ALMA MATER STUDIORUM • UNIVERSITÀ DI BOLOGNA

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

La nascita dell'Informatica Moderna:

le prime macchine e il linguaggio di programmazione

Tesi di laurea in

Storia dell'Informatica e dei dispositivi di calcolo

Relatore: Chiar.mo
Prof. Giorgio Casadei

Presentata da:
Gregorio Giacchetti

Sessione III

Anno accademico

2020-2021

“ They used to say that if Man was meant to fly, he’d have wings.

But he did fly. He discovered he had to. ”

– Capt. James T. Kirk

Indice

1.Introduzione

I primi calcolatori elettromeccanici sono stati introdotti sul mercato tra le due guerre mondiali come successivi aggiornamenti della selezionatrice di Hollerith sviluppata per elaborare i dati del censimento del 1890 in USA. L'elettronica è stata usata per la prima volta in modo parziale da Atanassov per realizzare un contatore di eventi in un esperimento di fisica nel 1942; solo col progetto di John Mauchly e Presper Eckert l'elettronica è stata usata in modo definitivo per costruire il calcolatore ENIAC. Tuttavia questo dispositivo era ancora incompleto in quanto non disponeva di un vero e proprio linguaggio macchina, infatti la definizione completa dei calcoli da eseguire richiedeva un aggiornamento parziale della struttura fisica. Per superare questo limite di ENIAC e per migliorarne le prestazioni, Mauchly ed Eckert avevano ipotizzato di costruire una macchina successiva. Venuto a conoscenza di questo progetto, von Neumann ha messo insieme tre argomenti rilevanti: le sue conoscenze di cibernetica (modello di neurone di Mc Culloch e Pitts e le potenzialità del tubo elettronico), la logica e la calcolabilità (risultati di Goedel e Turing) e l'esperienza di fisico-matematico maturata con la partecipazione al progetto Manhattan. In un report reso noto il 30 giugno del 1945, ha descritto a grandi linee il progetto del computer EDVAC. Il contributo

fondamentale, innovativo e originale di questo progetto consiste nella definizione dettagliata del linguaggio macchina che contiene due caratteristiche fondamentali: la Turing completezza e la presenza di comandi pensati per il problem solving; questa seconda caratteristica dimostra che von Neumann ha avvertito l'esigenza di disporre di uno strumento flessibile per rendere efficace ed efficiente l'attività di programmazione. La macchina che rappresenterà al meglio l'idea dell'architettura di von Neumann, sia nel corpo che nell'anima, sarà la macchina IAS, inaugurata ufficialmente nel 1952.

In questa tesi mi propongo di ricostruire le esperienze da cui ha preso corpo l'attività di programmazione negli anni in cui era presente il linguaggio di programmazione ma non era ancora disponibile il computer.

La prima parte, compilativa, prevede una descrizione degli sviluppi tecnologici che hanno portato alla creazione della scienza chiamata Informatica moderna e degli strumenti annessi; nella seconda parte viene affrontata sperimentalmente la progettazione di un algoritmo di ordinamento di n interi su una macchina di von Neumann, riproducendo la procedura formale indicata dallo scienziato.

2. ENIAC e l’arrivo dell’elettronica

La prima domanda che sorge spontanea quando si parla di Informatica è: ”qual è stato il primo computer?”

Questo quesito ha dato luce a tante discussioni, legali e non, in tutti i luoghi possibili, in particolar modo nei forum di Internet. Non esiste una risposta unica in quanto ognuna è relativa alla definizione data alla parola “computer”. Negli anni ‘40 questa domanda non avrebbe potuto avere senso perché con questa parola si intendeva una persona con la responsabilità di dover completare calcoli complessi; tant’è vero che le macchine di quel periodo venivano chiamate calcolatori automatici o macchine per calcoli.

È per questo motivo che gli studiosi tendono ad essere d’accordo nel definire l’inizio dell’Informatica moderna con la prima operazione effettuata dall’ENIAC, ovvero nel 1945. Questa decisione venne presa durante gli anni ’80, dove storici esperti raggiunsero un punto d’incontro stabilendo una serie di aggettivi che dovevano qualificare il “primo calcolatore” tra i molti nati negli anni ’40: “elettronico, general purpose e programmabile”, aggettivi che caratterizzarono perfettamente ENIAC.

Con il progetto ENIAC vennero introdotti vocaboli tecnici, tuttora in uso, come programmazione, programmi e le funzioni di automazioni di

alto livello come loop e ramificazione. Venne pubblicizzata in tutto il mondo, stimolando l'interesse per i calcolatori elettronici. I due designers principali fondarono la prima compagnia di computer elettronici e, prima ancora che la costruzione dell'ENIAC fosse finita, già si pensava al design del suo successore, l'EDVAC, il quale definirà l'architettura di tutte le macchine a venire.

Durante la grande depressione, John Mauchly, dottorando in fisica, stava affrontando una significativa ricerca, con considerevole mole di lavoro, presso un piccolo college. Uno dei suoi interessi era di utilizzare l'elettronica per contare e fare calcoli. L'occasione venne quando l'Università di Pennsylvania inaugurò, con l'aiuto del governo, una speciale classe in elettronica affinché giovani menti collaborassero per idee utili per la guerra in corso. Mauchly si distinse e riuscì a guadagnarsi una posizione di rilievo. Notò che centinaia di donne erano adoperate per il complesso calcolo di tabelle di fuoco, utili per determinare le variabili per la corretta traiettoria dei proiettili d'artiglieria ad una certa distanza. La creazione di una sola tabella poteva prendere mesi di lavoro a causa dei continui test pratici necessari, rendendo il ritmo molto difficile da mantenere. Questa circostanza diede a Mauchly l'occasione perfetta per giustificare la costruzione di un calcolatore automatico in grado di descrivere la traiettoria del proiettile, accelerando i tempi di creazione delle tabelle.

