ANNO ACCADEMICO 2024/2025

# Architettura degli Elaboratori II

## Teoria

Altair’s Notes

UNIVERSITÀ DI TORINO

DIPARTIMENTO DI INFORMATICA

Capitolo 1 Concetti di Base Pagina 5

1.1 Introduzione 5
Tassonomia delle architetture — 6 • Alcuni concetti fondamentali — 6

1.2 Una semplice macchina RISC 7
MIPS - Versione monociclo — 8 • Banco dei registri — 10 • Una semplice Control Unit — 11 • L'esecuzione di un'istruzione — 12

1.3 Dal monociclo al multiciclo 12
MIPS - Versione multiciclo — 12

Capitolo 2 Instruction Level Parallelism (ILP) Pagina 15

Capitolo 3 Caching Pagina 17

Capitolo 4 Architetture Parallele Pagina 19

Capitolo 5 Quantum Computing Pagina 21

Capitolo 6 GPU Pagina 23

# Premessa

## Licenza




## Formato utilizzato

**Box di "Concetto sbagliato":**

**Concetto sbagliato 0.1: Testo del concetto sbagliato**

Testo contente il concetto giusto.

**Box di "Corollario":**

**Corollario 0.0.1** Nome del corollario

Testo del corollario. Per corollario si intende una definizione minore, legata a un'altra definizione.

**Box di "Definizione":**

**Definizione 0.0.1: Nome delle definizione**

Testo della definizione.

**Box di "Domanda":**

**Domanda 0.1**

Testo della domanda. Le domande sono spesso utilizzate per far riflettere sulle definizioni o sui concetti.

**Box di "Esempio":**

**Esempio 0.0.1** (Nome dell'esempio)

Testo dell'esempio. Gli esempi sono tratti dalle slides del corso.

**Box di "Note":**

**Note:-**

Testo della nota. Le note sono spesso utilizzate per chiarire concetti o per dare informazioni aggiuntive.

**Box di "Osservazioni":**

**Osservazioni 0.0.1**

Testo delle osservazioni. Le osservazioni sono spesso utilizzate per chiarire concetti o per dare informazioni aggiuntive. A differenza delle note le osservazioni sono più specifiche.

# 1

# Concetti di Base

## 1.1 Introduzione

In questo corso verrà studiata l'architettura interna e il funzionamento dei processori moderni (con riferimento a cache e RAM).

**Note:-**

Lo scopo del corso è quello di spiegare il passaggio al multi-core, subito dopo la "Rivoluzione RISC".

