Le componenti fisiche di un computer: l'hardware

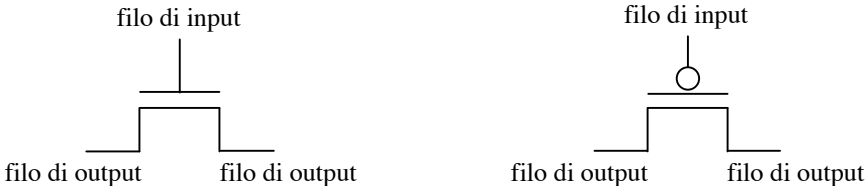
In questa sezione ci occuperemo di come è strutturato e come funziona l'hardware di un computer. In particolare, nella Sezione 1 ci occuperemo del "punto di incontro" tra hardware e software, ovvero di come è possibile "far ragionare" l'hardware mediante software. Nella Sezione 2 descriveremo la macchina di von Neumann descrivendo cosí la struttura generale di tutti i moderni computer e chiarendo i concetti di automatico, algoritmo e di macchina introdotti precedentemente. Nella Sezione 5 descriveremo i "cinque sensi" di un computer, ovvero descriveremo tutti gli apparati, oggidì più comuni, che permettono ad un computer di interagire con il mondo esterno ed in particolare con l'*utente* (la persona che usa il computer).

1. Il linguaggio binario e sua rappresentazione in hardware

Come noi umani, quando ragioniamo, pensiamo in una lingua composta da determinate parole in un certo alfabeto (tale lingua sarà l'italiano se siamo italiani, l'inglese se siamo inglesi, il cinese se siamo cinesi, ecc.), così un computer "ragiona" usando varie lingue tutte composte da parole nell'alfabeto binario costituito dalle sole due lettere 0 e 1. Nel caso dei computer, una lingua si chiama *codice* e una lettera di una parola di codice si chiama *bit*. Più specificatamente, **1 bit** è una lettera o variabile binaria (che quindi può assumere esclusivamente il valore 0 o 1) e **rappresenta l'unità di misura base della quantità di dati**. Le altre unità di misura per la quantità di dati fondamentali sono:

1 byte	= 8 bits		= 2^3 bits,
1Kilo byte	= 1024 bytes	= 2^{10} bytes	= 2^{13} bits,
1 Mega byte	= 1024 Kilo bytes	= 2^{10} Kilo bytes	= 2^{23} bits,
1 Giga byte	= 1024 Mega bytes	= 2^{10} Mega bytes	= 2^{33} bits,
1 Tera byte	= 1024 Giga bytes	= 2^{10} Giga bytes	= 2^{43} bits.

Figura 1: simboli schematici per i transistors.



transistor di tipo NPN transistor di tipo PNP

Di solito gli esseri umani rappresentano i numeri naturali nel sistema decimale ovvero come somme di potenze di 10 a coefficienti 0,1,..., 9. Ad esempio la rappresentazione decimale del numero 89 è $89 = \underline{8}\cdot 10^1 + \underline{9}\cdot 10^0$. La cifra più a destra è il numero di unità decimali di cui è composto il numero, la cifra appena a sinistra di questa è il numero di decine decimali, e così via per le centinaia, migliaia, eccetera. Viceversa, una sequenza finita di cifre decimali (ovvero, simboli da 0 a 9) può essere pensata come la rappresentazione decimale di un numero: quello il cui numero di unità decimali è pari alla cifra più a destra della sequenza, il cui numero di decine decimali è pari alla cifra appena a sinistra delle unità, e così via per le centinaia, migliaia, eccetera. Siccome l'alfabeto usato dai computer è quello binario, in essi ogni numero naturale è rappresentato nel

sistema binario; ovvero come somme di potenze di 2 a coefficienti 0 e 1. Ad esempio, la rappresentazione binaria del numero 89 è:

