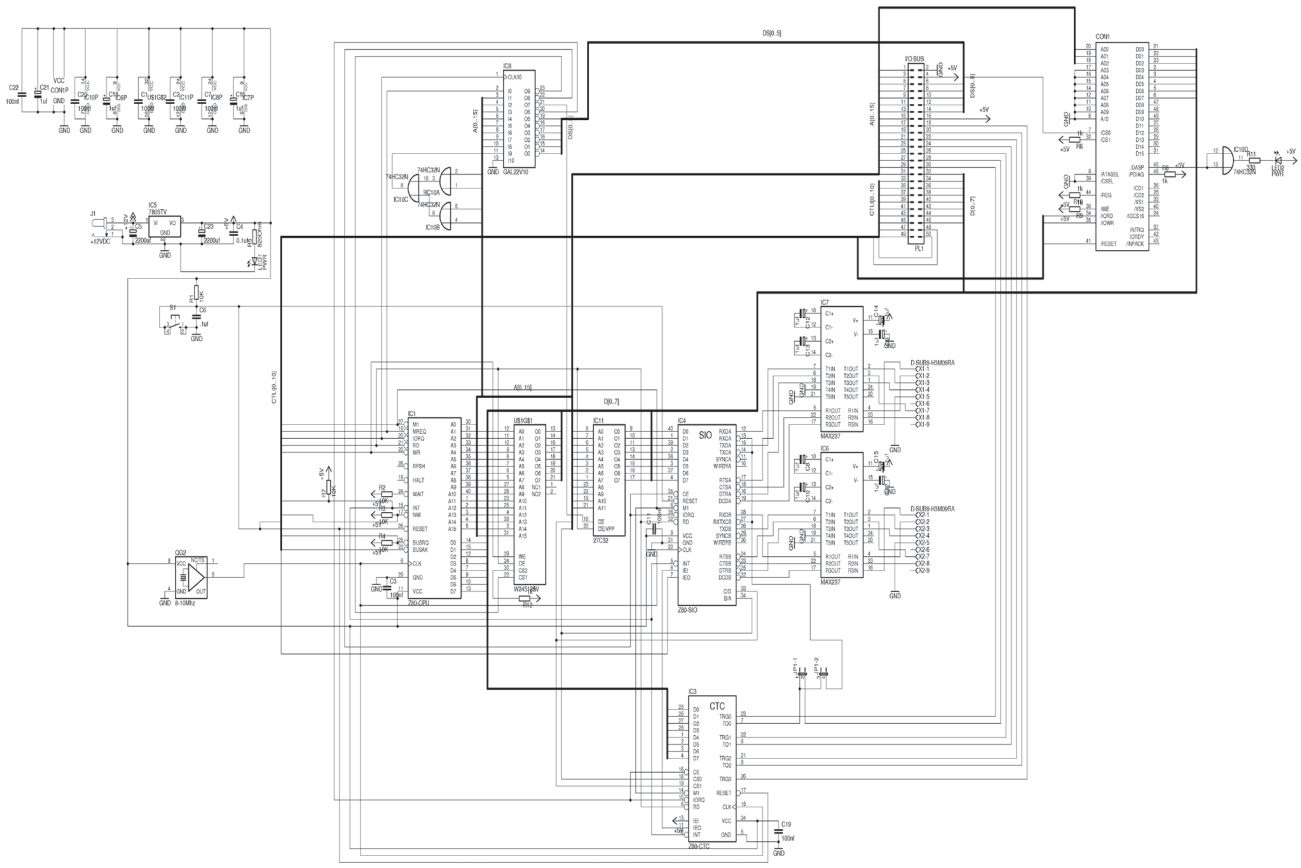
Allora l'hardware di base l'ho copiato da un sito molto completo [1].

La realizzazione

Ho rifatto il PCB in una versione che si potesse produrre con mezzi poveri [vedi foto 1 e 2] e ho iniziato a prendere dimestichezza con la programmazione in C utilizzando il Development Kit Z88DK [2] poi ho iniziato a pensare come si potesse migliorare il progetto, ad esempio lui ha usato una Eprom da 32k ma il suo monitor software occupava solo 5k il resto dello spazio era inutilizzato; inoltre è noioso caricare dalla seriale ogni volta il programma che si vuol far girare e per fare modifiche al monitor è necessario ogni volta riprogrammare la Eprom.

Ho quindi realizzato un hardware per interfacciare una compact-flash con il bus dello Z80, questa è semplicissima anche se vedendo altre cose che si trovano su internet non si direbbe. Le CF possono emulare un disco IDE (quindi con un bus a 16bit) e diverse persone che hanno realizzato computer con lo Z80 (o altre CPU a 8bit) ingenuamente hanno fatto interfacce piene di chipettini logici per accedere alla CF a 16bit, ma le CF sono memorie a 8bit (infatti io ho scritto che possono emulare un disco IDE) ma di fatto una CF si può interfacciare DIRETTAMENTE al bus dello Z80 senza nessuna logica frapposta e trasferire i dati 8 bit alla volta, l'unica cosa che bisogna fare è accedere al registro delle features e inizializzarla a 8bit.

