INTRODUZIONE AI SISTEMI DI ELABORAZIONE

Un SISTEMA DI ELABORAZIONE è un sistema che:

1 - Riceve / invia comandi e informazioni dall'esterno.
2 - Esegue operazioni di elaborazione.

I sistemi si basano su TRE IDEE CHIAVE:

1 - Rappresentazione Informazioni.
2 - Definizioni di operazioni e algoritmi.
3 - Memorizzazione dei dati.

ARCHITETTURA:

VON-NEUMANN: Unica memoria.
HARVARD: Memoria separata per dati e istruzioni.

! Nei calcolatori in generale si usa Vom-Neumann, però in alcune tipo di cache è possibile trovare la separazione dello spazio dedicato a istruzioni e dati (Hazvard).

I CIRCUITI INTEGRATI (IC)

Sono composti da: SILICIO, PACKAGE, PIN.
Ad un singolo IC corrispondono MILIONI di Transistor.

! COME È FATTO? (cit. 0)

1 - Da un wafer di silicio estraggo i die (core del IC) => Operazione di SLICING
2 - Li inserisco (i die) nel package
3 - Connetto i die e i loro pin con i pin del package => BONDING

Die, Core, dim. 2 cm^2
Package, Alloggio del die ☺, dim 20 cm^2.
↳ Posso inserire + die in 1 Package.

LEGGE di MOORE !

Il numero di Transistor integrati in un circuito raddoppia ogni 18/24 mesi.
=> Ciò permette di creare dei SYSTEM ON A CHIP (SoC), che integrano 1 o + processori, memoria, periferici in 1 singolo circuito. (Es: A14 Apple, 11.8 mld Transistor, Tecnologia 5nm)

MICROPROCESSORI

Basato su un solo IC, primo fu l'Intel 4004 (1971). (Capoprogetto l'italiano Federico Faggin.)
Un processore (CPU) esegue istruzioni.
Tipi di processori:

- CISC, set istruzioni ampio (>100), diversi colpi di clock per istruzione.
- RISC, (Reduced) set istruzioni ridotte (max 32 es.), elevato num. di registri, architettura pipeline
  1 o + clock per istruzione. (Es l'architettura MIPS è per microprocessori RISC)
- SUPERSCALARI: set istruzioni RISC, 1 clock per N istruzioni
- MULTICORE: Per ottimizzare il + possibile, controllando il consumo di potenza, si sono introdotti + processori in 1 dispositivo.

INSTRUCTION SET ARCHITECTURE (ISA)

I processori sono organizzati in famiglie, a livello software il codice sviluppato per il processore X di famiglia Y, può essere eseguito su TUTTI i processori della stessa famiglia successivi.

↳ L'ISA rappresenta SOLO le informazioni utili al programmatore (non specifiche di progetto per esempio)

MICROCONTROLLORI (MCU)

Unico IC, simile ai microprocessori ma integra altri moduli (I/O, memorie), orientati al special-purpose.

MCU = CPU + MEM (ROM, RAM, ..) + I/O + Contatori.

↳ Importanti anche consumi, dimensioni, affidabilità, etc. (il perché guarda su)

GPU (Graphic Processing Unit)

Utilizzate per applicazioni grafiche e scientifiche.

↳! Lavorano in parallelo su dati vettoriali!

Composta da Processing Element (PE, moduli elaborazione + mem), che eseguono in // la == operazione.
+ Veloci elaborazioni su immagini, matrici, etc.

SISTEMI SPECIAL PURPOSE

Progettati per eseguire una singola operazione/problema SPECIFICO.

- Soluzione SW: Compro un MCU, scrivo il software. (Software) => Soluzione meno efficiente in velocità, MA meno costosa, + flessibile (Adatta a piccoli progetti o prototipi)

- Soluzione HW: Progettazione di una Application Specific Integrated Circuit (ASIC), e relativo codice.
  => Prestazioni elevate, Ingombro minore, Consumo Ridotto, ~~Flessibilità~~ (dovrai riprogettare la scheda)
  (Costo progetto elevato, ma costo schede singole in produzione minore)
  => Utile per produzioni elevate (>1 MLN)

INTRODUZIONE AL LINGUAGGIO MIPS

Il processore è il modulo principale di un calcolatore, esegue istruzioni e interagisce con le interfacce

L'esecuzione di un'istruzione prevede 2 fasi:

1-FETCH, il codice viene letto dalla memoria

→ NEL MIPS (Ogni passo = 1 clock)

1.1, Invio l'indirizzo dell'istruzione (che è contenuto nel registro PC) all'Address (IAR) WE (MemWrite) nella C.U. viene settato a ϕ, leggo quindi il contenuto in memoria all'indirizzo specificato, e salvo il contenuto (l'istruzione) in RD.

