Dopo le macchine di Turing

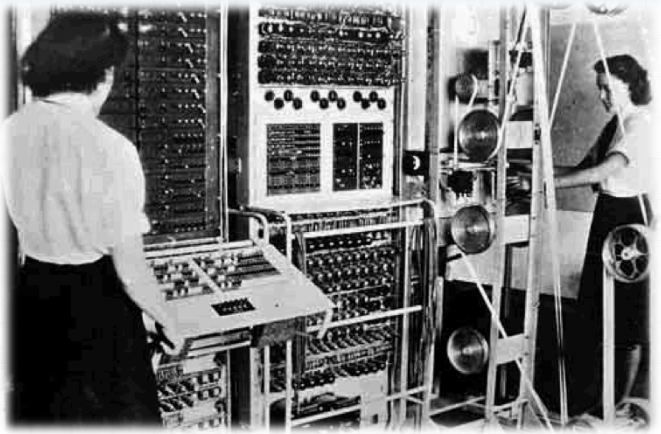
- La storia che porta dalle macchine di Turing ai moderni calcolatori digitali è lunga e articolata. Potremo solo fare dei cenni a qualche pietra miliare. Essi basteranno a far emergere il ruolo cruciale dei *matematici* e della *guerra* in questa storia.
- Il primo computer digitale "programmabile" è stato Colossus, entrato in funzione in Inghilterra nel 1944. Fu progettato e costruito dall'ingegnere Tommy Flowers in undici mesi, dietro richiesta e sotto la guida del grande topologo inglese Max Newmann, che era stato uno dei maestri di Turing a Cambridge.
- Nel 1942 Newmann aveva accettato di lavorare per il governo britannico a Bletchley Park, il centro militare per la decifrazione dei messaggi cifrati tedeschi.
- Gli inglesi avevano già decodificato i messaggi di Enigma, una macchina tedesca per cifrare messaggi. Colossus fu ideato e costruito per attaccare i messaggi della più recente macchina crittografica tedesca Lorenz SZ40/42.

Dopo le macchine di Turing

- Colossus fu costruito usando un gran numero di valvole, un'idea che Flowers aveva già sperimentato anni prima nel suo lavoro civile per la compagnia telefonica britannica. I dati erano immessi tramite un nastro di carta. Era "programmabile" per poter eseguire una gamma di operazioni logiche sui dati in ingresso.
- Anche Turing lavorò a Bletchley park durante la guerra. Oggi sappiamo che il suo contributo alla decifrazione dei codici tedeschi fu di grande importanza.

Tomas H. Flowers, 1905–1998

Maxwell H. A. Newman, 1897–1984



Il Colossus a Bletchley Park, 1944

Dopo le macchine di Turing

- Nel frattempo, gli alleati americani non erano rimasti indietro. Il primo computer elettronico "programmabile" negli Stati Uniti fu ENIAC (*Electronic Numerical Integrator and Computer*).
- ENIAC fu progettato e costruito all'Università della Pennsylvania da un gruppo guidato da J. Mauchly e J. Presper Eckert. Il progetto cominciò nel 1943; ENIAC entrò in funzione in via definitiva nel 1945.

J. Presper Eckert, 1919–1995

John W. Mauchly, 1907–1980

John von Neumann, 1903–1957
(con, da sinistra, Ulam e Feynman)

L’architettura di Von Neumann

- Colossus ed ENIAC erano concepiti in modo simile, sebbene ENIAC fosse di parecchio più grosso. Nonostante li si possa considerare computer “programmabili”, in realtà sarebbe forse più corretto definirli “configurabili”. Un dato algoritmo era codificato all’interno di ENIAC da uno specifico stato dei suoi cavi e dei suoi interruttori. Implementare con ENIAC un algoritmo anche realtivamente semplice poteva richiedere settimane di lavoro.
- Un grande passo avanti avvenne quindi con la successiva generazione di calcolatori. Tralasciando i dettagli, l’idea di fondo è che *sia i dati sia le istruzioni del programma che deve manipolarli risiedano nella memoria del calcolatore* (in gergo, “stored-program computer”). In particolare, progettare un calcolatore secondo questo modello vuol dire anche fissare il suo insieme di istruzioni: concettualmente, i passi elementari di calcolo che esso può eseguire.
- Il matematico John von Neumann, ungherese espatriato negli Stati Uniti, lavorava alla costruzione della bomba all’idrogeno durante la guerra, nei laboratori di Los Alamos, in Nuovo Messico.