Cosi' ho passato il tempo libero dell'estate a scrivere le routines in C per po-



ter accedere alla compact flash dallo Z80, prima ho scritto le routines a basso livello per inizializzarla, leggere un settore, scrivere un settore, cancellare un settore, ho fatto pesantemente uso di matematica a 32bit per poter calcolare gli indirizzi LBA, al momento uso solo 24bit effettivi di indirizzo + 8bit per impostare master/slave, quindi posso indirizzare 8Gb, ma il passo per usare un'indirizzo a 32bit intero e poter accedere a 512Gb è veramente minimo (anche se probabilmente inutile).

Finita la scrittura delle routines per accedere a basso livello alla CF ho modificato il monitor che stava in ROM cancellando quasi tutto il codice e trasformandolo in un bios che all'accensione non fa altro che caricare in ram i primi 48 settori della CF e poi lanciando l'esecuzione del primo indirizzo della ram, in questo modo potevo copiare in modo raw un mio binario sulla CF e la scheda Z80 all'accensione avrebbe eseguito quel programma. Ho rimodificato il monitor di base della MK2 per poter girare in RAM e l'ho espanso aggiungendo comandi e tutte le routines necessarie ad accedere alla CF e ho sviluppato poi un file system di testa mia, quindi ho ottenuto un vero e proprio DOS, il kernel del sistema operativo occupa i primi 48settori del disco e il file system inizia subito dopo, è quindi possibile riformattare la CF senza perdere il sistema operativo, questo metodo è molto simile a quello usato dai computer Apple.

Al momento nell'interprete dei comandi che ho scritto sono presenti pochi comandi di base che si vedono elencati nel video quando chiamo "help", ho implementato giusto ieri sera i parametri sulla linea di comando: nel video si vede ad esempio che se faccio mkdir, il comando mi dà un dialog dove mi chiede il nome della directory che voglio creare, ora invece posso fare direttamente "mkdir nome_cartella", il binario compilato occupa al momento 18k, io ho scritto 40k di codice sorgente che non sono pochi.

Lo stato del progetto

Al momento mi sono messo in pausa con lo sviluppo software perché devo lavorare ma soprattutto perché' ormai è il momento di evolvere l'hardware, ho già modificato lo schema del MK2 originale per ospitare una Eprom da soli 4k (più che sufficienti per il bios) e una RAM da 64k, senza fare le capriole con l'hardware i primi 4k della RAM non saranno accessibili, la scheda quindi disporrà di 60k di ram utilizzabile. Sarà integrato a bordo lo zoccolo per la compact flash e devo ancora decidere il tipo di connettore da usare per le schede di espansione, probabilmente a pettine, quello usato dal tizio inglese per l'MK2 è veramente scomodo da utilizzare e va aggiunto anche un circuito che tiene basso il reset qualche istante quando alimenti la scheda perché' la MK2 spesso quando viene alimentata si blocca e va resettata quasi tutte le volte. Il mio scopo finale è costruire una motherboard che sia autosufficiente con un sistema operativo facilmente aggiornabile e totalmente modificabile dove ci si possa divertire costruendo espansioni hardware e scrivendo il software per farle funzionare, io trovo che questo approccio sia molto più ludico rispetto ai vari Arduino o Raspberry, Arduino è senzaltro una bella cosa ma è una MCU e non puoi vedere fisicamente le varie parti (ram rom cpu seriali etc) come entità separate, sono schede più che altro pensate per far girare firmware per compiti specifici e non un sistema operativo, invece su raspberry è già tutto fatto, lo prendi ci installi Linux e sostanzialmente hai il software già scritto per fare praticamente ogni cosa, puoi aggiungere pezzi ma spesso pure quelli già preconfezionati ed infine svi-

luppare software dentro un sistema operativo cosi' complesso come Linux richiede anche di conoscerlo bene e di sapere come funziona invece su questa scheda con lo Z80 è tutto più semplice, metti una periferica connessa al BUS e dal programma C fai "outp(dato,indirizzo)" per mandargli un byte e vedere cosa fa, e "variabile = inp(indirizzo)" per leggere cosa ti manda indietro, fine, tutto semplicissimo. Non ho problemi ad ammettere che prima di costruirla non avevo la minima idea di come si programmava in C, e ora sono qui a scrivere un SO.

Il sito che ospita il progetto si trova al riferimento [4]. Nella sezione download del sito sono disponibili i sorgenti.

Conclusione

Penso che per ora quello che posso dirvi del mio progetto è tutto qui. Ho intenzione di migliorarlo cominciando con la realizzazione di una versione più professionale della scheda madre e proseguendo nel completamento del progetto come sistema completo stand-alone facilmente aggiornabile e programmabile.

