



Corso di Architettura degli Elaboratori a.a. 2021/2022

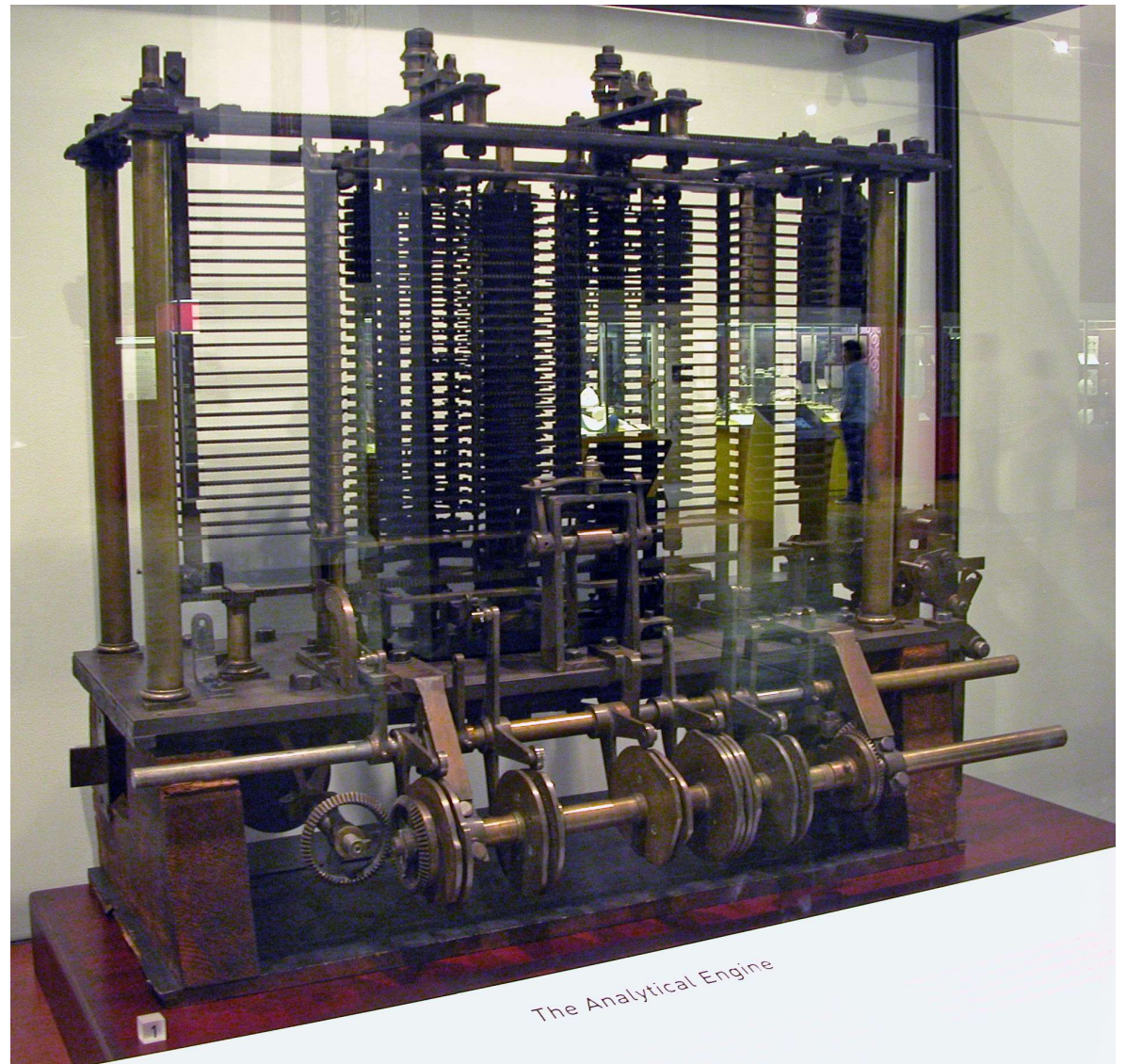
L'Instruction Set Architecture (ISA) RISC-V

Storia

Pillole di storia

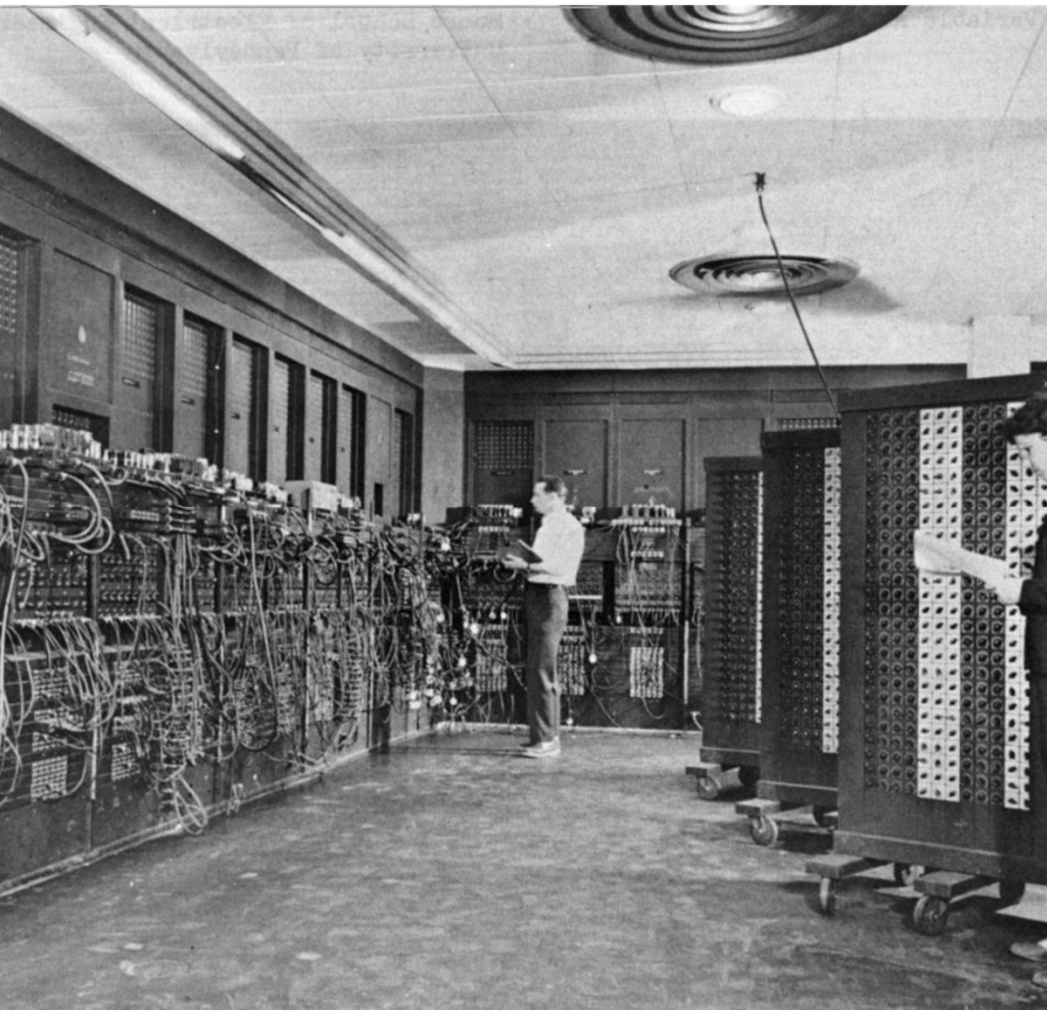
- **Pascal** (1623 – 1662): Calcolatrice meccanica per eseguire somme e sottrazioni (<https://it.wikipedia.org/wiki/Pascalina>)
- **Leibniz** (1646 – 1716): Macchina meccanica in grado di eseguire anche moltiplicazioni e divisioni (https://it.wikipedia.org/wiki/Stepped_Reckoner)
- **Babbage** (1792 – 1871): Difference engine: calcolare tabelle di numeri, eseguiva un solo algoritmo. Analytical engine: con un magazzino per memorizzare dati (https://it.wikipedia.org/wiki/Macchina_analitica)
- **Zuse** (1910 – 1995): negli anni '30 costruì macchine calcolatrici con relé elettromagnetici, distrutte da un bombardamento nel '44. Il calcolatore "Z1", completato da Zuse nel 1938, è considerato il primo computer moderno. A Konrad Zuse si deve anche l'invenzione del primo linguaggio di programmazione della storia, ideato per fornire le istruzioni allo Z1.

Pillole di storia

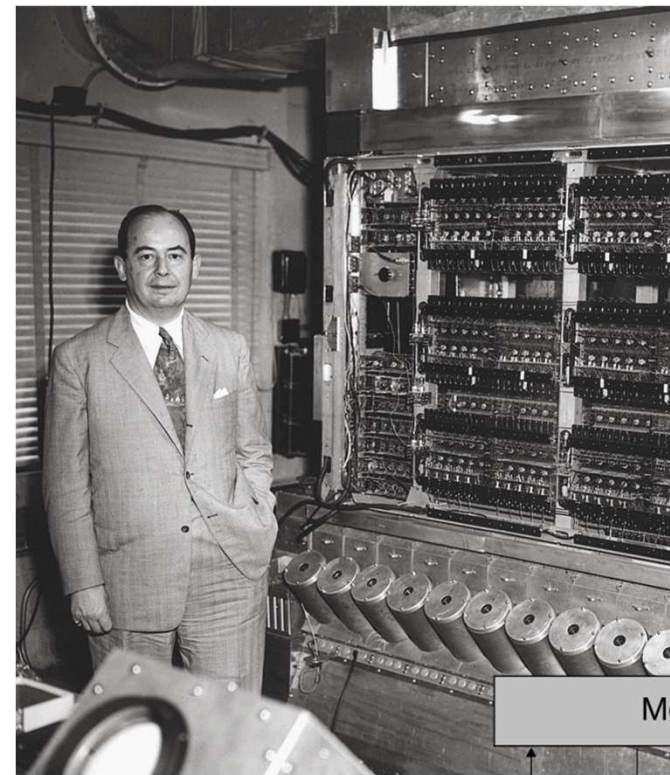


1^{ma} generazione – Valvole (1945 – 1955)

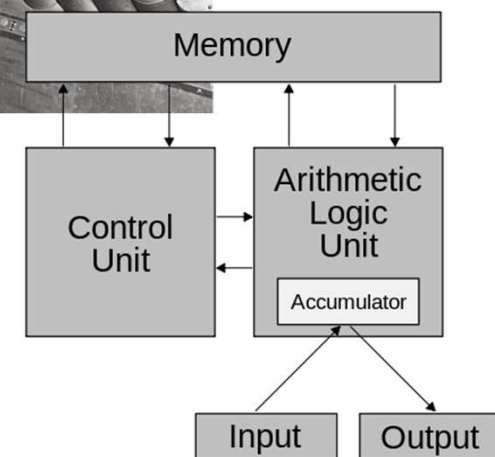
- **Colossus** per decifrare ENIGMA (Alan Turing)
- **ENIAC** (Mauchley – Eckert) 18.000 valvole termoioniche, 1.500 relé, 30 tonnellate, 140 Kw consumo di energia,
- **IAS** (von Neumann): aritmetica binaria, dati e istruzioni in memoria