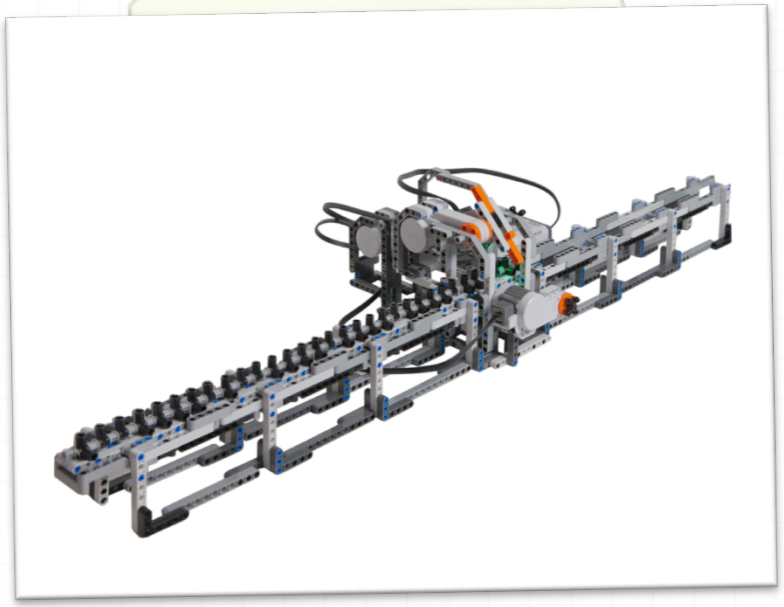
L'architettura di Von Neumann

- Von Neumann era in contatto con il gruppo che stava costruendo ENIAC all'Università della Pennsylvania. Conosceva inoltre bene il fondamentale lavoro di Turing del 1936.
- In effetti, nel suo lavoro Turing aveva concettualizzato in modo completo e dettagliato l'idea di un calcolatore programmabile a scopo generale, dimostrando l'esistenza di ciò che oggi chiamiamo una Macchina di Turing Universale, *in grado cioè di simulare il comportamento di qualunque altra macchina di Turing*.
- Turing usò quest'idea per esibire un semplice problema indecidibile, il problema dell'arresto, che diviene così il nostro secondo e ultimo esempio di problema indecidibile, dopo il Decimo di Hilbert.

Teorema (Turing 1936). Non esiste alcuna macchina di Turing che, ricevendo in ingresso (una codifica di) una qualsivoglia macchina di Turing M e una qualsivoglia quantità finita di dati D, decida se M si arresta ricevendo in ingresso D.

L'architettura di Von Neumann

- Mauchly ed Eckert, nell'agosto del 1944, cominciarono a lavorare al progetto di un nuovo calcolatore che superasse alcune limitazioni di ENIAC. Esso si sarebbe chiamato EDVAC (*Electronic Discrete Variable Automatic Computer*). EDVAC fu concepito secondo il modello genuinamente programmabile – e non meramente riconfigurabile – dello stored-program computer, cui si è accennato sopra.
- Quando von Neumann si unì al gruppo in qualità di consulente, si offrì di stendere un documento che riassumesse i principi di progettazione della nuova macchina.
- Lo fece in un lungo manoscritto, composto durante un viaggio di ritorno dalla Pennsylvania a Los Alamos. Il manoscritto, incompleto ma poi dattiloscritto e ampiamente diffuso, fu denominato *First Draft of a Report on the EDVAC*.
- Fra molte altre cose, von Neumman spiegava qui l'architettura degli stored-program computer, che in essenza è quella usata ancora oggi. A causa di questo documento, ci si riferisce spesso all'architettura dei calcolatori moderni come all'architettura di von Neumann.
- La storia che abbiamo raccontato dovrebbe chiarire che l'attribuzione è quantomeno discutibile.



