

Architettura del sistema di calcolo

hardware

Cristiana Bolchini

visione
astratta &
semplificata

sistema di calcolo

applicazioni

sistema operativo

unità
centrale di
elaborazione

memoria
di lavoro

mezzo di
comunicazione

periferiche

sistema fisico - hardware

memoria
di massa

ingresso

uscita

ingresso
/uscita

obiettivi

visione astratta e semplificata

capire come è organizzata un'architettura di un sistema di calcolo, con riferimento al modello di Von Neumann

- come funziona
- come è programmato
- come avviene l'esecuzione di un programma

architettura di Von Neumann

modello - cenni storici

idea chiave: memorizzare le istruzioni dei programmi da eseguire insieme ai dati

stored program computer

- 1943: ENIAC Eckert e Mauchl - first general electronic computer
 - programma hard-wired: valvole ed interruttori determinano il comportamento
- 1944: inizio dell'EDVAC
 - il programma è memorizzato
- 1945: John von Neumann e altri

architettura di Von Neumann

modello - cenni storici

- report sul concetto di programma memorizzato:
First Draft of a Report on EDVAC
- la struttura base del report è noto come il modello della macchina di Von Neumann
 - una memoria, che contiene istruzioni e dati
 - una unità di elaborazione, che effettua operazioni aritmetiche e logiche
 - una unità di controllo, che interpreta le istruzioni

al Politecnico di Milano

il primo calcolatore

- il Politecnico di Milano presenta la richiesta per l'assegnazione di un calcolatore digitale
- il prof. Luigi Dadda (1923-2012) va al Computer Research Corporation (CRC) e collabora alla realizzazione del calcolatore elettronico digitale acquistato (Crc 102A)
- nel 1954 porta in Italia, al Politecnico, il primo elaboratore funzionante in Italia e nell'Europa continentale

Crc 102A

**per risolvere equazioni
algebriche lineari**
facilitando i calcoli
necessari per la costruzione
di grandi opere soprattutto
nel campo edile e delle
costruzioni in genere



architettura di Von Neumann

modello

è un modello:

- descrizione funzionale/strutturale che racchiude le caratteristiche essenziali di un generico esecutore
- trascura aspetti architetturali che hanno un impatto sulle prestazioni (ad esempio il pipelining)

elementi fondamentali

serve un luogo per memorizzare le istruzioni e i dati

► MEMORIA DI LAVORO

poiché si leggono le istruzioni e si leggono e scrivono dati è importante che le operazioni elementari di lettura e scrittura richiedano poco tempo

elementi fondamentali

serve un elemento che elabori dati

- UNITÀ DI ELABORAZIONE

serve un elemento che controlli il flusso delle istruzioni

- UNITÀ DI CONTROLLO

le unità di elaborazione e di controllo sono contenute in un unico componente

- UNITÀ CENTRALE DI ELABORAZIONE
(CPU - CENTRAL PROCESSING UNIT)

