Il Manchester Baby

G. Lettieri

A/A 2017/18

I calcolatori moderni, nell'incessante sforzo di diventare sempre più efficienti, hanno accumulato una gran quantità di meccanismi molto complessi. Allo stesso tempo, una enorme pila di software è stata interposta tra il calcolatore e l'utente finale, allo scopo di semplificarne l'utilizzo. Questi sviluppi portano fuori strada chi si agginge oggi a studiare l'architettura di un elaboratore:

- lo studente parte essendosi già formato strane idee sul funzionamento interno del calcolatore, di cui ha esperienza solo attraverso il filtro del software moderno;
- quando si confronta con la complessità dell'architettura sottostante, non ne intuisce lo scopo e finisce per perdersi nel mare dei dettagli.

Ricordiamo che i calcolatori nascono, come dice il nome, per *calcolare*. Inizialmente si tratta di calcoli scientifici, e subito dopo anche commerciali. La loro caratteristica principale è di essere *programmabili*: la procedura di calcolo deve poter essere cambiata senza modificare l'hardware.

Anche se oggi le usiamo per tantissimi altri scopi, quando ne studiamo la struttura dobbiamo sempre tenere in mente questo: sono macchine che hanno lo scopo di eseguire programmi, e questi programmi servono a svolgere calcoli.

Per fissare più chiaramente i principi di base, che non sono cambiati molto dalla loro introduzione negli anni '40 del secolo scorso, può essere utile tornare, almeno brevemente, alle origini. Elimineremo in questo modo tutte le fonti di confusione che si sono accumulate nel tempo.

1 Il calcolatore SSEM

Lo Small-Scale Experimental Machine (Figura 1), anche noto come "Manchester Baby", fu costruito all'Università di Manchester nel 1948. Si tratta del primo prototipo di calcolatore elettronico (a valvole) funzionante secondo il principio del *programma memorizzato*. Tale principio fu introdotto dal gruppo che costruì l'ENIAC negli Stati Uniti e fu reso noto da una serie di articoli scritti dal matematico John von Neumann, consulente del progetto. Anche se è quasi unanimemente considerato un punto di svolta nell'evoluzione dei calcolatori, il concetto di programma memorizzato non è in realtà definito precisamente. Per

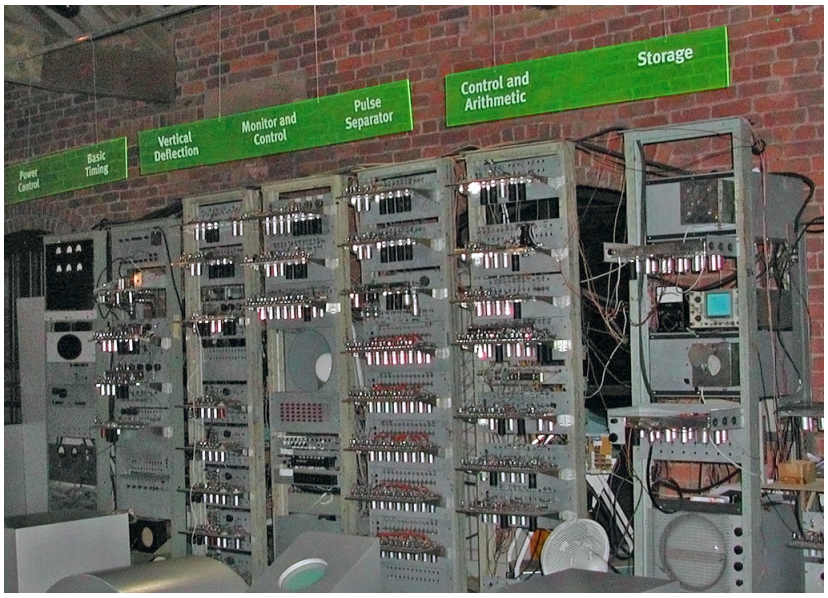


Figura 1: Una replica moderna del SSEM, soprannominato "Baby", in esposizione al *Museum of Science and Industry* di Castlefield, Manchester. (fonte: Wikipedia)

i nostri scopi è sufficiente dire che in un calcolatore a programma memorizzato il programma è codificato numericamente e memorizzato nella stessa memoria in cui si trovano anche i dati da elaborare.