Macchina di Turing in Lego, 2012

Cenni all'architettura dei calcolatori

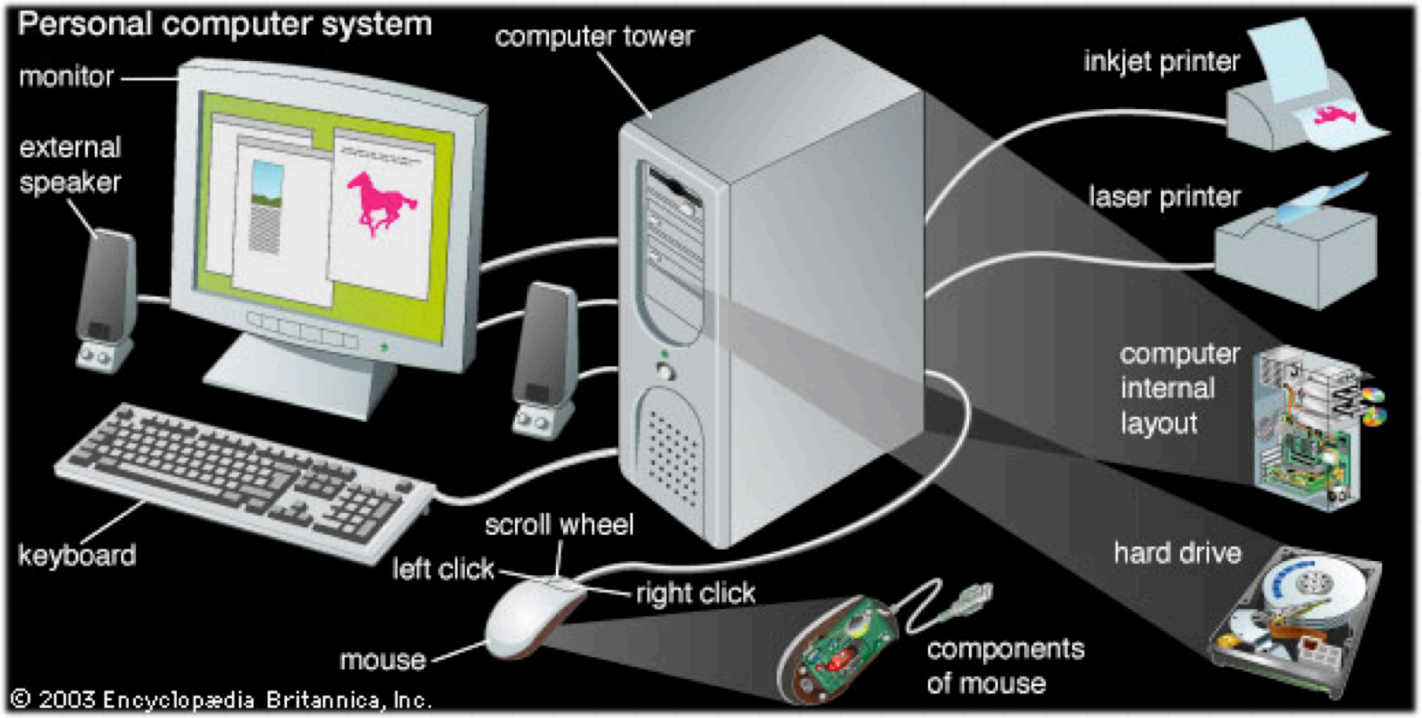
Il sostrato fisico della computazione nell'era digitale

Gli elementi fondamentali dell'architettura di un calcolatore sono i seguenti.

1. La CPU (Central Processing Unit), detta anche (micro)processore.
2. Il bus di sistema.
3. La RAM (Random Access Memory), detta anche memoria centrale.
4. La memoria di massa.
5. I dispositivi periferici per l'I/O, detti anche periferiche (monitor, tastiere…)

Gli elementi fondamentali dell'architettura di un calcolatore sono i seguenti.

