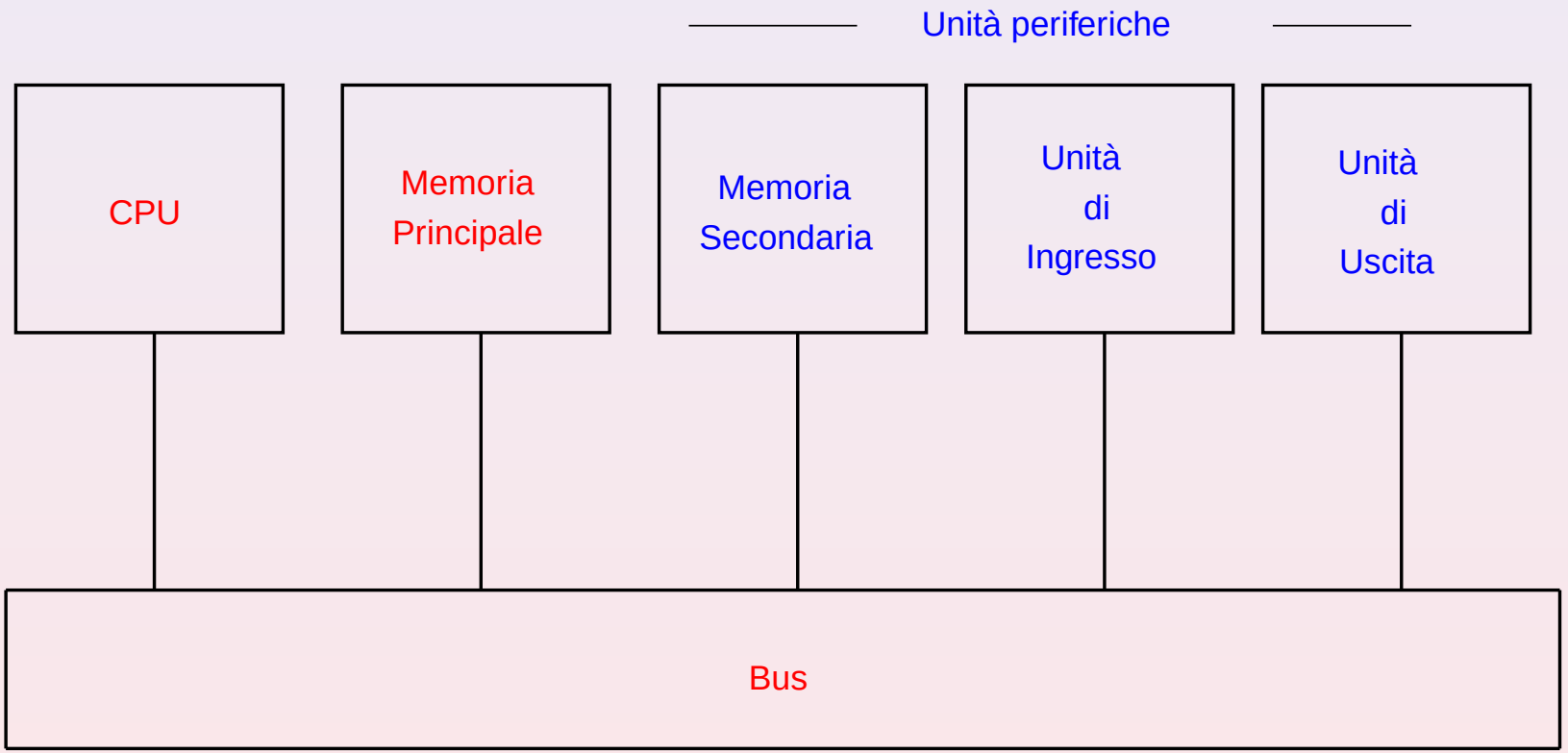
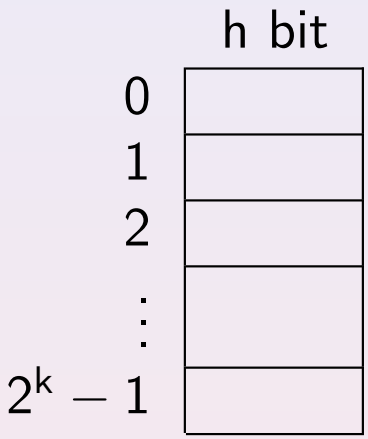
Architettura di Von Neumann