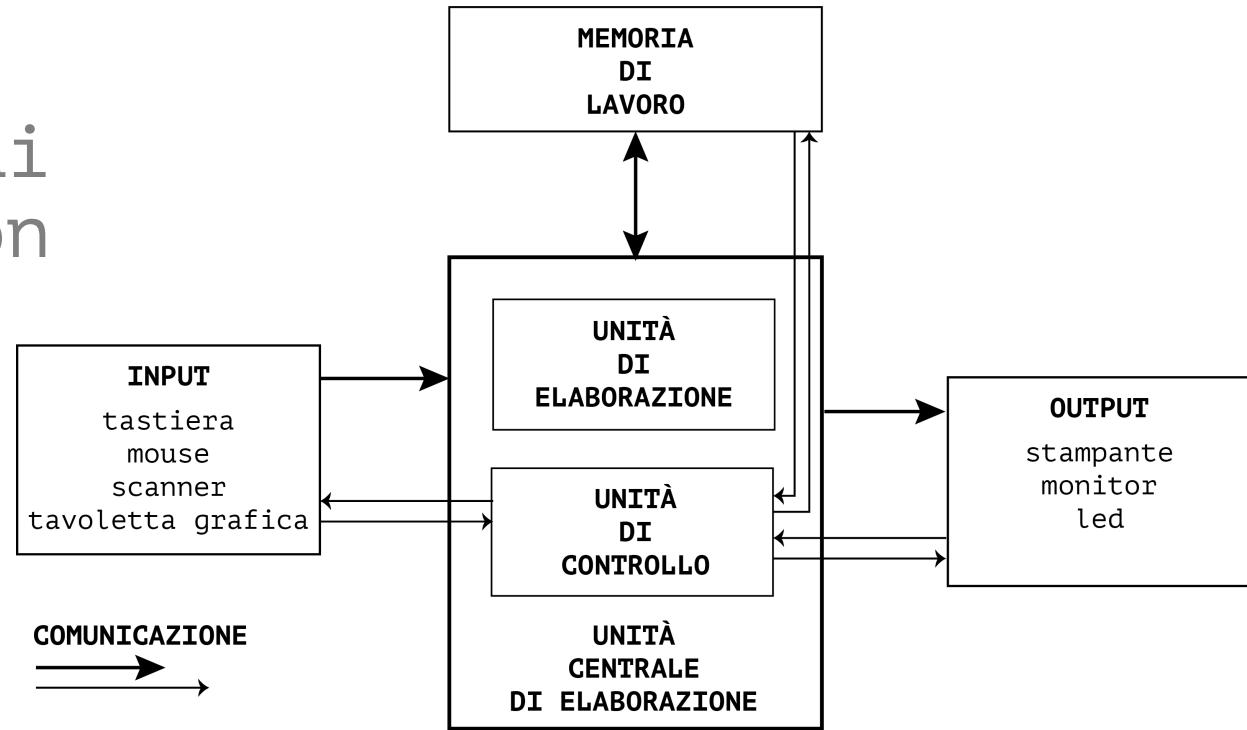
elementi fondamentali & utili

tutti gli elementi sono in comunicazione tra loro

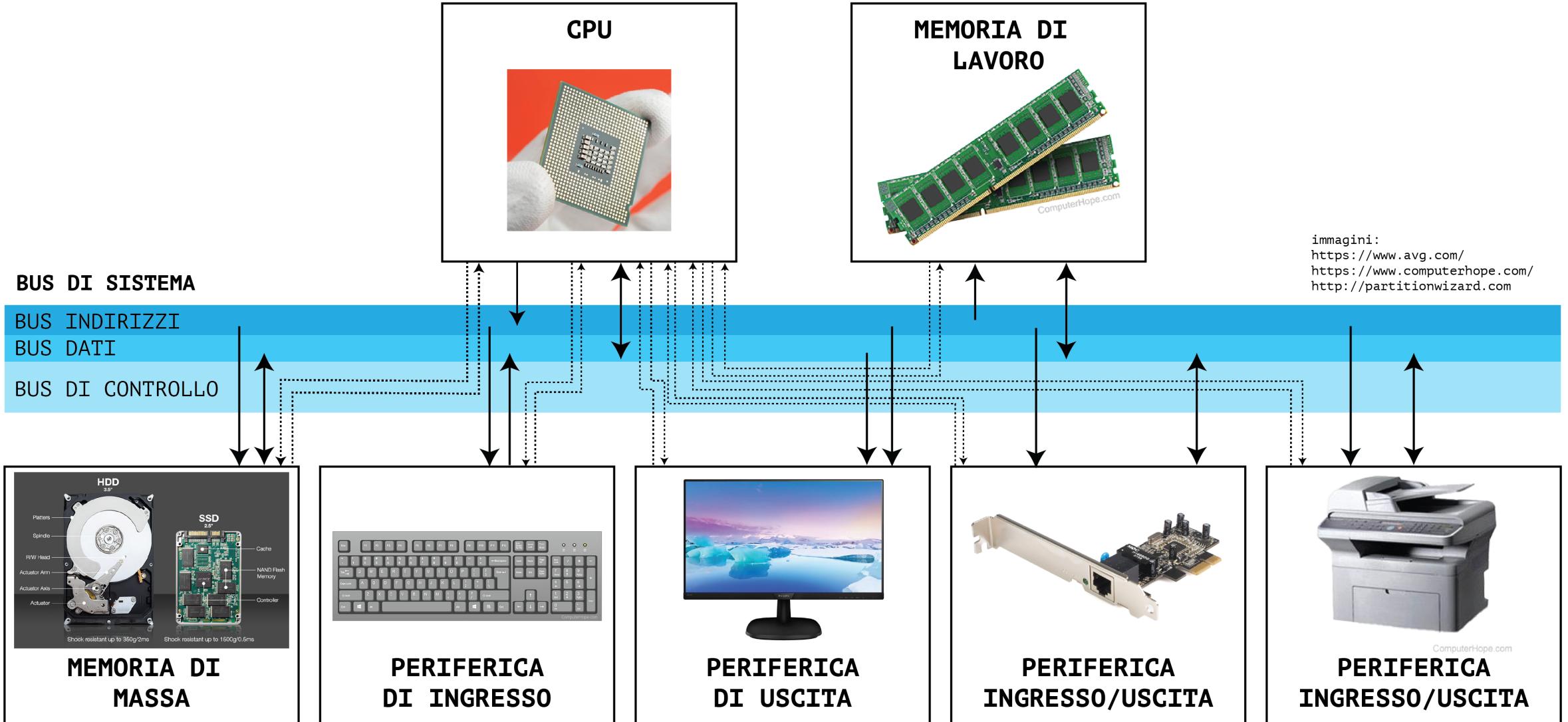
► MEZZO DI COMUNICAZIONE

è utile avere un insieme di elementi per interagire con il mondo esterno

► DISPOSITIVI/PERIFERICHE DI INGRESSO e USCITA (I/O - input/output)



panoramica



memoria di lavoro

MAIN MEMORY / RAM

- funzionalità: memorizza sia le istruzioni sia i dati del programma/programmi in esecuzione
- caratteristiche:
 - non persistente (volatile)
 - capacità "contenuta" (ordine di GB)
 - bassi tempi di accesso (ordine di 10-20 nanosecondi per i componenti più veloci)
 - costosa

memoria di lavoro

- contiene le informazioni (dati e istruzioni) necessarie per l'elaborazione in corso
- organizzata come una tabella, cui si accede "una riga alla volta"
 - riga: **cella** della memoria / **parola (word)**
 - numero di bit (**h**) della cella: 8, 16, 32 o 64 bit
 - la posizione della riga è individuata dal suo **indirizzo**
 - il tempo di accesso alle celle è costante e indipendente dall'indirizzo (random access memory)