1. La CPU (Central Processing Unit), detta anche (micro)processore.
2. Il bus di sistema.
3. La RAM (Random Access Memory), detta anche memoria centrale.
4. La memoria di massa.
5. I dispositivi periferici per l'I/O, detti anche periferiche (monitor, tastiere…)



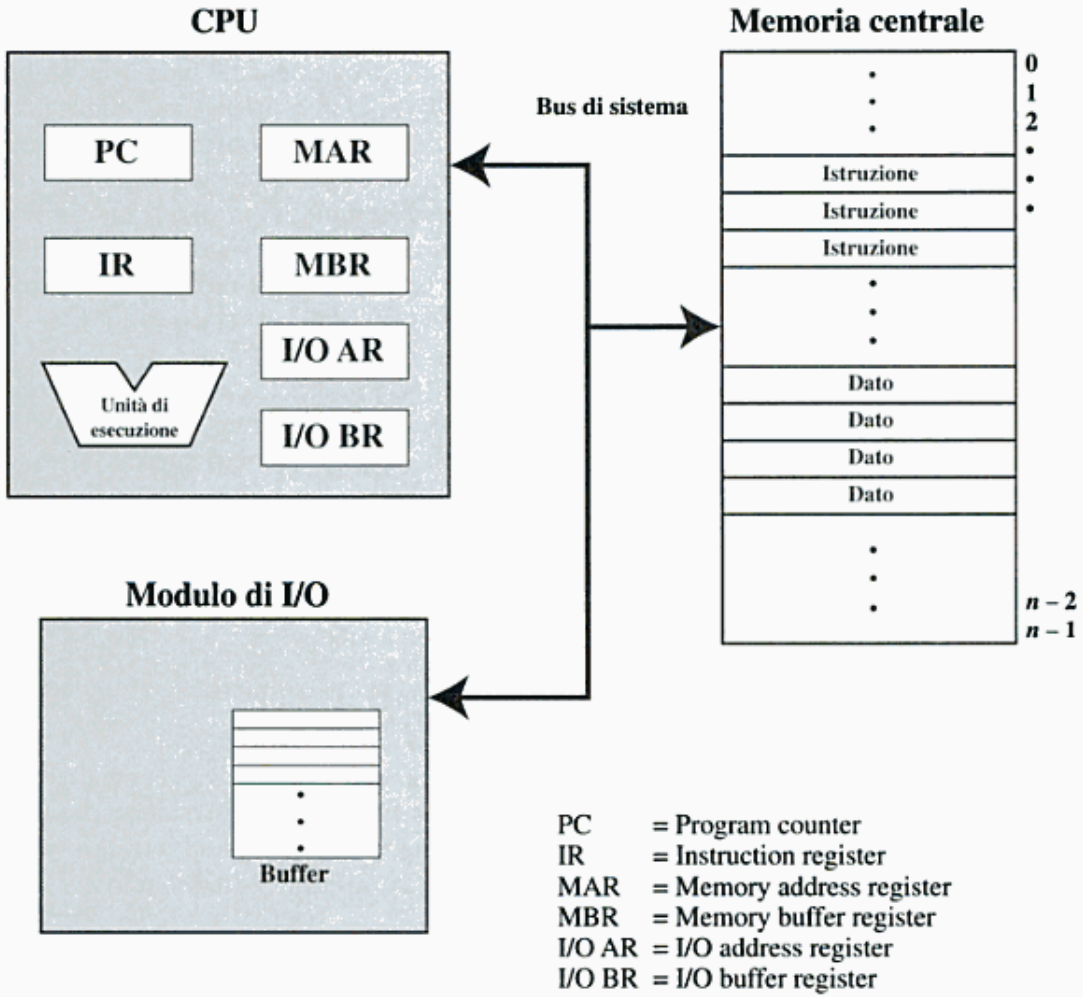


Figura 3.2 Componenti del calcolatore: visione ad alto livello.

Da W. Stallings, *Architettura e organizzazione dei calcolatori*, Pearson Education Italia, 6a ed., 2004