- L'architettura è ancora quella classica sviluppata da Von Neumann nel 1947.
- L'architettura di Von Neumann riflette le funzionalità richieste da un elaboratore:
  - memorizzare i dati e i programmi $\Longrightarrow$ memoria principale
  - i dati devono essere elaborati $\Longrightarrow$ unità di elaborazione (CPU)
  - comunicazione con l'esterno $\Longrightarrow$ unità di ingresso/uscita (periferiche)
  - le componenti del sistema devono scambiarsi informazioni $\Longrightarrow$ bus di sistema



Tra le periferiche evidenziamo la memoria secondaria.

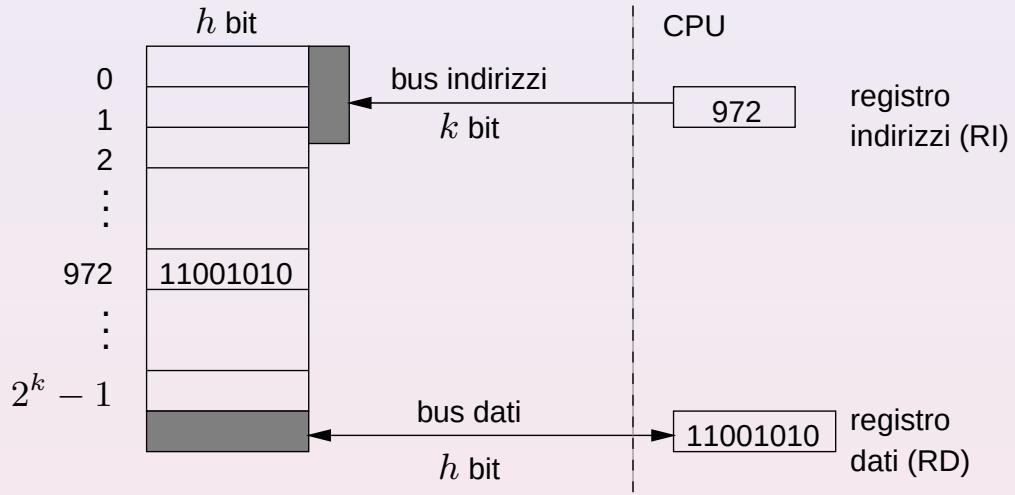
Memoria centrale (o RAM)



- è una sequenza di celle di memoria (dette parole), tutte della stessa dimensione
- ogni cella è costituita da una sequenza di bit
- il numero h di bit di una cella di memoria (dimensione) dipende dall'elaboratore, ed è un multiplo di 8: 8, 16, 32, 64
- ogni cella di memoria è identificata in modo univoco dal suo indirizzo
- il numero k di bit necessari per l'indirizzo dipende dal numero di celle di memoria

$$k \text{ bit} \Longrightarrow 2^k \text{ celle}$$

Memoria centrale



Operazione di lettura:

1. CPU scrive l'indirizzo della cella di memoria da cui leggere nel registro indirizzi (RI)
2. esegue l'operazione ("apre i circuiti")
3. il valore della cella indirizzata viene trasferito nel registro dati (RD)

Operazione di scrittura: al contrario

Memoria centrale

Caratteristiche principali

- è una memoria ad accesso casuale, ossia il tempo di accesso ad una cella di memoria è indipendente dalla posizione della cella $\Longrightarrow$ viene chiamata RAM (random access memory)
- può essere sia letta che scritta
  scrittura distruttiva – lettura non distruttiva
- alta velocità di accesso
- è volatile (si perde il contenuto quando si spegne il calcolatore)