Fedele collaboratore fu Presper Eckert, Dottore in ingegneria e maggiore responsabile per le innovazioni dal punto di vista architetturale. Difatti, come disse Mauchly: “i calcoli possono essere eseguiti ad alta velocità solo se le istruzioni sono fornite ad alta velocità” ed insieme tracciarono una strada alternativa ai vecchi e lenti calcolatori elettromeccanici. Con un utilizzo intensivo di valvole termoioniche, relè ed elettricità (il primo avvio di ENIAC causò un blackout ad un intero quartiere di Filadelfia) venne fatto l’importante passo verso il calcolatore completamente elettronico, di gran lunga più veloce dei suoi predecessori per via dell’enorme riduzione dell’intervento umano necessario. Si ridefinì il concetto di programmazione: se prima si intendeva uno stabilire un “programma” delle istruzioni da fare, ovvero una sequenza di azioni da far compiere (per esempio un programma di un concerto musicale indicava i pezzi musicali da eseguire) ora con lo stesso termine ci si riferisce all’atto di creare un programma per il calcolatore, ovvero una particolare configurazione della macchina per ottenere le operazioni necessarie per un compito. ENIAC non era il primo computer programmabile ma ebbe il primato di automatizzare la scelta di cosa fare dopo una sequenza finita di operazioni, evitando continui interventi umani che rallentavano il processo di soluzione. Per esempio, il calcolatore era in grado di ripetere una procedura numerica finché non si otteneva un

risultato abbastanza preciso per poi, automaticamente, passare al prossimo passo del calcolo. La capacità di selezione automatica tra differenti strade precostruite in base ai valori ottenuti nei calcoli precedenti verrà chiamata “ramificazione (conditional branching)” e sarà una principale caratteristica dei computer moderni.

Il distacco dalle vecchie macchine elettromeccaniche si evidenziò anche per quanto riguarda lo scopo del calcolatore: se prima il manufatto doveva essere strettamente utilizzato per uno specifico compito (special purpose), ora, con ENIAC, si poté ambire a macchine in grado di risolvere svariati compiti se configurati in maniera appropriata, aprendo la strada ai computer general purpose.

Anche se i nomi più rinomati e ricordati per ENIAC rimasero Eckert e Mauchly lo staff dietro questo manufatto è stato più numeroso di quanto si possa immaginare. Ad oggi, ENIAC viene ricordato come una macchina programmata da donne, per la precisione da sei donne: Kay McNulty, Betty Jennings, Betty Snyder Holberton, Marlyn Wescoff, Fran Bilas e Ruth Lichterman. Esse divennero famose come “le donne dell’ENIAC” e sono spesso riferite come le prime programmatrici, anche se non rispecchia appieno il loro vero lavoro. I dati di input e output venivano utilizzati e memorizzati attraverso schede perforate, le quali venivano gestite dal team di esperte tramite macchine IBM fatte ad

hoc. Con lavori di grandi dimensioni si potevano avere migliaia di schede intermedie che dovevano essere perforate, inserite e lette per poter comprendere l'output dei calcoli. Successivamente, seguendo degli apposti diagrammi di configurazione della macchina, riposizionavano l'hardware in modo da passare al problema successivo. Per questa loro approfondita esperienza di ENIAC venivano spesso chiamate per aiutare gli ingegneri e gli scienziati a formulare i problemi come sequenze di operazioni in ENIAC e successivamente a trasformare queste sequenze in diagrammi di configurazione.

Già nel '44, un anno prima che ENIAC compiesse la sua prima operazione, i suoi creatori stavano già stilando idee per il modello successivo. Compresero i limiti di ENIAC e proposero un nuovo schema basato su un nuovo tipo di memoria. Tale progetto prese il nome di EDVAC e attirò l'attenzione di un famoso matematico, John von Neumann, il cui nome divenne indelebile nella storia moderna del computer per via della sua descrizione, non tanto dell'hardware, ma della logica dietro la macchina. Tale scritto prese il nome di "First draft of a report on the EDVAC".

3. Passaggio da ENIAC a EDVAC

Con la descrizione della macchina EDVAC l'architettura di von Neumann vide la luce, portando con sé innovazioni che plasmarono la futura concezione di calcolatore.

Per raggiungere il suo obiettivo von Neumann si servì di competenze tecniche che aveva acquisito nel corso degli anni, grazie alla sua innata curiosità nello studiare diverse materie, in particolare:

1. Conoscenza di sistemi complessi organizzati in organi elementari derivabile da studi ed esperienze di cibernetica.
2. Conoscenza di linguaggi effettivi, semplici e potenti dal punto di vista espressivo derivati dagli studi di logica e calcolabilità.
3. Conoscenza di problemi complessi affrontabili con adeguate capacità computazionali derivate dalla necessità di risolvere sistemi di equazioni differenziali.

Grazie a queste conoscenze, lo scienziato poté realizzare la trasformazione di ENIAC in un vero e proprio computer, ovvero EDVAC.

Dal campo emergente della cibernetica, disciplina che asserisce l'equivalenza tra cervello e meccanismi meccanici di controllo, von Neumann utilizzò vocaboli del linguaggio tecnico della biologia descrivendo, per esempio, i circuiti di controllo dell'EDVAC come

neuroni, dividendo la struttura della macchina in organi e utilizzando una memoria (l'unico termine che rimarrà fino ai giorni nostri) per i programmi e i dati, i quali venivano memorizzati in segmenti di 32 bit chiamati parole.

Dalle attività di ricerca sui problemi di metamatematica formalizzati da Hilbert e dalla proposta di Turing sulla calcolabilità nasce l'idea di una macchina dotata di un linguaggio Turing completo. Infatti, con ENIAC, il programma era "costruito" con l'hardware in quanto si tratta di una macchina a programma cablato, dove la particolare struttura fisica della macchina ne determinava l'utilizzo e la memoria era data da dispositivi fisici esterni. Non era un vero e proprio calcolatore, ma piuttosto un kit di 40 moduli che potevano essere assemblati in modo diverso in funzione del tipo di calcolo da eseguire: preparare un nuovo programma per ENIAC equivaleva a realizzare un nuovo assemblaggio di questi moduli. Questo portava ad innumerevoli difficoltà e limitazioni, come la manutenzione (e l'affidabilità) dei componenti che formavano la macchina e la complessa gestione dei cablaggi e dei circuiti, indispensabile per poter creare e modificare il programma.

Il report di von Neumann sulla macchina EDVAC definì quello che mancava a ENIAC: un linguaggio macchina. Non un linguaggio macchina qualsiasi, bensì uno derivato direttamente dalla macchina virtuale di

Turing, rendendolo quindi Turing completo. La creazione di questo manufatto (macchina reale universale) fu resa possibile grazie all'idea di memoria elettronica riscrivibile, facendo nascere l'idea di computer a programma memorizzato, dove era possibile contenere contemporaneamente il programma che doveva essere eseguito e i dati manipolati: preparare un nuovo programma per EDVAC equivaleva a scrivere il programma in linguaggio macchina e copiarlo in memoria, rendendolo a tutti gli effetti il prototipo di computer. La Turing completezza era data dalla possibilità di sostituzione dei simboli, permettendo la modifica sia dei dati che del programma senza intervenire nell'hardware, cambiando il "software" per acquisire la capacità espressiva che ha la macchina virtuale di Turing e per essere modellato in funzione del suo utilizzo.