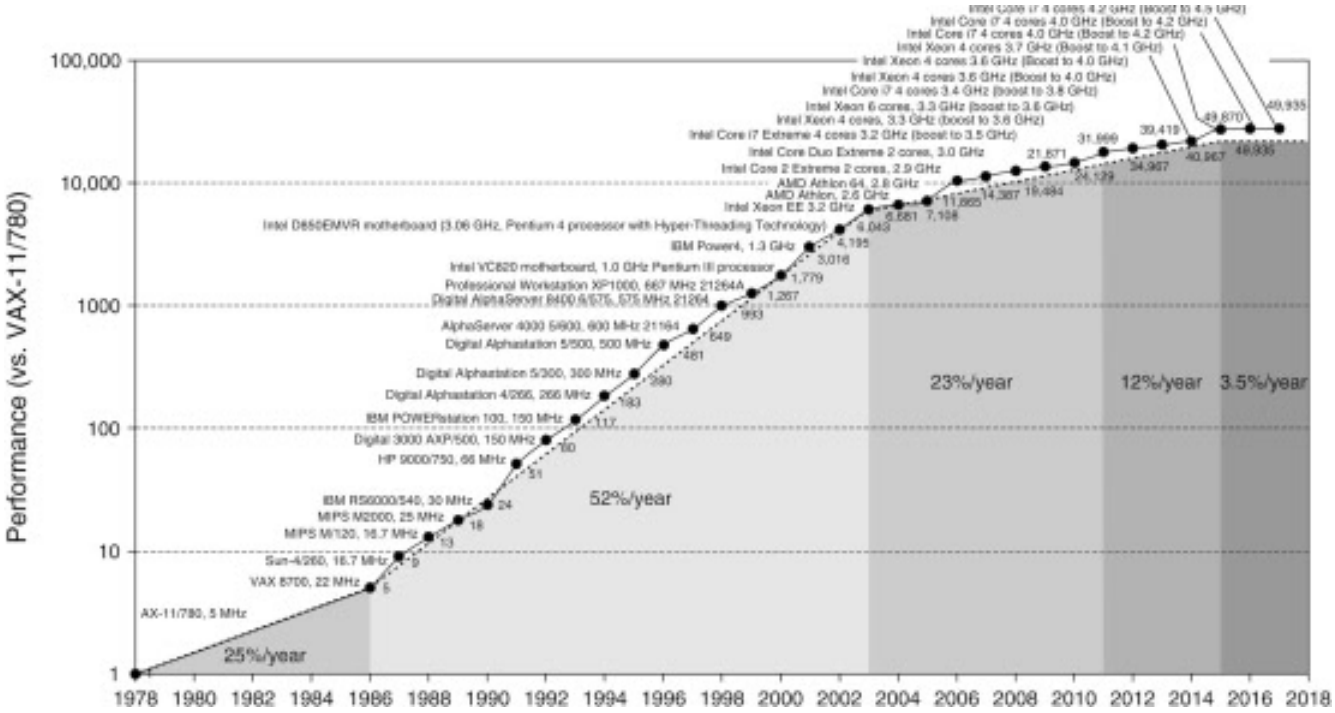


Figure 1.1: Nel 1986 ha inizio la "Rivoluzione RISC", mentre all'inizio degli anni 2000 si inizia a sfruttare l'idea di avere più "core".

### 1.1.1 Tassonomia delle architetture

Il contenuto del corso può essere descritto dalla "Tassonomia di Flynn".

**Definizione 1.1.1: Tassonomia di Flynn**

La Tassonomia di Flynn organizza i vari tipi di processori in base a determinate caratteristiche che verranno approfondite in questo corso.

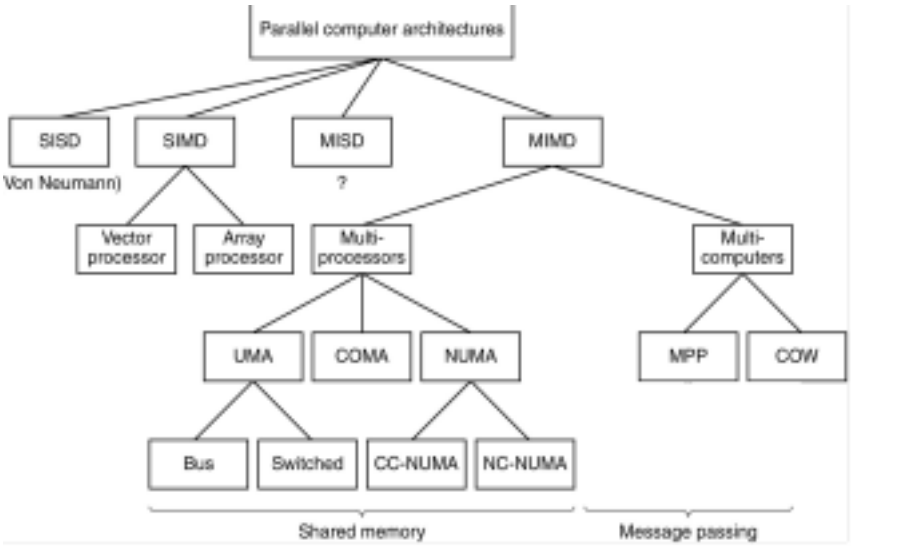


Figure 1.2: La Tassonomia di Flynn

### 1.1.2 Alcuni concetti fondamentali

**Definizione 1.1.2: Micoarchitettura**

L'architettura interna di un processore: com'è fatto a partire dal suo datapath.