$$\begin{aligned} 89 &= \underline{1}\cdot 2^6 + (89 - 64) = \underline{1}\cdot 2^6 + 25 \\ &= \underline{1}\cdot 2^6 + \underline{0}\cdot 2^5 + 25 \\ &= \underline{1}\cdot 2^6 + \underline{0}\cdot 2^5 + \underline{1}\cdot 2^4 + (25\text{-}16) = \underline{1}\cdot 2^6 + \underline{0}\cdot 2^5 + \underline{1}\cdot 2^4 + 9 \\ &= \underline{1}\cdot 2^6 + \underline{0}\cdot 2^5 + \underline{1}\cdot 2^4 + \underline{1}\cdot 2^3 + (9\text{-}8) = \underline{1}\cdot 2^6 + \underline{0}\cdot 2^5 + \underline{1}\cdot 2^4 + \underline{1}\cdot 2^3 + 1 \\ &= \underline{1}\cdot 2^6 + \underline{0}\cdot 2^5 + \underline{1}\cdot 2^4 + \underline{1}\cdot 2^3 + \underline{0}\cdot 2^2 + \underline{0}\cdot 2^1 + \underline{1}\cdot 2^0 \\ &= 1011001. \end{aligned}$$

Si noti che il funzionamento del sistema binario è esattamente lo stesso del sistema decimale, l'unica cosa che cambia è la base: nel sistema binario i numeri vengono espressi come somme di potenze di 2 anziché di 10. Come nel caso decimale, una sequenza finita di cifre binarie (ovvero, di simboli 0 e 1)) può essere pensata come la rappresentazione (binaria) di un numero. Ad esempio, la sequenza 1001011 rappresenta il numero

$$\begin{aligned} 1001011 \qquad &= \underline{1}\cdot 2^6 + \underline{0}\cdot 2^5 + \underline{0}\cdot 2^4 + \underline{1}\cdot 2^3 + \underline{0}\cdot 2^2 + \underline{1}\cdot 2^1 + \underline{1}\cdot 2^0 \\ &= \underline{1}\cdot 64 + \underline{0}\cdot 32 + \underline{0}\cdot 16 + \underline{1}\cdot 8 + \underline{0}\cdot 4 + \underline{1}\cdot 2 + \underline{1}\cdot 1 \\ &= 64 + 8 + 2 + 1 \\ &= 75. \end{aligned}$$

Per questi motivi, è anche possibile assumere che un computer ragioni pensando numeri naturali in quanto ragionare pensando numeri naturali o sequenze binarie sono due cose equivalenti.

Ogni componente hardware di un computer è costituito essenzialmente *da circuiti elettrici* ovvero da *transistors* (fatti di materiale semiconduttore) connessi tra loro da fili di metallo (alluminio o rame) detti *fili elettrici*. I fili elettrici memorizzano le lettere binarie costituenti una parola di codice oppure trasportano le lettere di una parola di codice da un transistor all'altro. I transistors permettono di manipolare (cambiare o meno) le lettere binarie memorizzate e/o trasportate dai fili. Ora, le lettere binarie sono rappresentate nei fili da due stati fisici particolari dei fili stessi: per convenzione, il filo trasporta e/o memorizza la lettera 0 se il suo potenziale elettrico è $GND = 0$ volt e trasporta e/o memorizza la lettera 1 se il suo potenziale elettrico è V_{DD} (oggidì, il valore di potenziale elettrico che comunemente si usa per rappresentare la lettera binaria 1 è circa $V_{DD} = 3$ volt, cinque anni fa era $V_{DD} = 5$ volt e fra cinque anni sarà $V_{DD} = 2{,}5$ volt o meno. Si cerca di ridurre tale valore per questioni di velocità e consumo di energia). Si noti che è proprio questo il punto di contatto tra hardware e software: l'hardware è costituito dai fili elettrici ed il software dallo stato fisico (specificatamente, dal valore di potenziale) dei fili elettrici.

Menzioniamo che, i circuiti elettrici in cui per convenzione hanno senso solo un numero finito di stati fisici (due nel nostro caso), sono detti *circuiti digitali*, mentre quelli per cui ha senso ogni stato fisico (che non faccia scoppiare la macchina) sono detti *circuiti analogici*. L'hardware di un computer è per lo più costituito da circuiti digitali binari (ovvero a due stati).

