Calcolatori

Angelo Perotti

1. INTRODUZIONE

Importanza dei calcolatori

- **Impatto economico**: Negli USA i calcolatori producono il 10% del PIL
- **Evoluzione storica**:
  - 1946: ENIAC (per calcolo traiettorie proiettili) - occupava un'intera stanza, causò blackout
  - 1969: Margaret Hamilton progettò sistemi IBM per esplorazioni lunari
  - Oggi: iPhone più potenti di computer del passato, grandi quanto una mano
- **Primi PC**: Anni '50

Categorie di calcolatori

1. **Personal Computer** (desktop/laptop)
   - Unico utente alla volta
   - Buone prestazioni a costo ridotto
   - Progettati per software di terze parti
2. **Server**
   - Multiutente (in rete)
   - Gestiscono grandi carichi di lavoro:
     - Poche applicazioni molto complesse
     - Tante applicazioni semplici
3. **Embedded (IoT)**
   - Microprocessori integrati con hardware
   - Applicazioni dedicate

Elementi che influenzano le prestazioni

- Algoritmi
- Linguaggi di programmazione
- Compilatori
- Architetture
- Processore
- Sistema di memoria
- Sistema operativo
- Hardware

Componenti fondamentali

- **Sistema operativo**: Programma principale che alloca memoria, gestisce I/O e multitasking
- **Compilatore**: Traduce linguaggio ad alto livello in linguaggio macchina
- **Porte logiche**: Componenti base (interruttori elettrici)
- **Bit**: Unità base di informazione (0 o 1)

Evoluzione linguaggi

1. **Codice binario**: Primi programmatori scrivevano direttamente in binario
2. **Assembly**: Linguaggio più intuitivo tradotto dall'assembler
    - Una linea di codice = una istruzione hardware
    - Viene tradotto (non compilato) da stringhe a bit
3. **Linguaggi alto livello**: Tradotti prima in assembly, poi in codice macchina

Architettura processore

- **Datapath (ALU)**: Esegue operazioni aritmetiche
- **Parte di controllo**: Coordina le operazioni
- **Memoria cache**: Vicina al processore, salva dati frequenti per ottimizzare accesso

Astrazioni

- **ISA (Instruction Set Architecture)**: Interfaccia che raggruppa funzioni hardware (I/O, registri)
- **ABI (Application Binary Interface)**: ISA + interfaccia sistema operativo, definisce uso registri software

Tipi di memoria

- **Volatile**: RAM
- **Non volatile**: Hard disk, SSD

2. PRESTAZIONI

Tipi di prestazioni

1. **Tempo medio di risposta**: Interesse singolo utente (tempo avvio-terminazione task)
2. **Throughput**: Interesse server (quanti task/utenti per unità di tempo)

Metriche temporali

- **Tempo di esecuzione CPU**: Tempo effettivo CPU sul task (esclude attese altri task)
- **Ciclo di clock**: Intervallo tra due colpi di clock (secondi)
- **Frequenza di clock**: Inverso del ciclo di clock (Hertz)

Formula prestazioni

Tempo CPU = Numero istruzioni × CPI × Ciclo di clock
Dove:

- CPI = Cicli di clock medi per istruzione
- Tempo CPU = (Numero istruzioni × CPI) / Frequenza clock

Fattori che influenzano tempo CPU

1. **Numero di istruzioni**
2. **CPI (Cycles Per Instruction)**
3. **Frequenza di clock**

Importante: Non si può giudicare un processore solo dalla frequenza di clock!

Problema "stop delle prestazioni"

- **Formula potenza**: P = Capacità × Tensione^2 × Frequenza
- Limite fisico: tensione sotto 1V causa problemi di scarica
- **Soluzione**: Parallelismo con processori multi-core

Sfide del parallelismo

- **Correttezza**: Difficile progettare programmi paralleli
- **Efficienza**: Mantenere carico bilanciato tra CPU

3. ARITMETICA DEI CALCOLATORI

Rappresentazione base

- **Bit**: Binary digit (0 o 1)
- **Byte**: 8 bit
- **Codifica**: Funzione che associa oggetti a sequenze di bit
- Con n bit si possono rappresentare 2 oggetti diversi

Sistemi di numerazione

- **Base b**: b cifre disponibili (0 a b-1)
- **Sistema binario**: Base 2 (perfetto per computer)
- **Esadecimale**: Base 16 (riduce numero cifre)
- **Conversione binario-esadecimale**: 4 bit = 1 cifra esadecimale