**Corollario 1.1.1** Datapath

Il percorso che compiono le istruzioni all'interno del processore per venire eseguite.

**Note:-**

Diversi tipi di istruzioni percorrono diverse parti del datapath per venire eseguite.

**Definizione 1.1.3: ISA**

L'Instruction Set Architetture (ISA) è l'insieme di istruzioni macchina di un processore.

**Note:-**

Due processori possono avere lo stesso ISA, ma microarchitetture diverse (e.g. AMD e Intel).

## 1.2 Una semplice macchina RISC

**Domanda 1.1**

Qual è la differenza tra un processore a 32 bit e un processore a 64 bit?

**Risposta:** il processore a 64 bit manipola in maniera naturale informazione scritta con 64 bit e il processore a 32 bit manipola in maniera naturale informazione scritta con 32 bit.

**Caratteristiche fondamentali dell'architettura RISC:**

⇒ le istruzioni hanno tutte la stessa lunghezza (o a 32 bit o a 64 bit);

⇒ le istruzioni sono semplici;

⇒ la Control Unit è semplice (poche porte logiche, quindi frequenze di clock più elevate).

**Note:-**

Ciò che verrà descritto in questa sezione è una versione semplificata di MIPS, la prima macchina RISC. Si considerano 32 registri a 32 bit e si ignorano le operazioni floating point.

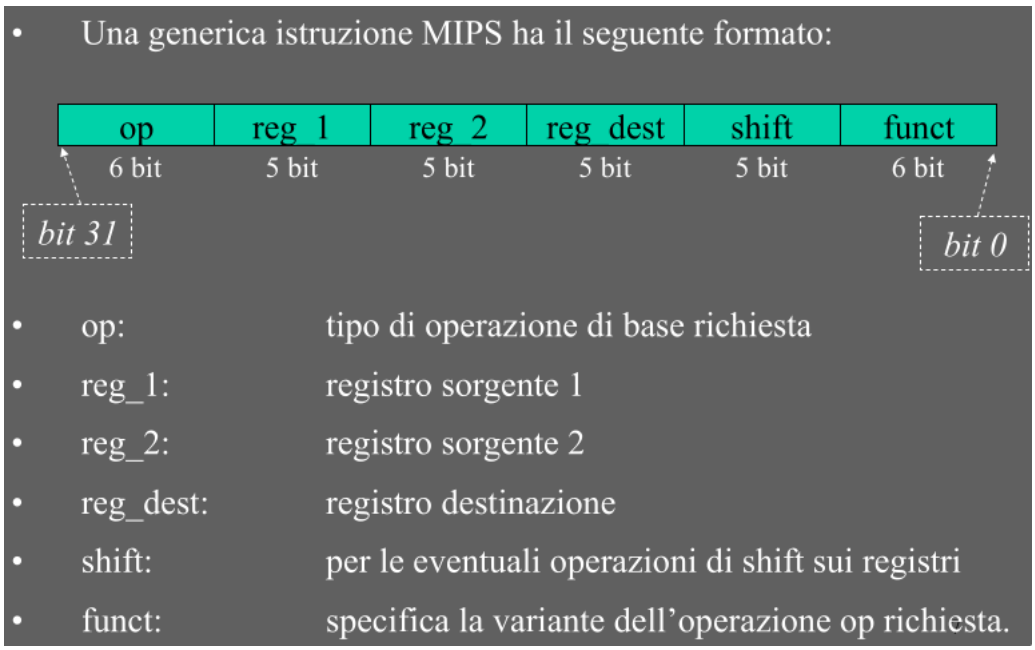


Figure 1.3: Istruzione MIPS

**Definizione 1.2.1: Istruzioni di tipo-R**

Le istruzioni di tipo-R usano due registri e restituiscono il risultato a un terzo registro. La convenzione prevede che il campo OP sia 0. L'operazione specifica si trova nel campo func.

**Note:-**

Solitamente si usa la lettera D quando si parla di dati interi (DADD, DSUB, etc.), F per i floating point.