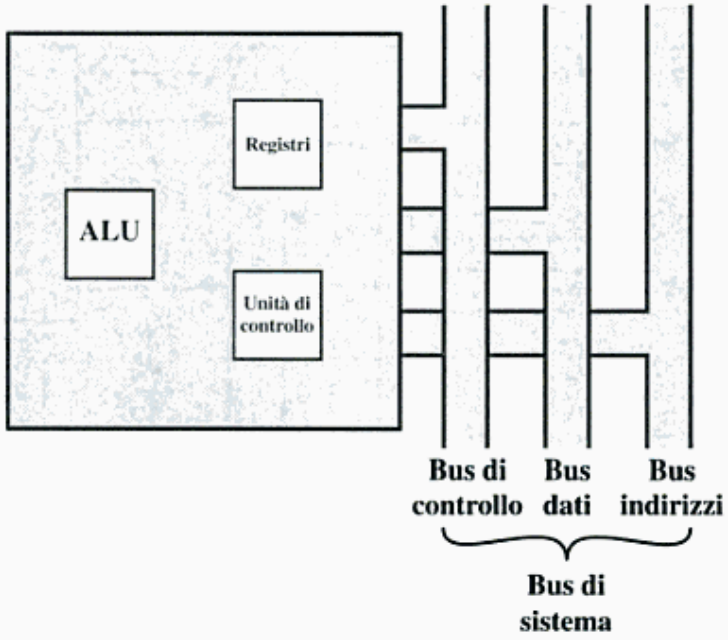


Figura 12.1 CPU e bus di sistema.

Da W. Stallings, *Architettura e organizzazione dei calcolatori*, Pearson Education Italia, 6a ed., 2004

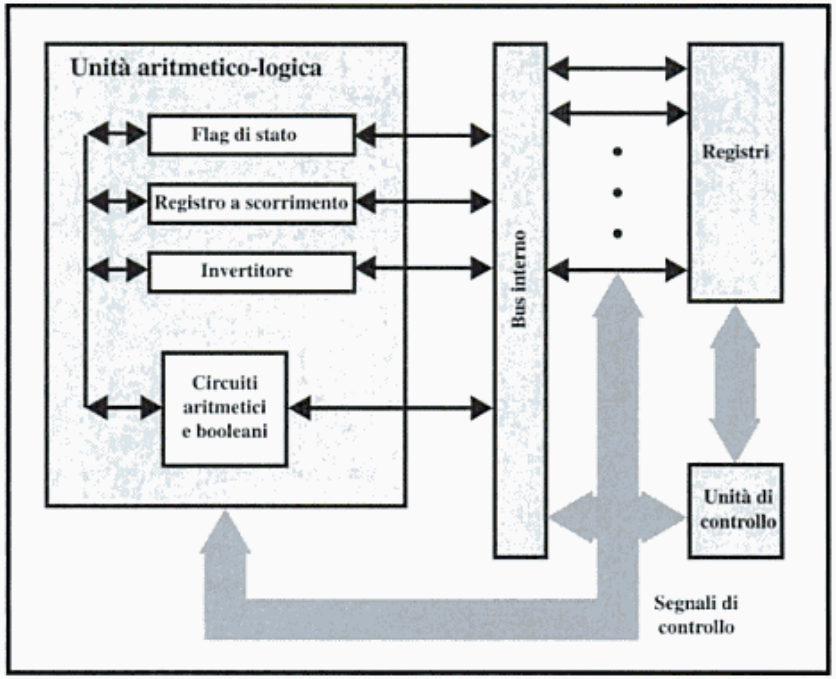


Figura 12.2 Struttura interna della CPU.

Da W. Stallings, *Architettura e organizzazione dei calcolatori*, Pearson Education Italia, 6a ed., 2004

La *Torre di Babele* di Brueghel il Vecchio, 1563

Linguaggi

La Babele digitale

La lingua madre del calcolatore è il linguaggio macchina: l'insieme di istruzioni che la CPU è in grado di *eseguire direttamente*.

Le istruzioni del linguaggio macchina sono codificate in notazione binaria, ed hanno un significato operazionale semplice come, per esempio: "Sposta il contenuto del registro *m* nel registro *n*."

Un programma in linguaggio macchina è un insieme (finito, naturalmente) di tali istruzioni.

La lingua madre del calcolatore è il linguaggio macchina: l'insieme di istruzioni che la CPU è in grado di *eseguire direttamente*.

Le istruzioni del linguaggio macchina sono codificate in notazione binaria, ed hanno un significato operazionale semplice come, per esempio: "Sposta il contenuto del registro *m* nel registro *n*."

Un programma in linguaggio macchina è un insieme (finito, naturalmente) di tali istruzioni.