Linguaggio macchina "derivato" da quello di Turing perché von Neumann trovò un altro vantaggio nel descrivere il linguaggio macchina: sebbene i comandi descritti da Turing avessero definito la portata massima della calcolabilità dei problemi non permetteva "scorciatoie" per la risoluzione di problemi, facili o difficili che fossero. Dalle esperienze di calcolo numerico connesse in particolare alla partecipazione al progetto Manhattan, nasce l'idea di un linguaggio di programmazione per rendere trattabili problemi di calcolo numerico e

non solo la soluzione di equazioni differenziali, proponendo un set di istruzioni ad hoc. Il calcolatore reale doveva avere a disposizione comandi ulteriori per facilitare la programmazione di soluzioni ai problemi presentati alla macchina. Per esempio, una macchina non sarà efficiente se per fare aritmetica deve ricorrere esclusivamente alle manipolazioni di simboli come definita da Turing. Von Neumann capì questo problema ed incluse nel linguaggio macchina di EDVAC le operazioni aritmetiche, in modo che la macchina fosse in grado di risolvere tutti i calcoli del linguaggio di Turing ed altre operazioni applicative, aprendo al mondo la strada all'idea di linguaggi di alto livello adeguati al problema che deve essere risolto.

Von Neumann, con il suo lavoro, separò l'hardware dal software del calcolatore, facendo nascere la figura del programmatore separata da quella del progettista del manufatto, dando a questa nuova figura lo strumento per poter esprimere tutto il potenziale dell'informatica: il Linguaggio di Programmazione.

Nonostante von Neumann sia stato il "Prometeo" dell'informatica, donando al mondo il fuoco della conoscenza, non ne fu il vero e proprio pioniere; già qualcun altro aveva pensato al linguaggio di programmazione come un vero e proprio strumento e non solo come accessorio.

Il Plankalkül (in tedesco: "piano di calcolo") è un linguaggio di programmazione sviluppato per applicazioni ingegneristiche da Konrad Zuse. Si pensa che Zuse progettò il linguaggio fra il 1942 e il 1946 ma non lo pubblicò per una serie di fattori fra cui i problemi derivanti dalla seconda guerra mondiale (il governo Nazista si rifiutò di sovvenzionare i progetti di Zuse in quanto la loro realizzazione, secondo i calcoli del Reich, avrebbe richiesto più tempo di quello che il governo tedesco contava di impiegare per porre fine alla guerra) ed il fatto che Zuse si concentrò piuttosto nella commercializzazione dello Z3 e dei suoi successori. Alla fine il Plankalkül venne pubblicato solo nel 1972 ed il primo compilatore venne implementato nel 2000 presso la Technische Universität Berlin, cinque anni dopo la morte di Zuse.

Zuse sostenne che il Plankalkül era stato il primo linguaggio ad alto livello mai progettato. Esso conteneva istruzioni di assegnamento, subroutine, salti condizionali, istruzioni di iterazione, istruzioni per il calcolo in virgola mobile, array, record, asserzioni, gestione delle eccezioni ed altre sofisticate caratteristiche. Se le affermazioni di Zuse fossero fondate, in effetti il Plankalkül sarebbe stato il primo linguaggio ad alto livello ad essere stato concepito, il primo linguaggio di programmazione moderno a situarsi al di sopra delle vecchie tecniche di programmazione in cui le istruzioni venivano

inserite utilizzando interruttori, connettori ed altri sistemi manuali o il semplice assembly.

4. Linguaggio Macchina di IAS

Nel 1946 von Neumann scrisse un'altra relazione, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument", che gli permise di approfondire ulteriormente le sue idee. La IAS machine è l'incarnazione di queste idee. La più grande differenza tra IAS machine e EDVAC è la gestione del processore, la quale è indicativa di come velocemente si stesse muovendo l'avanzamento tecnologico in quegli anni. EDVAC era stato costruito con un processore sequenziale, mentre IAS machine possedeva un processore parallelo dove, nonostante le parole venissero processate in serie, i bit di ogni parola venivano salvati e operati parallelamente. Mi sono riferito a questo fatto con il termine "indicativo" perché nel report del '45, scritto da von Neumann riguardo EDVAC, veniva espressa l'immensa difficoltà della costruzione di un processore parallelo, raccomandandone uno seriale.

IAS machine utilizzava parole da 40 bit, di cui 20 ad istruzione (dunque due istruzioni a parola), con memoria da 1024 parole e due registri general purpose. Era una macchina asincrona, ovvero non era presente un orologio che regolasse il ritmo delle istruzioni e quest'ultime venivano eseguite una dopo l'altra.

Punto cruciale è l'utilizzo effettivo dell'architettura di von Neumann, dove istruzioni e dati venivano salvati insieme in un'unica memoria, permettendo dunque modifiche sia a dati che alle stesse istruzioni durante l'esecuzione. Da qui deriva anche il problema meglio noto come "collo di bottiglia di von Neumann": avendo a disposizione un solo bus per il passaggio di dati e informazioni non si può prelevare (fetch) l'istruzione e, nello stesso tempo, operare nei dati. Questo causa un aumento dei tempi in cui la CPU resta in attesa, limitando l'efficacia della macchina. Questo problema si manifesta soprattutto quando la CPU ha poche istruzioni con una grossa mole di dati, in cui l'unico bus è impegnato al passaggio di dati facendo aspettare la CPU. Lato positivo di questa implementazione era la facilitazione nella realizzazione di loop, permettendo la modifica della condizione dei salti quando fosse stato necessario uscire dal ciclo.

La IAS machine è composta da 5 parti fondamentali:

1. Centrale aritmetica (esegue le operazioni aritmetiche)
2. Centrale di controllo (coinvolta nel passaggio di informazioni tra centrale aritmetica e memoria)
3. Memoria (composta da valvole termoioniche chiamate Selectron)
4. "Output/Input" (non nel senso moderno del termine)
5. Strumento di registrazione (come nastri magnetici)

Poiché il software era svincolato dall'hardware, questo schema permetteva la costruzione di macchine simili ma con differente utilizzo, mantenendo comunque vari gradi di compatibilità tra di loro. Difatti, il rilascio del progetto originale ad università, centri di studio e società portò alla realizzazione di molte macchine, contribuendo alla fama e alla conoscenza di questo importante passo tecnologico.