(=)

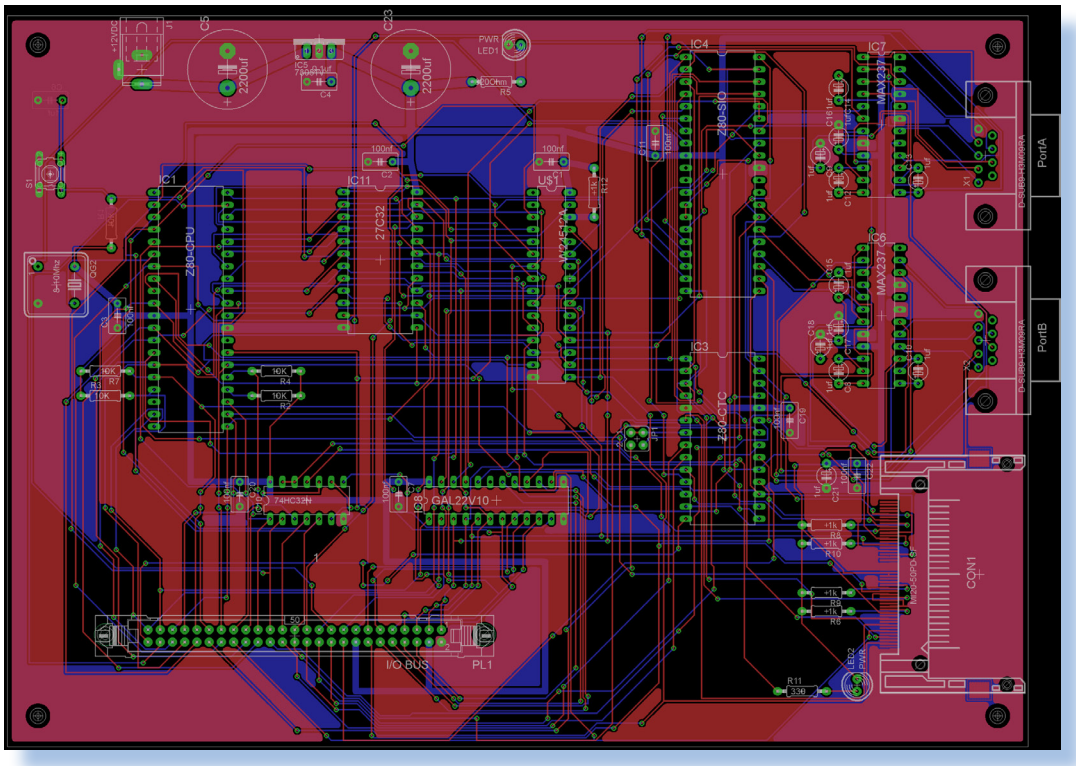
Riferimenti

[1] - http://www.allthingsmicro.com/index.php/projects/20-z80-based-computer

[2] - http://sourceforge.net/projects/z88dk/

[3] - http://www.gizmoblog.eu

[4] - http://www.zilog80.eu



DELETE. A Design History of Computer Vapourware

di Sonicher

Introduzione

Quando ho avuto notizia di questo volume non ci potevo credere: un libro che racconta il design delle apparecchiature informatiche! E soprattutto quel design che non ha mai visto la realizzazione industriale se non la fase prototipale (bruttissimo inglesismo).

L'ho letteralmente divorato, pagina dopo pagina, immagine dopo immagine, disegno dopo disegno... E' assolutamente fantastico!

Quando poi in redazione si è cominciato a parlare di un articolo o più articoli che prendessero spunto dalle macchine mai realizzate o mai commercializzate, ho colto la palla al balzo e ho preteso di lavorarci io a questa serie. Il primo "parto" è l'articolo sul Super QL che potete leggere in questo fascicolo.

E' un tema davvero affascinante: scoprire come non la realizzazione elettronica del computer ma l'idea di funzionalità e bellezza, cioè di uno "stile", sia stata perseguita da dei veri maestri, come appunto Paul Atkinson, autore del libro.

Delete, cioè i progetti cancellati

Dopo le prime realizzazioni di computer "home" o comunque personali, si cominciò a capire che non poteva il tutto esaurirsi in una tastiera da telescrivente; ci voleva di più, anche per distinguersi dai concorrenti e per portare avanti il "marchio di fabbrica" che distinguesse l'azienda.

Ogni realizzazione negli anni '80 si incanalava verso questa idea, con poche variazioni e qualche "alzata di ingegno" per evolvere. Se ci pensate l'Apple][è uguale al //e che è quasi uguale all'Apple /// e il //c, non dico che non sia indistinguibile, ma come idea generale si avvicina molto ai fratelli maggiori.

E il Commodore 64 non è il gemello del Vic20? Non parliamo poi delle cosiddette "macchine da ufficio": i vari TRS si clonano generazione dopo generazione (parliamo della linea estetica generale).

Sinclair era partito con il giocattolo ZX80 per cambiare in una livrea più seria con lo ZX81 e poi lo Spectrum che, d'accordo è diverso dai predecessori, ma poi mica troppo!

Sinclair fu uno dei primi a capirlo che un sistema doveva anche essere "bello". La paternità assoluta di certe idee è difficile da attribuire, diciamo che le idee sono nell'aria e qualcuno che sa coglierle al momento giusto le fa proprie. Steve Jobs era un maestro in questo!