Esempio: Linguaggio macchina MIPS

```
00000001001010100100000000100000
```

La lingua madre del calcolatore è il linguaggio macchina: l'insieme di istruzioni che la CPU è in grado di *eseguire direttamente*.

Le istruzioni del linguaggio macchina sono codificate in notazione binaria, ed hanno un significato operazionale semplice come, per esempio: "Sposta il contenuto del registro *m* nel registro *n*."

Un programma in linguaggio macchina è un insieme (finito, naturalmente) di tali istruzioni.

Esempio: Linguaggio macchina MIPS

```
00000001001010100100000000100000
Campi:  000000|01001|01010|01000|00000|100000
        OpCode|Reg Sr1|Reg Sr2|Reg Dst|Shift|Funz
```

La lingua madre del calcolatore è il linguaggio macchina: l'insieme di istruzioni che la CPU è in grado di *eseguire direttamente*.

Le istruzioni del linguaggio macchina sono codificate in notazione binaria, ed hanno un significato operazionale semplice come, per esempio: "Sposta il contenuto del registro *m* nel registro *n*."

Un programma in linguaggio macchina è un insieme (finito, naturalmente) di tali istruzioni.

Esempio: Linguaggio macchina MIPS

```
00000001001010100100000000100000
```

Campi: `000000|01001|01010|01000|00000|100000`

`OpCode|Reg Sr1|Reg Sr2|Reg Dst|Shift|Funz`

Significato: Somma i contenuti dei registri 9 e 10, e metti il risultato nel registro 8

Poter programmare un calcolatore in linguaggio macchina, come s'è già detto, è un enorme passo avanti rispetto, diciamo, alla prospettiva di dover riconfigurare un ENIAC.

La programmazione in linguaggio macchina, però, rimane pur sempre drammaticamente noiosa e soggetta ad errori materiali.

Le cose si possono migliorare di molto con un vecchio cavallo di battaglia dei matematici: un cambio di notazione.

Esempio: Linguaggio macchina MIPS

```
00000001001010100100000000100000
```

Campi: `000000|01001|01010|01000|00000|100000`

`OpCode|Reg Sr1|Reg Sr2|Reg Dst|Shift|Funz`

Significato: Somma i contenuti dei registri 9 e 10, e metti il risultato nel registro 8

Poter programmare un calcolatore in linguaggio macchina, come s'è già detto, è un enorme passo avanti rispetto, diciamo, alla prospettiva di dover riconfigurare un ENIAC.

La programmazione in linguaggio macchina, però, rimane pur sempre drammaticamente noiosa e soggetta ad errori materiali.

Le cose si possono migliorare di molto con un vecchio cavallo di battaglia dei matematici: un cambio di notazione.

Esempio: Linguaggio macchina MIPS

```
00000001001010100100000000100000
```

Campi: `000000|01001|01010|01000|00000|100000`

`OpCode|Reg Sr1|Reg Sr2|Reg Dst|Shift|Funz`

Significato: Somma i contenuti dei registri 9 e 10, e metti il risultato nel registro 8

```
add $8, $9, $10
```

Poter programmare un calcolatore in linguaggio macchina, come s'è già detto, è un enorme passo avanti rispetto, diciamo, alla prospettiva di dover riconfigurare un ENIAC.

La programmazione in linguaggio macchina, però, rimane pur sempre drammaticamente noiosa e soggetta ad errori materiali.

Le cose si possono migliorare di molto con un vecchio cavallo di battaglia dei matematici: un cambio di notazione.

Esempio: Linguaggio macchina MIPS

```
00000001001010100100000000100000
```

Campi: `000000|01001|01010|01000|00000|100000`

`OpCode|Reg Sr1|Reg Sr2|Reg Dst|Shift|Funz`

Significato: Somma i contenuti dei registri 9 e 10, e metti il risultato nel registro 8

```
add $8, $9, $10
add $t0, $t1, $t2
```

Poter programmare un calcolatore in linguaggio macchina, come s'è già detto, è un enorme passo avanti rispetto, diciamo, alla prospettiva di dover riconfigurare un ENIAC.

La programmazione in linguaggio macchina, però, rimane pur sempre drammaticamente noiosa e soggetta ad errori materiali.

Le cose si possono migliorare di molto con un vecchio cavallo di battaglia dei matematici: un cambio di notazione.