Operazioni binarie

- **Somma**: Regole base con riporto
- **Sottrazione**: Regole base con prestito
- **Shifting**: Moltiplicazione/divisione per potenze di 2

Codifiche numeri interi con segno

1. Modulo e segno

- Bit più significativo = segno (0=positivo, 1=negativo)
- Altri bit = valore assoluto
- **Range**: -(2^1-1) a +(2^1-1)
- **Problema**: Doppia codifica dello zero

2. Complemento a 1

- Bit più significativo = segno
- Negativo = complemento a 1 del positivo
- **Range**: -(2^1-1) a +(2^1-1)
- **Problema**: Doppia codifica dello zero

3. Complemento a 2 (PIÙ USATO)

- Bit più significativo = segno
- Complemento a 2 = complemento a 1 + 1
- **Range**: -2^1 a +(2^1-1)
- **Vantaggi**: Codifica unica dello zero, operazioni semplificate

Overflow

- Si verifica quando risultato supera range disponibile
- **Complemento a 2**: Overflow solo con operandi stesso segno
- **Matematica dell'orologio**: Operazioni modulo 2

Numeri reali

Virgola fissa

- n bit totali: k bit parte intera, (n-k) bit parte decimale
- **Limite**: Range ristretto per ordini di grandezza diversi

Virgola mobile (IEEE 754)

- **Formato**: Numero = Mantissa × 2Êsponente
- **32 bit**: 1 bit segno, 8 bit esponente, 23 bit mantissa
- **Numeri normalizzati**: Esponente ¿ 0
- **Numeri denormalizzati**: Esponente = 0 (per numeri molto piccoli)

Codifica del testo

- **ASCII**: 7 bit (128 simboli)
- **Extended ASCII**: 8 bit (256 simboli, non univoco)
- **Unicode**: Codifica univoca universale
  - UTF-32: 4 byte
  - UTF-16: 2 byte
  - UTF-8: Variabile

4. RETI LOGICHE

Principi base

- **Livelli logici**: Alto (1/vero), Basso (0/falso)
- **Transistor**: Controllano passaggio corrente
- **Tipi di reti**:
  - **Combinatorie**: Uscita dipende solo da ingresso
  - **Sequenziali**: Uscita dipende da ingresso + storico (memoria)

Algebra di Boole

- **AND**: Prodotto ($\cdot$) - 1 solo se tutti 1
- **OR**: Somma (+) - 0 solo se tutti 0
- **NOT**: Negazione (barra sopra) - inverte valore

Regole fondamentali

- **Identità**: $A+0=A$, $A\cdot 1=A$
- **Assorbimento**: $A+1=1$, $A\cdot 0=0$
- **Inversa**: $A+\bar{A}=1$, $A\cdot\bar{A}=0$
- **Commutativa**: $A+B=B+A$, $A\cdot B=B\cdot A$
- **Associativa**: $A+(B+C)=(A+B)+C$
- **Distributiva**: $A\cdot(B+C)=(A\cdot B)+(A\cdot C)$
- **De Morgan**: $A\cdot B = \bar{A}+B$, $A+B = \bar{A}\cdot B$

Circuiti logici

- **Decoder**: n bit input $\rightarrow$ 2 bit output
- **Multiplexer**: Seleziona tra input basandosi su selettore
- **PLA (Programmable Logic Array)**: Piano AND + piano OR

Reti sequenziali

- **Latch S-R**: Set-Reset con memoria
- **Latch D**: Elimina stato indecidibile
- **Flip-flop Master-Slave**: Aggiornamento su fronte clock
- **Registri**: Latch edge-triggered

Macchine a stati

- **Mealy**: Output dipende da stato + input
- **Moore**: Output dipende solo da stato

5. ASSEMBLY

Concetti base

- **Assembly**: Compromesso tra linguaggio macchina e alto livello
- **Assembler**: Compilatore Assembly → linguaggio macchina
- **Non portabile**: Dipende dalla CPU specifica
- **ISA**: Definisce sintassi, semantica, accesso dati/registri

Struttura esecuzione

1. **Fetch**: Preleva istruzione da memoria
2. **Decode**: Decodifica istruzione
3. **Execute**: Esegue istruzione

Tipi di dato

- **Costante (immediato)**
- **Contenuto in registri**
- **Contenuto in memoria**