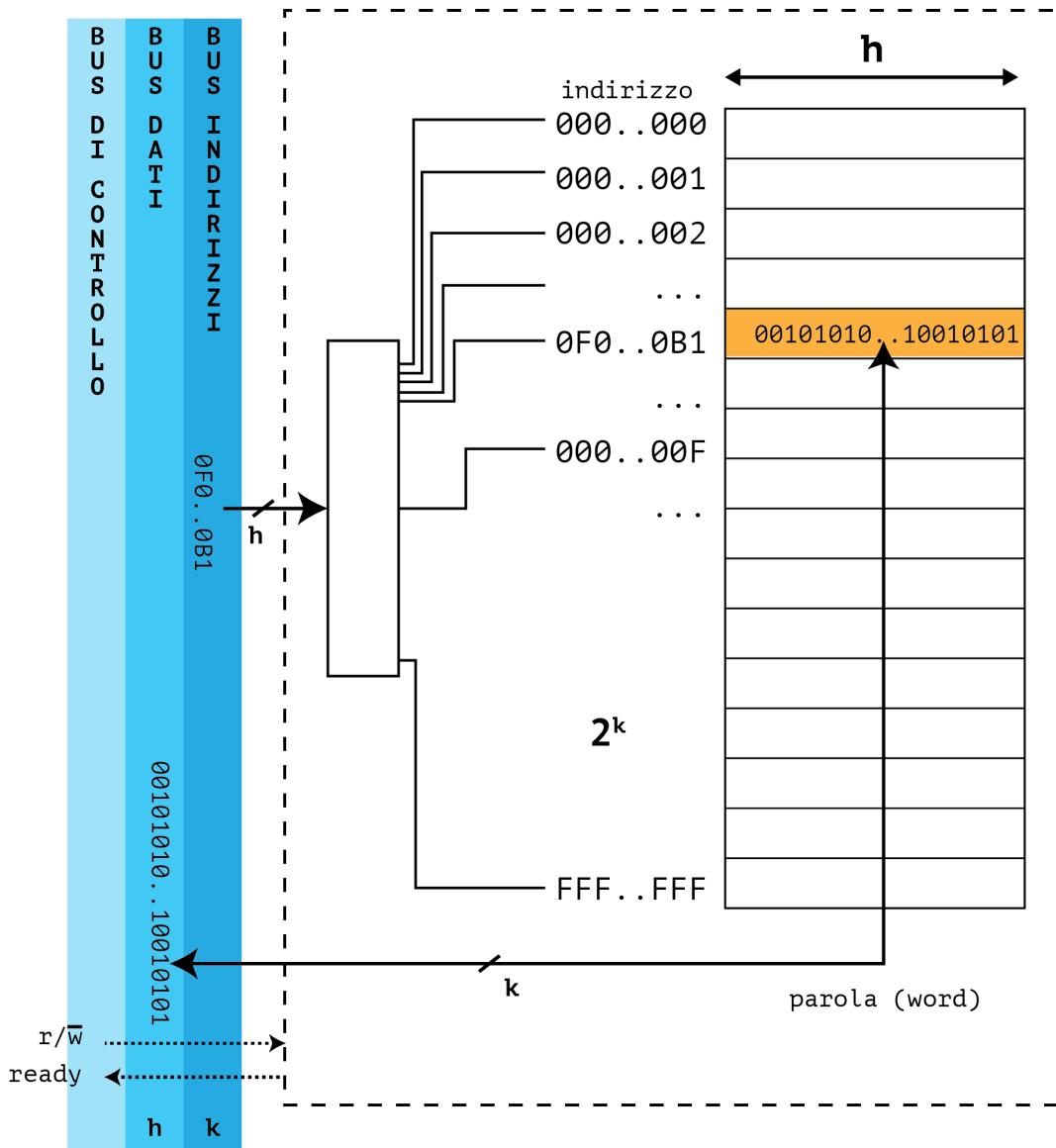
memoria di lavoro

- è possibile leggere dalla memoria (il contenuto di una cella viene reso disponibile sul bus) o scrivere (si sovrascrive il contenuto con quanto presente sul bus)
 - load: si legge dalla memoria un dato
 - store: si scrive in memoria un dato
- ogni operazione ha tre elementi:
 - specifica se leggere/scrivere
 - indirizzo di dove si vuole leggere/scrivere
 - dato scambiato (letto o da scrivere)

memoria di lavoro

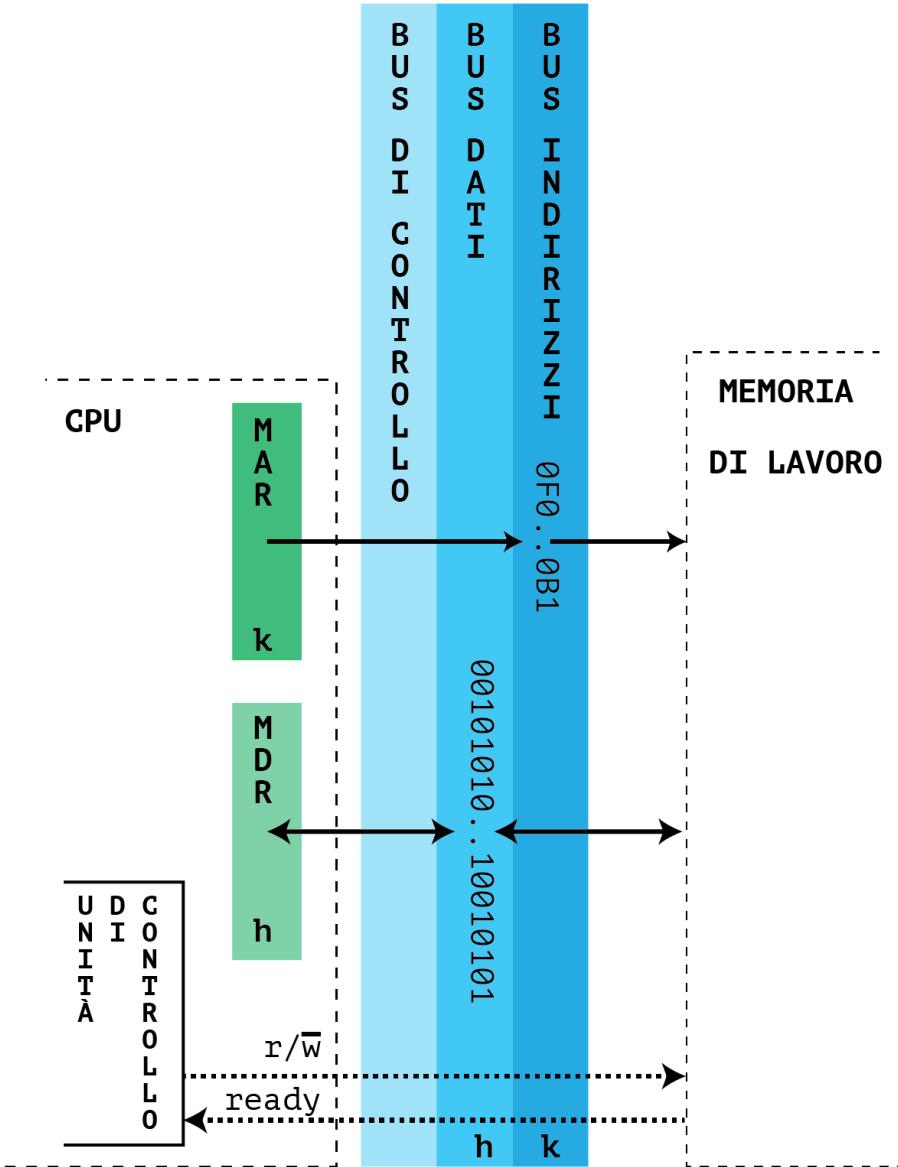
dimensioni/capacità:

- k : bit di indirizzo
- 2^k : spazio di indirizzamento
- h : dimensione della parola
- oggi almeno 32 bit di indirizzo



memoria di lavoro

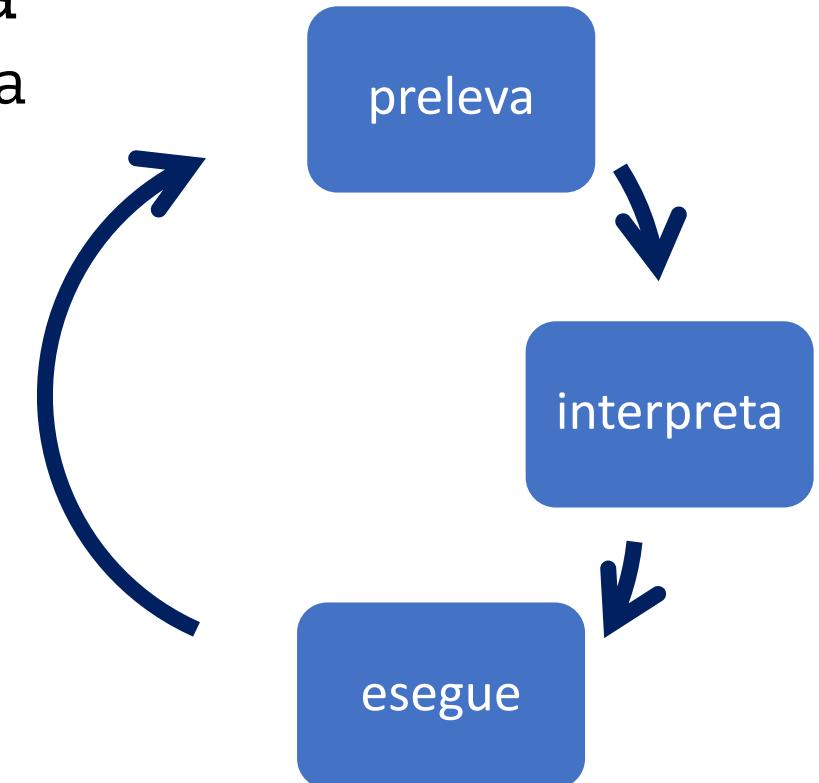
- nell'interazione memoria di lavoro-CPU si utilizzano due registri
 - registro dato - memory data register (MDR) o DR copia del dato da scrivere/letto
 - registro indirizzo - memory address register (MAR) o AR indirizzo di dove scrivere/leggere



unità centrale di elaborazione

CPU

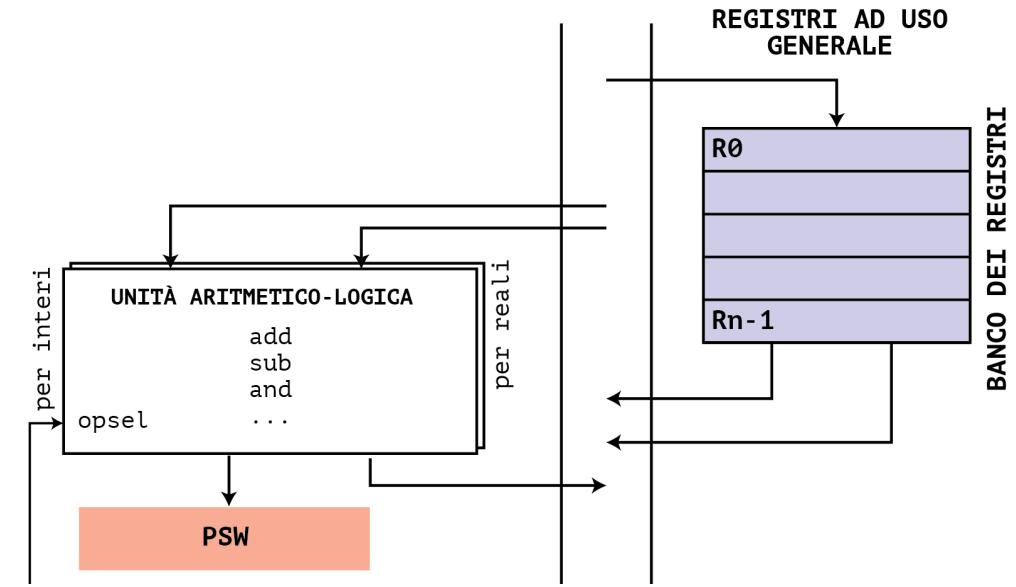
- obiettivo: eseguire un programma
 - preleva le istruzioni dalla memoria
 - le interpreta
 - realizza le operazioni elementari che corrispondono alle istruzioni
 - coordina il resto dei componenti
- la sequenza **fetch-decode-execute** viene ripetuta continuamente fino a quando si preleva l'istruzione **halt** che come esecuzione prevede il termine



unità centrale di elaborazione

unità di elaborazione

- effettua operazioni aritmetiche e logiche **Unità Aritmetico Logica** (per interi e per reali)
- utilizza registri di lavoro (ad uso generale), che costituiscono il **banco dei registri**
 - memoria volatile
 - quantità: nell'ordine della decina
 - accesso estremamente rapido
- informazioni sul risultato nel registro PSW



unità centrale di elaborazione

unità di controllo e decodifica:

- coordina l'insieme di azioni necessarie per l'esecuzione di una istruzione
 - preleva l'istruzione dalla memoria di lavoro
 - gestisce la comunicazione con le unità coinvolte
 - seleziona l'operazione specifica della ALU nel caso sia coinvolta
- utilizza **registri ad uso speciale** che contengono informazioni relative all'esecuzione dell'istruzione
- utilizza **registri di interfacciamento** con il bus di sistema

unità centrale di elaborazione

registri ad uso speciale

- program counter (PC): indirizzo alla prossima istruzione da eseguire
- instruction register (IR): copia dell'istruzione in esecuzione
- stack pointer: indirizzo della prima cella vuota dello stack
- processor status word (PSW): insieme di bit/flag che caratterizzano l'esito dell'operazione svolta dalla ALU
 - zero: il risultato è zero
 - sign: il risultato è negativo
 - ovf: si è verificato ovf
 - carry: c'è riporto

unità centrale di elaborazione

registri di interfacciamento con il bus

- memory address register (MAR): indirizzo posto sul bus indirizzi
- memory data register (MAR): contiene il dato trasferito dal/sul bus dati