**Definizione 1.2.2: Istruzioni di tipo-I**

Le istruzioni di tipo-I usano un valore immediato. La convenzione prevede che il campo op sia 8.

**Definizione 1.2.3: LOAD e STORE**

La LOAD carica in un registro un valore che si trova in memoria (op = 35). La STORE salva in memoria il valore di un registro (op = 43).

**Definizione 1.2.4: Salti condizionati (BRANCH)**

Salta solo se si verifica una determinata condizione (op = 5).

**Definizione 1.2.5: Salti incondizionati (JUMP)**

Salta sempre (op = 4).

### 1.2.1 MIPS - Versione monociclo

Generalmente i primi due passi di ogni istruzione sono:

1. usa il Program Counter (PC) per prelevare dalla "memoria di istruzioni[1]" la prossima istruzione da eseguire;
2. Decodifica l'istruzione e contemporaneamente legge i registri.

**Note:-**

I passi successivi dipendono dal tipo di istruzione (tutte usano la ALU.

⇒ LOAD e STORE accedono alla memoria dati e nel caso di LOAD viene aggiornato un registro;

⇒ le istruzioni logico-aritmetiche aggiornano un registro;

⇒ le istruzioni di salto possono alterare il valore di PC.

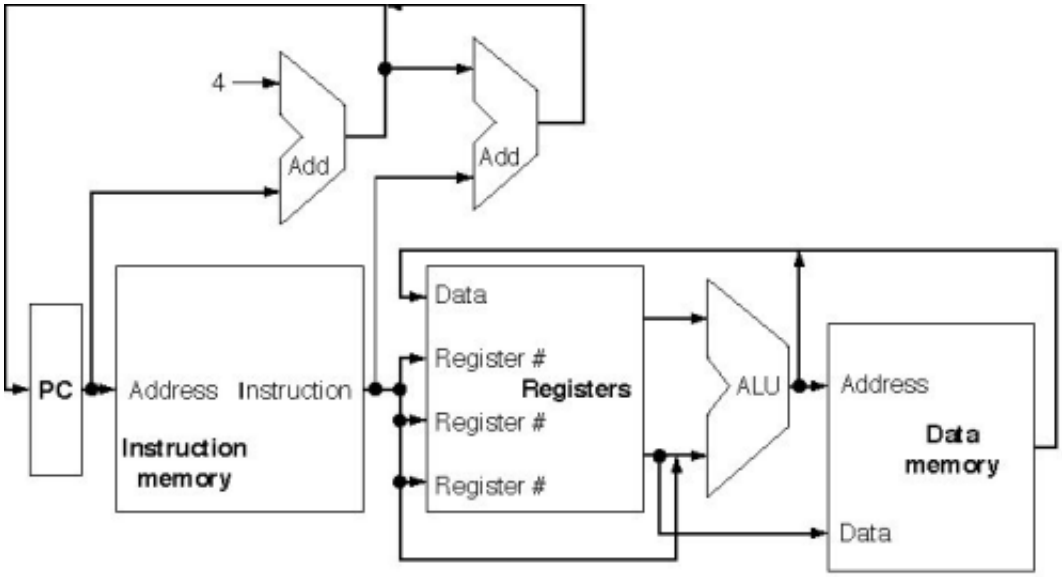


Figure 1.4: Schema ad alto livello del datapath MIPS

**Note:-**

Il fluire delle informazioni nel datapath deve essere controllato da una "Control Unit".

[1] Cache di primo livello.