Un simulatore Java del Baby può essere scaricato liberamente da `http://www.davidsharp.com/baby/`. Il Baby era soltanto un piccolo prototipo, non realmente utilizzabile per calcoli seri. Fu costruito allo scopo di testare una nuova tecnologia di memoria basata sui tubi catodici. In questa tecnologia, ormai non più in uso, i bit erano memorizzati come punti o linee su uno schermo fluorescente: venivano scritti dirigendo opportunamente un raggio catodico, e riletti tramite una griglia metallica che copriva lo schermo. Questa tecnologia offre un'interessante opportunità: se il segnale che pilota il raggio catodico della memoria viene inviato anche ad un normale schermo a tubo catodico (come quello dei vecchi televisori), diventa possibile osservare direttamente il contenuto della memoria stessa. La figura Figura 2 mostra il dettaglio dello schermo utilizzato proprio a questo scopo nel Baby. La Figura 3, invece, mostra lo schermo del Baby riprodotto sul simulatore. La memoria consiste soltanto di 32 locazioni, ciascuna di 32 bit. Una locazione occupa una intera riga della matrice. Ogni locazione è identificata da un numero da 0 a 31: quella più in alto è la 0 e le altre sono numerate a seguire. Questi identificatori, già allora chiamati *indirizzi*, non sono visibili da nessuna parte. Il dispositivo di memoria, però, dato un indirizzo, è in grado di selezionare la corrispondente riga, permettendo di

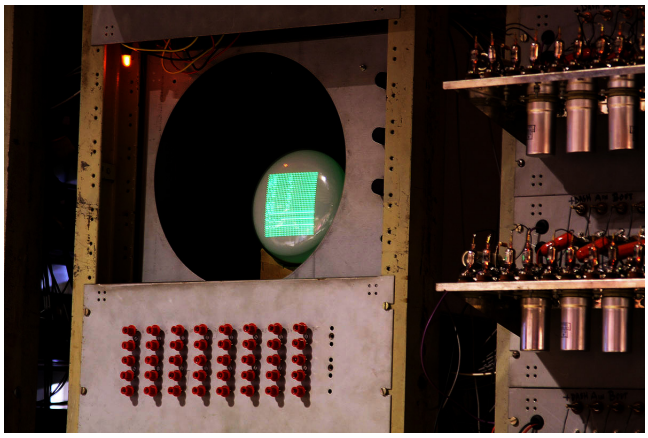


Figura 2: L'output su CRT (*Cathode Ray Tube*, tubo catodico). Lo schermo mostra il contenuto della memoria come una matrice di 32×32 punti o linee (fonte: Wikipedia)

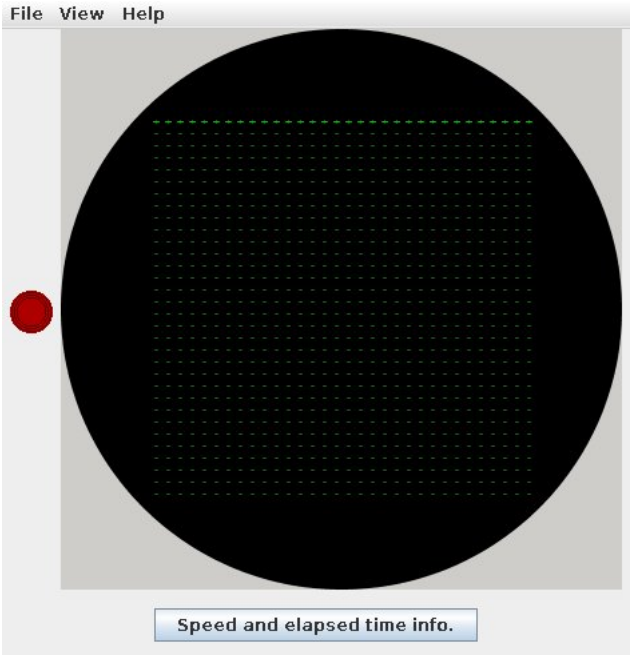


Figura 3: La memoria della SSEM come mostrata dal simulatore.