2. I transistors

I transistors opportunamente connessi tra loro manipolano (cambiano o meno) le lettere binarie (ovvero lo stato fisico) memorizzate e/o trasportate dai fili elettrici. Questa manipolazione avviene come descritto nella Sezione 3. In questa sezione diciamo che un transistor è un apparato a cui sono connessi tre fili: uno detto di input e due detti di output. Ora, se nel filo di input c'è 0 (oppure 1) i due fili di output non sono connessi tra loro. Se nel filo di input c'è 1 (oppure 0) i due fili di output sono connessi tra loro. Si può quindi dire che un transistor sia un interruttore elettrico controllato elettronicamente (invece che con un pulsante come gli usuali interruttori di lampadine che sono in ogni stanza di ogni casa). Ci sono quindi due tipi di transistors: quelli che sconnettono i due fili di output se nel filo di input c'è 0 e quelli che sconnettono i due fili di output se nel filo di input c'è 1. Il primo tipo di transistor è chiamato transistor di tipo NPN (o anche di tipo N) mentre il secondo tipo di transistor è chiamato transistor di tipo PNP (o anche di tipo P). I due tipi di transistors si comportano in maniera complementare e sono rappresentati graficamente come in Figura 1.

3. I gate booleani

In questa sezione vediamo come è possibile connettere tra loro i transistors in modo da manipolare le lettere binarie 0 e 1 e, per esempio, realizzare dei circuiti, detti *gate booleani*, che computino i connettivi logici fondamentali: NOT, AND e OR. Il connettivo NOT rappresenta la negazione ed è una funzione di una variabile binaria che a 0 associa NOT(0) = 1 e ad 1 associa NOT(1) = 0. Tale connettivo è rappresentato dalla seguente *tavola di verità*:

x	NOT(x)
0	1
1	0

Il connettivo AND rappresenta la congiunzione ed è una funzione di due variabili binarie definita dalla seguente *tavola di verità*:

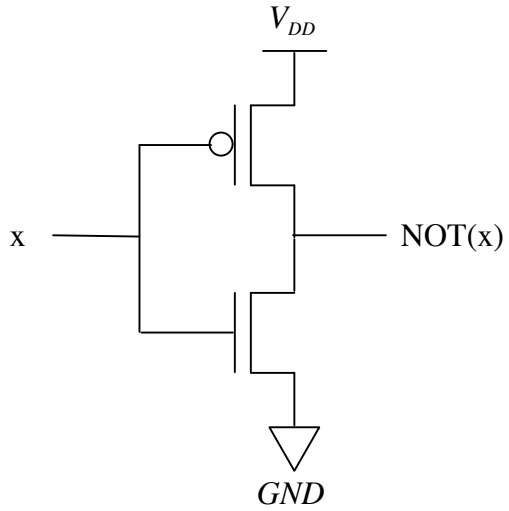
x	y	AND(x,y)
0	0	0
0	1	0
1	0	0
1	1	1

Il connettivo OR rappresenta la congiunzione ed è una funzione di due variabili binarie definita dalla seguente *tavola di verità*:

x	y	OR(x,y)
0	0	0
0	1	1
1	0	1
1	1	1

I connettivi NOT, AND e OR sono chiamati fondamentali perché componendo fra loro tali funzioni in modo opportuno è possibile ottenere tutte le funzioni di variabili binarie a valori binari.
Un modo per realizzare un gate booleano, detto anche *inverter*, che computi il connettivo logico NOT è quello definito dalla cosiddetta tecnologia CMOS (Complementary Metal Oxide Semiconductor) e raffigurato in Figura 2.
Come si può vedere dalla Figura 2, un inverter è costituito da un transistor di tipo PNP (sopra) e da un transistor di tipo NPN (sotto). Gli input dei due transistors sono connessi tra loro in un filo che trasporta il valore di input x. L'output di sopra del transistor di tipo PNP è ha V_{DD} volts (che corrisponde al valore logico binario 1). L'output di sotto del transistor di tipo NPN è ha GND volts (che corrisponde al valore logico binario 0). L'output di sotto del transistor di tipo PNP e l'output di sopra del transistor di tipo NPN sono connessi tra loro in un filo che trasporta il valore di output NOT(x). Vediamo come funziona il circuito. Se l'input x è 0, il transistor PNP ha i suoi fili di output connessi e il transistor NPN ha i suoi fili di output sconnessi. Quindi l'output dell'inverter è connesso con l'output di sopra del transistor PNP e quindi è al suo stesso potenziale elettrico, V_{DD} che rappresenta 1. Quindi se il filo di input all'inverter trasporta 0 il filo di output trasporterà 1. Viceversa, Se l'input x è 1, il transistor PNP ha i suoi fili di output sconnessi e il transistor NPN ha i suoi fili di output connessi. Quindi l'output dell'inverter è connesso con l'output di sotto del transistor NPN e quindi è al suo stesso potenziale elettrico, GND che rappresenta 0. Quindi se il filo di input all'inverter trasporta 1 il filo di output trasporterà 0.