Il risultato per la Sinclair Research poteva essere quello che viene mostrato nella copertina del libro e più in dettaglio nell'articolo che ho citato: un QL "evoluto" con degli inserti colorati e l'inclusione di una tecnologia nuova in un involucro sia funzionale al fattore di forma dell'oggetto elettronico stesso, ma anche parte "movimentante" della linea estetica del sistema.

Le pagine del volume scorrono rapide sotto le dita e gli occhi si riempiono di oggetti, spesso solo dei disegni o dei "concept project" costruiti di plastica, giusto per dare l'idea di come verranno. Qualcuno si riconosce, qualche idea è proprio quella che poi il produttore ha realizzato o che ritroviamo in soluzioni analoghe.

Il lavoro del design, così riassunto non viene valorizzato per intero. Sembra quasi che sia facile arrivare alla soluzione mostrata! Invece ci vogliono ore e ore di lavoro per spostare, limare, cancellare, confrontare soluzioni abbozzate... E poi si arriva alla fase di progettazione vera e propria e allora contano anche gli ingegneri, le macchine assemblatrici, il materiale, il costo di ogni elemento, comprese le viti che chiudono i gusci di plastica!

In questa fase si scopre spesso che "l'idea è buona ma irrealizzabile" o che "bello, ma costa troppo" e altre frasi di giustifica per il taglio del progetto.

Peccato però che IBM non abbia poi realizzato quei due coloratissimo home che vengono descritti nella pagine del volume e ci abbia invece rifilato un PC Junior davvero brutto (oltre che ciofeca dal punto di vista informatico).

Ma non solo computer: ci sono anche telefonini, abbozzi di tablet, e quant'altro è venuto in mente di fabbricare alle industrie elettroniche fino al 2000 nel comparto informatica e telecomunicazioni.

Conclusione

Il libro non è dei più economici (circa 40 Euro su Amazon) ma sono giustificati dalla ricchezza di immagini e soprattutto da informazioni e retroscena che difficilmente si possono reperire altrove.

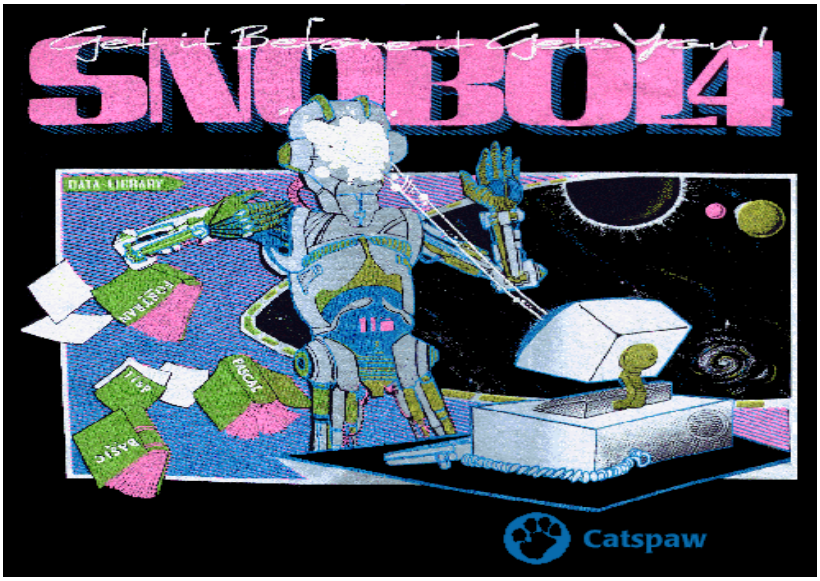
Se non intendete fare questo investimento per la vostra biblioteca di retro informatica, almeno cercate una biblioteca che lo possieda e prendetevi un pomeriggio per sfogliarlo; ne sarà valsa la pena.

(=)

Reference.

"Delete: A Design History of Computer Vapourware" *by Paul Atkinson*
ISBN-13: 978-0857853479
ISBN-10: 0857853473

SNOBOL (Parte 4)



di Salvatore Macomer

Il metodo migliore per imparare un linguaggio o anche solo per esplorarne le caratteristiche principali, è quello di analizzare i programmi via via più complessi.

Ad esempio vi ricordate l'algoritmo "Bubble Sort" per ordinare un vettore di elementi? L'algoritmo è semplice: si procede esaminando un elemento del vettore e il suo successivo e scambiandoli di posto se non risultano ordinati. Scandendo n volte il vettore e tenendo conto del numero di scambi che si effettuano, ci si ritrova con il vettore ordinato quando l'ultima scansione non ha prodotto scambi.

Come si programma un simile algoritmo in linguaggio SNOBOL?

Il metodo migliore per imparare un linguaggio o anche solo per esplorarne le caratteristiche principali, è quello di analizzare i programmi via vi più complessi.

Ad esempio vi ricordate l'algoritmo "Bubble Sort" per ordinare un vettore di elementi? L'algoritmo è semplice: si procede esaminando un elemento del vettore e il suo successivo e scambiandoli di posto se non risultano ordinati. Scandendo n volte il vettore e tenendo conto del numero di scambi che si effettuano, ci si ritrova con il vettore ordinato quando l'ultima scansione non ha prodotto scambi.