unità centrale di elaborazione

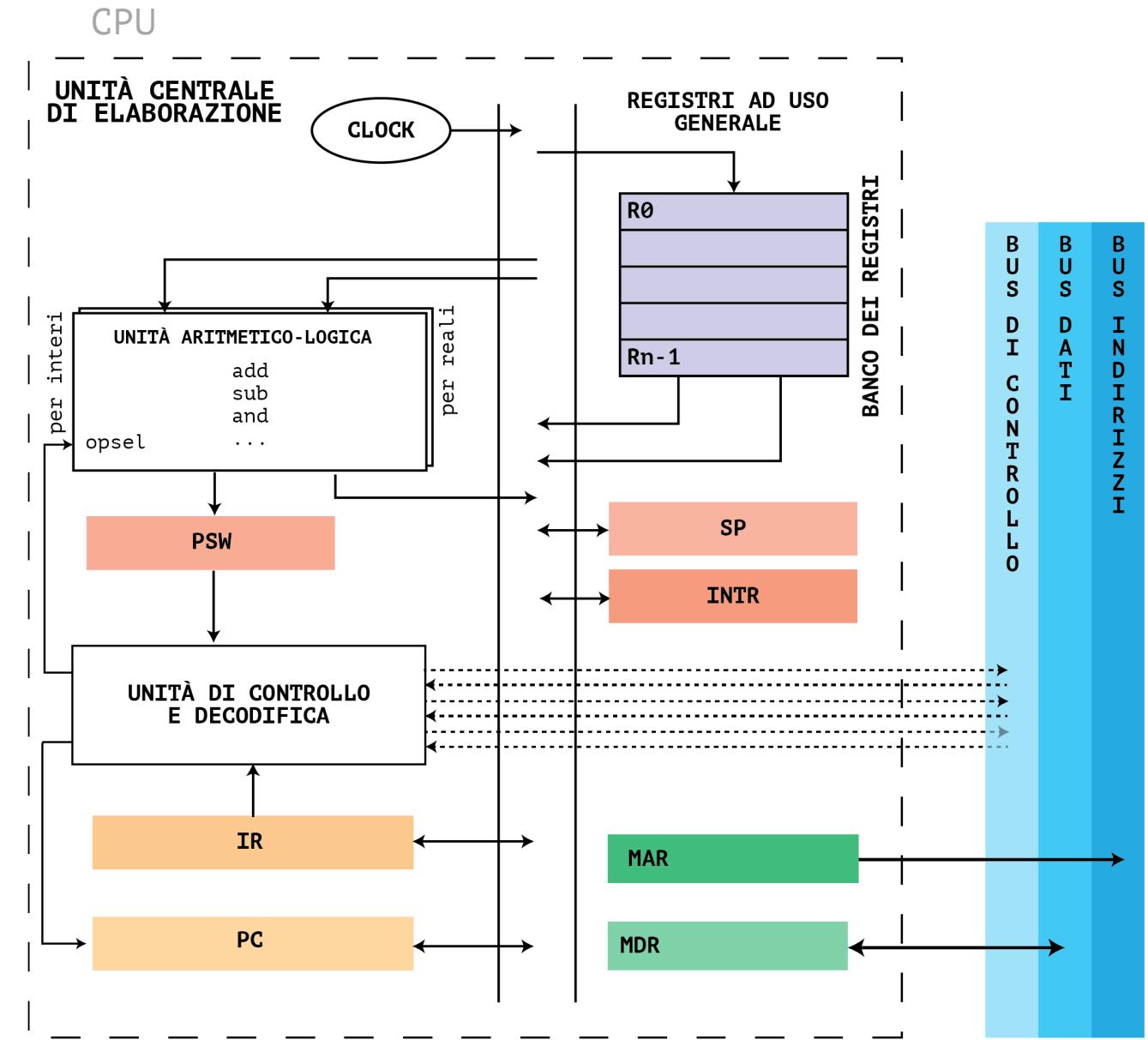
sincronismo & coordinamento

- tutte le operazioni sono scandite dal segnale di clock che determina la frequenza di funzionamento
- segnali di controlli da/verso l'unità di controllo e decodifica per coordinare le unità interne alla CPU e quelle esterne (sono parte del bus di controllo)

unità centrale di elaborazione

elementi:

- unità di elaborazione
- unità di controllo (e decodifica)
- **registri ad uso speciale**
- registri ad uso generale
- **registri di interfacciamento**
- segnale di clock
- bus interno



esecuzione di una istruzione

fetch

- prelievo dalla memoria di lavoro

MAR \leftarrow PC

R/ \bar{W} \leftarrow 1 (lettura dalla memoria)

READY

MDR \leftarrow MEM[MAR]

IR \leftarrow MDR

PC \leftarrow PC + 1

esecuzione di una istruzione

decode

- si analizza il codice operativo dell'istruzione presente nell'IR e si dà inizio alla sequenza di passi elementari per realizzare l'istruzione vera e propria

esecuzione di una istruzione

execute

- sequenza di passi
- ad esempio: LOAD R3, ind1
carica nel registro R3 il contenuto all'indirizzo ind1

MAR \leftarrow OP2(IR)