Figura 2: NOT gate o inverter.



Analoghi circuiti si possono progettare per i connettivi logici AND e OR e per celle di memoria (ovvero circuiti che memorizzino una lettere dell'alfabeto binario ovvero i bit di dati).

4. l'architettura di von Neumann

Tutti i più moderni computer sono strutturati secondo il modello di macchina di von Neumann e sono realizzati con una tecnologia che prende il nome di VLSI (Very Large Scale Integration). Tale tecnologia permette di miniaturizzare i circuiti elettrici.
In VLSI, tutti i circuiti elettrici costituenti la logica di un computer (o un sistema in genere) sono suddivisi in vari pezzettini rettangolari di silicio detti *chips* connessi tra loro da fili elettrici e situati su un circuito stampato o scheda (circuit board). Oggi, ogni chip può contenere all'incirca fino a 1200 milioni = 1,2 miliardi (ad esempio il chip Dual-Core Intel Itanium 2 dell'INTEL) di transistors ed ha dimensioni fino a 1,5 cm^2. Questo perché la più piccola forma che può essere delineata su di un chip (the minimum feature patternable) ha una grandezza di $\lambda = 45$ nm (1 nm = 1 nanometro = un miliardesimo di metro = un millesimo di micron. Si noti che le dimensioni di un virus si aggirano intorno a 18-20 nm). È importante notare che tale parametro λ tende a diminuire con lo sviluppo della tecnologia VLSI e presto saranno sul mercato processori prodotti con un processo a $\lambda = 32$ nm.
Esponiamo che cosa è la macchina di von Neumann; chiarendo in tal modo il concetto di automatico senza ambiguità. Diamo la seguente Definizione.
Definizione di macchina di von Neumann (modello di calcolo pratico): una macchina di von Neumann è una terna

$$(\mathbf{N}, IS, P)$$

Dove

N = {0,1,2,…} è l'insieme dei numeri naturali e rappresenta l'alfabeto della macchina. Si noti che questa componente è fissa ovvero uguale per ogni macchina.

IS = {ZERO, INC, SOM, SOT, MOL, DIV, UGUALE, MINORE; SALCOND, ALT} (Instruction Set) è l'insieme di istruzioni generiche della macchina. Si noti che anche questa componente è fissa ovvero uguale per ogni macchina. Tale insieme di istruzioni generiche è definito come segue. Le prime otto istruzioni implementano le funzioni a valori in **N** così definite:

$$\text{ZERO}: \mathbf{N} \to \mathbf{N}$$
$$n \to \text{ZERO}(n) = 0,$$

$$\text{INC}: \mathbf{N} \to \mathbf{N}$$
$$n \to \text{INC}(n) = n+1,$$

$$\text{SOM}: \mathbf{N} \times \mathbf{N} \to \mathbf{N}$$
$$(n,m) \to \text{SOM}(n,m) = n + m,$$

$$\text{SOT: } \mathbf{N} \times \mathbf{N} \to \mathbf{N}$$
$$(n,m) \to \text{SOT}\ (n,m) = \begin{cases} n - m & \text{se } n \geq m, \\ 0 & \text{se } n < m \end{cases},$$

$$\text{MOL: } \mathbf{N} \times \mathbf{N} \to \mathbf{N}$$
$$(n,m) \to \text{MOL}\ (n,m) = n \times m,$$

$$\text{DIV: } \mathbf{N} \times \mathbf{N} \to \mathbf{N}$$
$$(n,m) \to \text{DIV}\ (n,m) = \lfloor n / m \rfloor,$$

dove $\lfloor x \rfloor$ indica la parte intera del numero reale x, ovvero il numero intero più grande minore od uguale ad x.