ENIAC

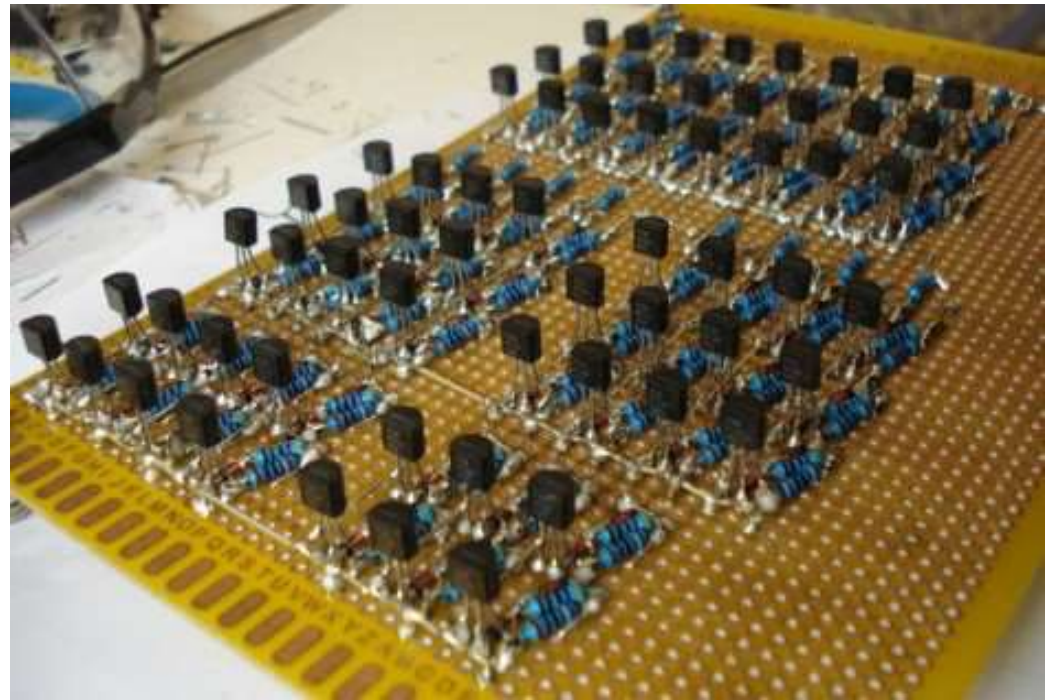


IAS con Von Neumann
e la sua architettura



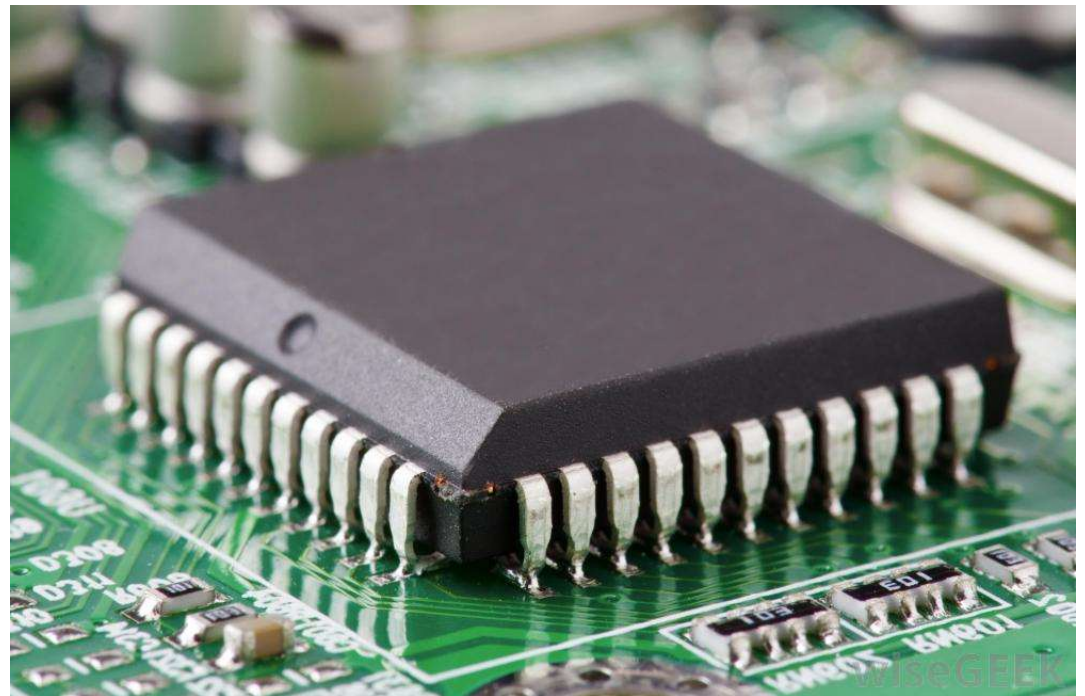
Le generazioni successive

- Seconda generazione – Transistor (1955 – 1965)



Le generazioni successive

- Terza generazione –
Circuiti Integrati (1965 –
1980)



Le generazioni successive

- Quarta generazione –
Integrazione su vasta
scala (1980 - ...)
 - VLSI (Very Large Scale
Integration)

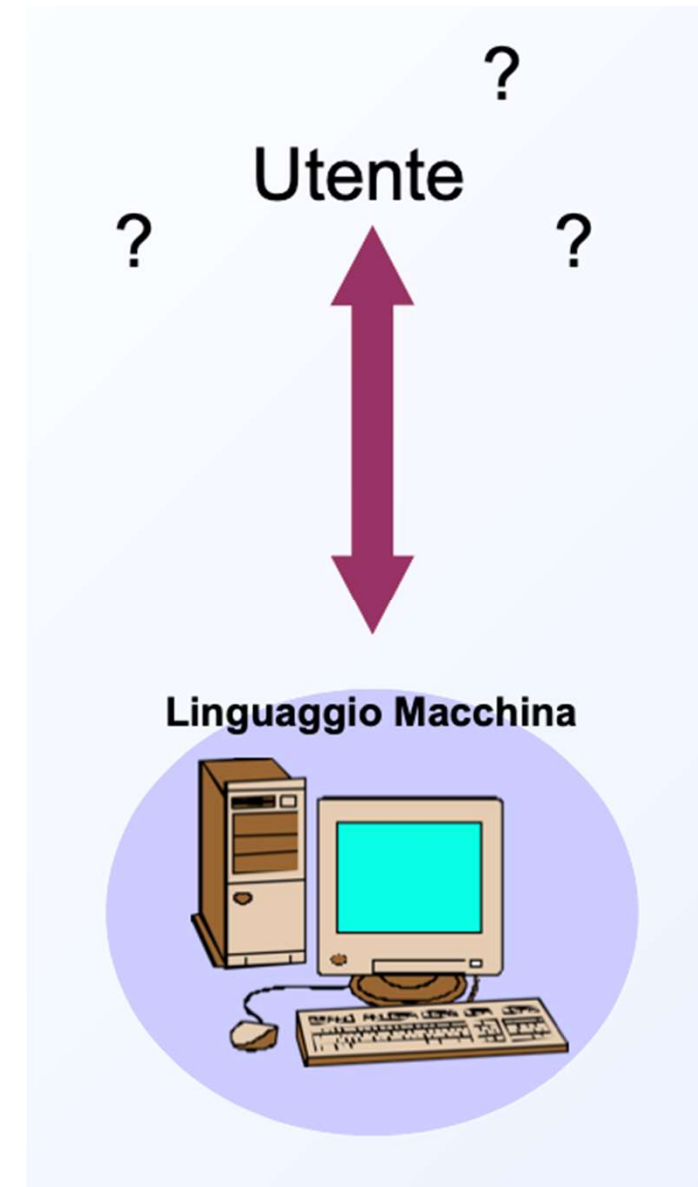


Come faccio a programmare...

- ... un insieme di transistor?
- Programma è sequenza di istruzioni che descrive come portare a termine un certo compito.
- Un processore capisce Java/C/Python/...?
- Se un processore non può capire Java o C, come fa a funzionare?
- Perché usiamo Java o C e non il linguaggio del processore?

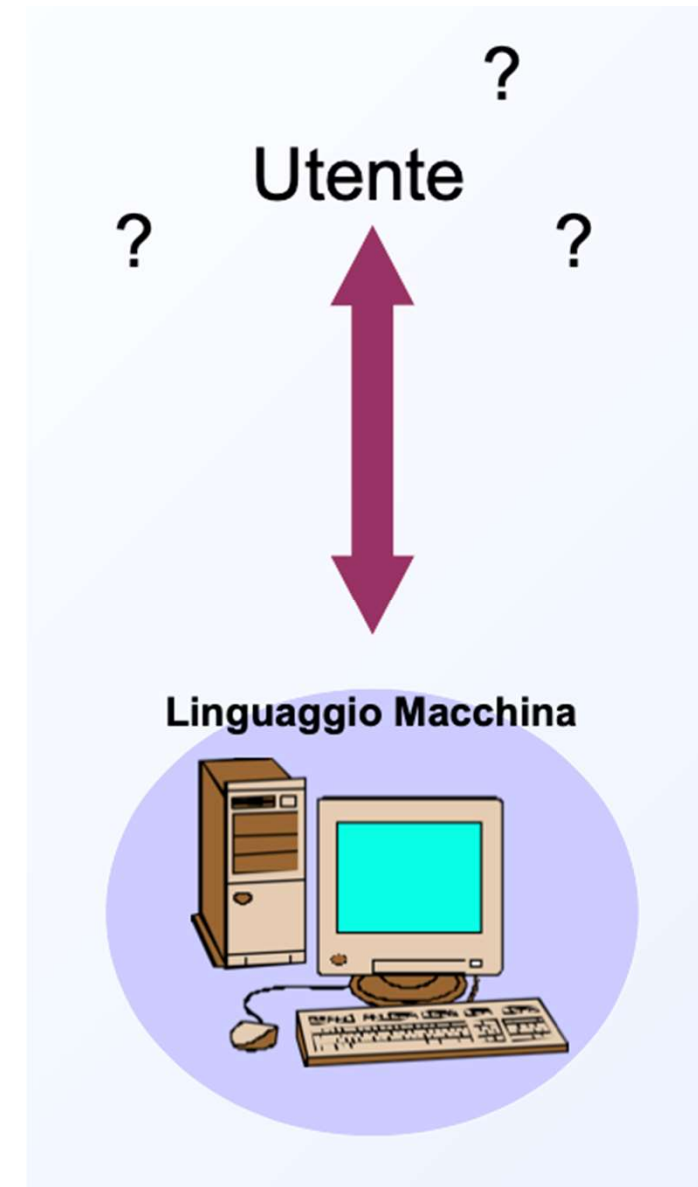
Livelli come macchine virtuali

- Un computer è una macchina programmabile, tuttavia esso non è direttamente utilizzabile da parte degli utenti poiché richiederebbe la conoscenza sull'organizzazione fisica della specifica macchina e del suo linguaggio macchina.
- Ogni macchina avrebbe le sue differenti caratteristiche
- Il linguaggio macchina è estremamente complicato e non di facile gestione



Livelli come macchine virtuali

- Il linguaggio macchina è estremamente complicato (per un umano) e non di facile gestione
- Cosa vuol dire?
- Un programma risiede in memoria ed è composto da una sequenza di bit
- Come comprenderlo?
 - Creare una rappresentazione simbolica delle istruzioni macchina

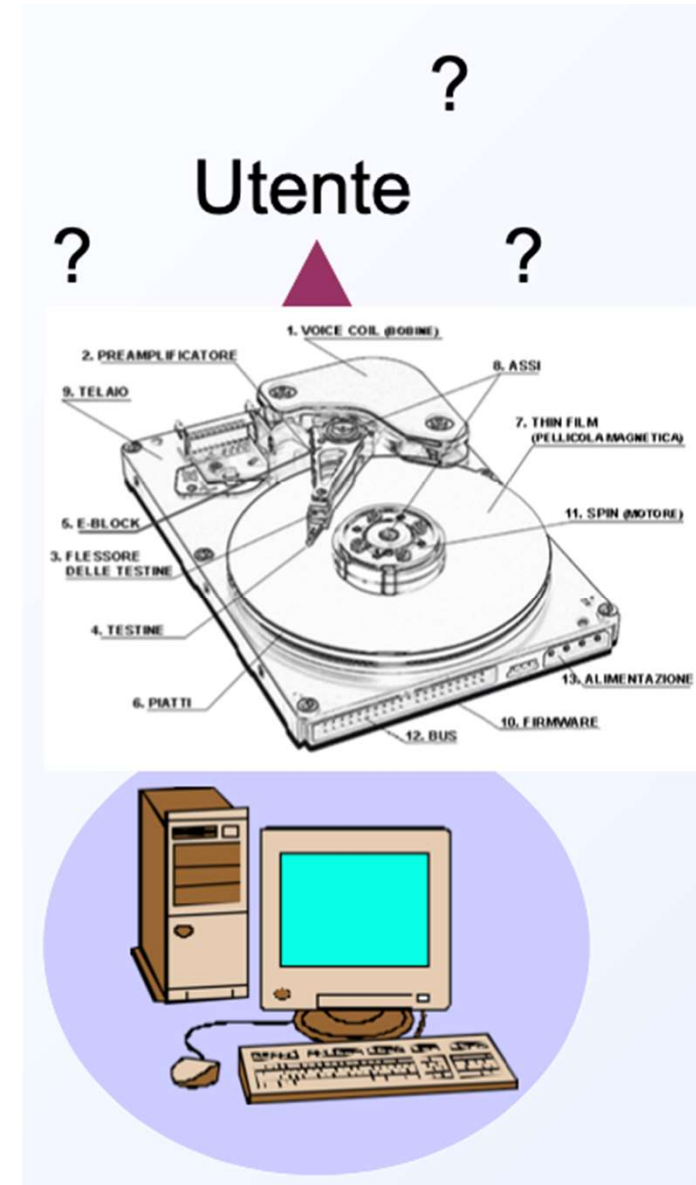


Linguaggio macchina/assembler

Ling. macchina	Assembler	Significato
0001 0101 0110 1100	LOAD R5 108	M[108] -> R5
0001 0110 0110 1101	LOAD R6 109	M[109] -> R6
0101 0000 0101 0110	ADD R0 R5 R6	R5 + R6 -> R0
0011 0000 0101 1110	STORE R0 110	R0 -> M[110]
1100 0000 0000 0000	Halt	Halt