R/W \leftarrow 1

READY

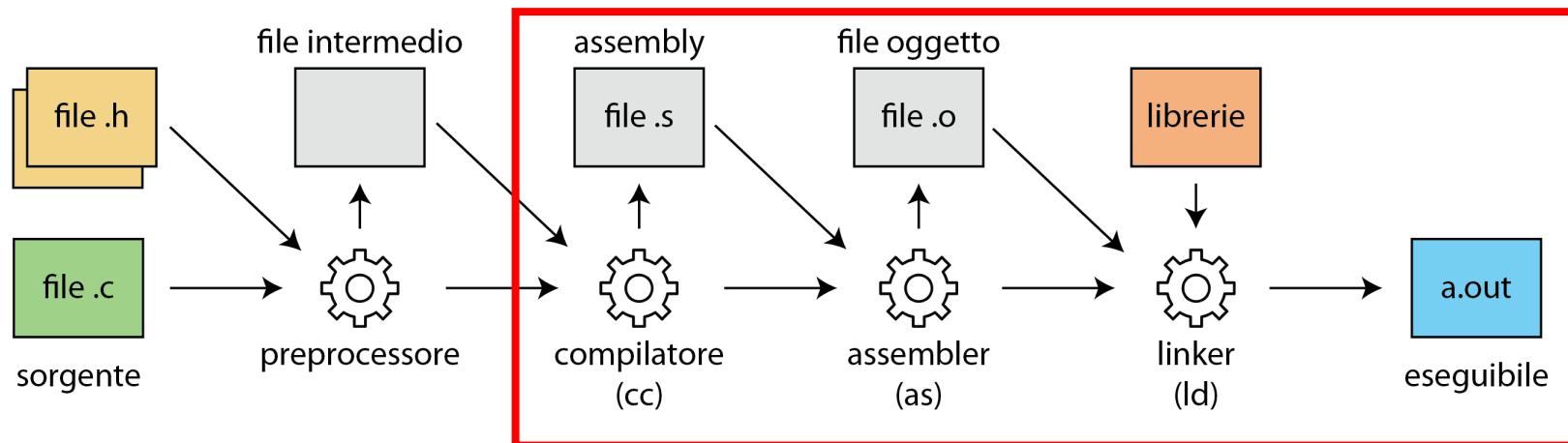
MDR \leftarrow MEM[MAR]

REG[OP1(IR)] \leftarrow MDR

decodifica / insieme di istruzioni

decode

- ogni CPU è progettata per interpretare ed eseguire un insieme di istruzioni scelto
- qualsiasi programma per essere eseguito deve essere tradotto in una sequenza di istruzioni appartenenti all'insieme delle istruzioni per quella CPU



istruzioni

ogni istruzione è composta da due parti

- codice operativo (OPCODE)
- operando/i

istruzioni

- effettive: manipolano dati e producono nuovi dati
(somma, sottrazione, and, ...)
- di controllo: determinano quale sia la prossima istruzione da eseguirsi

insieme delle istruzioni

esempio

| Codice | Effetto |
|----------------|---|
| add RX, RY, RZ | $RX \leftarrow RY + RZ$ |
| sub RX, RY, RZ | $RX \leftarrow RY - RZ$ |
| and RX, RY, RZ | $RX \leftarrow RY \& RZ$ |
| or RX, RY, RZ | $RX \leftarrow RY RZ$ |
| xor RX, RY, RZ | $RX \leftarrow RY ^ RZ$ |
| shl RX, RY, RZ | $RX \leftarrow RY \ll RZ$ |
| shr RX, RY, RZ | $RX \leftarrow RY \gg RZ$ |
| ld RX, ADDR | $RX \leftarrow \text{MEM}[ADDR]$ |
| st RX, ADDR | $\text{MEM}[ADDR] \leftarrow RX$ |
| ldc RX, CONST | $RX \leftarrow \text{CONST}$ |
| jz etichetta | $PC \leftarrow \text{etichetta}$ se zero. |
| jnz etichetta | $PC \leftarrow \text{etichetta}$ se non zero. |
| jg etichetta | $PC \leftarrow \text{etichetta}$ se positivo. |
| jng etichetta | $PC \leftarrow \text{etichetta}$ se non positivo. |
| jc etichetta | $PC \leftarrow \text{etichetta}$ se carry. |
| jnc etichetta | $PC \leftarrow \text{etichetta}$ se non carry. |
| jmp etichetta | $PC \leftarrow \text{etichetta}$ |
| push RD | $\text{MEM}[SP] \leftarrow RD; SP \leftarrow SP - 1$ |
| pop RD | $SP \leftarrow SP + 1; RD \leftarrow \text{MEM}[SP]$ |
| call proced | $\text{MEM}[SP - k] \leftarrow PC + h; SP \leftarrow SP - k; PC \leftarrow \text{proced}$ |
| ret | $SP \leftarrow SP + k; PC \leftarrow \text{MEM}[SP]$ |
| halt | termine |

generazione del codice assembly

dal C all'assembly

```
#include <stdio.h>

int main(int argc, char * argv[])
{
    int val, num;
    int ris;

    scanf("%d", &val);
    scanf("%d", &num);

    ris = val * num;
    ris++;
    printf("%d\n", ris);
    return 0;
}
```

gcc -S test.c

Intel Core i5

```
.section      __TEXT,__text,regular,pure_instructions
.build_version macos, 11, 3      sdk_version 11, 3
.globl _main
## -- Begin function
main
.p2align     4, 0x90
_main:
.cfi_startproc
## %bb.0:
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset %rbp, -16
movq %rsp, %rbp
.cfi_def_cfa_register %rbp
subq $48, %rsp
movl $0, -4(%rbp)
movl %edi, -8(%rbp)
movq %rsi, -16(%rbp)
leaq L_.str(%rip), %rdi
leaq -20(%rbp), %rsi
movb $0, %al
callq _scanf
leaq L_.str(%rip), %rdi
leaq -24(%rbp), %rsi
movl %eax, -32(%rbp)
movb $0, %al
callq _scanf
movl -20(%rbp), %ecx
imull -24(%rbp), %ecx
## 4-byte Spill
```

generazione del codice assembly



dal C all'assembly

```
#include <stdio.h>

int main(int argc, char * argv[])
{
    int val, num;
    int ris;

    scanf("%d", &val);
    scanf("%d", &num);

    ris = val * num;
    ris++;
    printf("%d\n", ris);
    return 0;
}
```

gcc -S test.c

```
.section __TEXT,__text,regular,pure_instructions
.build_version macos, 13, 0      sdk_version 13, 0
.globl _main
; -- Begin function main
.p2align 2
_main:                                ; @main
    .cfi_startproc
; %bb.0:
    sub    sp, sp, #64
    stp    x29, x30, [sp, #48]           ; 16-byte Folded
    Spill
    add    x29, sp, #48
    .cfi_def_cfa w29, 16
    .cfi_offset w30, -8
    .cfi_offset w29, -16
    mov    w8, #0
    str    w8, [sp, #16]                ; 4-byte Folded
    Spill
    stur   wzr, [x29, #-4]
    stur   w0, [x29, #-8]
    stur   x1, [x29, #-16]
    mov    x9, sp
    sub    x8, x29, #20
    str    x8, [x9]
    adrp   x0, l_.str@PAGE
    add    x0, x0, l_.str@PAGEOFF
    str    x0, [sp, #8]                 ; 8-byte Folded
    Spill
    bl     _scanf
    ldr    x0, [sp, #8]                 ; 8-byte Folded
    Reload
    mov    x9, sp
    add    x8, sp, #24
```

generazione del codice assembly



dal C all'assembly

```
#include <stdio.h>

int main(int argc, char * argv[])
{
    int val, num;
    int ris;

    scanf("%d", &val);
    scanf("%d", &num);

    ris = val * num;
    ris++;
    printf("%d\n", ris);
    return 0;
}
```