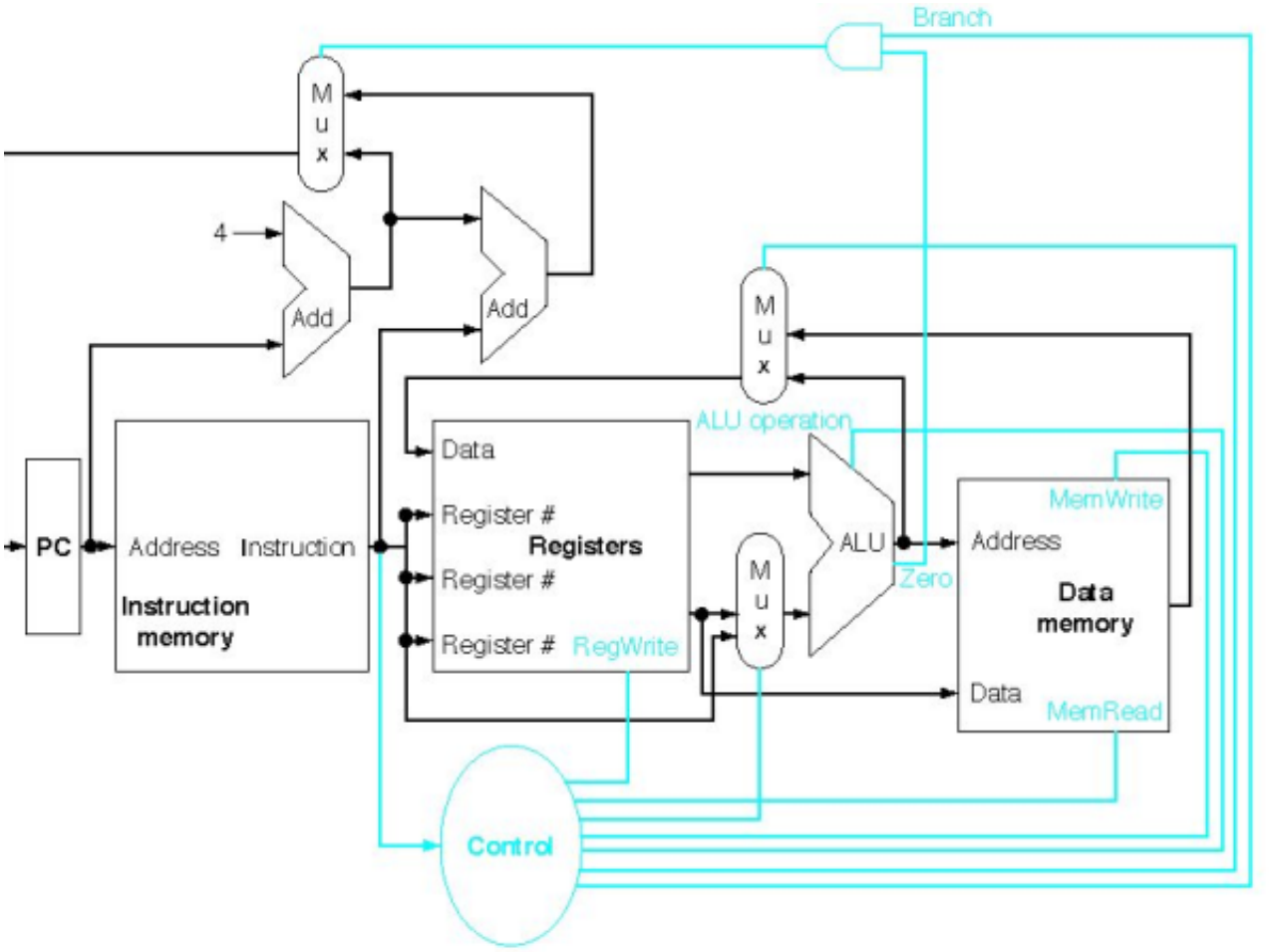


Figure 1.5: Versione modificata del MIPS con una Control Unit

**Note:-**

L'esecuzione di ciascuna istruzione può avvenire in un unico ciclo di clock, purché sia stato adeguatamente dimensionato.

**Per capire come funziona il datapath di una macchina monociclo si osserva che esso è composto da due tipi di elementi logici:**

⇒ gli elementi di *stato*;

⇒ gli elementi di tipo *combinatorio*.