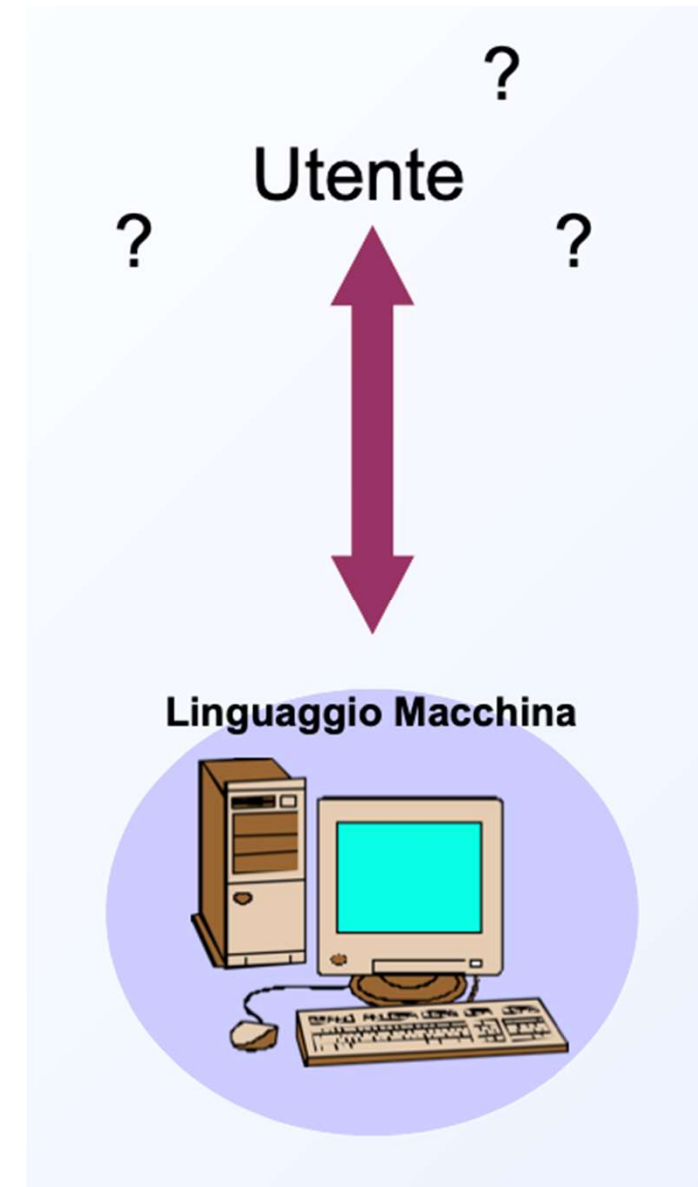
Livelli come macchine virtuali

- Un programma non fa solo calcoli:
interagisce anche con i dispositivi di input/output
- Problemi
 - I dispositivi sono complicati
 - I dispositivi sono diversi anche su macchine con stessa CPU
 - Le operazioni sono le stesse per tutti i programmi (leggere/scrivere file)
 - Stesse operazioni su dispositivi di tipo diverso: e.g. file su HD, USB key, etc...
- Soluzione
 - Sistema operativo!



Livelli come macchine virtuali

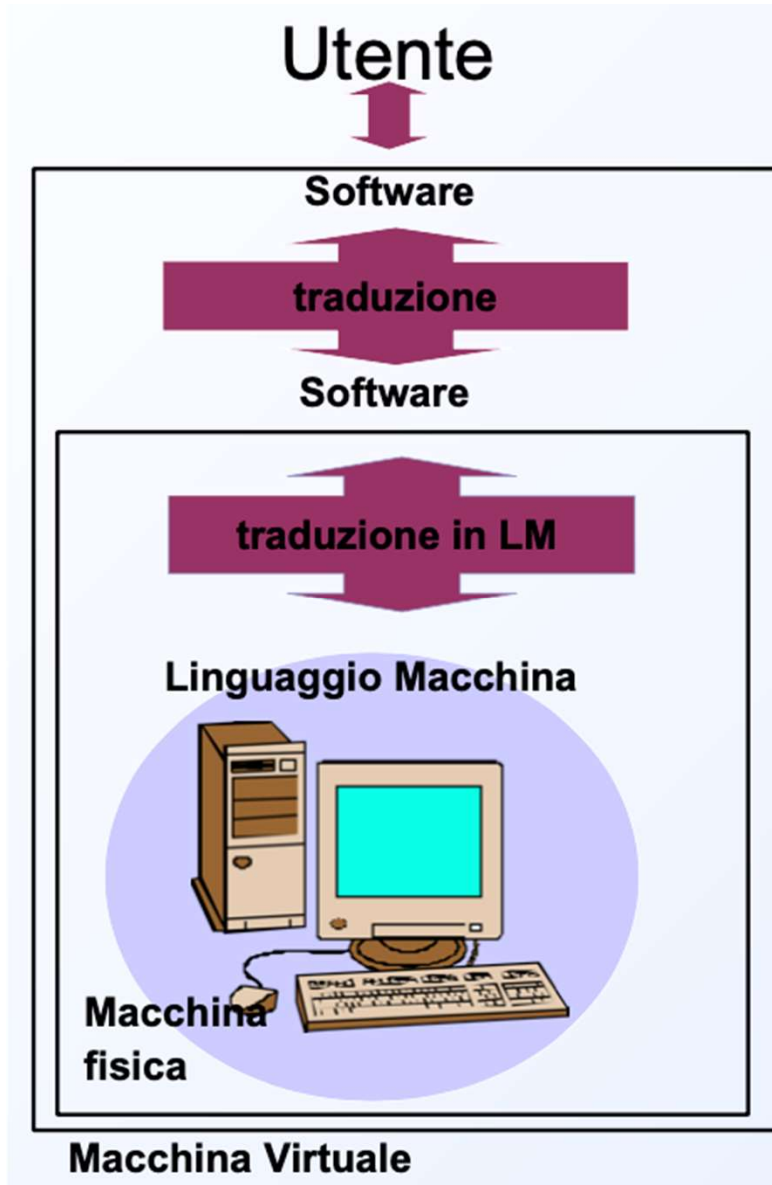
- Il linguaggio assembler è troppo primitivo per programmare
- Un linguaggio ad alto livello ha:
 - Tipi di dati complessi
 - Controllo dei tipi
 - Programmazione strutturata (while, for vs goto)
 - Polimorfismo
 - ...



Livelli come astrazioni

- Desideriamo astrarci dai dettagli fisici della macchina in oggetto e dal suo specifico linguaggio macchina
- L'idea è quella di realizzare al di sopra della macchina reale una macchina virtuale astratta che abbia le funzionalità desiderate e che sia facile da utilizzare per l'utente
- L'utente interagisce con la macchina virtuale, ogni comando viene poi tradotto nei corrispondenti comandi sulla macchina fisica
- La macchina virtuale è realizzata mediante software (programmi)

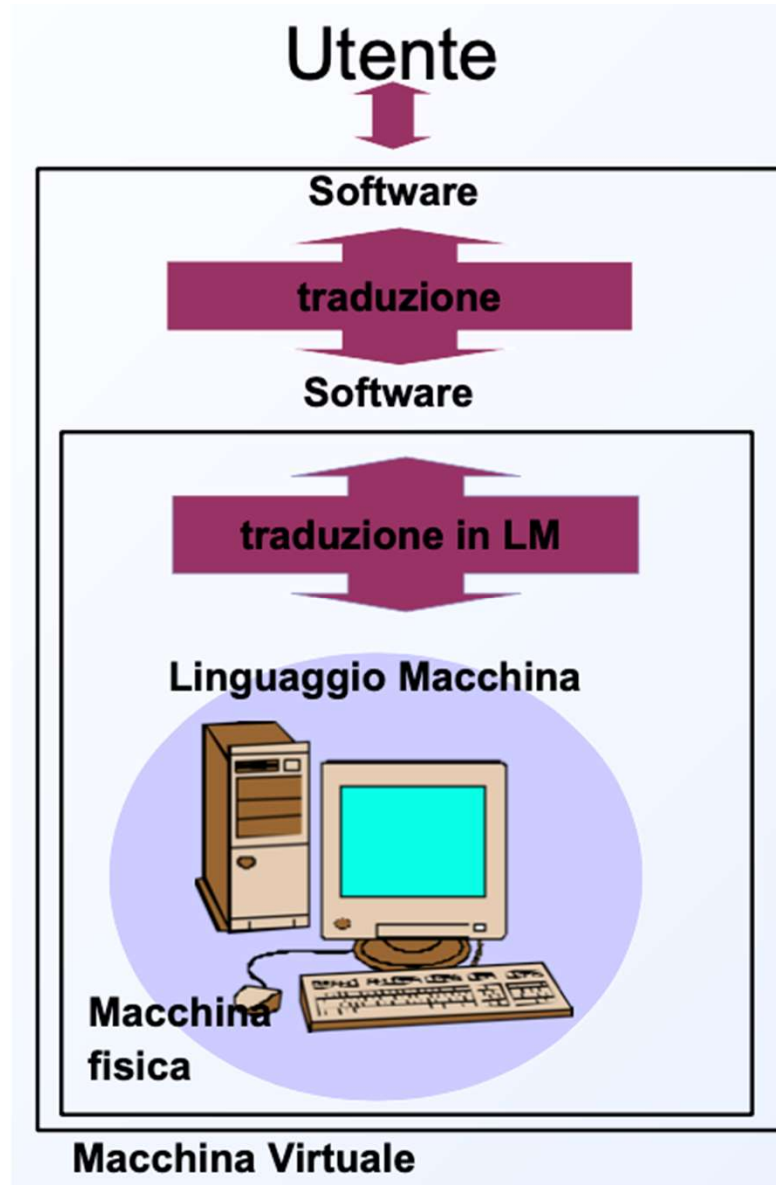
RISC-V Instruction Set



Livelli come astrazioni

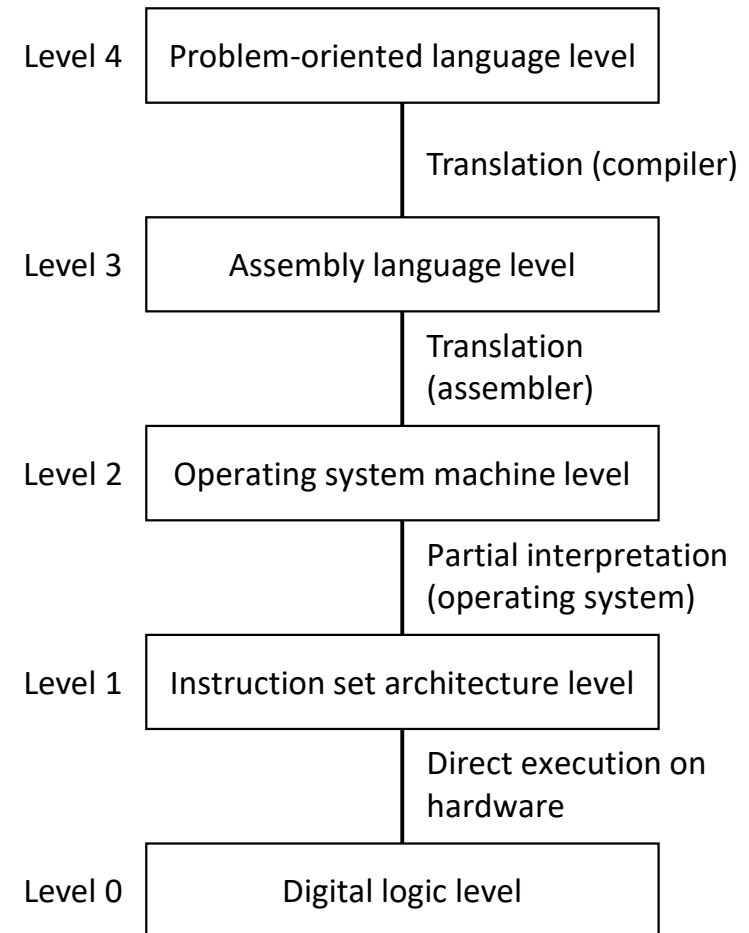
- La macchina virtuale viene realizzata in genere mediante il software di base:
- **Sistema Operativo:** file system, memoria, cpu, risorse ausiliarie, comunicazione
- **Linguaggi e ambienti di programmazione ad alto livello:** interpreti e compilatori
- Non vi sono limiti al numero e al tipo di macchine virtuali che possono essere realizzate
- In genere nelle macchine moderne sono strutturate su più livelli (**struttura a cipolla**)