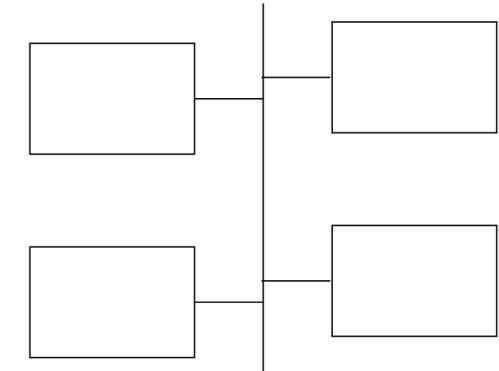
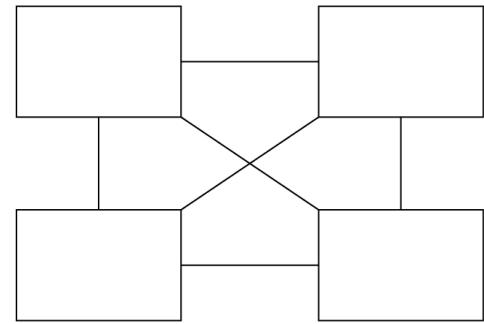
gcc -S test.c

Arm

```
.arch armv8-a
.file "testas.c"
.text
.section .rodata
.align 3
.LC0:
.string "%d"
.align 3
.LC1:
.string "%d\n"
.text
.align 2
.global main
.type main, %function
main:
stp x29, x30, [sp, -64]!
add x29, sp, 0
str w0, [x29, 28]
str x1, [x29, 16]
adrp x0, :got:_stack_chk_guard
ldr x0, [x0, #:got_lo12:_stack_chk_guard]
ldr x1, [x0]
str x1, [x29, 56]
mov x1, 0
add x1, x29, 44
adrp x0, .LC0
add x0, x0, :lo12:.LC0
bl __isoc99_scanf
add x1, x29, 48
adrp x0, .LC0
add x0, x0, :lo12:.LC0
bl __isoc99_scanf
ldr w1, [x29, 44]
```

mezzo di comunicazione

- funzionalità: collega i dispositivi e veicola le informazioni tra questi
- due alternative:
 - ogni dispositivo è direttamente collegato agli altri
connessione completa
 - c'è un unico canale comune cui sono collegati tutti i dispositivi
bus di sistema



mezzo di comunicazione

| Conessione completa | Bus di sistema |
|--|--|
| elevato parallelismo (possibilità di trasferire molti dati per unità di tempo) | serialità nelle comunicazioni (il bus è accessibile per un trasferimento alla volta) |
| controllo complicato delle comunicazioni (ci potrebbero essere conflitti, ogni componente deve saper comunicare con gli altri) | controllo semplice delle comunicazioni (in ogni istante solo una coppia di elementi comunica) |
| rigidità rispetto alle possibilità di espansione (è necessario prevedere quali elementi comporranno il sistema) | flessibilità (è possibile teoricamente aggiungere un numero elevato di componenti) |
| gestione dei guasti indipendente per ogni coppia | poca robustezza nel caso di un guasto del mezzo condiviso |

mezzo di comunicazione

bus di sistema

organizzazione logica:

- bus indirizzi: transitano gli indirizzi dello spazio in memoria (o del dispositivo) cui si vuole far accesso
- bus dati: transita l'informazione (dato o istruzione) che viene scambiata
- bus di controllo: segnali che vengono utilizzati per coordinare lo scambio delle informazioni (modo, sincronizzazione, ...)

mezzo di comunicazione

bus di sistema

bus indirizzi

- monodirezionale: la CPU imposta l'indirizzo
- la dimensione ha un impatto sugli indirizzi utilizzabili per individuare le celle di memoria e gli altri dispositivi

bus dati

- bidirezionale: la CPU legge e scrive, gli altri dispositivi possono leggere o scrivere in base alla funzionalità
- la dimensione ha un impatto sulla quantità di informazione trasferibile in un unico ciclo di bus

mezzo di comunicazione

bus di sistema

bus di controllo

- linee monodirezionali, alcune dalla CPU verso un dispositivo, altre dal dispositivo verso la CPU
 - Read/Write: dalla CPU verso la memoria
 - Ready: dalla memoria verso la CPU
 - ...

mezzo di comunicazione

bus di sistema

- unico gestore del bus: CPU
 - determina da quale componente leggere informazioni o a quale componente inviare informazioni
 - è sempre coinvolta nello scambio**
- ad esempio: dati da I/O a memoria di lavoro
 - da I/O a CPU
 - da CPU a memoria di lavoro
- in ogni momento un solo dispositivo invia informazioni, tutti possono leggerle (ma solo uno è interessato)