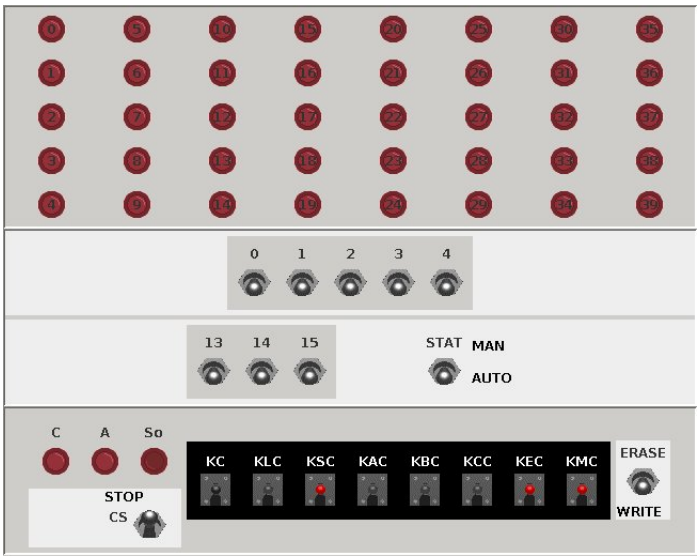


Figura 4: Il pannello di controllo della SSEM nel simulatore.

leggerne il contenuto (converire i corrispondenti punti e linee in impulsi elettrici) o cambiarne il contenuto (convertire impulsi elettrici in punti e linee).

Ogni locazione può memorizzare indifferentemente un numero intero (rappresentato in complemento a 2 su 32 bit) oppure una istruzione. Niente, nella memoria, permette di distinguere un numero da una istruzione: il significato dei bit dipende solo dall'uso che se ne fa. Tutto ciò è vero anche per le memorie moderne. Anzi, l'immagine di Figura 3 è un buon modo di visualizzare la memoria di un qualunque calcolatore. Su una sola cosa la memoria del Baby è un po' particolare: in ogni locazione, il bit meno significativo si trova a sinistra.

L'ingresso/uscita della macchina è ridotto al minimo. Le uniche uscite sono lo schermo stesso e la lampadina visibile in alto a sinistra in Figura 2, rappresentata in rosso in Figura 3. Lo schermo permette di vedere tutto il contenuto della memoria (e anche dei registri, come vedremo), mentre la lampadina si accende quando la macchina si ferma, presumibilmente perché il programma è terminato. L'unico dispositivo di ingresso è dato dal pannello visibile parzialmente in Figura 2, subito sotto lo schermo, e mostrato in Figura 4 come riprodotto nel simulatore. A parte alcuni interruttori per cancellare la memoria e i registri e avviare o fermare l'esecuzione, notiamo le cinque file di bottoni rossi in alto e i cinque interrutori numerati da 0 a 4 subito sotto. Questi 5 interruttori permettono di selezionare una riga qualsiasi della memoria scrivendone il numero, compreso tra 0 e 31, in binario. Anche qui il bit meno significativo è a sinistra. Con la levetta in basso a destra impostata su WRITE, i primi 32 bottoni rossi permettono di accendere il corrispondente bit della riga selezionata (i bottoni da 33 a 39 non servono a niente). Se la levetta è impostata su ERASE, gli stessi bottoni permettono invece di azzerare il corrispondente bit.

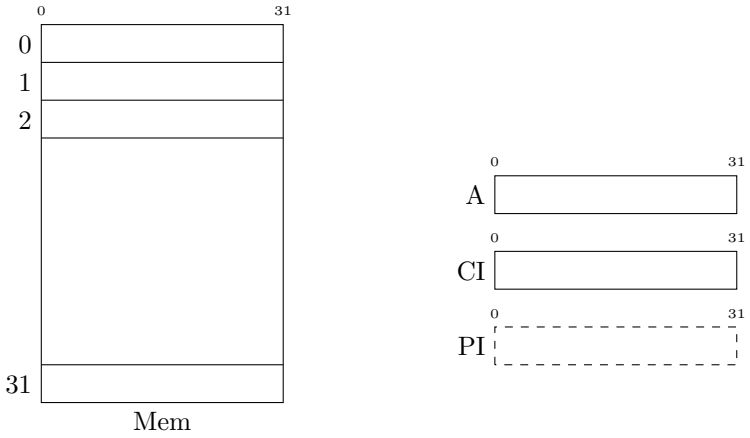


Figura 5: La memoria (a sinistra) e i registri (a destra) del SSEM.