RISC-V Instruction Set



Organizzazione a livelli

- La maggior parte dei moderni computer consiste di 2 o più livelli (nel nostro caso 5)
- Livello 0
 - rappresenta l'**hardware della macchina** i cui circuiti eseguono i programmi scritti nel linguaggio macchina del livello superiore
 - porte logiche di base, costituite da **transistor**, dotate di 1 o più input digitali (segnali corrispondenti ai valori 0 e 1) che calcolano semplici funzioni dei valori in ingresso
 - le porte si possono combinare per formare
 - **circuito chiamato ALU**, capace di effettuare semplici operazioni aritmetiche
 - **memorie e registri** in grado di memorizzare le informazioni

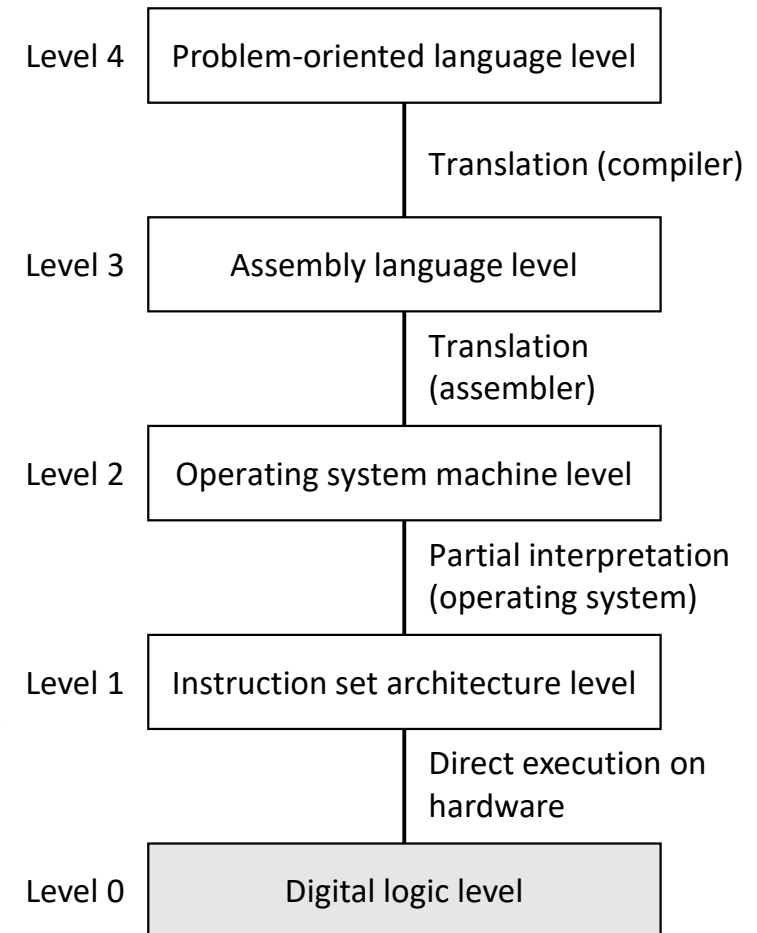


Organizzazione a livelli

• Livello 0

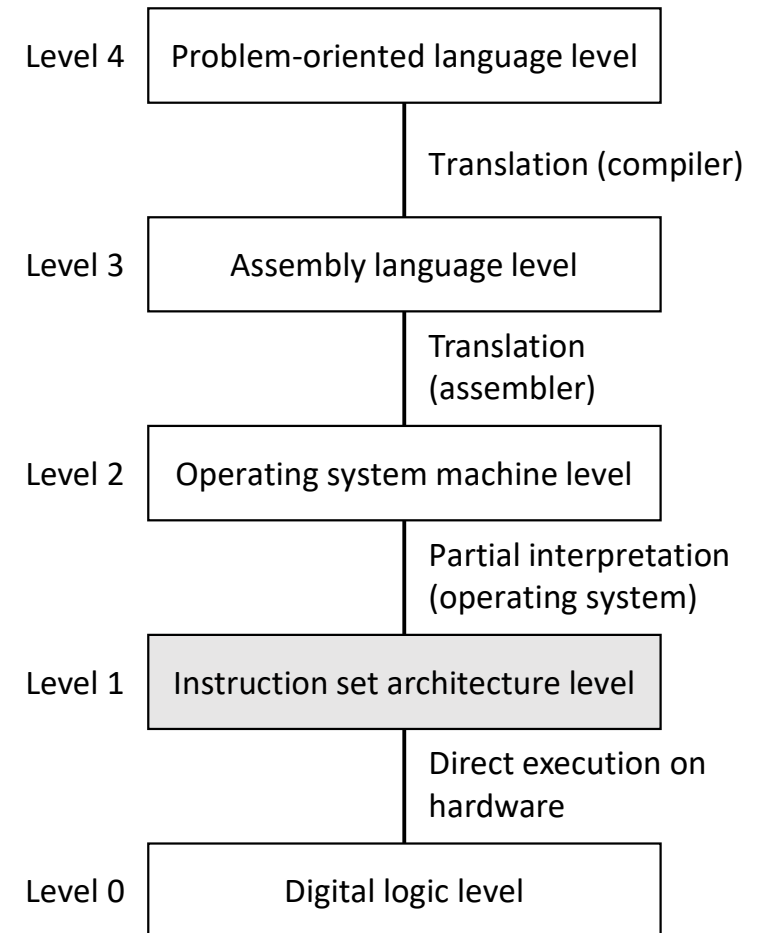
Bu J

- I registri sono connessi alla ALU per formare il percorso dati lungo il quale i dati si spostano
- In alcune macchine le operazioni del percorso dati sono controllate da un programma: il microprogramma (logicamente sarebbe presente un livello in più)
- mentre in altre è controllato direttamente dall'hardware (come nel nostro corso)
- In passato era quasi sempre rappresentato da un interprete software, oggi invece è spesso controllato in modo diretto dall'hardware.



Organizzazione a livelli

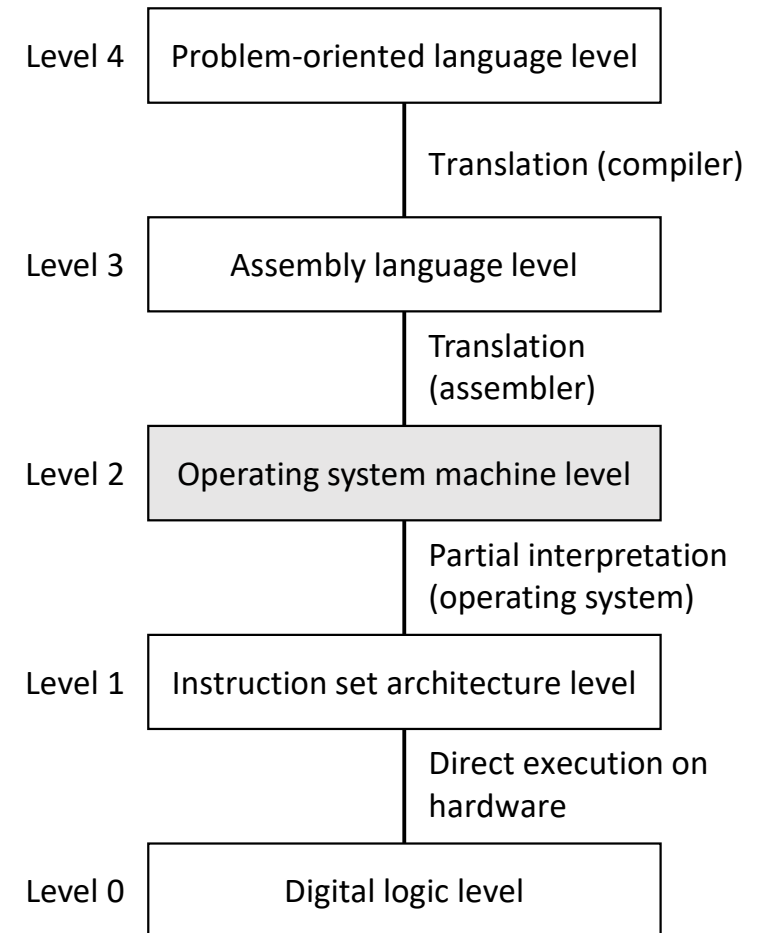
- Livello 1 – ISA (Instruction Set Architecture)
 - livello del linguaggio macchina
 - I manuali che descrivono le istruzioni macchina presentano le istruzioni di questo livello, eseguite direttamente dall'hardware (o in modo interpretato dal microprogramma)



Organizzazione a livelli

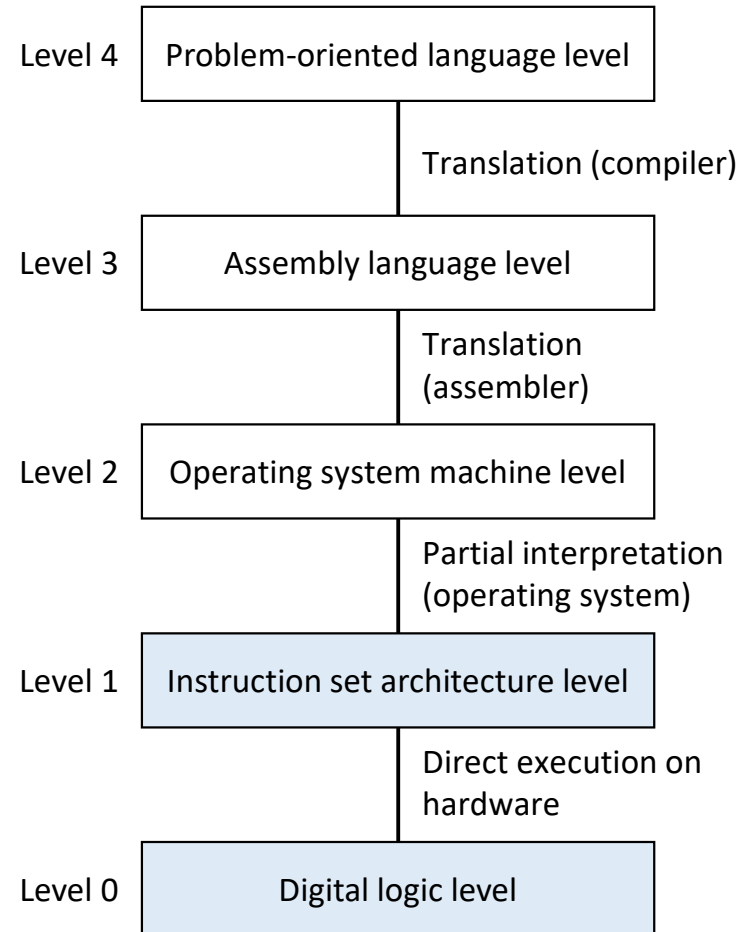
- Livello 2 – Sistema operativo

- la maggior parte delle istruzioni del livello 2 fa parte anche del livello ISA
- vi sono inoltre nuove istruzioni, diversa organizzazione della memoria, capacità di eseguire i programmi in modo concorrente, ...