Esempio: Linguaggio macchina MIPS

```
00000001001010100100000000100000
```

Campi: `000000|01001|01010|01000|00000|100000`

```
OpCode|Reg Sr1|Reg Sr2|Reg Dst|Shift|Funz
```

Significato: Somma i contenuti dei registri 8 e 9, e metti il risultato nel registro 10

Codice Assembly:

```
add $8, $9, $10
add $t0, $t1, $t2
```

Il codice o liguaggio assembly è una traduzione mnemonica del linguaggio macchina. Sebbene nella pratica le cose siano un po' più intricate, concettualmente è corretto assumere l'esistenza di una *biiezione* che associ ciascuna istruzione assembly alla sua corrispondente istruzione macchina, e viceversa.

L'esperienza dimostra che questo semplice cambio di notazione facilita enormemente l'attività del programmatore.

Naturalmente, rimane un problema: chi si occuperà di tradurre per la CPU il codice assembly in codice macchina? Ossia, chi computerà la biiezione di cui sopra?

Il codice o liguaggio assembly è una traduzione mnemonica del linguaggio macchina. Sebbene nella pratica le cose siano un po' più intricate, concettualmente è corretto assumere l'esistenza di una *biiezione* che associ ciascuna istruzione assembly alla sua corrispondente istruzione macchina, e viceversa.

L'esperienza dimostra che questo semplice cambio di notazione facilita enormemente l'attività del programmatore.

Naturalmente, rimane un problema: chi si occuperà di tradurre per la CPU il codice assembly in codice macchina? Ossia, chi computerà la biiezione di cui sopra?

Risposta. La traduzione è eseguita per via automatica da un opportuno programma, scritto in codice macchina una volta per tutte.

Questo programma-traduttore si chiama assemblatore (o assembler).

Si noti che l'assemblatore si comporta come un traduttore, non come un interprete (simultaneo): esso riceve in ingresso l'*intero* testo da tradurre (=il programma scritto in assembly) e produce in uscita il testo tradotto (=il programma in linguaggio macchina eseguibile dalla CPU).

Il processo di traduzione eseguito dall'assemblatore è detto assemblaggio, o, con un termine generico che si applica anche ad altri contesti, compilazione.

Grazie all'architettura di von Neumann, il programma assembly da tradurre può risiedere nella memoria centrale, al pari di qualunque altro dato e programma, incluso l'assemblatore.

Per scrivere materialmente il programma assembly si usano programmi ausiliari, gli editor (=semplici programmi per l'elaborazione dei testi.)

Una volta capito il trucco, però, ci vuol poco ad annoiarsi anche della programmazione in assembly: e si generalizza.

Possiamo concepire un linguaggio di programmazione le cui istruzioni abbiano un significato operazionale più complesso di quelle tipiche dell'assembly e del codice macchina.

Queste istruzioni più astratte – o di più alto livello, come si dice in gergo informatico – potranno poi essere tradotte in opportune sequenze di istruzioni assembly, e da lì in codice macchina.

Nascono così i linguaggi di programmazione moderni, o semplicemente, in questo contesto, i linguaggi.

Una volta capito il trucco, però, ci vuol poco ad annoiarsi anche della programmazione in assembly: e si generalizza.

Possiamo concepire un linguaggio di programmazione le cui istruzioni abbiano un significato operazionale più complesso di quelle tipiche dell'assembly e del codice macchina.

Queste istruzioni più astratte – o di più alto livello, come si dice in gergo informatico – potranno poi essere tradotte in opportune sequenze di istruzioni assembly, e da lì in codice macchina.

Nascono così i linguaggi di programmazione moderni, o semplicemente, in questo contesto, i linguaggi.

Come per le lingue naturali, vi sono molti più linguaggi di programmazione in uso di quanti non sia possibile anche solo menzionare qui. Citiamo solo tre linguaggi storicamente importanti, e tuttora in uso.

- 1957: FORTRAN (Formula Translating System). Pensato per applicazioni al calcolo scientifico. Inventore principale: John Backus. Finanziatore: IBM.

- 1959: COBOL (Common Business-Oriented Language). Pensato per applicazioni gestionali e finanziarie. Inventrice principale: Grace Hopper. Finanziatori: Un consorzio di industrie e università statunitensi, cofinanziate dal governo.