Come si programma un simile algoritmo in linguaggio SNOBOL?

Nel box della pagina a fronte l'esempio tratto dal libro "THE SNOBOL4 PROGRAMMING LANGUAGE" (vedi ref.), che discuteremo per capire la sintassi del linguaggio.

Penso che tutti abbiano capito che le righe che incominciano con l'asterisco sono dei commenti. Il sorgente è scritto in maiuscolo perché, forse lo sapete, una volta i calcolatori non avevano le minuscole né come caratteri in memoria e ovviamente nemmeno sulle tastiere.

A parte la banale definizione di una variabile "globale" TRIM inizializzata al valore 1, il sorgente definisce tre funzioni con lo statement DEFINE. Le tre funzioni sono: SORT, SWITCH e BUBBLE.

La definizione di una funzione prevede di indicare il nome della funzione, fra parentesi i parametri di ingresso chiusi nelle parentesi e come secondo argomento della DEFINE, e la lista delle variabili locali che saranno usate nel corpo della funzione.

```
*       BUBBLE SORT PROGRAM

                $TRIM           = 1
                DEFINE('SORT(N) I')
                DEFINE('SWITCH(I)   TEMP')
                DEFINE('BUBBLE(J)')

*       GET NUMBER OF ITEMS TO BE SORTED
                N       = INPUT                 : F(ERROR)
                A       = ARRAY(N)

*       READ IN THE ITEMS
READ            I       = I + 1
                A<I>    = INPUT                 : F(GO) S (READ)


*       SORT THE LIST
GO              SORT(N)


*       PRINT SORTED LIST
                M       = 1
PRINT           OUTPUT = A<M>                   : F(END)
                M       =  M + 1                : (PRINT)

*       FUNCTIONS
SORT            I       = LT(I, N - 1)  I + 1   : F(RETURN)
                LGT(A<I>,A<I + 1>)              : F(SORT)
                SWITCH(I)
                BUBBLE(I)                       : (SORT)

SWITCH          TEMP            = A<I>
                A<I >           = A<I + 1>
                A<I + 1>        = TEMP          : (RETURN)

BUBBLE          J = GT(J, 1)  J - 1             : F(RETURN)
                LGT(A<J>, A<J + 1>)             : F(RETURN)
                SWITCH(J)                       : (BUBBLE)

END
```

DEFINE ('SWITCH(I) TEMP') definisce la funzione con nome SWITCH alla quale sarà passato un valore e la funzione stessa userà TEMP come area di memoria per fare il suo lavoro. Lo scopo è semplicemente quello di scambiare due elementi del vettore da ordinare.

L'istruzione A = ARRAY(N) definisce A come nome di una variabile con indice ad una sola dimensione (un vettore) con N elementi. Si noti che il valore N viene letto da terminale e quindi le matrici in SNOBOL hanno dimensione definita a run-time.

Il ciclo:

```
READ    I = I + 1
        A<I>    = INPUT
                : F(GO) S (READ)
```

riempie gli elementi del vettore A con i valori che passiamo da INPUT.
Notate come l'indicizzazione degli elementi del vettore A faccia uso di parentesi angolari <>, sintassi piuttosto rara nei linguaggi di programmazione.

A questo punto si chiama la funzione SORT passando come parametro di input la lunghezza del vettore N, letta precedentemente da console.

```
SORT I = LT(I, N - 1)  I + 1
                       : F(RETURN)
   LGT(A<I>,A<I + 1>)
                       : F(SORT)
   SWITCH(I)
   BUBBLE(I)    : (SORT)
```

Lo SNOBOL utilizza come operatori di confronto gli stessi del FORTRAN e cioè LT per LESS THAN (minore di), GT (GREATER THAN) per maggiore, etc...
L'istruzione LT(I, N - 1) controlla se la scansione degli elementi è terminata. Se vero, cioè se l'indice I, che per la funzione SORT() è una variabile locale e quindi inizializzata all'ingresso, ha raggiunto la fine degli elementi del vettore A<>, la funzione ritorna. Altrimenti si entra nell'algoritmo e cioè si confrontano due elementi contigui del vettore e si procede allo scambio degli stessi se necessario.

Con LGT(A<I>,A<I + 1>) che è il test di controllo se l'elemento A<I> è minore o uguale all'elemento A<I+1>, allora si continua la scansione, altrimenti si procede allo scambio dei due elementi tramite la funzione SWITCH().

Gli altri elementi del programma sono auto esplicativi per chi conosce un po' di programmazione "old style".

Oltre alla costruzione delle matrici tramite ARRAY, lo SNOBOL prevede un tipo strutturato chiamato TABLE che è una struttura indicizzata unidimensionale di tipo CHIAVE - VALORE.

```
T = TABLE()
```

crea la variabile T di tipo TABLE e con

```
T<NOME> = 'MARIO'
```

ci si riferisce all'elemento A della tabella e assegna ad esso il valore MARIO. Possiamo dire che TABLE è un record, nella moderna tipologia dei linguaggi odierni.