Panoramica del corso

Numerico
↑
Controllo gruppi di
circuiti in sequenza

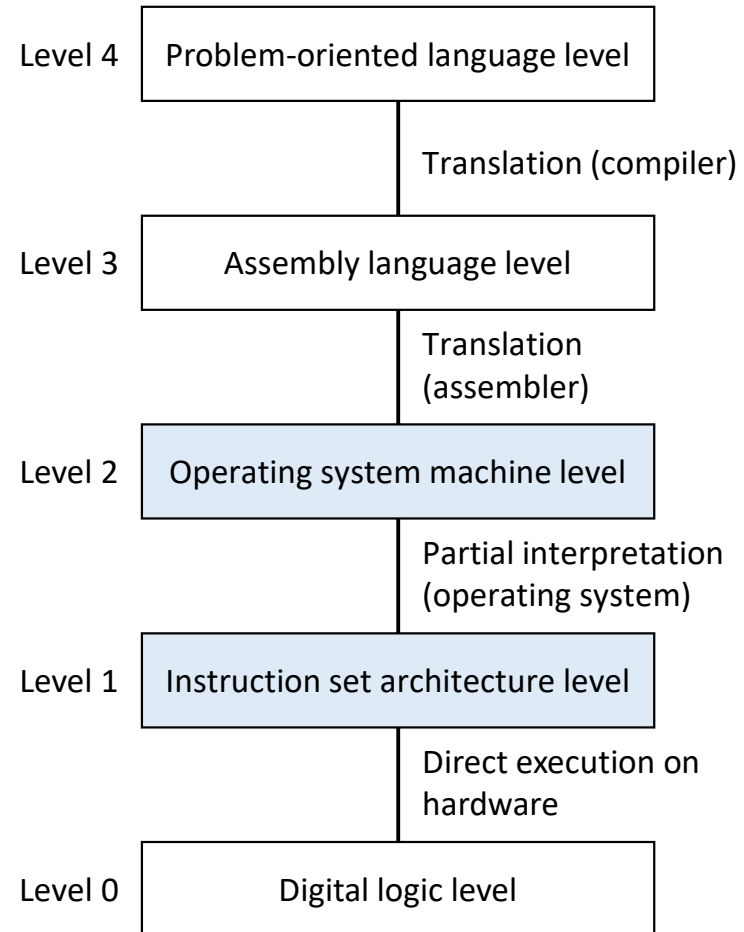


Panoramica del corso

CPU+I/O+Memoria



CPU

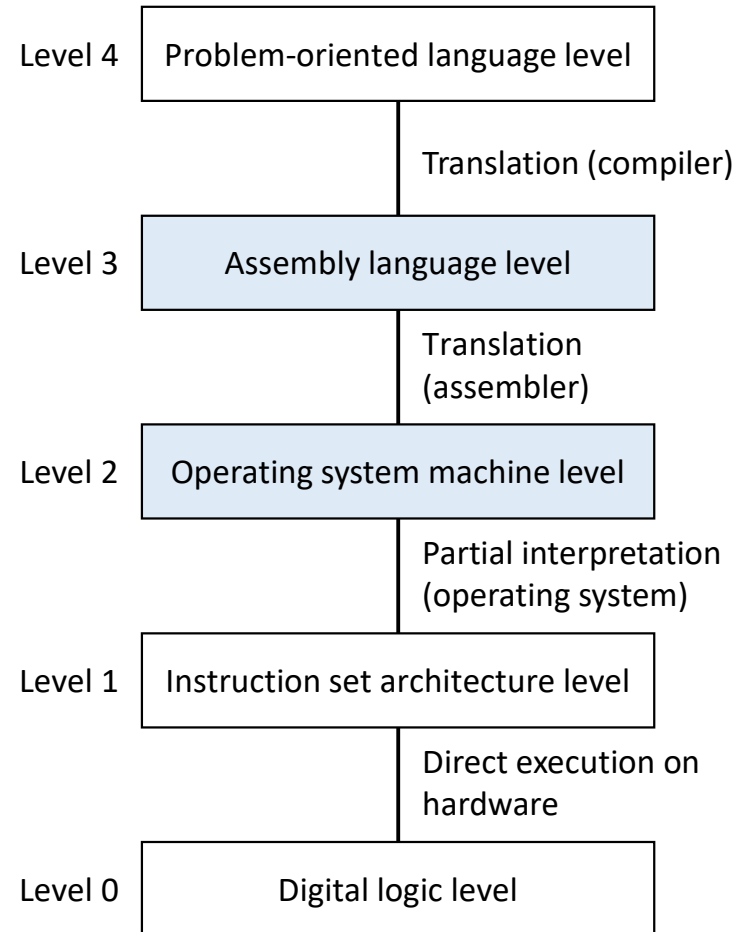


Panoramica del corso

Simbolico



Numerico

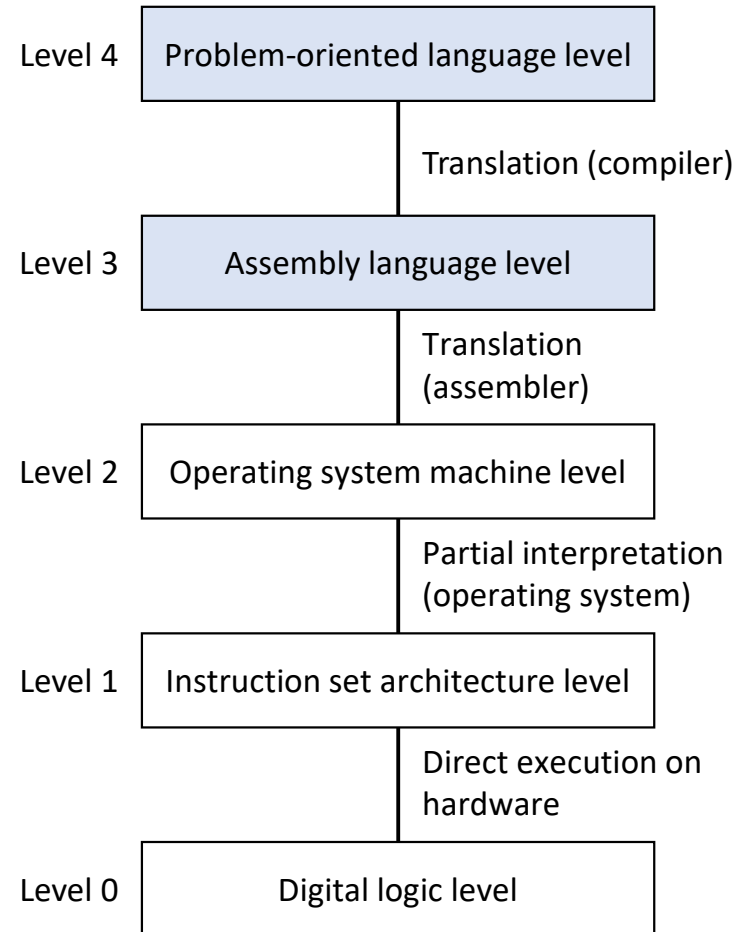


Panoramica del corso

High level language



Low level language

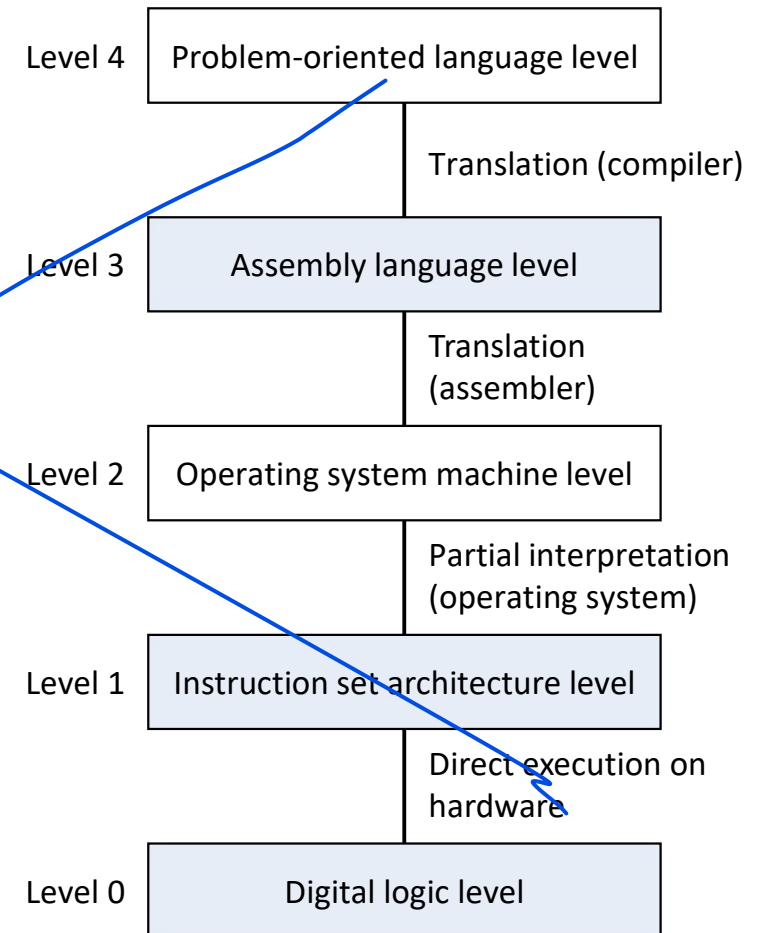


salute guardo



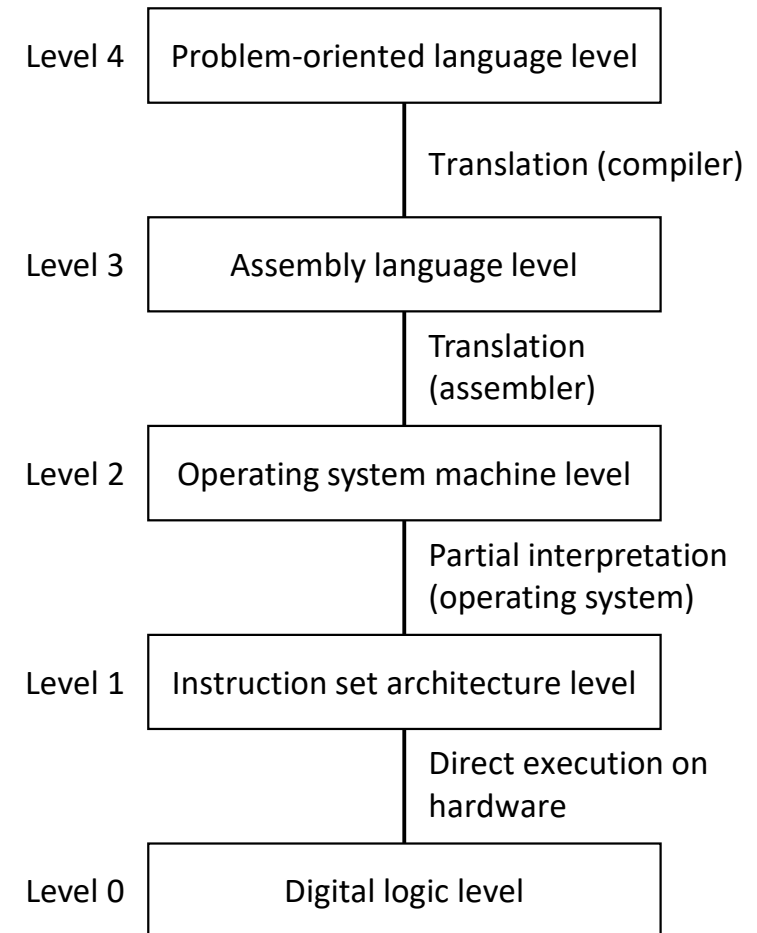
Panoramica del corso

- Organizzazione strutturata del calcolatore
- Studieremo i livelli 0, 1 e un po' del 3
- Nel secondo anno studierete il livello 2 (Sistemi Operativi). Studierete anche i principi della traduzione (LFT)
- Nei corsi di Programmazione 1 e 2 avete incominciato a studiare il livello 4



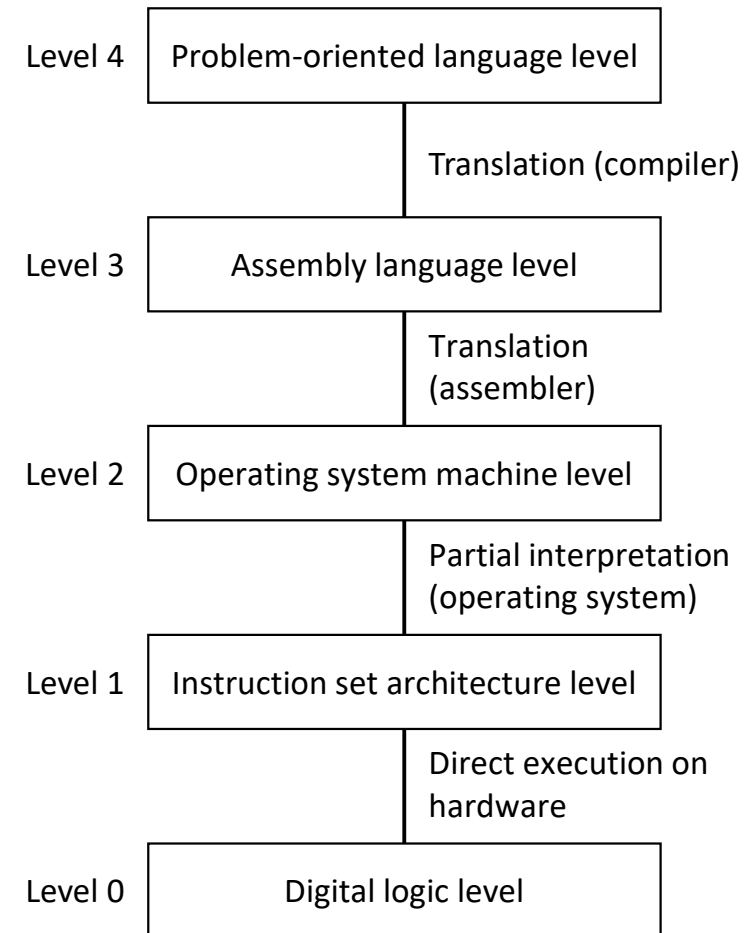
Panoramica del corso

- Livello 0: Logico-Digitale
 - porte
 - registri
 - memoria
 - Arithmetic Logic Unit (ALU)
 - Data Path
- Livello 1: Instruction set (ISA)
 - Linguaggio Macchina
 - Supporti architetturali