Di seguito vengono rappresentate tutte le istruzioni utilizzabili, raggruppabili in cinque gruppi funzionali:

- Trasferimento di dati
- Salto incondizionato
- Salto condizionato
- Aritmetica
- Modifica dell'indirizzo

Table 1. IAS computer instruction set.

Inst #	Inst name	Abbrev	Description
1	S(x) → Ac +	x	Copy number in Selectron location x into A.
2	S(x) → Ac –	x –	Same as #1 but copy the negative of the number.
3	S(x) → AcM	xM	Same as #1 but copy the absolute value.
4	S(x) → Ac–M	x – M	Same as #1 but subtract the absolute value.
5	S(x) → Ah +	xh	Add number in Selectron location x into A.
6	S(x) → Ah –	xh –	Subtract number in Selectron location x from A.
7	S(x) → AhM	xhM	Same as #6, but add absolute value.
8	S(x) → Ah–M	xh – M	Same as #7, but subtract absolute value.
9	S(x) → R	xR	Copy number in Selectron location x into R.
10	R → A	A	Copy number in R to A.
11	S(x) * R → A	x ×	Multiply number in Selectron location x by the number in R. Place the left half of the result in A and the right half in R.
12	A/S(x) → R	x ÷	Divide the number in A by the number in Selectron location x. Place the quotient in R and the remainder in A.
13	Cu → S(x)	xC	Continue execution at the left-hand instruction at Selectron location x.
14	Cu′ → S(x)	xC′	Continue execution at the right-hand instruction at Selectron location x.
15	Cc → S(x)	xCc	If the number in A is > = 0, continue as in #13. Otherwise, continue normally.
16	Cc′ → S(x)	xCc′	If the number in A is > = 0, continue as in #14. Otherwise, continue normally.
17	At → S(x)	xS	Copy the number in A to Selectron location x.
18	Ap → S(x)	xSp	Replace the right-hand 12 bits of the left-hand instruction at Selectron location x by the right-hand 12 bits of A.
19	Ap′ → S(x)	xSp′	Same as above, but modifies right-hand instruction.
20	R	R	Shift the number in A to the right 1 bit (leftmost bit is copied).
21	L	L	Circularly left shift the bits in A and R as an 80-bit quantity, leaving the most-significant bit of A unchanged.

A: accumulator
Cc: control conditional (conditional branch in modern parlance)
Cu: control unconditional (unconditional branch or jump in modern parlance)
R: arithmetic register
S: Selectrons (memory in modern parlance)

La IAS machine ha diverse features da tenere a mente:

- Mancanza di I/O: contenendo programma e dati nella memoria, così anche input e output vengono memorizzati insieme al programma effettivo, costringendo i tecnici a controllare il tutto a fine esecuzione
- Gestione della memoria e formato delle istruzioni: la memoria è organizzata in 2.048 40-bit parole, consentendo la coesistenza di

dati e istruzioni. Le istruzioni occupano però solo 20 bit, permettendo alla macchina di salvare, per ogni parola, due istruzioni. Questa gestione ci costringe a fare distinzioni tra istruzione a sinistra e a destra.

- Codice auto-modificante: fin da subito fu riconosciuta l'importanza dell'eseguire la stessa computazione su diversi gruppi di dati. Questo obiettivo è raggiunto tramite la capacità di modificare il codice in maniera autonoma durante l'esecuzione, rappresentata dalle istruzioni 18 e 19 della tabella delle istruzioni

4.a. I 15 problemi di von Neumann

Von Neumann e Goldstine elencarono 15 problemi da poter risolvere con la IAS machine per illustrare i differenti campi di pertinenza del manufatto. A distanza di 65 anni, grazie all'utilizzo di un emulatore ad hoc, si è potuto esaminare, con conoscenze odierne, le applicazioni dei problemi nella macchina dell'epoca, mettendo in risalto il design dei set di istruzioni, effetti collaterali e persino mancanze, tipografiche e non, dandoci un'interessante finestra dei tempi passati e sottolineando, nonostante la qualità del team coinvolto, la difficoltà nel programmare.

Il tipo di errore più frequente è quello di carattere tipografico, compilazione o traduzione, mancanze dall'apparenza non gravi ma che, in assenza di revisione, avrebbero potuto propagarsi in molto peggio in

caso di effettivo utilizzo. Non mancano però errori logici causati o per probabile distrazione o per effettivi dubbi riguardo il comportamento della macchina. Sottolineo che i problemi trovati non sono di tipo matematico, inerenti allo svolgimento teorico del quesito, ma sono strettamente legati alla programmazione, ovvero all'atto di trasformare la soluzione teorica trovata in qualcosa di effettivo per la macchina.

Nel primo campo rientra probabilmente quello presente nel terzo problema dove viene utilizzata iterativamente una funzione per calcolare valori multipli; infatti, manca l'atto di salvare il risultato di un'operazione. La soluzione più semplice è quella di aggiungere una parola intera per effettuare tale salvataggio, mantenendo allo stesso tempo la parità delle parole successive (l'aggiunta della singola istruzione SAVE avrebbe modificato l'ordine di parità). Nota particolare di questo quesito è l'utilizzo di un particolare comando di parziale sostituzione che modifica l'istruzione subito dopo, nell'altra metà della stessa parola. Ciò è un azzardo in quanto il programma fa un prefetch di entrambe le istruzioni nella parola nello stesso momento, anche se il design della macchina prevede e permette tale operazione senza problemi. Questo design anticipa le complicazioni della gestione dei dati e del branching ritardato nelle moderne architetture di pipeline.

Errore causato da un ragionamento non corretto si può invece notare nel problema 13, l'utilizzo dell'interpolazione di Lagrange per funzioni tabulate in tre differenti situazioni date da diverse distribuzioni di punti. Viene utilizzato il comando di sostituzione in più punti per poter effettuare calcoli nei passi successivi. Affinché ci sia il corretto funzionamento di tale comando l'indirizzo bersaglio deve essere zero prima che la macchina lo alteri, in quanto il comando di sostituzione sostituisce solo una parte della parola. La situazione non è quella desiderata perché gli indirizzi bersaglio sono occupati da istruzioni passate, rendendo il codice incorretto. La soluzione più rapida è trovare due indirizzi inizializzati a zero inutilizzati, senza influenzare il resto del programma. Non si può dare lo stesso debug ai due sotto-problemi successivi poiché i comandi di sostituzione avvengono in loop tra loro, rendendo necessaria una riparazione più elaborata con pulizia esplicita della memoria utilizzata e conseguente gestione della parità delle parole.