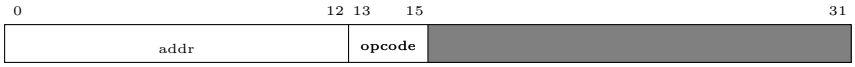
Il fatto che il Baby fosse solo un piccolo prototipo lo rende ideale per il nostro esempio, in quanto ci permette di eliminare un gran numero di distrazioni: abbiamo tutta la memoria sott'occhio e possiamo direttamente accendere o spegnere tutti i bit che vogliamo. Per quanto riguarda la memoria, dunque, non ci sono segreti.

Il Baby è ridotto all'osso anche per quanto riguarda il supporto alla programmazione. Le Figure 5 e 6 mostrano tutto ciò che il programmatore deve sapere per programmare la macchina (le informazioni equivalenti per un moderno processore Intel si trovano in una serie di volumi di quasi 4000 pagine in totale). La Figura 5 mostra la memoria, che già conosciamo, e i tre registri disponibili: un accumulatore (A), il contatore di programma (CI) e un registro non utilizzabile direttamente dal programmatore, destinato a contenere l'istruzione che la macchina sta eseguendo (PI, *Present Instruction*). I registri sono realizzati con la stessa tecnologia della memoria, anche se consistono ciascuno di un'unica riga. I bottoni C, A e So in basso a sinistra in Figura 4 permettono di vedere sullo schermo il contenuto di CI e PI (bottone C), di A (bottone A) o della memoria (bottone So).

Quando l'interruttore STOP/CS è impostato su CS, la macchina:

1. incrementa CI di 1;
2. legge dalla memoria la locazione di indirizzo CI e la copia in PI;
3. esegue l'istruzione contenuta in PI;
4. se l'istruzione non era di stop, torna al punto 1; altrimenti, accende la lampadina e non fa altro.

Se una istruzione non modifica il contenuto di CI (al punto 3), la macchina esegue in sequenza le istruzioni contenute in memoria, grazie all'auto-incremento



opcode	mnemonico	*effetto*
000	JMP	$\mathsf{CI} \leftarrow \mathsf{Mem}[\mathsf{addr}]$
100	JRP	$\mathsf{CI} \leftarrow \mathsf{CI} + \mathsf{Mem}[\mathsf{addr}]$
010	LDN	$\mathsf{A} \leftarrow -\mathsf{Mem}[\mathsf{addr}]$
110	STO	$\mathsf{Mem}[\mathsf{addr}] \leftarrow \mathsf{A}$
-01	SUB	$\mathsf{A} \leftarrow \mathsf{A} - \mathsf{Mem}[\mathsf{addr}]$
011	CMP	*se* $\mathsf{A} < 0$, $\mathsf{CI} \leftarrow \mathsf{CI} + 1$
111	STP	*halt*

Figura 6: Il formato e l'insieme delle istruzioni del SSEM.

di CI (punto 1). La macchina è un po' particolare, dal momento che l'incremento di CI avviene in ogni caso, prima di prelevare l'istruzione. In particolare, se all'accensione CI contiene zero, la prima istruzione che verrà eseguita sarà quella che si trova all'indirizzo 1. A parte alcuni dettagli (come quest'ultimo), un processore moderno si comporta essenzialmente nello stesso modo del processore del Baby, da quando lo accendiamo a quando lo spegniamo.

Le possibili istruzioni sono solo 7 e sono mostrate in Figura 6. In alto in figura è mostrato anche il *formato* delle istruzioni, cioè come queste devono essere codificate in memoria affinchè la macchina sia in grado di interpretarle. Abbiamo detto che ogni istruzione occupa una riga (32 bit), ma in realtà i 16 bit più significativi sono ignorati. Dei 16 bit meno significativi, i primi 12 possono contenere un indirizzo di memoria (*addr*). Dal momento che la memoria è di sole 32 locazioni, solo i primi 5 bit di *addr* sono realmente utilizzati e gli altri sono ignorati. I bit 13, 14 e 15 identificano l'istruzione da eseguire (*opcode*, codice operativo). L'istruzione 010 (LDN) legge la cella di memoria di indirizzo *addr* e la copia nel registro A. Si noti che ne cambia anche il segno, che è un'altra peculiarità di questa macchina. L'operazione 110 fa la copia nell'altro verso (ma senza cambiare il segno). I codici 001 e 101 sono utilizzati per la stessa istruzione, che esegue una sottrazione tra il contenuto di A e il contenuto della locazione di memoria puntata da *addr*, scrivendo il risultato in A. Si noti che le istruzioni permettono di scrivere o leggere solo intere locazioni (intere righe, quindi). In particolare, non esistono i byte.