$$\text{UGUALE: } \mathbf{N} \times \mathbf{N} \to \mathbf{N}$$
$$(n,m) \to \text{UGUALE}\ (n,m) = \begin{cases} 1 & \text{se } n = m, \\ 0 & \text{se } n \neq m \end{cases},$$

$$\text{MINORE: } \mathbf{N} \times \mathbf{N} \to \mathbf{N}$$
$$(n,m) \to \text{MINORE}\ (n,m) = \begin{cases} 1 & \text{se } n < m, \\ 0 & \text{se } n \geq m \end{cases},$$

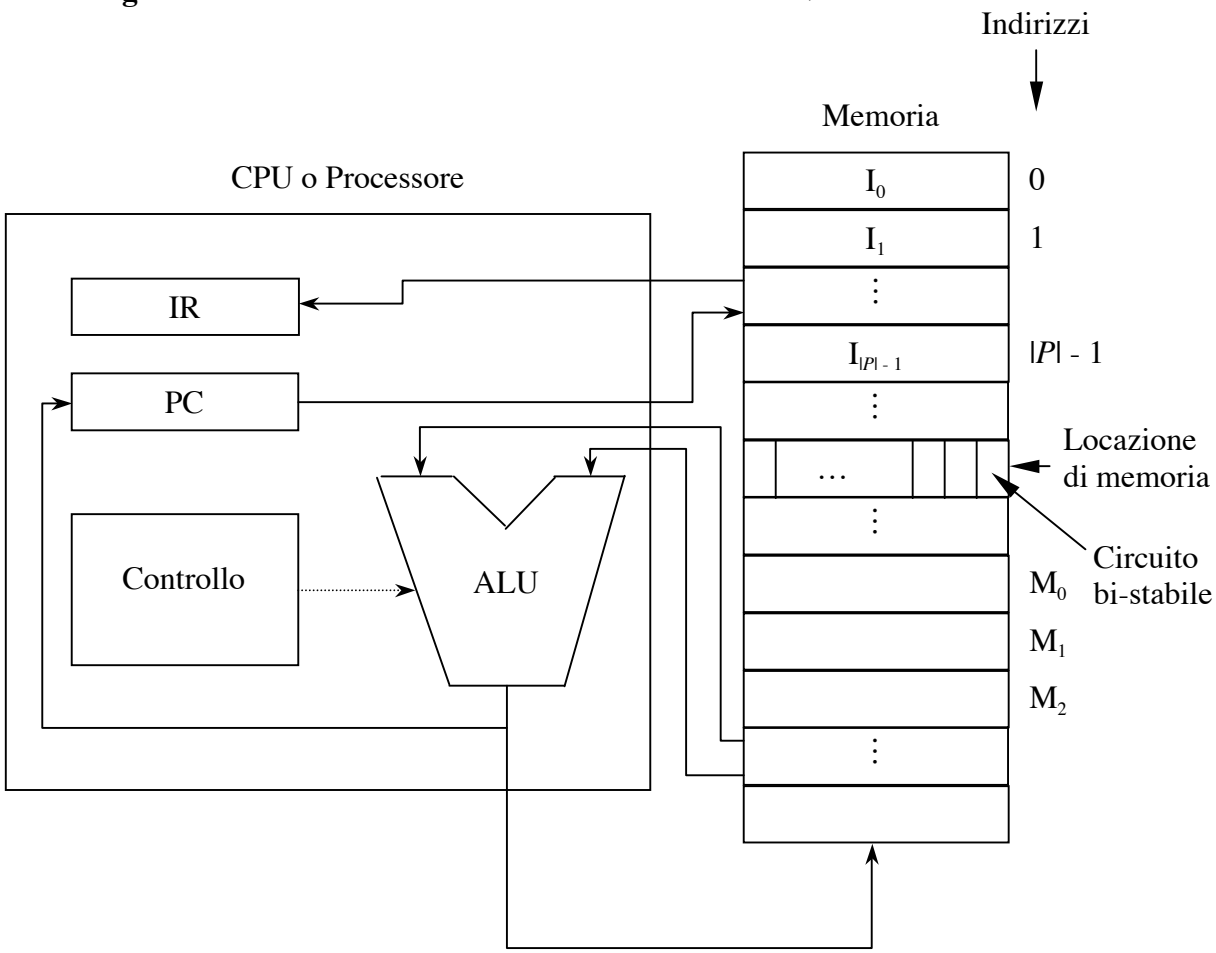
La nona istruzione, SALCOND, è un istruzione speciale, detta istruzione di salto condizionato. Essa è una funzione che ad un numero naturale ed un indice, detto indirizzo di *P*, associa l'azione di "saltare all'istruzione di *P"* specificata dall'indice. Questo, condizionatamente al fatto che il numero naturale sia diverso da zero o meno. Formalmente,