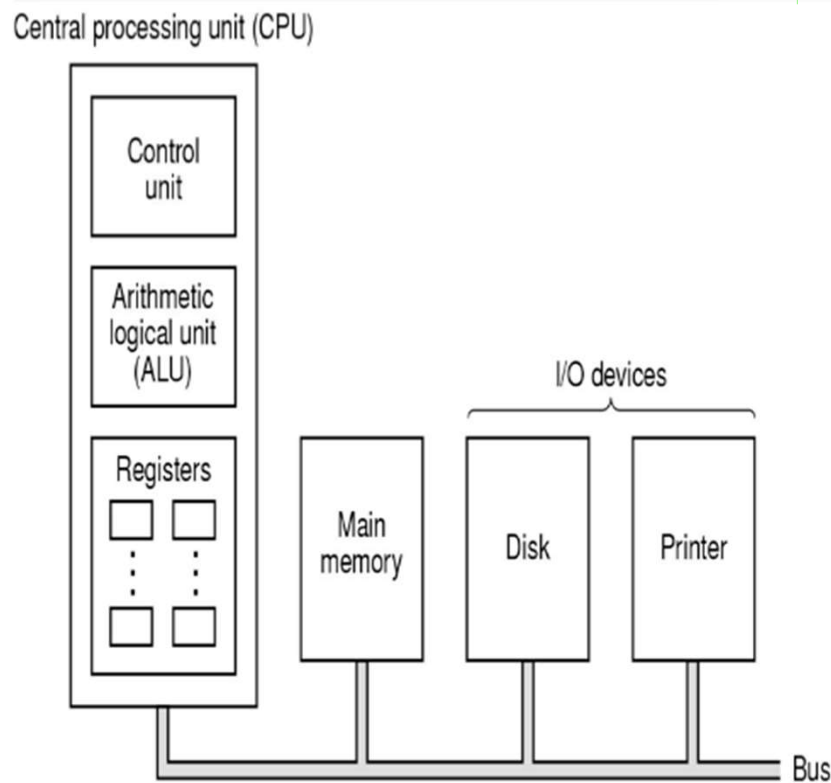


Architettura e organizzazione

- L'insieme di tipi di dati, operazioni e caratteristiche di ogni livello si chiama **architettura**
- Lo studio di come progettare le parti di un sistema che sono visibili ai programmatori si chiama **architettura dei calcolatori**
- **Organizzazione**, relazioni strutturali tra le unità funzionali di una data architettura (non visibile al programmatore)
- Spesso il termine **organizzazione** è usato come sinonimo di architettura



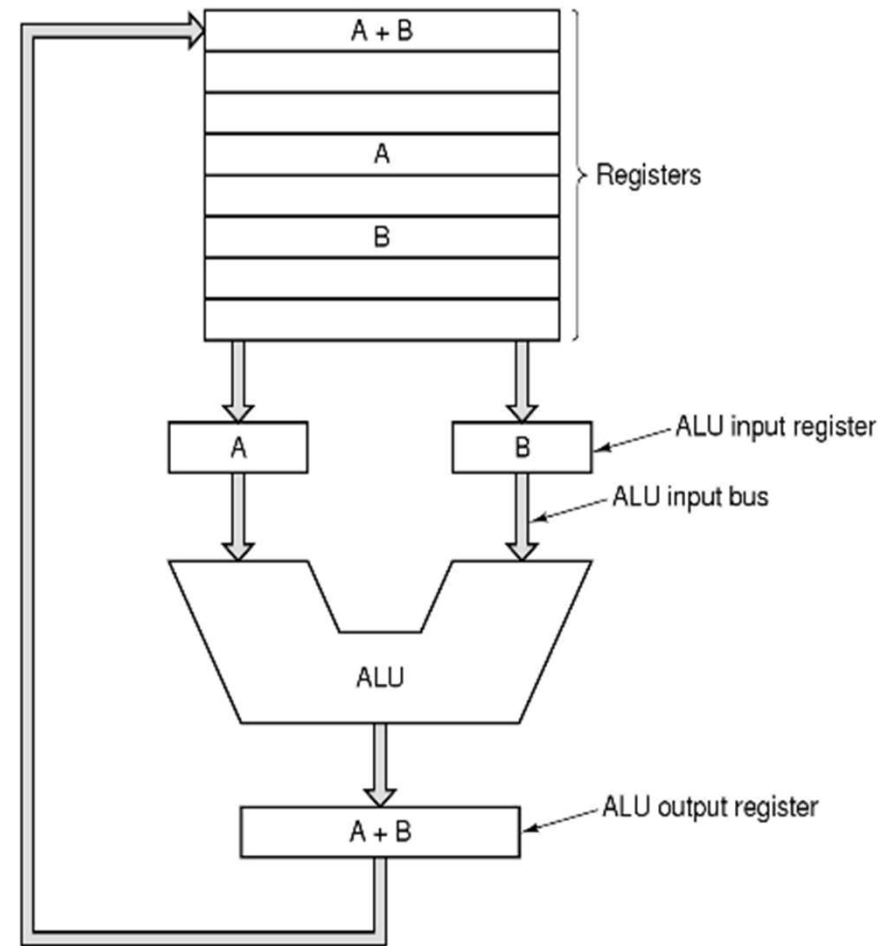
Organizzazione della CPU in una macchina di Von Neumann



- La CPU si compone di diverse parti distinte: unità di controllo, unità aritmetico-logica, registri
- I registri, l'unità aritmetico-logica e alcuni bus che li collegano compongono il data path
- Due registri importanti: Program Counter (PC) e Instruction Register (IR)
- La main memory contiene sia istruzioni sia dati usando sequenze di bit.

CPU di Von Neumann

- **Data Path:** organizzazione interna di una CPU (registri, ALU, bus interno)
- **Registro:** memoria veloce per dati temporanei
- Istruzioni registro-registro e registro-memoria
- Ciclo del data path: processo di far passare due operandi attraverso l'ALU e memorizzarne il risultato



Esecuzione delle istruzioni: ciclo di fetch-decode-execute

La CPU esegue ogni istruzione del livello 1 (ISA) per mezzo di una serie di passi elementari:

1. Prendi l'istruzione seguente dalla memoria e mettila nel registro delle istruzioni
2. Cambia il program counter per indicare l'istruzione seguente
3. Determina il tipo dell'istruzione appena letta
4. Se l'istruzione usa una parola in memoria, determina dove si trova
5. Metti la parola, se necessario, in un registro della CPU
6. Esegui l'istruzione
7. Torna al punto 1 e inizia a eseguire l'istruzione successiva

Esecuzione delle istruzioni: ciclo di fetch-decode-execute

```
static int PC, AC;
static int instr, instr_type;
static int data_loc, data;

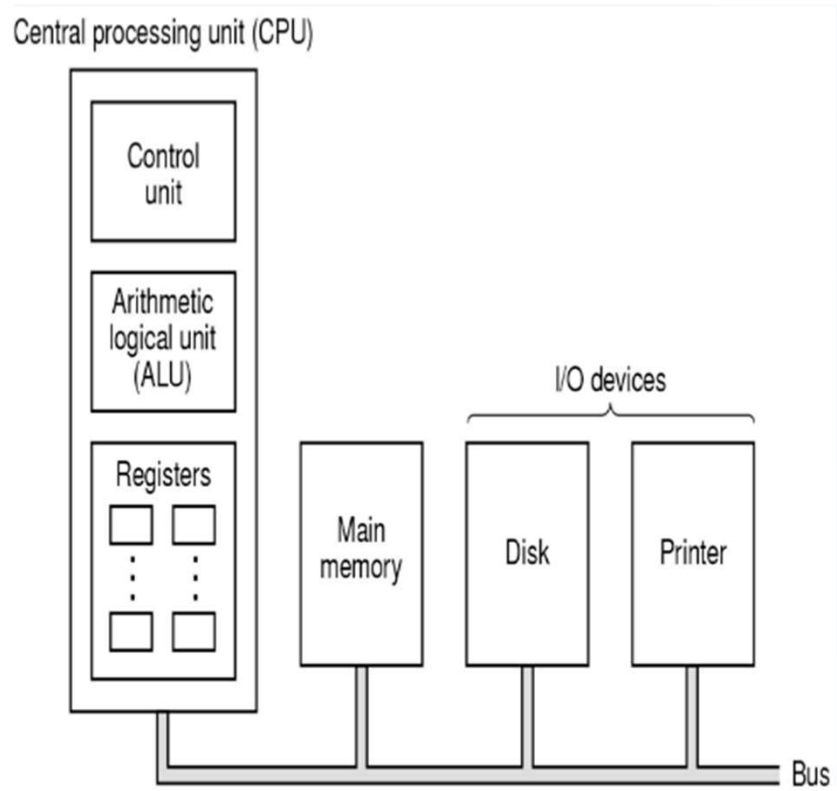
public static void interpret(int memory[], int starting_address)
{
    PC = starting_address;

    while (true) {
        instr = memory[PC];
        PC = PC + 1;

        instr_type = get_instr_type(instr);
        data_loc = find_data(instr, instr_type);
        if (data_loc >= 0)
            data = memory[data_loc];

        execute(instr_type, data);
    }
}
```

Organizzazione della CPU in una macchina di Von Neumann

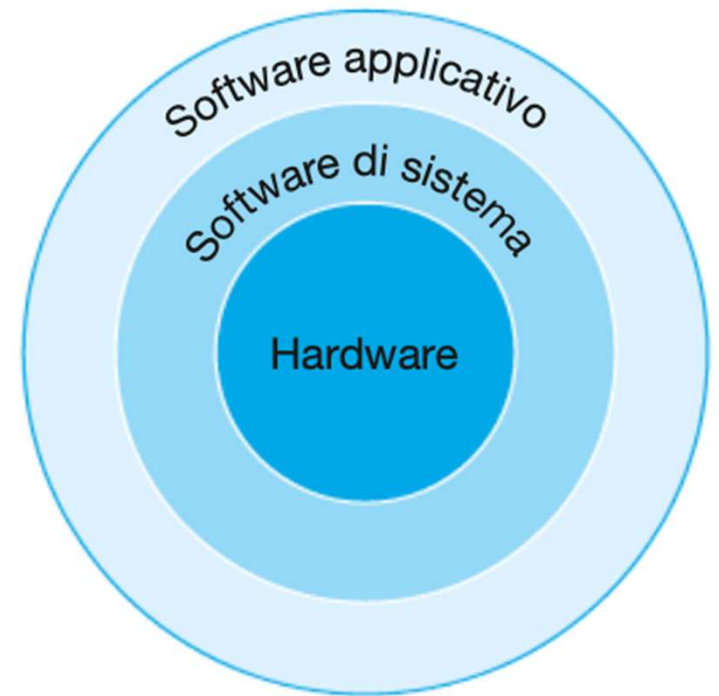


- È l'unità di controllo che esegue il ciclo di fetch- decode-execute: cioè, legge le istruzioni dalla memoria centrale (fetch), ne determina il tipo (decode) e provvede ad eseguirle (execute)
- L'unità di controllo può essere vista come un programma che permette di interpretare le istruzioni ed impostare in maniera corrispondente il data path

riscv.visit

Il ruolo del software

- Organizzazione a livelli
- Software applicativo
 - Tipiche applicazioni utilizzate dall'utente finale
- Software di sistema
 - Sistema operativo
 - Compilatore
- Hardware

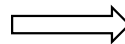


Ciclo di vita del software

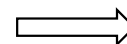
- Come si passa da un programma scritto con un linguaggio ad alto livello ad uno scritto in linguaggio macchina?

```
scambia(size_t v[], size_t k) {  
    size_t temp;  
    temp = v[k];  
    v[k] = v[k+1];  
    v[k+1] = temp;  
}
```

Linguaggio C



???



```
00000000001101011001001100010011  
00000000011001010000001100110011  
000000000000000110011001010000011  
00000000100000110011001110000011  
00000000011100110011000000100011  
00000000010100110011010000100011  
0000000000000000100000001100111
```

RISC-V linguaggio macchina

Ciclo di vita del software

- **Compilatore**

- Traduce un programma scritto in un linguaggio **ad alto livello** (C, Java, ...) in un programma scritto in linguaggio **assembler**

- **Assemblatore**

- Traduce programmi scritti in **assembler** (notazione simbolica) in istruzioni in **linguaggio macchina**

```
scambia(size_t v[], size_t k) {  
    size_t temp;  
    temp = v[k];  
    v[k] = v[k+1];  
    v[k+1] = temp;  
}
```

Linguaggio C

COMPILATORE

```
scambia:  
    slli x6, x11, 3  
    add x6, x10, x6  
    ld x5, 0(x6)  
    ld x7, 8(x6)  
    sd x7, 0(x6)  
    sd x5, 8(x6)  
    jalr x0, 0(x1)
```

RISC-V linguaggio assembler

RISC-V Instruction Set

ASSEMBLATORE

```
00000000001101011001001100010011  
00000000011001010000001100110011  
00000000000000110011001010000011  
00000000100000110011001110000011  
0000000011100110011000000100011  
00000000010100110011010000100011  
0000000000000000100000001100111
```

RISC-V linguaggio macchina

RISC-V Instruction set

- Proposto e sviluppato all'Università di Berkeley dal 2010
- Obiettivi
 - Costruire un hardware semplice
 - Compilatori efficienti
 - Massimizzare le prestazioni
 - Minimizzare costi
 - Minimizzare il consumo energetico
- Standard aperto

↳ detto 1roff.