Dimensione della memoria: misurata in byte (1 byte=8 bit)

Kilobyte	$=$	2^{10}	$\sim$	10^3 byte
Megabyte	$=$	2^{20}	$\sim$	10^6 byte
Gigabyte	$=$	2^{30}	$\sim$	10^9 byte
Terabyte	$=$	2^{40}	$\sim$	10^{12} byte

Memoria secondaria

- non volatile
- capacità maggiore della memoria centrale (decine di GB)
- tempo di accesso lento rispetto alla memoria centrale
- accesso sequenziale e non casuale
- tipi di memoria secondaria: dischi rigidi, floppy, CDROM, CDRW, DVD, nastri, ...

Bus di sistema suddiviso in tre parti:

- bus indirizzi: k bit
- bus dati: h bit
- bus comandi: trasferisce i comandi tra le varie unità

  $\Longrightarrow$ parallelismo (attualmente si arriva a 128 bit)

CPU (Central Processing Unit)

- coordina le attività di tutte le componenti del calcolatore
- interpreta ed esegue le istruzioni del programma
- 3 componenti principali:

  unità logico-aritmetica (ALU): effettua i calcoli

  unità di controllo: coordinamento di tutte le operazioni

  registri: celle di memoria ad accesso molto veloce
  - registro istruzione corrente (IR): contiene l'istruzione in corso di esecuzione
  - contatore di programma (PC): contiene l'indirizzo della prossima istruzione da eseguire
  - accumulatori: utilizzati dalla ALU per gli operandi ed il risultato
  - registro dei flag: memorizza alcune informazioni sul risultato dell'ultima operazione (carry, zero, segno, overflow, ...)
  - registro interruzioni: utilizzato per la comunicazione con le periferiche
  - registro indirizzi (RI) e registro dati (RD) per il trasferimento da e verso la memoria centrale

CPU

Ciclo dell'unità di controllo

- Tutte le attività interne alla CPU sono regolate da un orologio (clock) che genera impulsi regolari ad una certa frequenza (ad es. 800 MHz, 1 GHz, 2 GHz, ...).
- Il programma è memorizzato in celle di memoria consecutive, sulle quali l'unità di controllo lavora eseguendo il ciclo di

  prelievo — decodifica — esecuzione

```
while macchina in funzione do
preleva dalla memoria l'istruzione indirizzata da PC
e caricala in IR
  (aggiorna PC in modo che indirizzi la prossima istruzione)
decodifica l'istruzione in IR
esegui l'istruzione
endwhile
```

CPU - Ciclo dell'unità di controllo

1. fase di prelievo (fetch)
   l'unità di controllo acquisisce dalla memoria l'istruzione indirizzata da PC e aggiorna PC in modo che indirizzi la prossima istruzione
   $$PC = PC + n$$
   dove n è la lunghezza in byte dell'istruzione prelevata
2. fase di decodifica
   viene decodificato il tipo di istruzione per determinare quali sono i passi da eseguire per la sua esecuzione
3. fase di esecuzione
   vengono attivate le componenti che realizzano l'azione specificata

Istruzioni

Ogni istruzione è costituita da:

Tipi di istruzione

- istruzioni di trasferimento dati
  - da e verso la memoria centrale
  - ingresso/uscita
- istruzioni logico/aritmetiche
- istruzioni di controllo
  - istruzioni di salto

Le istruzioni dettano il flusso del programma. Vengono eseguite in sequenza, a meno che non vi sia un'istruzione di controllo che altera il normale flusso (istruzione di salto).

Dal codice sorgente al codice macchina

I concetti di algoritmo e di programma permettono di astrarre dalla reale struttura del calcolatore, che comprende sequenze di 0 e 1, ovvero un linguaggio macchina.