1.2, Invio valore contenuto in RD, in IR e faccio PC = PC + 4 (Setto PcWrite=1 e faccio una ADD con ALUSrcB=01)

2, EXECUTE, il codice viene decodificato ed eseguito. Comporta l'accesso a 1 o + operandi.

NEL MIPS:

- Invio alla C.U. i bit 31:26 (Op Code) e i bit 5:0 (Funct Code), così la C.U. abiliterà i giusti segnali per la funzione (altri vengono settati a ϕ oppure DONTCARE)
- Il resto dipende dal tipo di istruzione.

REGISTRI

Tempo accesso Memoria >> Tempo elaborazione dati per la CPU ⇒ Accesso in memoria = BOTTLENECK

ALLORA, ∃ Registri ∈ CPU, celle di memoria veloci che contengono gli operandi e i risultati.

→ Ricorda, la CPU richiede il dato in memoria, a sua insaputa viene controllato se il dato è presente in cache (prima L1, poi L2 ed L3), se presente ed è 'aggiornato' lo manda alla CPU, altrimenti lo si richiede in memoria RAM e si salva anche in cache. (→ CACHE HIT, → CACHE MISS)

Alla fine qualunque dato va a finire in almeno 1 registro (non per forza dei 32 disponibili al programmatore...)

ASSEMBLY (in MIPS)

Rappresentazione simbolica/mnemonica del linguaggio macchina.
Prevede SOLO semplici istruzioni dati.

VANTAGGI:
- Visibilità diretta HW (registri etc.)
- Possibilità di sfruttare al 100% l'HW
- Ottimizzazione Prestazioni

SVANTAGGI:
- Mancanza portabilità programmi
- Maggior costo di sviluppo (anche di Tempo)
- Codice + lungo e + complicato da leggere rispetto a linguaggi alto-livello

---

MIPS

Progettato secondo 4 principi:

1 - Semplicità per favorire la regolarità (e v.v.):
Formato delle istruzioni unico (o quasi, 2 reg sorg e 1 dest opp 1 immediato).
Ciò favorisce l'HW nella decodifica dell'istruzione ⇒ decode + veloce

2 - Rendere il caso comune + veloce:
Mips include solo le istruzioni + semplici e usate. Le + complesse sono in realtà eseguite attraverso + istruzioni semplici ⇒ MIPS È RISC, 8086 È CISC

!! RICORDA, MIPS ha 32 registri da 32 bit.

3 - Più piccolo ⇒ Più veloce

REGISTRI:

Nome	Uso
$0	Costante che contiene 0
$at	
$v0-v1	Per valori di ritorno (o per syscalle)
$t0-t9	Registri Temp (il chiamante si assicura che non contenyamo cose importanti, altrimenti li mette in $s0-$s7 opp. stack
$s0-s7	Registri save (una funzione chiamata se vuole usarli deve prima salvare il loro contenuto nello STACK)
$k0-k1	
$gp	
$sp	Puntatore a stack (TIPS: Aggiungi un immediato negativo di byte ad $sp, poi fai le sw, in modo speculare poi fai lw e poi add imm positivo)
$fp	
$ra	Indirizzo di ritorno (della funct. chiamante, il PC+4 rispetto alla chiamata)
$a0-a3	Argomenti da passare alla funz.

4) Buon design richiede compromessi:
C'è flessibilità nel Tipo di istruzioni e operandi che esse richiedono, lo vedrai in dettaglio Tra 1 pagina nel capitolo

ACCESSO in MEMORIA (Mips)

∀ programma ∃ molti dati ⇒ 32 registri mips non possono gestire tutto.
I dati vengono quindi prelevati dalla MEMORIA e salvati nei registri.
→ MIPS è byte-addressed e l'indirizzo di una parola in memoria è sempre multiplo di 4
* Lo sempre per il motivo che ∀ word contiene 32 bit = 4 byte.