La funzione di costruzione del tipo TABLE prevede due parametri:

```
T = TABLE(M, N)
```

dove M è la dimensione iniziale desiderata e N è l'incremento eventuale che si desidera avere se la dimensione massima è insufficiente.

Ad esempio

```
T = TABLE(20, 10)
```

crea la tabella T con dimensione iniziale di 20 elementi e un incremento progressivo di dieci elementi quando necessario.

Nel box di questa pagina è riportato il codice di un programma che conta le parole in un testo. Esso fa uso di una TABLE, inizialmente definita di 20 caratteri e poi incrementata di 10 elementi all'occorrenza, che sarà indicizzata con la parola del testo e come valore il numero di occorrenze di quella parola nel testo.

Si noti in questo sorgente la semplicità di isolare la parola nel testo; si usa il pattern matching:

```
TOKEN = END . WORD GAP
```

che ha come scopo restituire una parola del testo, usata poi nell'indicizzazione della tabella.

L'istruzione:

```
COUNT          = CONVERT(COUNT, 'ARRAY')
```

Converte la TABLE in un ARRAY, necessario per stampare in output l'elenco delle parole trovate e la relativa occorrenza.

(...continua...)

```
                $ANCHOR     = 1
                SEPARATOR   = ' .,;:!?-'
                END         = BREAK(SEPARATOR)
                GAP         = SPAN(SEPARATOR)
                TOKEN       = END . WORD GAP
                COUNT       = TABLE(20,10)
READ            LINE        = INPUT         : F(PRINT)
                OUTPUT      = LINE
                LINE         GAP  =         : F(PRINT)
NEXTT           LINE        TOKEN =         : F(READ)
                COUNT<WORD>  = COUNT<WORD> + 1 : (NEXTT)
PRINT           OUTPUT      =
                OUTPUT      = 'WORD COUNTS ARE:'
                OUTPUT      =
                COUNT       = CONVERT(COUNT, 'ARRAY') : F(END)
                I           = 1
NEXTC           OUTPUT      = COUNT<I,1> , = , COUNT<I,2> : F(END)
                I           = I + 1
END
```

Geriatric Linux

by Emery Fletcher

Presentazione

Questa personale storia di Emery Fletcher fa parte di una raccolta di articoli denominata "Geriatric Linux", ospitata su OpenSource.org.

Lo scopo della serie è raccogliere le esperienze degli utilizzatori "senior" che si avvicinano a Linux, spesso delusi dalle incongruenze dei sistemi Microsoft, e che trovano nel software libero motivi di impegno e occasione di divertimento.

Vogliamo ospitarne qualcuna sulla nostra rivista come stimolo per chi ha paura a cimentarsi in prima persona con un calcolatore e come testimonianza del valore dell'informatica non precotta, ma quella che si guadagna giorno dopo giorno con l'impegno, la curiosità e la voglia di imparare.

Proprio come succedeva a noi trenta anni e più fa!

My Linux Story

Sono nato nel 1933, un'epoca dove non esisteva ancora il computer, ma nemmeno la TV e l'energia atomica... Nei miei primi settanta anni non ho avuto alcun desiderio di cimentarmi con un calcolatore!

Quando mi ritirai in pensione nel 2002 andai a vivere in una località di montagna distante cinquanta miglia dalla città più vicina. Volendo proseguire in una attività di part-time per la ditta dove avevo lavorato, fui istruito alle basi del computer e cominciai a utilizzarlo.

Avrei voluto approfondire l'utilizzo di questa macchina perché capivo le sue potenzialità quale strumento di collegamento con il mondo, abitando in una località così lontana da un ambiente cittadino. Purtroppo mi resi conto che avrei avuto bisogno di aiuto e non era facile trovarlo nella mia situazione.

Finché non conobbi un vicino di casa che gestiva una attività commerciale di supporto informatico nella città vicina. Accettò di

consigliarmi una macchina adatta alle mie esigenze, di configurarla e di assistermi per un prezzo che era la metà di quanto normalmente chiedeva ai suoi clienti.

Fu così che diventai un felice possessore di un nuovo PC Compaq e utilizzatore del simpatico nuovo sistema operativo Windows XP.

La possibilità di connettersi ad Internet era limitata ad un servizio telefonico a 2 Kbit/sec, ma siccome dovevo inviare solo dei documenti, la cosa non mi disturbava eccessivamente.

Questo durò finché nel tempo libero che la mia attività part-time mi concedeva venni attratto dall'esplorazione di un mondo completamente nuovo per me: Internet.

Avevo sempre pensato che fosse una specie di sala giochi per ragazzini, ma dopo qualche anno realizzai che incredibile risorsa era la Rete e che tipo di potere detenevo sotto le mie dita.

Il computer era diventato un fedele compagno e giorno dopo giorno avevo voglia di imparare sempre di più, anche perché non mi sembrava così misterioso come all'inizio del mio incontro con esso.