Le istruzioni 000, 100, 011 e 111 permettono di alterare il flusso di esecuzione delle istruzioni. L'istruzione 111 semplicemente lo arresta, in modo che l'utente possa osservare il risultato del programma. Le istruzioni 000 e 100 scrivono un nuovo valore in CI, e sono dunque dei salti incondizionati: 000 è assoluto e 100 è relativo al valore corrente di CI. L'istruzione 011 è molto importan-

istr.	commento
LDN X	carichiamo X, ottenendo $-x$ nell'accumulatore
CMP	se $-x$ è negativo, x era positivo e non dobbiamo fare niente
STO X	altrimenti sostituiamo x con $-x$
STP	fermiamo la macchina

Figura 7: Un esempio di programma. La locazione X deve contenere un valore (x) che alla fine deve essere sostituito dal suo valore assoluto.

te: decide di saltare o meno l'istruzione successiva in base al segno del numero contenuto in A. La possibilità di eseguire azioni diverse in base al risultato di elaborazioni precedenti è ciò che permette alla macchina di eseguire potenzialmente qualunque algoritmo (avendo a disposizione sufficiente memoria). Anche se le istruzioni del Baby hanno alcune peculiarità (la decisione di avere solo un caricamento con cambio di segno, il salto condizionale che permette di saltare una sola istruzione), sono comunque rappresentative del tipo di istruzioni che si trovano comunemente in tutti i processori.

Non c'è altro da sapere sul processore del Baby, e dunque anche qui non ci sono segreti. Dato lo stato iniziale della memoria, tutto ciò che la macchina farà è completamente determinato.

2 Un esempio di programmazione

Programmare la macchina significa decidere lo stato iniziale della memoria, che conterrà sia il programma, sia i dati che il programma dovrà elaborare.

Proviamo a scrivere un programma molto semplice. Supponiamo di avere un numero x, memorizzato in una locazione X. Vogliamo scrivere un programma che sostituisca x con il suo valore assoluto.

Il dato da elaborare, in questo caso, è il solo numero x. Il programma può essere scritto in tanti modi. Un modo è il seguente: se carichiamo x nell'accumulatore, il Baby cambierà anche il segno. Se abbiamo ottenuto un numero negativo, vuol dire che x era positivo, e dunque già uguale al suo valore assoluto. Se invece otteniamo un numero positivo, vuol dire sia che x era negativo, sia che ora abbiamo il suo valore assoluto nell'accumulatore. Ci basta dunque scrivere il contenuto dell'accumulatore all'indirizzo X, e abbiamo terminato.

Questa idea si traduce nel programma mostrato in Figura 7.

Per fare in modo che la macchina lo esegua, dobbiamo fare tre cose:

1. decidere dove mettere il programma e i dati all'interno della memoria;
2. tradurre tutto (programma e dati) in binario;
3. inizializzare la memoria con i bit ottenuti al passo precedente.

Occupiamoci del punto 1. Ricordiamo che il processore eseguirà per prima l'istruzione che si trova all'indirizzo 1, e quindi procederà automaticamente agli

indirizzi successivi, se non incontra istruzioni di salto. Conviene dunque caricare il nostro programma a partire dall'indirizzo 1 fino all'indirizzo 4. Il nostro programma opera su un solo dato, il numero che si trova alla locazione X. Dobbiamo decidere dove allocare questo dato. Le locazioni di memoria sono tutte equivalenti, quindi possiamo sceglierne una qualunque, con l'unico vincolo di non sceglierne una già usata per qualcos'altro. Nel nostro caso le uniche locazioni già occupate saranno quelle da 1 a 4. Scegliamo, per esempio, $X = 20$. All'indirizzo 20 dovremo caricare il numero x di cui vogliamo calcolare il valore assoluto. Il nostro programma dovrebbe poter funzionare con un qualunque x, ma per eseguirlo dobbiamo sceglierne uno. Scegliamo, per esempio, $x = -1$.