Livelli di astrazione ai quali possiamo vedere i programmi:

- Linguaggio macchina (o codice binario): livello più basso
  - un programma è una sequenza di 0 e 1 (suddivisi in parole) che codificano le istruzioni
  - dipende dal calcolatore
- Linguaggio assemblativo: livello intermedio
  - dipende dal calcolatore e le sue istruzioni sono in corrispondenza 1-1 con le istruzioni in linguaggio macchina
  - istruzioni espresse in forma simbolica $\Longrightarrow$ comprensibile da un umano
- Linguaggi ad alto livello: (e.g. C, Pascal, C++, Java, Fortran, ...)
  - si basano su costrutti non elementari, comprensibili da un umano
  - istruzioni più complesse di quelle eseguibili da un calcolatore (corrispondono a molte istruzioni in linguaggio macchina)
  - in larga misura indipendenti dallo specifico elaboratore

Per arrivare dalla formulazione di un problema all'esecuzione del codice che lo risolve, bisogna passare attraverso diversi stadi:

problema (specifica)
⇓
algoritmo (pseudo-codice)
⇓
codice sorgente (linguaggio ad alto livello)
⇓ compilazione — [compilatore]
codice oggetto (simile al codice macchina, ma con riferimenti simbolici)
⇓ collegamento tra le diverse parti — [collegatore (linker)]
codice macchina (eseguibile)
⇓ caricamento — [caricatore (loader)]
codice in memoria eseguito

Esempio: dati due interi positivi X ed Y, eseguire il loro prodotto usando solo le operazioni di somma e sottrazione

```
Input(X);
Input(Y);
somma = 0;
contatore = 0;
while (contatore < Y)
{
    somma = somma + X;
    contatore = contatore + 1;
}
Output(somma);
```

Codifica dell'algoritmo in C

```
#include <stdio.h>
 void main (void) {
  int x, y;
  int cont = 0;
  int somma = 0;
  printf("Introduci due interi da moltiplicare\n");
  scanf("%d%d", &x, &y);
  while (cont < y) {
    somma = somma + x;
    cont = cont + 1;
  }
  printf("La somma di %d e %d e' pari a %d\n", x, y, somma);
}
```

Linguaggio assembler

Vediamo per comodità un esempio di linguaggio assembler: (corrisponde 1-1 al codice in linguaggio macchina, ma è più leggibile).

Caricamento dati

LOAD R1 X LOAD R2 X	Carica nel registro R1 (o R2) il dato memorizzato nella cella di memoria identificata dal nome simbolico X
LOAD R1 #C	Carica nel registro R1 la costante numerica C

Somma e Sottrazione

SUM R1 R2 SUB R1 R2	Somma (sottrae) il contenuto di R2 al contenuto di R1 e memorizza il risultato in R1

Memorizzazione

STORE R1 X STORE R2 X	Memorizza il contenuto di R1 (R2) nella cella con nome simbolico X

Linguaggio assembler

Controllo

JUMP A	La prossima istruzione da eseguire è quella con etichetta A
JUMPZ A	Se il contenuto di R1 è uguale a 0, la prossima istruzione da eseguire è quella con etichetta A
STOP	Ferma l'esecuzione del programma

Lettura/Scrittura

READ X	Legge un dato e lo memorizza nella cella di nome simbolico X
WRITE X	Scrive il valore contenuto nella cella di nome simbolico X