Cominciai a leggere molto sul computer e rimasi affascinato da un progetto di sistema operativo che sembrava sfidare tutti i modelli commerciali: Linux.

Mi affascinava l'idea della libertà, del fatto che fosse stato ideato da uno studente liceale e sviluppato da volontari e che venisse usato in ambiti molto diversi e da persone con diverso skill.

Mi sarebbe piaciuto provare ad usarlo ma temevo di rovinare qualcosa sulla mia macchina, che mi serviva anche per lavoro e non potevo permettermi di rischiare di perdere qualcosa o di dover chiedere la reinstallazione daccapo di tutto il sistema e dei programmi (mi sarebbe costato una piccola fortuna!).

Chiesi al mio vicino di casa, con il quale avevo stipulato un contratto di assistenza e che gestiva il mio sistema, se conoscesse Linux.

La sua risposta fu: "So come si scrive".

Capii dal tono in cui lo disse che la pensava come Steve Ballmer: "Linux era il cancro dell'informatica e bisognava starne alla larga".

Giudicai prudente non approfondire oltre: il computer che ormai occupava una parte così importante del mio tempo dipendeva da quel vicino! Era lui che una volta all'anno lo aggiornava e lo rimetteva in ordine per l'ot-

timale funzionamento e era sempre lui che mi risolveva qualsiasi problema mi trovassi ad affrontare, come l'aggiunta di una stampante, l'aggiornamento del programma di scrittura, il collegamento ad Internet con il modem,... insomma tutto!

Giudicai pertanto saggio non irritare questa persona; in fondo il suo business era l'assistenza informatica e viveva grazie ai servizi e al software a pagamento che offriva alla sua clientela. Era logico che non vedesse di buon occhio una filosofia che metteva al centro l'utente e che prometteva di liberarlo del costo e delle pastoie delle licenze e della necessità di pagare degli esperti per continuare a lavorare.

Però, sia perché sono curioso di natura e sia perché dalle letture tecniche che avevo fatto, non volevo lasciar perdere l'approfondimento di quell'argomento. Così ho continuato a leggere su Linux: ho comprato un paio di libri e delle riviste, scoprendo (se la letteratura diceva il vero) che non sembrava poi così difficile installarlo su un PC e proseguire poi con l'utilizzo "normale" della macchina, pur con programmi e comportamenti diversi.

"Prima o poi" - mi ripetevo - "farò questo passo!".

Il momento fatidico venne prima di quanto immaginassi: fu in occasione della pulizia annuale del mio PC (si era nel 2007) che il mio vicino di casa me lo restituì dichiarando che si stava trasferendo e che non avrebbe più potuto seguirmi.

Fui preso dal panico! Come avrei fatto ora, lontano cinquanta miglia dalla città più vicina e senza conoscere nessuno che mi avrebbe aiutato nell'aggiornamento della macchina e nella pulizia annuale? E i virus? Cosa conoscevo dei virus e quando avrei saputo che il computer necessitava di aggiornamento e chi me lo avrebbe eseguito?

Ero convinto che Windows non fosse affatto un sistema semplice: sì, era semplice da usare ma tutt'altro da configurare, gestire e aggiornare! Non era un prodotto "fai da te", insomma!

C'era Linux, lo sapevo e avevo letto moltissimo su di esso, ma ero intimidito: con Windows il messaggio era: "se hai problemi, lasciali all'esperto", con Linux sarebbe diventato: "Sei tu l'esperto!".

Avevo letto abbastanza per dissuadermi dal provare Linux direttamente sul mio PC e non abbastanza sicuro per lanciarmi in partizionamento del disco, menù di boot, etc... (tutte cose che adesso sono banali per me). Tuttavia scoprì presto che un PC analogo al mio, con Pentium 4, da usato valeva quello che pagavo di manutenzione annuale! Così me ne procurai uno nell'aprile del 2009 da destinare alle prove.

Quasi tutti i libri e le riviste su Linux includevano un CD-ROM o un DVD-ROM utile all'installazione con varie distribuzioni. La

mia idea era di procedere all'installazione una ad una di questi sistemi sulla macchina di prova. L'idea era di imparare il più possibile su questo sistema, prima di diventarne un utilizzatore effettivo.

Mentre mi dilettavo con il mio nuovo giocattolo, accadde una cosa che mi mise in crisi: il mio Windows si beccò il virus Conficker. Provai a contattare il sito della Norton Antivirus o qualsiasi altro sito ufficiale di Windows che mi potesse aiutare, ma la mia connessione era troppo lenta e non potevo proseguire per questa strada. Che fare? Mi decisi a prendere la macchina, guidare per cinquanta miglia per arrivare alla biblioteca della città vicina, scaricare un pulitore adatto e rimuovere quindi il fastidioso ospite dal mio sistema.

La cosa si svolse con una naturalezza che non immaginavo! Solo qualche mese prima mi sarei rassegnato a consegnare la macchina ad un negozio e ripassare a prenderla una settimana dopo. Ora invece sapevo cosa andava fatto e lo feci!

Senza volerlo ero diventato l'amministratore del mio Windows!