**Definizione 1.2.6: Elementi di stato**

Gli elementi di stato sono quelli in grado di memorizzare uno *stato* (e.g. flip flop, registri e memorie). Un elemento di stato possiede almeno 2 ingressi e un'uscita. Gli ingressi richiedono:

⇒ il valore da scrivere nell'elemento;

⇒ il clock per determinare quando scriverlo.

Il dato presente in uscita è sempre quello scritto in un ciclo di clock precedente.

**Note:-**

Solitamente esiste un terzo ingresso "di controllo" che stabilisce se l'elemento di stato può effettivamente memorizzare l'input.

**Definizione 1.2.7: Elementi combinatori**

Gli elementi combinatori sono quelli in cui le uscite dipendono solamente dai valori d'ingresso in un dato istante (e.g. ALU e Multiplexer).

### 1.2.2 Banco dei registri

**Definizione 1.2.8: Banco dei registri**

Nelle immagini precedenti i registri della CPU sono rappresentati da un'unità funzionale detta *register file* (o banco dei registri). Essa è un'unità di memoria molto piccola e veloce.

**Note:-**

Si può accedere a ognuno dei 32 registri (da 0 a 31) specificando il suo indirizzo. Ogni registro può essere letto o scritto.

**Operazione di scrittura:** quando il segnale di controllo (RegWrite) è a 1, il valore proveniente dalla ALU o dalla Data Memory e presente in input in *DST data* viene memorizzato nel registro di destinazione specificato da *DST addr*.

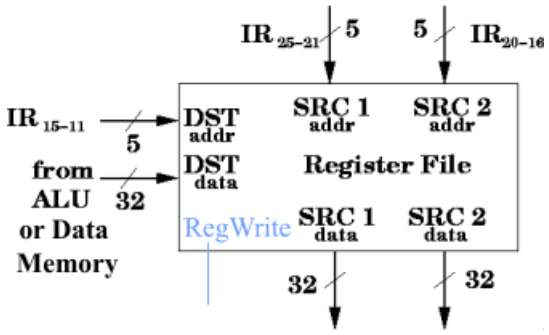


Figure 1.6: Operazione di scrittura

**Operazione di lettura:** le letture sono immediate. In qualunque momento alle uscite *SRC1 data* e *SRC2 data* è presente il contenuto dei registri i cui numeri sono specificati da *SRC1 addr* e *SRC2 addr*.

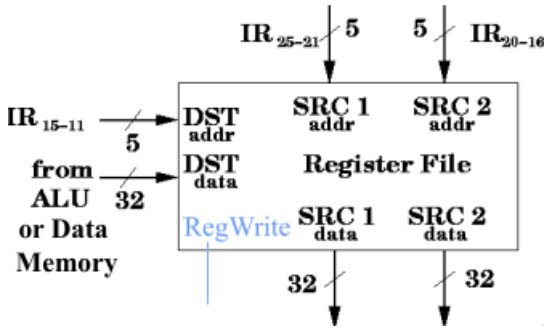


Figure 1.7: Operazione di lettura

### 1.2.3 Una semplice Control Unit

**Definizione 1.2.9: Control Unit**

La Control Unit riceve in input i 6 bit del campo op dell'istruzione e deve generare in output i segnali per comandare:

⇒ la scrittura dei registri;

⇒ la lettura/scrittura della memoria dati (MemRead/MemWrite);

⇒ i Multiplexer che selezionano gli input da usare;

⇒ la ALU (ALUOp) che deve eseguire ciascuna operazione aritmetico-logica appropriata per la specifica istruzione in esecuzione.

**Corollario 1.2.1** Segnale ALUOp

Il segnale ALUOp dipende:

⇒ dal tipo di istruzione in esecuzione, specificato nel campo op;

⇒ dalla specifica operazione da eseguire, determinata dal campo funct.

**Esempio 1.2.1** (Istruzioni di tipo-R)

⇒ Se op == load || op == store allora la ALU deve eseguire una *somma*;

⇒ se op == beq allora la ALU deve eseguire una *sottrazione* (per controllare se il risultato è 0);

⇒ Se op == tipo-R allora è il campo *func* a stabilire l'operazione che deve eseguire la ALU.