RISC-V tech roadmap

RISC-V Innovation Roadmap

Industry Adoption

Proliferation of RISC-V CPUs across performance and application spectrum
RISC-V dominant in universities
Strategic and growing adoption in HPC, automotive, transportation, cloud, industrial, communications, IoT, enterprise, consumer, and other applications

Test Chips
Software tests
Linux port

Proof of Concept SoCs
Minion processors for power management, communications
Bare metal software

IoT SoCs
Microcontrollers
RTOS, Firmware
Development tools
Technical Steering Committee,
HPC SIG,
GlobalPlatform partnership

AI SoCs, Application processors, Linux Drivers, AI Compilers
Dev Board program
Development Partners
RISC-V Labs, Security response process, AI SIG, Graphics SIG, Android SIG, Communications SIG

2010 - 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025 →

ISA Definition
RISC-V Foundation

RV32

RV32I and RV64I
Base instructions: Integer, floating point, multiply and divide, atomic, and compact instructions

Priv modes, Interrupts, exceptions, memory model, protection, and virtual memory

Arch compatibility framework, Processor trace

Zfinx
ZiHintPause
BitManip
Vector
RISC-V Profiles & Platforms
Crypto Scalar
Virtual Memory
Hypervisor & Advanced interrupt architecture
Cache mgt ops
Code size reduction*
Trusted Execution Environment*
P (Packed SIMD)*

RV32E and RV64E
64 bit and 128 bit addresses*
Vector Atomic and quad-widening*
Quad floating point in integer registers*
Crypto Vector*
Trusted Execution phase 2*
Jit pointer masking & I/D synch*
BitManip phase 2*
Cache management phase 2*
... and more

Technical Deliverables



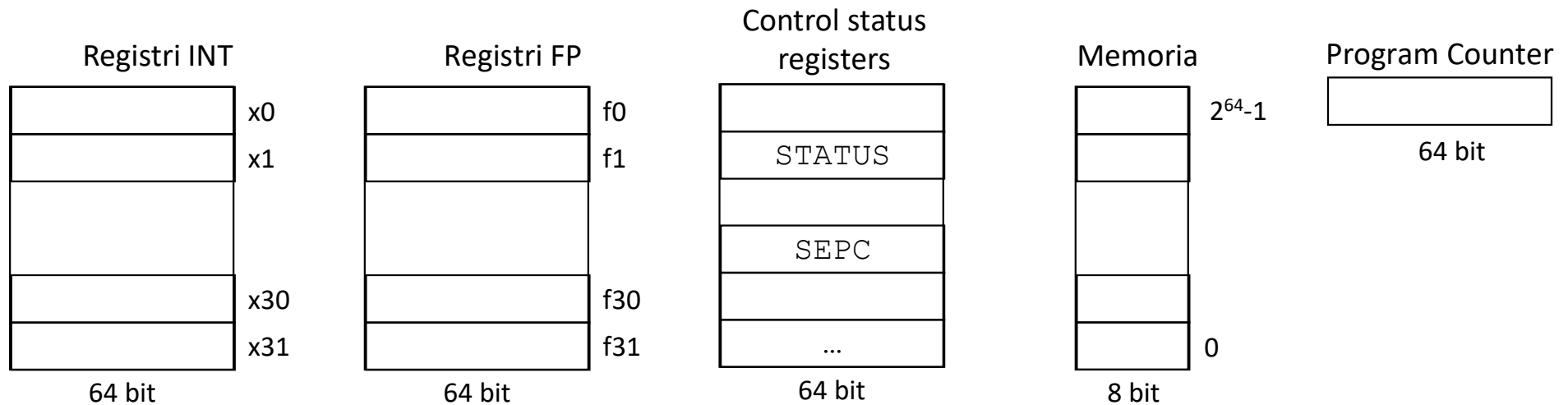
* On track, subject to change

RISC-V Instruction set

- RISC
 - Reduced Instruction Set Computer
- Principi di progettazione
 1. La semplicità favorisce la regolarità
 2. Minori sono le dimensioni, maggiore è la velocità
 3. Un buon progetto richiede buoni compromessi

reg 1

RISC-V Registri e memoria



Parola (word): 32 bit

Parola doppia (doubleword): 64 bit

- 32 registri per gli interi
- 32 registri per i numeri in virgola mobile
- Program Counter
- 4096 control status registers
- Memoria centrale

RISC-V Registri e memoria

Parola (word): 32 bit

Parola doppia (doubleword): 64 bit

- Registri per gli interi
 - Quantità: 32, indicati con x0 .. X31
 - Dimensione: 64 bit
 -

x0	zero (<u>costante</u>) <u>read-only</u>
x1	Return address (ra) • (dove deve tornare <u>ind. istruzioni</u>)
x2	Stack pointer (sp) <u>ultima locazione stack</u> (<u>push</u> e <u>pop</u> di metodi)
x3	Global pointer (gp)
x8	Frame pointer (fp)
x10-x17	Registri usati per il passaggio di parametri nelle procedure e valori di ritorno
x5-x7, x28-x31	Registri temporanei, non salvati in caso di chiamata
x8-x9, x18-x27	Registri da salvare: il contenuto va preservato se utilizzati dalla procedura chiamata

RISC-V Instruction set

- Tipologie di istruzione
 - Aritmetiche
 - Logiche
 - Accesso alla memoria
 - Condizionali

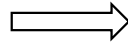
Istruzioni aritmetiche

- Notazione rigida
 - Tutte le istruzioni aritmetiche hanno esattamente 3 operandi
 - L'ordine degli operandi è fisso

- Addizione

`a = b + c`

Linguaggio C



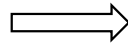
`add a, b, c`

RISC-V assembler

- Sottrazione

`a = b - c`

Linguaggio C



`sub a, b, c`

RISC-V assembler

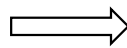
Istruzioni aritmetiche

- Operandi

- Gli operandi devono essere sempre **tre registri x0..x31**
- La ALU ha come input solo il contenuto di registri
- **I numeri interi sono rappresentati in complemento a due**

`a = b + c`

Linguaggio C



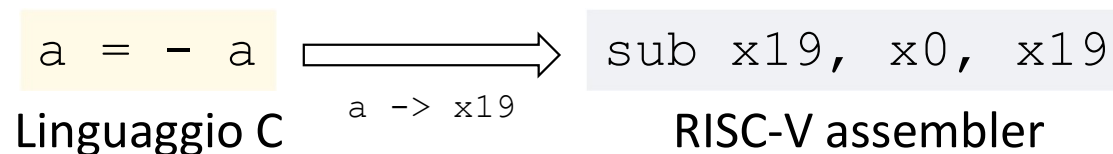
a → x5
b → x20
c → x21

`add x5, x20, x21`

RISC-V assembler


Istruzioni aritmetiche

- Spesso si ha la necessità di **cambiare segno** al valore di un registro
- L'istruzione `sub` può essere utilizzata, memorizzando il valore 0 nel secondo operando
- In RISC-V il registro **x0** contiene sempre il valore 0



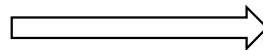
Istruzioni aritmetiche

- Il processo di traduzione di codice ad alto livello in linguaggio assembler è svolto dal compilatore
- Un'unica istruzione in un linguaggio ad alto livello può corrispondere a diverse istruzioni assembler



```
f = a + b - c
```

Linguaggio C



```
f -> x19  
a -> x20  
b -> x21  
c -> x22
```

```
add x19, x20, x21  
sub x19, x19, x22
```

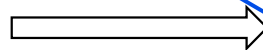
RISC-V assembler

Istruzioni aritmetiche

- Il processo di traduzione di codice ad alto livello in linguaggio assembler è svolto dal compilatore
- Un'unica istruzione in un linguaggio ad alto livello può corrispondere a diverse istruzioni assembler

`f = (g + h) - (i + j)`

Linguaggio C




?

RISC-V assembler

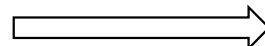
Istruzioni aritmetiche

- Il processo di traduzione di codice ad alto livello in linguaggio assembler è svolto dal compilatore
- Un'unica istruzione in un linguaggio ad alto livello può corrispondere a diverse istruzioni assembler



```
f = (g + h) - (i + j)
```

Linguaggio C



```
f -> x19  
g -> x20  
h -> x21  
i -> x22  
j -> x23
```

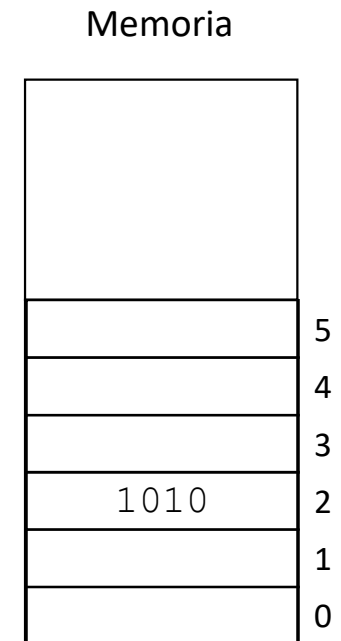
temporary

```
add x5, x20, x21  
add x6, x22, x23  
sub x19, x5, x6
```

RISC-V assembler

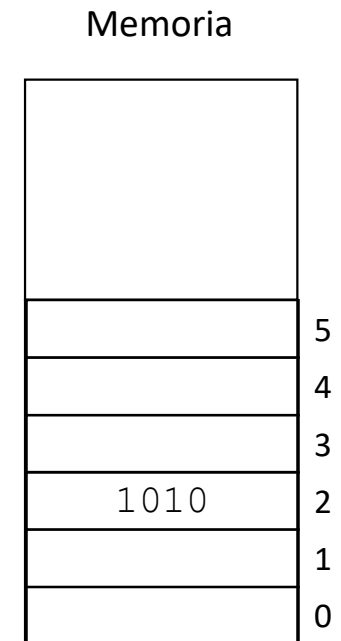
Istruzioni di accesso alla memoria

- Che cosa accade quando
 - Le variabili utilizzate in un programma sono maggiori del numero di registri a disposizione
 - Si utilizzano strutture dati complesse (vettori, liste, ecc)
- I dati sono salvati in memoria centrale
- La **memoria** centrale può essere **astratta** come un grande **vettore monodimensionale**
- Nell'esempio, la quarta cella di memoria ha valore 1010
 - $M[2] = 1010$



Istruzioni di accesso alla memoria

- La ALU può leggere e scrivere solo dai registri
- L'accesso alla memoria è più lento rispetto a quello dei registri
- Il compilatore si occupa di individuare la strategia più efficiente per le operazioni di caricamento e salvataggio dei dati tra registri e memoria
- Variabili utilizzate più di frequente devono rimanere il più possibile salvate nei registri



Istruzioni di accesso alla memoria

- L'istruzione `load` copia un dato dalla memoria ad un registro
- L'indirizzo del dato in memoria viene specificato da:
 - Indirizzo base (contenuto in un registro)
 - Scostamento o offset (compreso tra -2048 e +2047)
- L'istruzione **`ld`** (`load doubleword`) carica una **parola doppia** dalla memoria in un registro

```
long a;  
long v[10];  
...  
a = v[3]
```

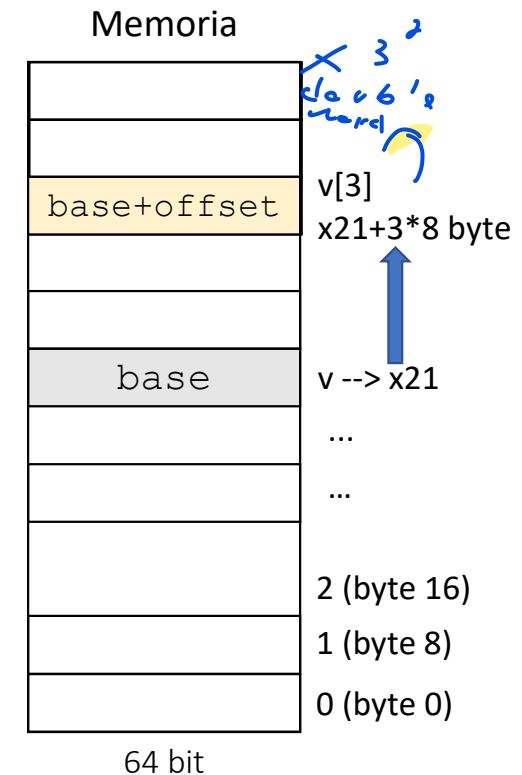
Linguaggio C

→
a -> x5
v -> x21

`ld x5, 24(x21)`

RISC-V assembler

RISC-V Instruction Set



Istruzioni di accesso alla memoria

- L'istruzione `ld` (load doubleword) carica una **parola doppia** dalla memoria in un registro
- Le celle dell'array possono essere memorizzate con differenti orientamenti