Nonostante l'inevitabile presenza di errori, causati principalmente per mancanza di adeguata strumentazione, von Neumann tracciò un solco importante nella strada della programmazione, pubblicando insieme ai problemi sopracitati la procedura formale seguita per effettuare un'ottima traduzione della soluzione teorica in linguaggio macchina.

5. L’inizio della programmazione

La procedura formale per programmare con la macchina IAS prevedeva diverse fasi di elaborazioni dell’algoritmo desiderato, dalla formula scritta in un linguaggio matematico fino ad una sequenza di istruzioni compatibili con la macchina.

Le varie fasi della procedura formale sono le seguenti:

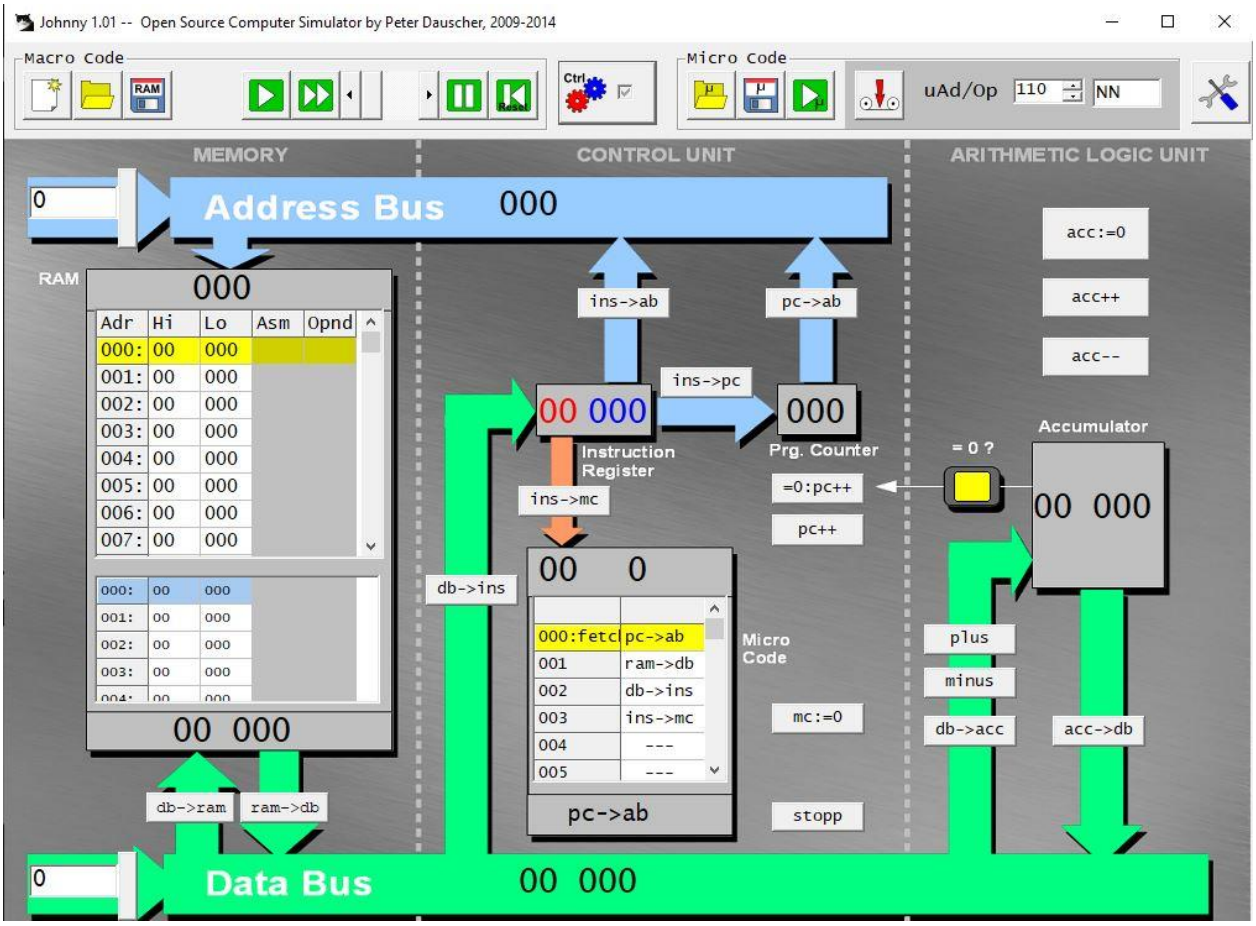
1. Descrivere l’algoritmo desiderato con un linguaggio misto inglese e matematico
2. Trasformare quanto scritto in un diagramma di flusso, rappresentando logica e operazioni ed etichettando i vari blocchi per ottenere un’enumerazione preliminare delle istruzioni da usare
3. Tradurre il diagramma di flusso etichettato in una tabella, rendendo più formale l’idea delle istruzioni da usare, cosicché fosse possibile effettuare una panoramica della dimensione del programma e degli indirizzi da assegnare
4. Arrivati a questo punto un ingegnere (ovvero il programmatore ante litteram) poteva tradurre l’enumerazione finale in codice binario e caricare il programma

Con la descritta procedura formale si ha un esempio di come il concetto di compilatore fosse già intrinsecamente forte nella mente degli

scienziati, nonostante non sia stato dichiarato esplicitamente. Infatti il lavoro della procedura formale corrispondeva esattamente a quello di un moderno compilatore: tradurre l'algoritmo da un linguaggio specifico di alto livello, utilizzato dagli specialisti per descrivere il problema, ad un linguaggio macchina comprensibile per il calcolatore (e per l'ingegnere che doveva inserirlo), mantenendo la semantica nel processo. Per il primo vero compilatore che prenderà piede nel mondo dovremo aspettare il '57 con il team Fortran, presso l'IBM, guidato da John Backus.

Utilizzando la procedura formale e forte di un simulatore, mi sono voluto immedesimare in uno scienziato dell'epoca, seguendo la tecnica descritta nel progettare e testare un programma che ordina n interi.

Johnny Simulator è un simulatore di una macchina basata sull'architettura di Von Neumann, in grado di restituire una panoramica su ciò che accade all'interno del manufatto. Il simulatore è costituito da tre parti: la memoria (RAM), l'unità aritmetico-logica e l'unità di controllo.