Filosofie architetture

RISC (Reduced Instruction Set Computer)

- **Caratteristiche**:
  - Architettura semplice
  - Istruzioni agiscono solo su registri
  - Poche istruzioni, meno potenti
  - Semplifica implementazione CPU
  - Codifica fissa

CISC (Complex Instruction Set Computer)

- **Caratteristiche**:
  - Architettura complessa
  - Operandi/destinazione anche in memoria
  - Molte istruzioni, più potenti
  - Facilita sviluppo programmi
  - Codifica variabile

Modalità di indirizzamento

1. **Assoluto**: Indirizzo costante
2. **Indiretto**: Indirizzo in registro
3. **Base + Spiazzamento**: Registro + costante
4. **Base + Indice**: Somma di due registri
5. **Base + Indice + Spiazzamento**: Combinazione completa

Registri

- **General purpose**: Uso generico
- **Specializzati**: Uso specifico
- **ABI**: Convenzioni software per uso registri

6. RISC-V

Principi di progettazione

1. **Semplicità**: Istruzioni a 3 operandi
2. **Velocità**: Solo 32 registri a 64 bit
3. **Buoni compromessi**: Istruzioni a 32 bit

Caratteristiche registri

- **32 registri a 64 bit**
- **Double word**: 64 bit
- **Word**: 32 bit
- **Registro x0**: Sempre contiene 0

Operazioni base

- **Aritmetiche**: add, sub, addi
- **Logiche**: and, or, xor, sll, srl, sra
- **Memoria**: load (ld), store (sd)
- **Controllo**: beq, bne, blt, bge

Formato istruzioni

- **R-type**: Operazioni registro-registro
- **I-type**: Operazioni immediate
- **S-type**: Store
- **B-type**: Branch

Istruzioni condizionali

- **beq**: Branch if equal
- **bne**: Branch if not equal
- **blt**: Branch if less than
- **bge**: Branch if greater equal

Procedure

- **Convenzioni registri**:
  - x10-x17: Parametri e valori ritorno
  - x1: Indirizzo di ritorno
  - x2: Stack pointer
  - x8: Frame pointer

Gestione stack

- **Prologo**: Salva registri usati
- **Epilogo**: Ripristina registri
- **Stack frame**: Segmento stack per funzione

Ottimizzazioni

- **Funzioni foglia**: Evitano prologo/epilogo se possibile
- **Ricorsione di coda**: Evita salvataggi non necessari
- **Registri temporanei**: x5-x7, x28-x31 (non salvati)

7. Intel x86

Caratteristiche Generali

- **Tipo**: CISC (Complex Instruction Set Computer)
- **Utilizzo**: Maggior parte di laptop, desktop e server moderni
- **Evoluzione**: Compatibilità backwards (CPU 64-bit eseguono codice 8-bit)
- **Complessità**: Elevata per mantenere compatibilità

Registri Intel

- **Prefisso**: Tutti i registri iniziano con %
- **Numero**: 16 registri a 64 bit
- **Caratteristica**: Nomi rispecchiano lo scopo del registro

Struttura Gerarchica dei Registri

- **64 bit**: %rax - %r15 (estensioni complete)
- **32 bit**: %eax - %r15d (parte bassa dei 64 bit)
- **16 bit**: %ax - %r15w (parte bassa dei 32 bit)
- **8 bit**: Solo primi 4 registri (%rax-%rdx) hanno 2 registri da 8 bit:
  - High: %ah (parte alta di %ax)
  - Low: %al (parte bassa di %ax)

Registri Speciali

- **Instruction Pointer**: %rip (visibile al programmatore)
- **Flag Register**: %rflags (per istruzioni di salto condizionale)

Istruzioni Intel

Caratteristiche Generali

- **Operandi**: Prevalentemente 2 operandi
- **Risultato**: Salvato nel secondo operando (destinazione)
- **Limitazione**: Non entrambi gli operandi in memoria

Sintassi Operandi

- **Primo operando (sorgente)**:
  - Costanti: $valore
  - Registri: %registro
  - Memoria: indirizzo
- **Secondo operando (destinazione)**: NO costanti

Accesso alla Memoria

- **Sintassi**: scala(base, indice, offset)
- **Base**: Sempre necessaria
- **Scala**: Default = 1
- **Resto**: Default = 0