E' escluso che ci sarei riuscito senza le esperienze accumulate nell'installazione delle distribuzioni Linux.

Quando qualche mese dopo arrivò la possibilità di una connessione DSL, fui in grado di configurare tutto da solo sia su Windows che su Linux.

Cominciai ad interessarmi anche di hardware e di aggiornamento delle mie ormai diventate 6 macchine con nuova RAM, schede grafiche più potenti, etc...

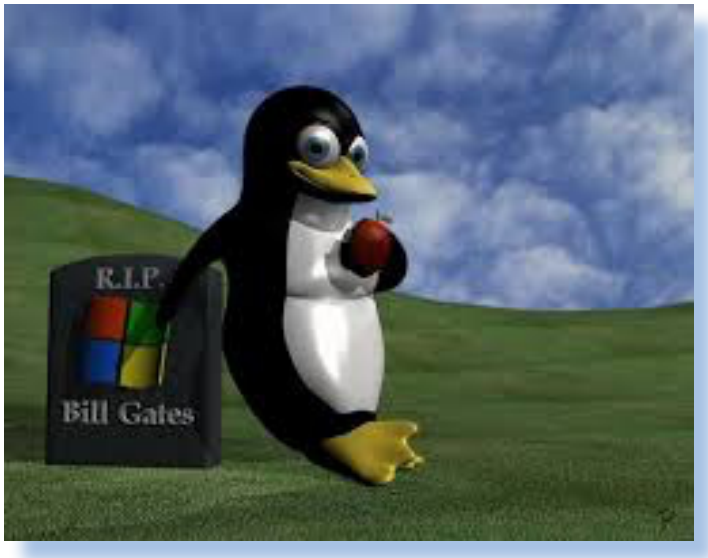
Per il mio 79^ compleanno, nel 2012, mi regalai i pezzi per costruirmi da solo un computer con CPU a 8 core, una scheda grafica "da paura" e tanta, tantissima RAM!

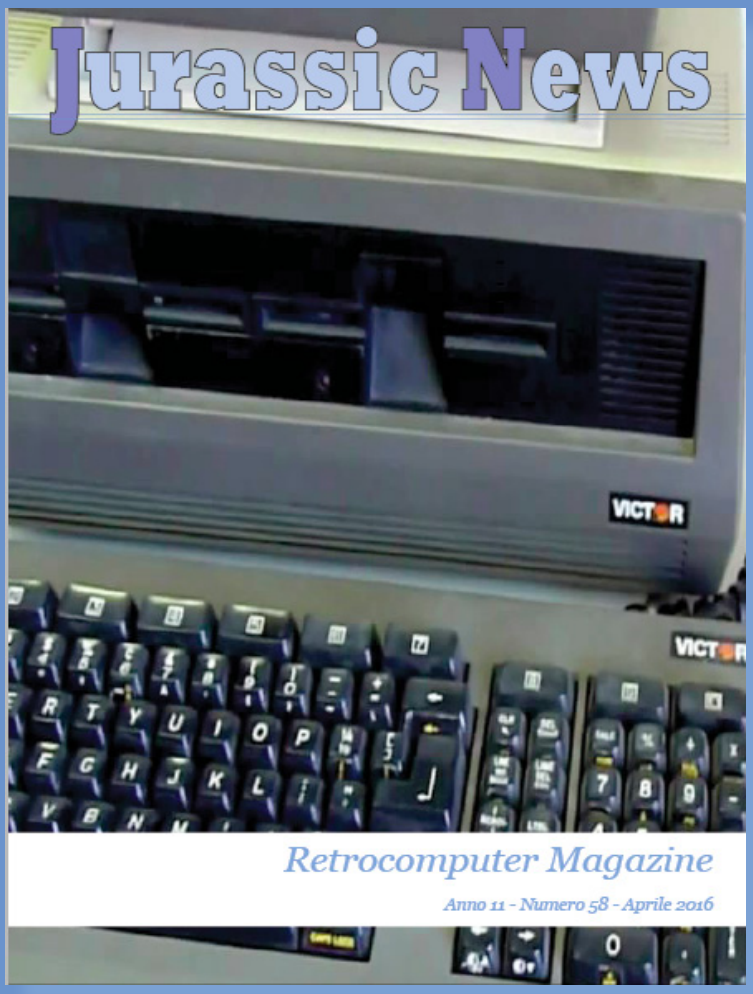
Ben presto rimossi quello che restava di Windows sul Compaq, ormai vecchio di 12 anni, che ora esegue una tranquilla distribuzione MX-14 e lo uso come server casalingo.

E' incredibile quanto ho imparato con Linux dal 2003 in poi! Invece di usare il PC passivamente, come un televisore o come Windows vuole che tu lo utilizzi, ora ho un ruolo attivo e non permetterò a Microsoft o a qualcun altro di influenzare le mie decisioni informatiche!

E' stato l'inizio del mio pensionamento attivo!

(=)





Nel prossimo numero l'ultima creatura di Chuck Peddle: il sistema professionale Sirius-1/Victor 9000