A sinistra abbiamo La memoria ad accesso casuale (RAM) è formata da 1000 locazioni, ciascuna ha la capacità di memorizzare numeri compresi nell'intervallo 0 …19999. Di conseguenza, tre cifre decimali sono sufficienti per indirizzare ogni locazione. La colonna Hi e la colonna Lo rappresentano, rispettivamente, il codice operativo e l'indirizzo dell'operando della macroistruzione. Premendo il pulsante "ram- >db" i dati possono essere copiati dalla locazione di memoria sul bus dati; premendo "db->ram" avviene il contrario. Le locazioni sono modificabili utilizzando la GUI; le macroistruzioni possono essere scelte utilizzando il menù. Nella GUI sono mostrate due sezioni della RAM (che possono

sovrapporsi). Così le istruzioni e i dati possono essere mostrati contemporaneamente.

A destra abbiamo l'Unità Aritmetico Logica, la quale è formata soltanto dall'accumulatore. L'accumulatore può essere resettato ("acc:=0"), incrementato ("acc++"), decrementato ("acc—"). "db->acc" trasporta un dato dal bus nell'accumulatore; "acc->db" fa il contrario. Un valore del data bus può essere aggiunto ("plus") o sottratto ("minus").

La parte più complessa del processore è l'unità di controllo (control unit), rappresentata dalla sezione centrale. Si compone dal registro delle istruzioni (instruction register), dal contatore di programma (program counter) e dal microcodice. Con "db->ins" il contenuto del data bus viene trasferito nel registro delle istruzioni. Per implementare l'istruzione di salto l'indirizzo dell'istruzione può essere messo direttamente sul bus indirizzi ("ins->ab") o trasferito nel program counter ("ins->pc"). Il program counter stesso può essere copiato sul bus indirizzi con "pc->ab". "pc++" incrementa il contatore; "=0: pc++" fa lo stesso, ma solo se l'accumulatore ha valore zero. "ins->mc" imposta il contatore di microistruzioni che si trova sopra il microcodice. Questo contatore è costituito da un valore a due cifre e uno ad una cifra. Il valore a due cifre contiene il codice operativo contenuto nel registro delle istruzioni; il valore ad una cifra conta i passaggi di ogni fase e

viene azzerato ogni volta che si completa o la fase di fetch oppure l'esecuzione dell'istruzione. "mc:=0" resetta il contatore di micro istruzioni; "stopp" forza il simulatore a terminare il programma visualizzando il relativo messaggio.

Le istruzioni (chiamate anche macro) usate sono solo 10. Tutte le istruzioni utilizzano un indirizzamento assoluto (un indirizzo per l'istruzione) per ragioni di semplicità. Il codice dell'operazione (OP) è composto dalle migliaia e dalle centinaia; la prima posizione rappresenta l'indirizzo (ADD 42 è rappresentato da 02 042).

Il set di macroistruzioni include 10 comandi standard:

- TAKE: Il valore della locazione (indicato dall'indirizzo assoluto) è copiato nell'accumulatore.
- SAVE: Il valore presente nell'accumulatore è copiato nella locazione di memoria indicata dall'indirizzo assoluto.
- ADD: Il valore della locazione (indicato dall'indirizzo assoluto) è sommato al contenuto dell'accumulatore.
- SUB: Il valore della locazione (indicato dall'indirizzo assoluto) è sottratto al valore contenuto nell'accumulatore.
- INC: Il valore della cella (dato dall'indirizzo assoluto) è incrementato di 1.

- DEC: Il valore della cella (dato dall'indirizzo assoluto) è decrementato di 1.
- NULL: Il valore nella locazione di memoria indicata (in indirizzo assoluto) è impostato a zero.
- TST: Se la cella indicata (in indirizzo assoluto) ha come valore lo zero, la successiva istruzione verrà saltata, altrimenti sarà eseguita.
- JMP: prosegue saltando alla locazione indicata.
- HLT: Il programma termina visualizzando un messaggio

Sono date le possibilità di modificare quelle già esistenti e di creare nuove macro attraverso la sequenza di microistruzioni offerte dal simulatore

5.a. Algoritmo di ordinamento su Johnny

1. DESCRIZIONE DEL PROBLEMA

Il problema presentato è il seguente: ordinare, in senso crescente, una lista di n interi.

L'algoritmo pensato si basa concettualmente su Bubble Sort, ovvero confrontare gli elementi dell'array a due a due, procedendo in un verso prestabilito. Per esempio, saranno confrontati il primo e il secondo elemento, poi il secondo e il terzo, poi il terzo e il quarto, e così via fino al confronto fra il penultimo e l'ultimo elemento. Ad ogni confronto, se

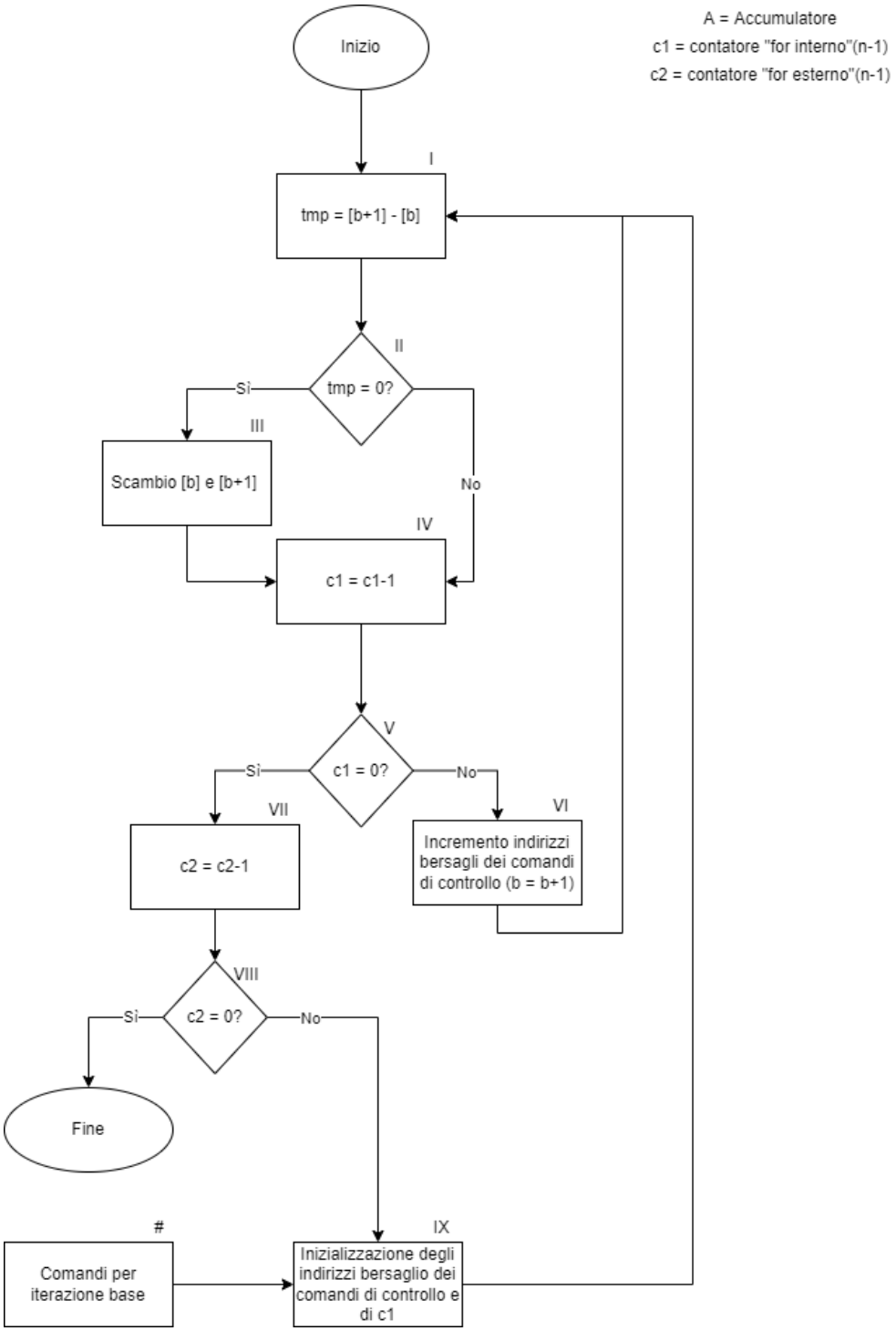
i due elementi esaminati non sono ordinati secondo il criterio prescelto, vengono scambiati di posizione. L'algoritmo continua nuovamente a ri-eseguire questi passaggi su tutta la lista, tralasciando di volta in volta il massimo del sottoinsieme esaminato, fino a quando il sottoinsieme esaminato non diventa vuoto, situazione di arrivo che indica che la lista è ordinata. Ho scelto questo algoritmo in quanto lo ritengo più consono al design del set delle istruzioni presenti.