Programma per il prodotto in linguaggio assembler

	Etic.	Istr. assembler	Istruzione C	Significato
0		READ X	scanf	Leggi valore e mettilo nella cella *X*
1		READ Y	scanf	Leggi valore e mettilo nella cella *Y*
2		LOAD R1 #0	cont = 0	Inizializzazione di *cont*; metti 0 in *R*1
3		STORE R1 CONT		Metti il valore di *R*1 in *CONT*
4		LOAD R1 #0	somma = 0	Inizializzazione di *SOMMA*; metti 0 in
5		STORE R1 SOMMA		Metti il valore di *R*1 in *SOMMA*
6	INIZ	LOAD R1 CONT	while (*cont* < *y*)	Esecuzione del test:
7		LOAD R2 Y		Metti in *R*1 (*R*2) il valore di *CONT* (*Y*
8		SUB R1 R2	(se cont=y,	Sottrai *R*2 (ossia *Y*) da *R*1
9		JUMPZ FINE	vai a FINE)	Se *R*1 = 0 (quindi CONT=Y) vai a FIN
10		LOAD R1 SOMMA	somma =	Metti in *R*1 il valore di *SOMMA*
11		LOAD R2 X	somma + x	Metti in *R*2 il valore di *X*
12		SUM R1 R2		Metti in *R*1 la somma tra *R*1 ed *R*2
13		STORE R1 SOMMA		Metti il valore di *R*1 in *SOMMA*
14		LOAD R1 #1	cont =	Incremento contatore; metti 1 in *R*1
15		LOAD R2 CONT	cont + 1	Metti in *R*2 il valore di *CONT*
16		SUM R1 R2		Metti in *R*1 la somma tra *R*1 ed *R*2
17		STORE R1 CONT		Metti il valore di *R*1 in *CONT*
18		JUMP INIZ		Salta a *INIZ*
19	FINE	WRITE SOMMA	printf	Scrive il contenuto di *SOMMA*
20		STOP		Fine dell'esecuzione

Osservazioni sul codice assembler

- ad una istruzione C corrispondono in genere più istruzioni assembler (e quindi linguaggio macchina)

  Esempio: somma = somma + x

  $\Longrightarrow$ 1. carica il valore di *X* in un registro
  2. carica il valore di *SOMMA* in un altro registro
  3. effettua la somma tra i due registri
  4. memorizza il risultato nella locazione di memoria di *SOMMA*
- JUMP e JUMPZ interrompono la sequenzialità delle istruzioni
- In realtà il compilatore (ed il linker) genera linguaggio macchina
  - ogni istruzione è codificata come una sequenza di bit
  - ogni istruzione occupa una (o più celle di memoria)
  - istruzione costituita da 2 parti:
    codice operativo
    operandi

Un esempio di linguaggio macchina

Per semplicità consideriamo istruzioni con al più un operando (un indirizzo di memoria ind)

Istruzione assembler	Codice operativo
LOAD R1 ind	0000
LOAD R2 ind	0001
STORE R1 ind	0010
STORE R2 ind	0011
SUM R1 R2	0100
SUB R1 R2	0101
JUMP ind	0110
JUMPZ ind	0111
READ ind	1000
WRITE ind	1001
STOP	1011
LOAD R1 #c	1100

	Indirizzo	Codice operativo	Indirizzo operando	Istr. assembler
0	00000	1000	10101	READ X
1	00001	1000	10110	READ Y
2	00010	1100	00000	LOAD R1 #0
3	00011	0010	11000	STORE R1 CONT
4	00100	1100	00000	LOAD R1 #0
5	00101	0010	10111	STORE R1 SOMMA
6	00110	0000	11000	LOAD R1 CONT
7	00111	0001	10110	LOAD R2 Y
8	01000	0101	-----	SUB R1 R2
9	01001	0111	10011	JUMPZ FINE
10	01010	0000	10111	LOAD R1 SOMMA
11	01011	0001	10101	LOAD R2 X
12	01100	0100	-----	SUM R1 R2
13	01101	0010	10111	STORE R1 SOMMA
14	01110	1100	00001	LOAD R1 #1
15	01111	0001	11000	LOAD R2 CONT
16	10000	0100	-----	SUM R1 R2
17	10001	0010	11000	STORE R1 CONT
18	10010	0110	00110	JUMP INIZ
19	10011	1001	10111	WRITE SOMMA
20	10100	1011	-----	STOP
21	10101			X
22	10110			Y
23	10111			SOMMA
24	11000			CONT