memoria di massa

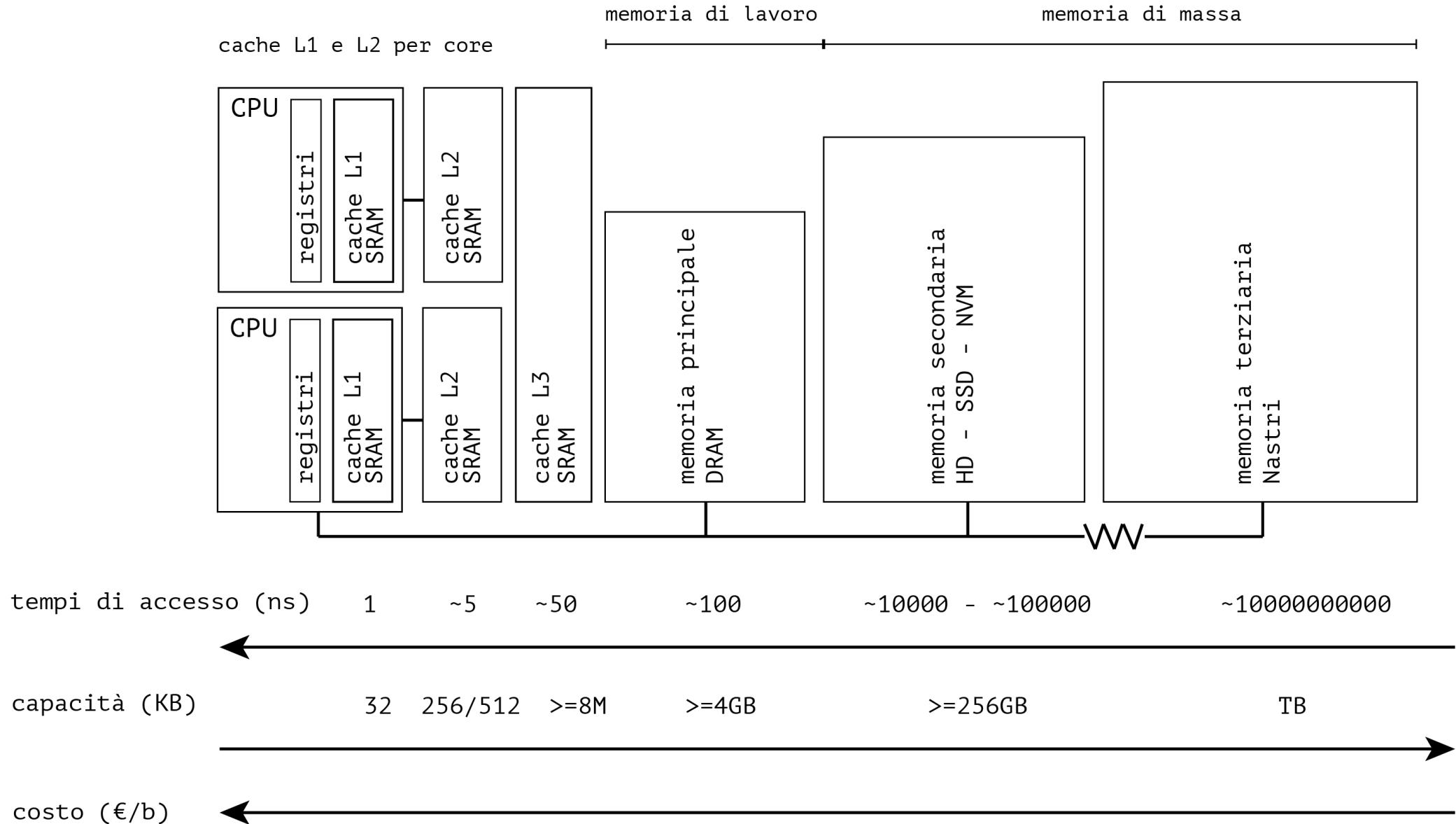
storage / HARD DISKS, DISCHI STATO SOLIDO, ...

- dispositivo di I/O costituito da memoria **non volatile**
 - le informazioni contenute rimangono memorizzate anche in assenza di alimentazione
- utilizzata
 - come deposito dati
 - per estendere la memoria di lavoro (memoria virtuale)

gerarchia di memoria

- idealmente si vorrebbe avere un sistema di calcolo con grandi quantità di memoria, caratterizzata da tempi di accesso piccoli, affidabile e poco costosa
- nella realtà si realizza un compromesso con diverse tipologie di memoria caratterizzate da:
 - capacità
 - persistenza/volatilità
 - modalità di accesso
 - tempi di accesso
 - costi

gerarchia di memoria



dispositivi di ingresso/uscita

periferiche - I/O

- l'ambiente esterno interagisce con tempi e modi diversi dall'interazione CPU-memoria di lavoro
- dipende dal tipo di dispositivo con cui si interagisce
- la CPU sta eseguendo un programma che ha bisogno di acquisire/inviare un dato ad una periferica

dispositivi di ingresso/uscita

periferiche - I/O

- in genere sono composti da
 - una parte meccanica
 - una parte elettronica: il controller
- il controller è connesso al bus
 - comanda il dispositivo e comunica con il processore
 - sono mappati sullo spazio di indirizzamento visibile dal processore
 - ha dei registri di controllo che vengono utilizzati per le comunicazioni con il processore
 - Registro Comandi Periferica (PCR) - collegato al bus di controllo
 - Registro Dati della Periferica (PDR) - collegato al bus dati
 - Registro Stato Periferica (PSR) - pronto, occupato, errore, ...

dispositivi di ingresso/uscita

periferiche - I/O

- modalità di scambio informazioni
 - a controllo di programma (polling)
 - ad interruzione
 - con accesso diretto alla memoria (Direct Memory Access - DMA)

a controllo di programma

- la CPU "da programma" interrompe l'elaborazione in corso, salva lo stato, e gestisce scambi dati previsti
 - se la periferica è pronta: trasferisce il dato
 - se la periferica non è pronta: la CPU procede facendo altro e ricontrollerà in futuro se il dato è pronto
- il processo che attende il dato dalla periferica rimane bloccato

ad interruzione

- la CPU attiva la periferica con cui deve scambiare il dato e prosegue poi nell'attività
- quando la periferica è pronta, attiva un segnale di interruzione
- la CPU rileva la richiesta di interruzione, conclude l'esecuzione corrente (sospendendola)
 - salva il contesto
 - esegue la sequenza di operazioni per trasferire il dato
 - ripristina il contesto
- la CPU prosegue nell'eseguire quanto sospeso

con accesso diretto alla memoria **

Direct Memory Access (DMA)

caso di trasferimento di più dati tra memoria di lavoro e periferica (memoria di massa)

- si utilizza un dispositivo programmabile dalla CPU, controllore DMA, specificando
 - sorgente dati
 - destinazione dati
 - quantità di dati da trasferire
- la CPU continua nell'elaborazione (di altro)
- il controllore DMA "ruba" cicli di bus per trasferire i dati quando il bus non è utilizzato dalla CPU
- segnala alla CPU il termine del trasferimento

riferimenti usati

immagini e spunti presi da

- Carl Hamacher "Computer Organization"

corsi presso:

- Stony Brook University
- Carnegie Mellon University