Alla luce di questa elaborazione, sappiamo che dobbiamo gestire in maniera particolare la memoria in questi casi:

- Poiché abbiamo a che fare con una lista di dimensioni variabili, è bene collocare tale lista alla fine dei comandi funzionali, in modo da non sovrascrivere istruzioni successive.
- Bisogna memorizzare i comandi con cui si inizia la prima iterazione in una parte di memoria non coinvolta, in modo da poterle usare come “stampino” per iniziare le iterazioni in sicurezza.
- Dobbiamo utilizzare la feature del codice auto-modificante per poter effettuare le stesse operazioni su indici diversi.

2. DIAGRAMMA DI FLUSSO ETICHETTATO

Di seguito viene proposto il problema tramite diagramma di flusso, con successiva etichettatura dei blocchi:



3. ENUMERAZIONE PRELIMINARE DELLE ISTRUZIONI

Una volta che il diagramma di flusso è completato ed etichettato si può procedere ad esaminare i vari blocchi, utilizzando un'enumerazione per definire i comandi per replicare l'effetto descritto dai blocchi del diagramma.

Blocco del diagramma, sequenza	Comando	Indirizzo di memoria bersaglio simbolico
I,1	TAKE	[b + 1]
I,2	SUB	[b]
I,3	SAVE	Tmp
II,1	TEST	Tmp
III, 1	TAKE, SAVE	[b + 1], [b], Tmp
IV,1	SUB	C1
V,1	TEST	C1
VI, 1	INC	Blocco I e blocco III
VI, 2	JUMP	Blocco I
VII,1	SUB	C2
VIII,1	TEST	C2
IX, 1	TAKE, SAVE	Blocco I e blocco III
IX, 2	JUMP	Blocco I

4. REALIZZAZIONE PROGRAMMA E INSERIMENTO

Una volta completata l'enumerazione preliminare delle istruzioni si può procedere all'assegnazione degli indirizzi a comandi, interi da ordinare ed elementi di supporto, conoscendo ora la dimensione dell'algoritmo e gli indirizzi da utilizzare. L'ingegnere incaricato dell'inserimento del programma provvederà a completare il compito assegnatogli memorizzando le istruzioni e i dati nei Selectron predisposti, avviando ufficialmente l'algoritmo che fino a quel momento era rimasto su carta e permettendo un'eventuale fase di debugging in caso di bisogno.

Di seguito l'algoritmo inserito e testato nel simulatore Johnny:

INDIRIZZO DI MEMORIA	ISTRUZIONE/DATO	COMMENTO
00	TAKE 71	Prendo b+1 e lo metto nell'accumulatore
01	SUB 70	Sottraggo b al valore presente nell'accumulatore
02	SAVE 69	Salvo il risultato della sottrazione in 69 (tmp)
03	TEST 69	Faccio test su 69: •se diverso da 0 allora

		ordine giusto e faccio comando in 4 •se uguale a 0 devo fare lo scambio di variabili, dunque comando in 5
04	JUMP 6	Salto all'indirizzo 6
05	JUMP 15	Faccio il salto all'indirizzo 15 (funzione di scambio interi)
06	SUB 68	Decremento contatore 1
07	TEST 68	Faccio test su 68 (contatore 1): •se diverso da 0 vuol dire che ho ancora dei cicli da fare, vado al comando in 8 •se uguale a 0 vuol dire che ho finito i cicli del for interno, vado al

		comando in 9
08	JUMP 55	Faccio il salto a 55 (funzione di incremento degli indirizzi)
09	SUB 67	Decremento contatore 2
10	TEST 67	Faccio test su 67 (contatore 2): •se diverso da 0 vuol dire che ho ancora dei cicli da fare, vado al comando in 11 •se uguale a 0 vuol dire che ho finito i cicli del for esterno, vado al comando in 12
11	JUMP 23	Salto a 23 (funzione di rinizializzazione degli indirizzi e di contatore

		1)
12	HALT	Fine programma, insieme ordinato
SCAMBIO INTERI		
15	TAKE 70	Prendo il primo valore e lo metto nell'accumulatore
16	SAVE 69	Salvo il valore nell'accumulatore in 69
17	TAKE 71	Metto nell'accumulatore il secondo valore
18	SAVE 70	Salvo il valore nell'accumulatore nella posizione del primo valore
19	TAKE 69	Metto nell'accumulatore il primo valore salvato in 69
20	SAVE 71	Salvo il valore

		nell'accumulatore nella posizione del secondo valore
21	JUMP 6	Salto a 6
RI-INIZIALIZZAZIONE DEGLI INDIRIZZI DEI CONTROLLI E DELLA FUNZIONE DI SCAMBIO		
23	TAKE 67	Prendo il valore del contatore 2
24	SAVE 68	Salvo il valore del contatore 2 nel contatore 1
25	TAKE 40	Prendo il comando base di 00 da 40
26	SAVE 00	Salvo il comando base di 00 in 00
27	TAKE 41	Prendo il comando base di 01 da 41
28	SAVE 01	Salvo il comando base di 01 in 01
29	TAKE 42	Prendo il comando base di 15 da 42