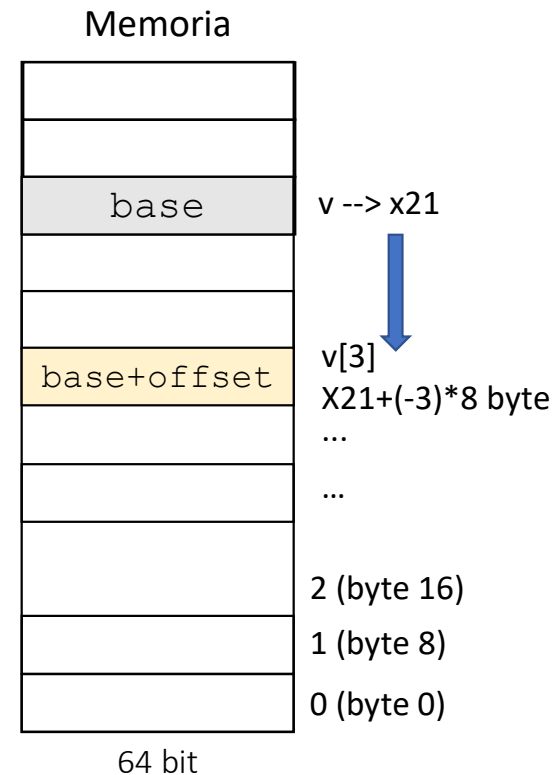
```
long a;  
long v[10];  
...  
a = v[3]
```

Linguaggio C

→
a -> x5
v -> x21

```
ld x5, -24(x21)
```

RISC-V assembler



Istruzioni di accesso alla memoria

- L'istruzione `store` copia un dato da un registro alla memoria
- L'indirizzo di destinazione in memoria viene specificato da:
 - Indirizzo base (contenuto in un registro)
 - Scostamento o offset (compreso tra -2048 e +2047)
- L'istruzione **`sd`** (`store doubleword`) salva una **parola doppia** in memoria

```
long a;  
long v[10];  
...  
v[2] = a
```

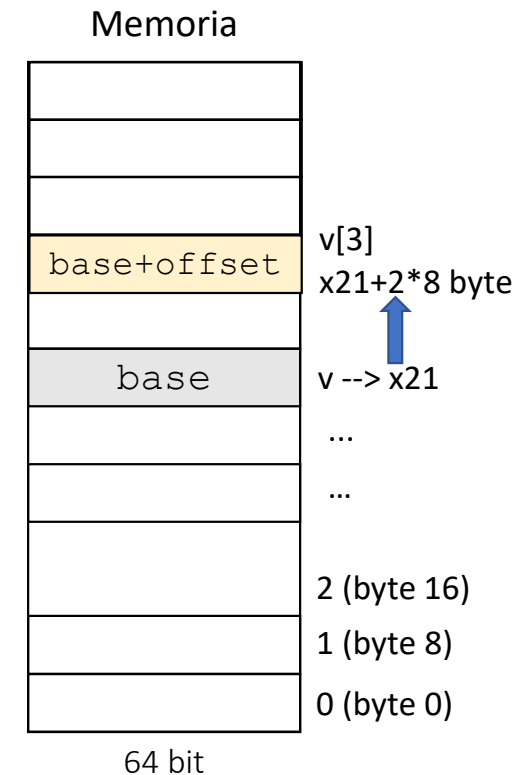
Linguaggio C

→
a -> x5
v -> x21

`sd x5, 16(x21)`

RISC-V assembler

RISC-V Instruction Set



Istruzioni di accesso alla memoria

- Un esempio

es

```
long g, h, f;
long v[10];
...
g = h + v[3]
v[6] = g - f
```

Linguaggio C

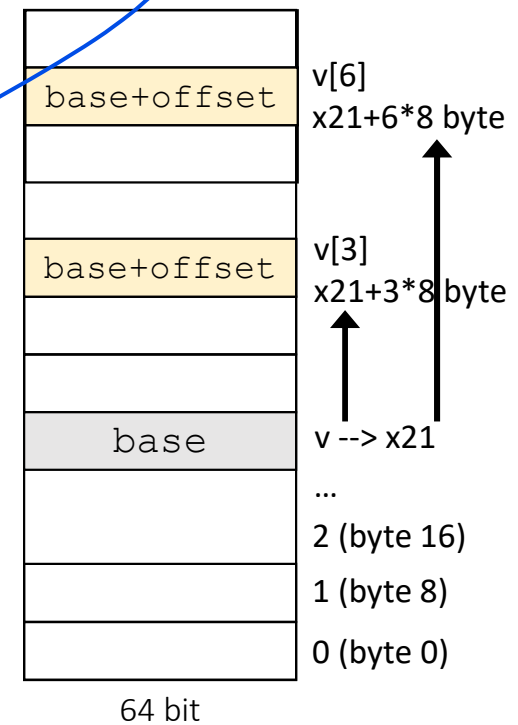


~~ld x5, 24(x21)
~~ld x17, ?(x18), x5
~~sub x6, x17, x16~~~~~~

RISC-V assembler

~~sd x6, 48(x21)~~

Memoria



Istruzioni di accesso alla memoria

- Un esempio

```
long g, h, f;  
long v[10];  
...
```

```
g = h + v[3];  
v[6] = g - f;
```

Linguaggio C

a → x5
h → x9
v → x21
f → x19

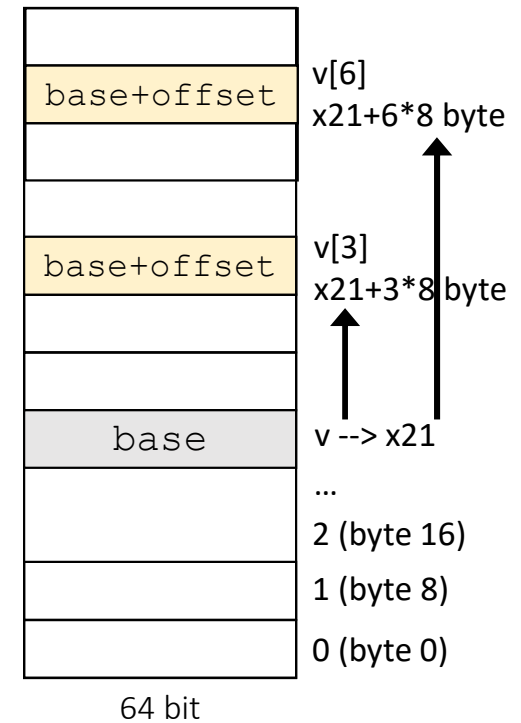
```
ld x10, 24(x21)  
add x5, x9, x10  
sub x5, x5, x19  
sd x5, 48(x21)
```

RISC-V assembler

offset

registro base

Memoria



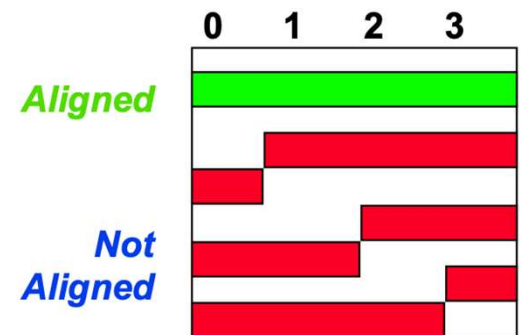
Istruzioni di accesso a byte, half-word e word

- Per accedere al singolo byte sono a disposizione
 - (Utile per le stringhe di caratteri ASCII)
 - `lb x5, 0(x6)` “load byte”
 - `sb x5, 0(x6)` “store byte”
- Per accedere alla half-word (16 bit) ci sono
 - (Utile per le stringhe di caratteri UNICODE (es. in Java))
 - `lh x5, 0(x6)` “load half-word”
 - `sh x5, 0(x6)` “store half-word”
- Per accedere alla word (32 bit) ci sono
 - `lw x5, 0(x6)` “load word”
 - `sw x5, 0(x6)` “store half-word”

Nota: in fase di caricamento (load), dovendo porre la quantità da 8/16/32 bit in 64 bit, viene automaticamente effettuata **l'estensione del segno**. Se ciò non si vuole, si devono usare `lbu` (al posto di `lb`) e `lhu` (al posto di `lh`) e `lwu` (al posto di `lw`) ed estensione con 0

Restrizioni sull'allineamento degli indirizzi

- La memoria è classicamente indirizzata “al byte”
- Quindi, le istruzioni di load e store usano indirizzi al byte, però
 - `lw`, `lwu` e `sw` trasferiscono 32 bit
 - `lh`, `lhu` e `sh` trasferiscono 16 bit
 - solo `lb`, `lbu`, `sb` trasferiscono 8 bit
- E' conveniente pertanto che l'indirizzo sia opportunamente allineato...
 - per `lw`, `lwu`, `sw` dovrebbe essere allineato ad un multiplo di 4
 - per `lh`, `lhu`, `sh` dovrebbe essere allineato ad un multiplo di 2
- Esempi di dati ALLINEATI e NON ALLINEATI “alla word”



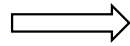
Nota: se si specifica un indirizzo non allineato rispetto a quanto l'istruzione desidera, il RISC-V genera un warning (avvertendo che il tempo per l'accesso al dato risulterà 2 volte più lento)

Istruzioni di accesso alla memoria

- Un esempio con variabili a 32 bit

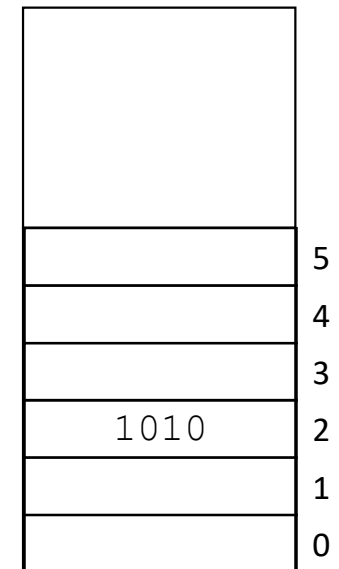
```
int g,h,f;  
int v[10];  
...  
g = h + v[3]  
v[6] = g - f
```

Linguaggio C



RISC-V assembler

Memoria



e)

Istruzioni di accesso alla memoria

- Un esempio con variabili a 32 bit

```
int g, h, f;
int v[10];
```

...

```
g = h + v[3]
v[6] = g - f
```

Linguaggio C

```
a → x5
h → x9
v → x21
f → x19
```

```
lw x10, 12(x21)
add x5, x9, x10
sub x5, x5, x19
sw x5, 24(x21)
```

RISC-V assembler

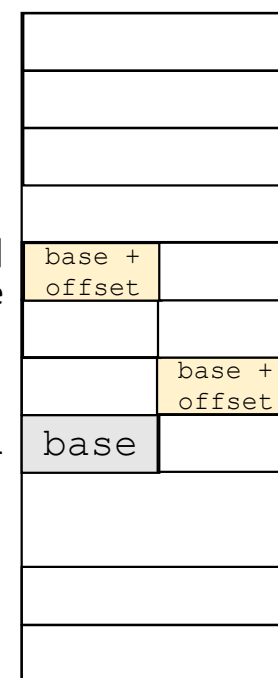
offset

registro base

v[6]
x21+6*4 byte

v --> x21

Memoria



v[3]
x21+3*4 byte

...
2 (byte 16)
1 (byte 8)
0 (byte 0)

64 bit

Operandi immediati e costanti

- In più della metà delle operazioni aritmetiche, uno degli operandi è una costante (benchmark SPEC CPU2006)
- I valori delle costanti solitamente sono molto piccoli
 - $a = a + 1$
 - $b = b + 5$
- Es: l'operazione $b = b + 5$ può essere rappresentata con due istruzioni assembler

```
b = b + 5
```

Linguaggio C

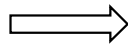
\Rightarrow
 $b \rightarrow x5$

```
ld x9, indirizzoCostante5(x3)
add x5, x5, x9
```

RISC-V assembler

Operandi immediati e costanti

- Alternativa: istruzioni aritmetiche in cui uno degli operandi è una costante
- L'istruzione di somma immediata è chiamata `addi` (add immediate)

<code>b = b + 5</code>		<code>addi x5, x5, 5</code>
Linguaggio C	$b \rightarrow x5$	RISC-V assembler

- La costante può assumere valori tra `-2048 e +2047`
- La sottrazione immediata non esiste: si usano le costanti con valore negativo