$$\text{SALCOND: } \mathbf{N} \times J \to \textit{Azione}$$
$$(n, j) \to \text{SALCOND}\ (n, j) = \begin{cases} \text{salta a } j & \text{se } n \neq 0, \\ \text{non salta a } j & \text{se } n = 0 \end{cases},$$

essendo $J = \{j \in \mathbf{N}: 0 \leq j < |P| - 1\}$ un insieme finito di indici.
La decima istruzione, ALT, è una costante uguale all'azione di fermare, terminare la computazione della macchina.

P $= \{I_0, I_1, I_2, \ldots, I_{|P|-1}\}$ una sequenza finita non vuota di istruzioni specifiche, detta *programma* della macchina. Per istruzione specifica intendiamo ogni istruzione (ovvero, elemento di *IS*) in cui si siano specificati particolari valori alle variabili indipendenti. L'insieme di tali valori, al variare delle istruzioni specifiche nel programma, è detto insieme dei dati del programma.

Figura 3: hardware costituente la macchina di von Neumann.



Si assume che la computazione della macchina avvenga eseguendo le istruzioni del programma, nell'ordine definito dal programma (prima si esegue la prima istruzione, poi la seconda, ecc), a meno di salti condizionati. Tutto ciò nel seguente modo. Si immagina che l'hardware della macchina sia costituito da una *memoria* e da un *processore*, come in Figura 3. La memoria è un sistema costituito da una sequenza sufficientemente grande di *locazioni di memoria*, ognuna contenente un numero naturale. Oggi che ogni computer ha un'architettura a 64 bits, ogni locazione di memoria è costituita da 64 circuiti bi-stabili (ovvero, lampadine) e può quindi contenere 64 bits = 8 bytes di dati. Ciò fa si che, in realtà, ogni locazione di memoria possa contenere solo tutti i numeri da 0 a 2^{64}-1. Ad ogni locazione di memoria è associato l'indice della locazione nella sequenza detto *indirizzo* della locazione. Nella memoria sono registrati il programma e i dati del programma. Il programma (o, per meglio dire, una codifica naturale del programma) è registrato nelle locazioni di memoria i cui indirizzi vanno da 0 a $|P| - 1$ in modo che l'i-esima istruzione del programma è registrata nella locazione di memoria il cui indirizzo è i. In un'altra parte della memoria sono registrati i dati necessari per l'esecuzione del programma, ovvero, i dati del programma. Il processore è un sistema

costituito da quattro parti fondamentali: il *contatore di programma* (in inglese Program Counter o PC), il registro delle istruzioni (in inglese, Instruction Register o IR), l'*unità aritmetica e logica* (in inglese Arithmetic and Logic Unit o ALU) ed il *controllo*. Il contatore di programma è una locazione di memoria contenente l'indirizzo dell'istruzione da eseguire. Il registro delle istruzioni è una locazione di memoria contenente l'istruzione da eseguire. L'unità aritmetica e logica è un sistema che esegue l'istruzione indicata dal PC e registrata in IR. Il controllo è un sistema che, attraverso una sequenza di cambiamenti di stato, "eccita e/o inibisce" opportunamente determinate parti del processore (tra cui l'ALU) facendo avvenire **l'esecuzione di un'istruzione** tramite l'esecuzione delle seguenti azioni.