30	SAVE 15	Salvo il comando base di 15 in 15
31	TAKE 43	Prendo il comando base di 17 da 43
32	SAVE 17	Salvo il comando base di 17 in 17
33	TAKE 44	Prendo il comando base di 18 da 44
34	SAVE 18	Salvo il comando base di 18 in 18
35	TAKE 45	Prendo il comando base di 20 da 45
36	SAVE 20	Salvo il comando base di 20 in 20
37	JUMP 00	Salto a 00 per riniziare controllo dall'inizio dell'insieme
COMANDI BASE		
40	TAKE 71	Comando base di 00
41	SUB 70	Comando base di 01
42	TAKE 70	Comando base di 15

43	TAKE 71	Comando base di 17
44	SAVE 70	Comando base di 18
45	SAVE 71	Comando base di 20
INCREMENTO INDIRIZZI		
55	INC 00	Incremento indirizzo bersaglio del TAKE (es. TAKE 71->TAKE 72)
56	INC 01	Incremento indirizzo bersaglio del SUB (es. SUB 70->SUB 71)
57	INC 15	Incremento indirizzo bersaglio del TAKE (es. TAKE 70-> TAKE 71)
58	INC 17	Incremento indirizzo bersaglio del TAKE (es. TAKE 71->TAKE 72)
59	INC 18	Incremento indirizzo bersaglio del SAVE (es. SAVE 70->SAVE 71)
60	INC 20	Incremento indirizzo bersaglio del SAVE (es.

		SAVE 71->SAVE 72)
61	JUMP 00	Salto ad inizio programma con i due interi successivi (70,71->71, 72,...)
VARIABILI		
67	N-1	Valore contatore 2. È il contatore del ciclo for esterno.
68	N-1	Valore contatore 1. È il contatore del ciclo for interno.
69	00	Variabile di supporto per fare lo scambio di interi e per fare la verifica dell'ordine
70, 71, ...	INSIEME DI N	Da indirizzo 70 (compreso) in poi interi dell'insieme da ordinare

6. Conclusione

Il problema di progettare e implementare un algoritmo seguendo il processo di problem solving utilizzato negli anni ’40 ha offerto una vista interessante sia del passato sia del futuro della computazione, richiedendo uno studio approfondito dell’architettura di von Neumann e mettendo in risalto le difficoltà tecniche sorte all’alba della programmazione moderna. La presenza (e il successivo utilizzo) dell’emulatore mi ha permesso di seguire la procedura “a cuor leggero”, senza la pressione che i vincoli hardware dell’epoca potevano imporre al team di programmazione, evidenziando comunque la buona qualità della strategia proposta e dunque una buona qualità del codice progettato. Sfida particolare è stata la comprensione e l’utilizzo della particolare tipologia di memoria offerto dall’architettura proposta; avere dati e istruzioni soggette a potenziali (e indesiderate) modifiche per via di comandi mal interpretati è una situazione anormale ma allo stesso tempo d’obbligo utilizzo per via della gestione di loop e di subroutine tramite codice auto-modificante.

Ecco alcune lezioni importanti da sottolineare:

- Automatizzazione della programmazione: completare il vuoto semantico tra l’algoritmo scritto su carta e l’impulso elettrico è stato un problema a cui neanche le grandi menti del dopo guerra

hanno saputo mettere una toppa; infatti la traduzione dal diagramma di flusso all'immissione nella macchina del codice ha portato inevitabilmente degli errori. Con il progredire dell'hardware si è riusciti a risolvere questa problematica tramite la creazione di strumenti automatizzati per la programmazione e di linguaggi di alto livello, in modo da poter rendere la programmazione più a portata delle persone, diminuendo gli errori causati dall'uomo.

- Semplicità e ortogonalità del design del set di istruzioni: a causa della mancanza degli strumenti di programmazione disponibili ai tempi, molti errori sono causati da effetti collaterali di comandi, semantica inconsistente e mancanza di ortogonalità (di come le istruzioni si influenzano tra di loro) delle interazioni tra istruzioni. Aggiungere elementi di supporto a livello di set di istruzioni per specifici lavori computazionali come la conversione da binario a decimale significa invece aggiungere complessità hardware alla macchina, rendendo la corretta programmazione a livello di linguaggio assembly più difficile. Raggiungere l'equilibrio giusto tra hardware e firmware sembra essere un'eterna sfida per l'informatica.
- La sfida del testare su un computer non esistente: poiché la IAS machine entro in azione dal 1951 i problemi presentati da von

Neumann non poterono essere testati al momento del rilascio del report, il 1947. Essendo, ovviamente, l'emulazione un sogno lontano, i ricercatori poterono basare le loro visioni solo sull'idea di come la macchina avrebbe potuto lavorare.

Questa esplorazione delle prime macchine dell'Informatica moderna ci fa chiedere se continui studi come questo qua rappresentato possa avere in sé una svolta significativa per il futuro del design del computer. Del resto, chi non impara dalla storia è condannato a ripeterne gli errori.

7. Riferimenti bibliografici e sitografici

Barry Fagin e Dale Skrien," Debugging on the Shoulders of Giants: Von Neumann's Programs 65 Years Later", https://ieeexplore.ieee.org/document/6152078, 2012.

Juliet Kemp, "John von Neumann, EDVAC, and the IAS machine", http://www.linuxvoice.com/issues/004/lv4-neumann.pdf, 2015.

Maurizio Gabbrielli e Simone Martini, "Linguaggi di programmazione: Principi e paradigmi", McGraw-Hill Education, seconda edizione 2011.

Peter Dauscher, "Johnny Simulator", https://sourceforge.net/projects/johnnysimulator/, 2012.

Pierre Marchal, "John von Neumann: The Founding Father of Artificial Life", https://ieeexplore.ieee.org/document/6788156, ", The MIT Press, 1998.

Thomas Haigh e Paul E. Cerazzi, "A New History of Modern Computing", The MIT Press, 2021

8. Ringraziamenti

Sarò breve.

Grazie di cuore a tutti quelli che mi hanno sopportato.