Suffissi per Larghezza Dati

- `b`: byte (8 bit)
- `w`: word (16 bit)
- `l`: long word (32 bit)
- `q`: quad word (64 bit)

Istruzioni Utili

- `lea`: Load Effective Address (somma indirizzi senza sovrascrivere)
- `inc`/`dec`: Incremento/decremento di 1 (risparmia bit rispetto a `add $1`)

8. Architettura ARM

Caratteristiche Generali

- **Tipo**: RISC pragmatico
- **Utilizzo**: Tablet, smartphone, sistemi embedded
- **Evoluzione**: Nata a 32 bit, nuova architettura 64 bit (non compatibile)
- **Obiettivo**: Risparmio energetico

Registri ARM

- **Numero**: 16 registri da 32 bit (R0-R15)
- **General Purpose**: Solo R0-R14 (R15 non è GP)

Registri con Nome Simbolico

- R13: `SP` (Stack Pointer)
- R14: `LR` (Link Register - return address)
- R15: `PC` + flags (Program Counter e vari flag)

Convenzioni di Chiamata ARM

- **Parametri**: R0-R3 (4 registri, non preservati)
- **Parametri extra**: Nello stack
- **Registri preservati**: R4-R11
- **Valori di ritorno**: R0 e R1
- **Non preservati**: R0-R3, R12, R9 (in alcune ABI)

Istruzioni ARM

Caratteristiche

- **Operandi**: Prevalentemente 3 argomenti
- **Limitazione**: NO operandi in memoria
- **Operando sinistro**: Sempre registro
- **Operando destro**: Immediato o registro (eventualmente shiftato)

Accesso alla Memoria

- Due operazioni dedicate:
  - Singolo registro
  - Registri multipli

Modalità di Indirizzamento

- **Base**: Indirizzo di partenza
- **Offset**: Valore immediato o indice in registro (shiftato)
- **Aggiornamento**: Opzionale del registro base
- **Tipi**:
  - **Post-indexed**: `[base], offset` (base sempre aggiornata)
  - **Pre-indexed**: `[base, offset]!` (aggiornamento opzionale con `!`)

Esecuzione Condizionale

- Tutte le istruzioni possono essere condizionali:
  - `eq`: equal
  - `ne`: not equal
  - `hs`: higher or same
  - `lo`: lower
  - `mi`: minus

Istruzioni Comuni

- **Costanti**: Prefisso `#`
- **Aritmetiche**: `add`, `adc` (con carry), `sub`, `rsb` (reverse sub)
- **Logiche**: `and`, `orr`, `eor` (xor), `bic` (and not)
- **Sottoinsiemi**: Suffissi `b` e `h` per accesso a sotto-registri

9. Toolchain di Compilazione

Fasi di Compilazione

1. **Preprocessore** → Sostituisce macro
2. **Compilatore** → C → Assembly
3. **Assembler** → Assembly → Oggetto
4. **Linker** → Oggetto → Eseguibile

GCC (GNU Compiler Collection)

Sottoprogrammi

1. `cpp`: C preprocessor
2. `cc`: C compiler
3. `as`: Assembler
4. `ld`: Linker

Opzioni di Compilazione

- `-S`: Ferma dopo compilazione (genera `.s`)
- `-c`: Ferma dopo assemblaggio (genera `.o`)
- **Default**: Eseguibile completo

File Oggetto (.o)

Segmenti

- **Header**: Dimensioni e posizioni
- **Segmento testo**: Codice macchina
- **Segmento dati**: Dati statici e dinamici
- **Tabella simboli**: Simboli → indirizzi relativi
- **Tabella rilocazione**: Istruzioni da "patchare"

Linker (ld)

Funzioni

- **Rilocazione**: Assegna indirizzi assoluti
- **Risoluzione simboli**: Collega simboli non definiti
- **Patch**: Corregge istruzioni con indirizzi assoluti

Tipi di Simboli

- **Definiti**: Con indirizzo relativo
- **Non definiti**: Definiti in altri file
- **Locali**: Non utilizzabili in altri file

Librerie

Statiche (.a)

- **Caratteristiche**: Collezione di file oggetto
- **Vantaggi**: Eseguibile autocontenuto
- **Svantaggi**: Dimensioni maggiori

Dinamiche (.so)

- **Caratteristiche**: Linking a runtime
- **Vantaggi**: Eseguibile piccolo, aggiornabilità
- **Svantaggi**: Dipendenze esterne, caricamento complesso