- 1969–1972: C. Pensato come strumento per la programmazione di sistema all'interno del sistema operativo UNIX. Inventori principali: Dennis Ritchie e Ken Thompson.

- 1957: FORTRAN (Formula Translating System). Pensato per applicazioni al calcolo scientifico. Inventore principale: John Backus. Finanziatore: IBM.

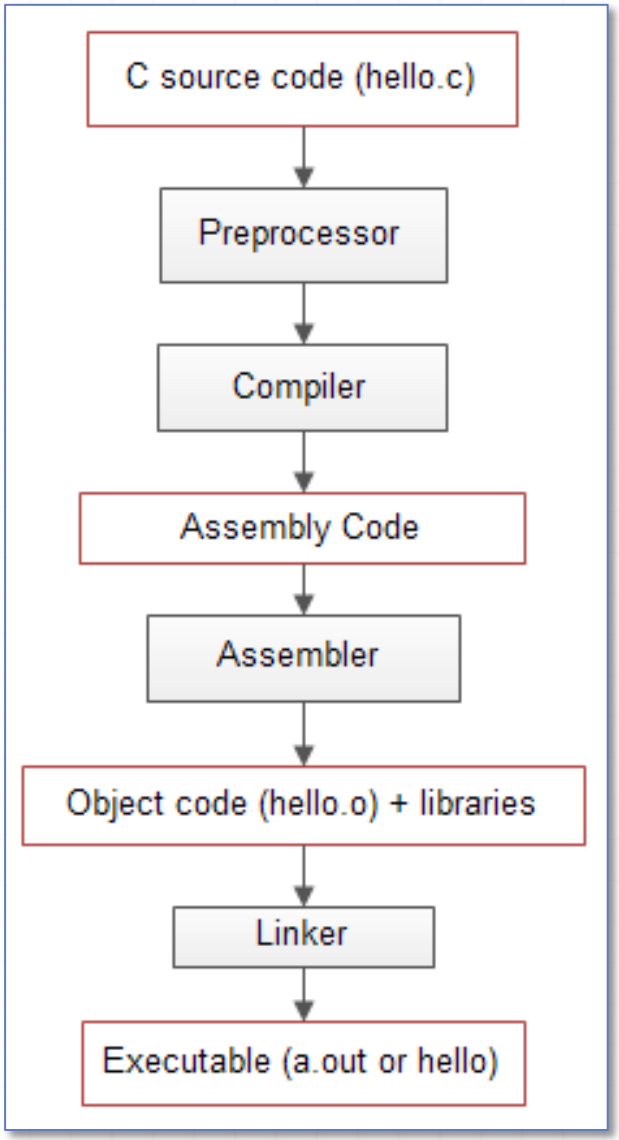
- 1959: COBOL (Common Business-Oriented Language). Pensato per applicazioni gestionali e finanziarie. Inventrice principale: Grace Hopper. Finanziatori: Un consorzio di industrie e università statunitensi, cofinanziate dal governo.

- 1969–1972: C. Pensato come strumento per la programmazione di sistema all'interno del sistema operativo UNIX. Inventori principali: Dennis Ritchie e Ken Thompson.

K. Thompson e D. Ritchie

1. **Tassonomia**. I linguaggi si possono categorizzare in molti modi diversi: compilati *vs*. interpretati, alto livello *vs*. basso livello, imperativi *vs*. dichiarativi, procedurali *vs*. orientati agli oggetti, e così via *ad libitum*. Come descrizione di massima: **Il C è un linguaggio compilato, imperativo, procedurale, di basso livello**.

2. **Standard**. Tutti i linguaggi di programmazione di uso corrente hanno una definizione formale precisa. Il linguaggio C non fa eccezione. Il libro di testo del corso si basa sullo *standard ANSI C* del 1989-1990, cui ci si riferisce come C90, o a volte come C89. Le versioni correnti del compilatore `gcc` rispettano tipicamente il successivo standard C99, del 1999. Ai nostri fini non è importante far riferimento allo standard più recente del 2011, il C11, che pochi compilatori implementano compiutamente. **Faremo di solito riferimento allo standard C99**.



Il ciclo di compilazione in C