**Definizione 1.2.10: ALU Control**

Parte della Control Unit che indica le operazioni da eseguire.

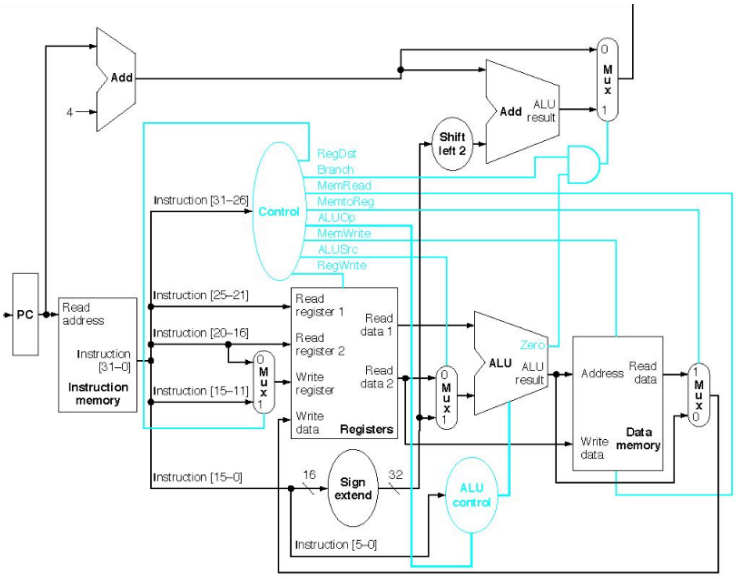


Figure 1.8: Versione modificata del MIPS la ALU Control

### 1.2.4 L'esecuzione di un'istruzione

1. Il contenuto del PC viene utilizzato per indirizzare la instruction memory e produrre in output l'istruzione da eseguire;
2. Il campo *op* dell'istruzione viene mandato in input alla Control Unit, mentre i campi *reg_1* e *reg_2* vengono usati per indirizzare il register file;
3. La CU produce in output i nove segnali di controllo relativi ad una operazione di tipo-R, ed in particolare ALUOp=10, che, dati in input alla ALU control insieme al campo *funct* dell'istruzione stabiliscono l'effettiva operazione di tipo-R che deve eseguire la ALU;
4. Nel frattempo, il register file produce in output i valori dei due registri *reg_1* e *reg_2*, mentre il segnale ALUSrc=0 stabilisce che il secondo input della ALU deve provenire dal secondo output del register file;
5. La ALU produce in output il risultato della computazione, che attraverso il segnale *MemtoReg* viene presentato in input al register file (Write data).

**Note:-**

Tutti questi passi devono essere eseguiti nello stesso ciclo di clock.

## 1.3 Dal monociclo al multiciclo

Le macchine monociclo sono estremamente inefficienti perché portano a sprecare cicli di clock nel caso di operazioni semplici. Per trasformare una macchina da monociclo in multiciclo si scompone l'esecuzione di ciascuna istruzione in un insieme di passi ognuno dei quali eseguibile in un singolo ciclo di clock (per cui ogni passo deve richiedere più o meno lo stesso tempo di esecuzione, perché la durata del clock va commisurata alla durata del passo più lungo).

**Note:-**

Le parole "passo", "fase", "stadio" e "stage" sono intercambiabili.