10. Il Processore

0.1 Ciclo di Esecuzione Istruzioni

1. **Fetch**: Prelievo istruzione dalla memoria
2. **Decode**: Lettura registri operandi
3. **Execute**: Esecuzione specifica per istruzione

0.2 Utilizzo della ALU

Tutti i tipi di istruzioni usano la ALU:

- **Memoria**: Calcolo indirizzi
- **Aritmetico-logiche**: Operazioni dirette
- **Salti condizionali**: Confronti

0.3 Datapath

Componenti principali:

- **Multiplexer**: Selettore dati (controllo flusso)
- **Unità funzionali**: ALU, registro, memoria
- **Unità di controllo**: "Direttore d'orchestra"

0.4 Temporizzazione

- **Clock**: Sincronizzazione operazioni
- **Elementi di stato**: Registri, memoria (flip-flop D-latch)
- **Metodologia**: Sensibile ai fronti di clock

0.5 Unità di Controllo ALU

Input

- **funz7**: 7 bit funzione
- **funz3**: 3 bit funzione
- **ALUOp**: 2 bit controllo
  - `00`: Somma (load/store)
  - `01`: Sottrazione (confronti)
  - `10`: Operazioni tipo R

Livelli di Controllo

1. **Unità controllo generale**: Genera ALUOp
2. **Unità controllo ALU**: Genera segnali ALU (4 bit)

11. Pipeline

0.6 Principio di Funzionamento

- **Ispirazione**: Catena di montaggio Ford
- **Concetto**: Componenti CPU lavorano in parallelo su istruzioni diverse
- **Vantaggio**: Throughput migliorato significativamente

0.7 Esempio Numerico

- **Singolo ciclo**: 4 task in 16 unità tempo $\rightarrow$ 1/4 task/tempo
- **Pipeline**: 4 task in 7 unità tempo $\rightarrow$ 4/7 task/tempo

0.8 Vantaggi RISC per Pipeline

1. **Istruzioni uniforme**: Stessa lunghezza (1 word)
2. **Operandi fissi**: Posizioni fisse facilitano decode
3. **Memoria separata**: Solo load/store accedono memoria
4. **Accessi allineati**: Un ciclo per trasferimento

0.9 Hazard (Criticità)

Hazard Strutturali

- **Causa**: Conflitti hardware
- **Soluzione**: Memoria istruzioni + memoria dati separate

Hazard sui Dati

- **Causa**: Dipendenza tra istruzioni
- **Soluzione**: Operand forwarding
- **Limitazione**: Stallo necessario per load seguita da uso

Hazard sul Controllo

- **Causa**: Salti condizionali
- **Soluzioni**:
  - **Stallo**: Attendere decisione
  - **Predizione**: Assumere esito più probabile
  - **Predittori avanzati**: Basati su storia precedente

12. Gerarchia di Memoria

0.10 Classificazione Memorie

Memoria Indirizzata Direttamente

- **Tipo**: RAM, Cache
- **Caratteristiche**: Volatile, accesso diretto CPU
- **Limite**: Spazio di indirizzamento processore

Memoria Indirizzata Indirettamente

- **Tipo**: Hard disk, SSD, USB
- **Caratteristiche**: Permanente, gestita da SO
- **Limite**: Spazio "software" illimitato

0.11 Parametri Memoria

- **Tempo di accesso**: Lettura/scrittura singola
- **Tempo di ciclo**: Tra operazioni consecutive
- **Accesso casuale**: Tempo costante (RAM)
- **Accesso sequenziale**: Tempo dipendente da posizione (dischi)

0.12 RAM (Random Access Memory)

SRAM (Static RAM)

- **Caratteristiche**: Veloce, basso consumo
- **Svantaggi**: Costosa, molti componenti per cella
- **Utilizzo**: Cache

DRAM (Dynamic RAM)

- **Caratteristiche**: Economica, alta densità
- **Svantaggi**: Refresh periodico, consumi elevati
- **Utilizzo**: Memoria principale

Miglioramenti DRAM

- **FPM**: Fast Page Mode (accessi consecutivi)
- **SDRAM**: Sincrona (buffer + clock)
- **DDR**: Double Data Rate (fronti positivi e negativi)

0.13 Principi di Località

- **Spaziale**: Accesso a locazioni vicine
- **Temporale**: Riutilizzo locazioni recenti