A1 Legge il contenuto del PC.
A2 Fa pervenire nel processore (in inglese, to fetch) l'istruzione contenuta nella locazione di memoria programma il cui indirizzo e contenuto nel PC.
A3 Decodifica tale istruzione, ovvero
  A3.1 "capisce" che istruzione è tra le dieci possibili, ovvero invia dei segnali all'ALU (ed eventuali altre parti del processore) in modo che questa esegua la funzione caratteristica della data istruzione, e
  A3.2 acquisisce dalla memoria i dati specificati dall'istruzione necessari per eseguire l'istruzione. In un'istruzione (specifica) del programma, i dati sono specificati in maniera indiretta tramite gli indirizzi delle locazioni di memoria contenenti i dati dell'istruzione. Ad esempio, un istruzione del programma può essere SOM(M_2,M_0) in cui M_2 e M_0 sono i due indirizzi delle locazioni di memoria contenenti i due numeri (dati) da sommare. A ciò fa eccezione l'istruzione SALCOND per quanto riguarda il secondo argomento (operando) che viene dato in maniera diretta. Ad esempio un operazione specifica SALCOND può essere SALCOND(M_1,12302).
A4 Aspetta che l'ALU calcoli il risultato.
A5 Registra il risultato nella locazione di memoria specificata dall'operando più a sinistra dell'istruzione. Se, ad esempio, l'istruzione in esecuzione è SOM(M_2,M_0), il controllo registra il risultato nella locazione di memoria il cui indirizzo è M_2. Nel caso in cui, l'istruzione in esecuzione è SALCOND(*n*,*j*) ed *n* è diverso da zero, il controllo registra il numero *j* (che rappresenta l'indirizzo della prossima istruzione da eseguire) nel PC.
A 6 Incrementa il PC a meno che l'istruzione in esecuzione non sia un SALCOND(*n*,*j*) con *n* diverso da zero o un ALT.

Il ciclo di esecuzione di un'istruzione si ripete fin tanto che non si incontra l'istruzione ALT, per cui la macchina si ferma. Nei computer moderni un chip contiene la CPU (ed eventualmente anche parte della memoria), mentre un altro chip (o più chips) contiene la memoria. Le comunicazioni tra la CPU e la memoria (rappresentate dalle frecce in Figura 3) avviene attraverso un canale di comunicazione composto da più fili paralleli detto *bus*.
Diamo ora un esempio di programma per la macchina di von Neumann che calcola se un numero è divisibile per 4 o meno. Si noti che un numero *n* è divisibile per 4 se, e solo se, il resto, *r*, della divisione tra *n* e 4 è 0. Si noti che tale resto è proprio dato da $r = n - 4 \cdot \lfloor n / 4 \rfloor$. Assumiamo che il numero *n* sia registrato nella locazione di memoria M_0. Il seguente

programma scrive il risultato nella locazione di memoria M_1. In particolare, scrive 0 se il numero non è divisibile per 4 e scrive 1 se il numero è divisibile per 4.

```
0  ZERO(M1)
1  INC(M1)
2  INC(M1)
3  INC(M1)
4  INC(M1)
5  ZERO(M2)
6  SOM(M2,M0)
7  DIV(M2,M1)
8  MOL(M2,M1)
9  SOT(M0,M2)
10 SALCOND(M0,14)
11 ZERO(M1)
12 INC(M1)
13 ALT
14 ZERO(M1)
15 ALT
```

} scrive 4 in M_1.

} copia il contenuto di M_0 (n) in M_2.

} calcola $r = n - 4 \cdot \lfloor n/4 \rfloor$ e mette il risultato in M_0.

} se $r \neq 0$ salta a 14.

} scrive 1 in M_1 se $r = 0$ e si ferma.

} scrive 0 in M_1 se $r \neq 0$ e si ferma.