Ora passiamo al punto 2. Per tradurre il programma in binario, esaminiamo ogni istruzione utilizzando la tabella di Figura 6. La prima istruzione da convertire è "LDN 20". In binario, 20 è 1010_2 $(16 + 4)$, e dunque i bit 0–4 dell'istruzione dovranno valere 01010 (si ricordi che in questa macchina il bit meno significativo va a sinistra, quindi il numero appare al contrario rispetto a come siamo abituati). Dalla tabella di Figura 6 vediamo che il campo *opcode* (bit 13–15) deve valere 010. Gli altri bit della riga verranno ignorati e potremmo assegnarvi un valore qualsiasi, ma non abbiamo motivo per assegnarvi un valore diverso da 0. Si passa poi all'istruzione "CMP", che non usa il suo campo *addr*, quindi dobbiamo preoccuparci solo dei bit 13–15, che devono valere 011. Procediamo così anche per le altre due istruzioni, ottenendo la tabella seguente:

riga	0	1	2	3	4	13	14	15
1	0	1	0	1	0	0	1	0
2						0	1	1
3	0	1	0	1	0	1	1	0
4						1	1	1

Dobbiamo anche convertire il numero x. Il valore -1, rappresentato in complemento a 2 su 32 bit corrisponde a 32 bit tutti pari a 1.

Terminiamo infine con il punto 3. Per caricare la riga 1 nella memoria del Baby, supponendo che inizialmente sia copletamente azzerata, selezioniamo la riga 1 tramite le levette centrali (levetta 0 in basso, le altre in alto). Quindi, con la levetta "ERASE/WRITE" impostata su "WRITE", premiamo i bottoni rossi numero 1, 3 e 14. Procediamo così per tutte le altre righe, ottenendo la configurazione di memoria illustrata in Figura 8.

A questo punto possiamo avviare l'esecuzione (levetta "STOP/CS" su "CS") e, se non abbiamo commesso errori, si accenderà la lampada di stop e alla locazione 20 dovremmo osservare il valore 1 (un solo bit acceso a sinistra) al posto di -1.

Questa laboriosa procedura deve essere eseguita in modo sostanzialmente equivalente anche quando si programma un calcolatore moderno. La differenza principale consiste nel fatto che possiamo utilizzare dei programmi già pronti (assemblatore, collegatore e caricatore) per automatizzare le varie fasi.

0	00000000000000000000000000000000	
1	00101000000000100000000000000000	LDN 20
2	00000000000000110000000000000000	CMP
3	00101000000001100000000000000000	STO 20
4	00000000000001110000000000000000	STP
5	00000000000000000000000000000000	
6	00000000000000000000000000000000	
7	00000000000000000000000000000000	
8	00000000000000000000000000000000	
9	00000000000000000000000000000000	
10	00000000000000000000000000000000	
11	00000000000000000000000000000000	
12	00000000000000000000000000000000	
13	00000000000000000000000000000000	
14	00000000000000000000000000000000	
15	00000000000000000000000000000000	
16	00000000000000000000000000000000	
17	00000000000000000000000000000000	
18	00000000000000000000000000000000	
19	00000000000000000000000000000000	
20	11111111111111111111111111111111	-1
21	00000000000000000000000000000000	
22	00000000000000000000000000000000	
23	00000000000000000000000000000000	
24	00000000000000000000000000000000	
25	00000000000000000000000000000000	
26	00000000000000000000000000000000	
27	00000000000000000000000000000000	
28	00000000000000000000000000000000	
29	00000000000000000000000000000000	
30	00000000000000000000000000000000	
31	00000000000000000000000000000000	

Figura 8: Programma di Figura 7 caricato in memoria. Abbiamo scelto $X = 20$ e $x = -1$.

In genere questi programmi sono eseguiti sul calcolatore stesso su cui poi verrà eseguito anche il programma finale. Questa circostanza, sicuramente molto comoda, genera però gran confusione negli studenti alle prime armi, che sono indotti a pensare che l'assemblatore (o, più tipicamente, il compilatore) continui a fare qualcosa anche mentre il programma finale è in esecuzione, oppure che per eseguire un programma sia necessaria la contemporanea presenza di tutti questi strumenti. L'unica cosa che conta, invece, ai fini dell'esecuzione di un programma, è come è stata inizializzata la memoria al suo avvio (Figura 8). Per avere le idee un po' più chiare, è molto meglio immaginare che tutta l'operazione di traduzione e caricamento venga svolta a mano, come abbiamo fatto ora, anche quando usiamo compilatori, assemblatori, collegatori, caricatori e persino ambienti integrati di sviluppo.