### 1.3.1 MIPS - Versione multiciclo

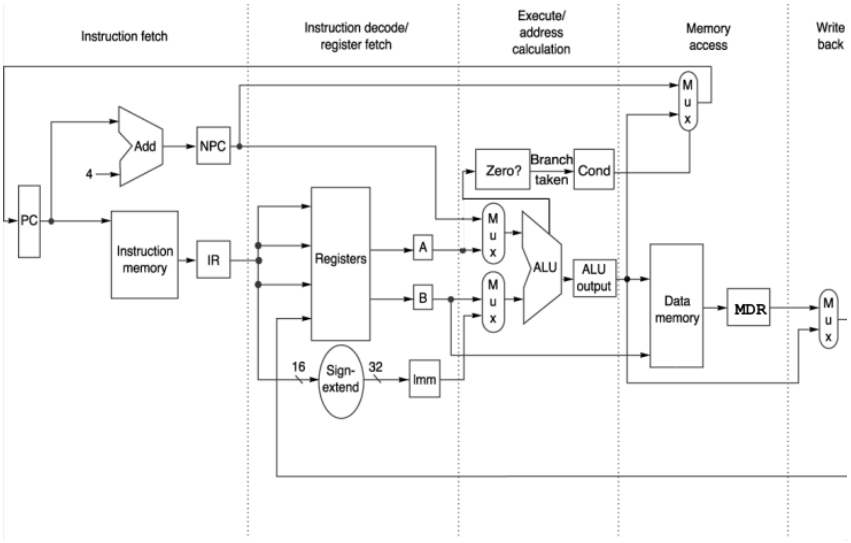


Figure 1.9: Il Datapath suddiviso nelle 5 fasi standard

**Note:-**

Instruction Memory e Data Memory corrispondono alla cache di primo livello (L1).

1. *Instruction Fetch*: il PC indirizza le istruzioni (IR) e contemporaneamente incrementa il PC di 4 (salvato in NPC che verrà instradato alla ALU per eventuali istruzioni di salto). Da notare che la ALU non viene utilizzata quindi si potrebbe pensare di usarla al posto dell'adder, ma quando si introdurrà il concetto di *pipeline* si vedrà che la ALU sarà occupata dalla fase di esecuzione di un'altra istruzione;
2. *Instruction Decode / Register Fetch*: vengono sempre prelevati i valori dei 2 registri di destinazione (anche se l'istruzione non è di tipo-R) e messi in 2 *registri nascosti* A e B. Inoltre il registro Imm viene memorizzato nell'eventualità che sia un'istruzione di tipo-I. Oltre a ciò si memorizza un'eventuale salto (usando un adder apposta);
3. *Execute / Address Calculation*: la ALU fa i suoi calcoli in base al tipo di istruzione;
4. *Memory Access*: nei casi di load e store si accede alla memoria. Per la load si preleva il risultato della fase di esecuzione e viene salvato nel registro MDR. Se si sta eseguendo un'istruzione di tipo-R o di tipo-I questa fase viene saltata;
5. *Write Back:* il risultato viene scritto sul registro di destinazione. Non è necessaria per le store.

**Note:-**

Non necessariamente l'esecuzione di un'istruzione racchiude tutte le fasi. Ogni fase avviene in un ciclo di clock.

**Definizione 1.3.1: Registro "nascosto"**

I risultati di ciascuna fase sono memorizzati da registri "nascosti" in cui viene salvato il risultato dell'output di una fase in modo che diventi l'output della fase successiva. Non sono indirizzabili e servono solo a velocizzare l'esecuzione delle istruzioni.

**Domanda 1.2**

Quali fasi sono coinvolte nell'esecuzione di una "load"?

**Risposta:** tutte.

**Domanda 1.3**

Quali fasi sono coinvolte nell'esecuzione di un'istruzione di tipo-R?

**Risposta:** 4, perché non si deve accedere alla memoria.

**Domanda 1.4**

Quali fasi sono coinvolte nell'esecuzione di una "store"?

**Risposta:** 4, perché non esiste la frase di Write Back.

**Domanda 1.5**

Quali fasi sono coinvolte nell'esecuzione di un'istruzione di salto?

**Risposta:** 3, Instruction Fetch, Instruction Decode e Address Calculation.

**Note:-**

Tutto questo è più efficiente della versione monociclo perché ogni fase usa circa $\frac{1}{5}$ del ciclo di clock e non sempre si devono attraversare tutte le fasi.

# 2

# Instruction Level Parallelism (ILP)

# 3
# Caching

# 4

# Architetture Parallele

# 5

# Quantum Computing

# 6
# GPU