0.14 Metriche Prestazioni

- **Hit rate**: Probabilità di trovare dato nel livello superiore
- **Miss rate**: 1− Hit rate
- **Tempo di hit**: Accesso al livello superiore
- **Penalità di miss**: Tempo trasferimento + tempo di hit

13. Cache

0.15 Tipi di Cache

Mappatura Diretta

- **Principio**: Ogni indirizzo memoria → posizione cache fissa
- **Calcolo**: Modulo numero blocchi cache
- **Componenti**: Tag + bit validità
- **Vantaggio**: Semplice
- **Svantaggio**: Conflitti frequenti

Completamente Associativa

- **Principio**: Qualsiasi indirizzo → qualsiasi blocco
- **Ricerca**: Parallela su tutti i blocchi
- **Vantaggio**: Flessibilità massima
- **Svantaggio**: Costosa (n comparatori)

Set-Associativa

- **Principio**: Via di mezzo tra le precedenti
- **Struttura**: n vie per set
- **Ricerca**: Parallela all'interno del set
- **Compromesso**: Flessibilità vs complessità

0.16 Gestione Scritture

Write-Through

- **Principio**: Scrittura immediata in memoria principale
- **Vantaggi**: Consistenza garantita
- **Svantaggi**: Scritture costose

Write-Back

- **Principio**: Scrittura solo in cache, update ritardato
- **Vantaggi**: Efficiente per scritture multiple
- **Svantaggi**: Complessità gestione consistenza

0.17 Politiche di Rimpiazzo

- **FIFO**: First In First Out
- **LRU**: Least Recently Used (richiede bit aggiuntivi)

14. Dispositivi I/O

0.18 Classificazione

- **Comportamento**: Read/Write
- **Partner**: Tipo di comunicazione
- **Velocità**: Latenza vs throughput

0.19 Tipi di Bus

Bus Processore/Memoria

- **Caratteristiche**: Specializzato, corto, veloce
- **Utilizzo**: Collegamento diretto CPU-RAM

Bus I/O

- **Caratteristiche**: Lungo, eterogeneo
- **Utilizzo**: Periferiche diverse

0.20 Sincronizzazione Bus

Bus Sincrono

- **Controllo**: Clock comune
- **Vantaggi**: Semplice, veloce
- **Svantaggi**: Tutte periferiche stessa velocità

Bus Asincrono

- **Controllo**: Segnali di handshake
- **Vantaggi**: Periferiche diverse velocità
- **Svantaggi**: Circuiteria complessa

0.21 Comunicazione con I/O

Memory-Mapped I/O

- **Principio**: Locazioni memoria speciali per controllo
- **Accesso**: Solo sistema operativo

Istruzioni Speciali

- **Principio**: Istruzioni dedicate I/O
- **Utilizzo**: Alternativa a memory-mapped

0.22 Gestione Trasferimenti

Polling (Attesa Attiva)

- **Principio**: Controllo continuo stato periferica
- **Vantaggi**: Semplice
- **Svantaggi**: Spreco risorse CPU

Interrupt

- **Principio**: Periferica segnala quando pronta
- **Vantaggi**: CPU libera per altro
- **Svantaggi**: Hardware speciale necessario

15. Eccezioni e Interrupt

0.23 Tipi di Eccezioni

Interrupt

- **Causa**: Eventi esterni (periferiche)
- **Caratteristiche**: Asincrone, ripresa programma
- **Gestione**: Tra istruzioni

Trap (Eccezioni)

- **Causa**: Eventi interni (overflow, errori)
- **Caratteristiche**: Sincrone
- **Gestione**: Riesecuzione o abort

Environment Call / Break

- **ecall**: Richiesta servizio sistema
- **ebreak**: Debug/diagnostica

0.24 Gestione Eccezioni

Metodi di Gestione

1. **Salto diretto**: Routine unica analizza causa
2. **Vettore interruzioni**: Array di routine specifiche

Salvataggio Stato

- **Dove**: Stack, registri ausiliari, registri speciali
- **RISC-V**: Stack + Control Status Register (CSR)

Registri CSR RISC-V

- **mepc**: Salvataggio PC
- **mcause**: Causa eccezione
- **mtvec**: Indirizzo routine gestione
- **mstatus**: Stato macchina

0.25 Eccezioni RISC-V

- **Istruzione non valida**
- **Malfunzionamenti hardware**
- **Gestione**: Come hazard sul controllo