↳ OPERAZIONI:

lw $s0, 8($t1) → Salva in $s0 la parola contenuta in memoria all'indirizzo ($t1 + 8), con $t1 che contiene l'indirizzo iniziale (offset)

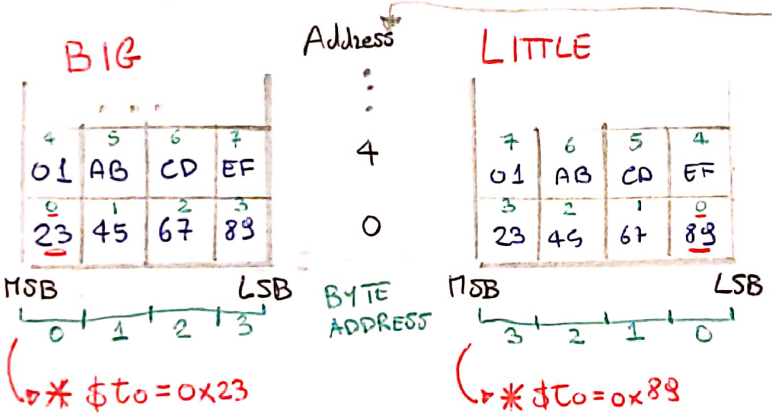
sw $s0, 8($t1) → Salva in memoria all'indirizzo ($t1 + 8) il contenuto del registro $s0.

lb, sb → == a lw, sw ma con un solo byte!

* WORD ALIGNMENT, gli indirizzi specificati per lw/sw devono essere multipli di 4
↳ lw $s0, 7($t0) ⇒ NO! ILLEGAL INSTRUCTION

BIG-ENDIAN & LITTLE ENDIAN

L'indirizzo è sempre multiplo di 4, ma come viene salvata la word? Es: word = 0x23456789



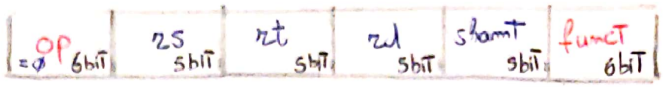
word in 0 = 0x23456789
word in 4 = 0x01ABCDEF

Se Faccio:
① li $s0, 0
② lb $t0, 0($s0)
Cosa avrò in $t0? *

LINGUAGGIO MACCHINA e ISTRUZIONI (4° principio Mips)

Esistono 3 Tipologie di istruzioni, Tutte da 32 bit (1° principio):

- **R-TYPE**, operando registri

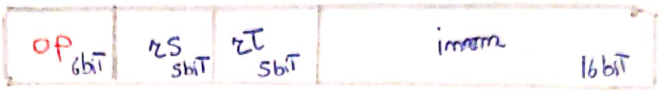


↳ max 2^6 R-Type Istruzioni

  - rs, rt registri sorgente
  - rd // destinazione
  - op, operation code (sempre =0 in R-TYPE)
  - funct, specifica la precisa R-TYPE
  - shamt specifica il numero di bit da shiftare ⇒ 5bit ⇒ $2^5 = 32$ bit max di shift (32 bit nei registri)

  ↳ es: add $s0, $s1, $s2 ⇒ 000000 | 10001 | 10010 | 10000 | 00000 | 100 000

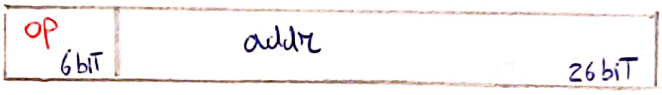
- **I-TYPE**, operazioni con immediati



  - rs reg. sorg
  - rt reg dest
  - imm 16 bit complemento a 2
  - op, indica le diverse funzioni I-TYPE

  ↳ Essendo le operazioni su registri a 32bit, faccio ESTENSIONE DEL SEGNO, copiando l'ultimo bit sugli altri 16

- **J-TYPE**, salti



  - op code
  - address, su 26 bit (ma non salvo i primi 2 bit indirizzo perché le istruzioni sono ogni 4 byte, non sono quindi utili) ⇒

    ↳ riesco a fare un salto del tipo $PC = PC \left[\pm 2^{27}\right]$ ⇝ 26 + 2 bit = 2^{28} bit di salto, ma essendo complemento a 2 $\pm 2^{27}$ e non $+2^{28}$

INTERPRETAZIONE DEL CODICE

Solitamente le istruzioni sono salvate dall'indirizzo 0x00400000, ie PC si salva questi indirizzi e viene incrementato.

SEQUENZA:

1) Fetch Istruzione
2) Analizza OPCODE (bit 31:26) per fare le parse dei restanti bit:

```
if (opcode == 0) {
    // R-Type Instr.
      see (funct) // to understand operation
} else {
    // I or J-Type
      see (opcode) //
}
```