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".

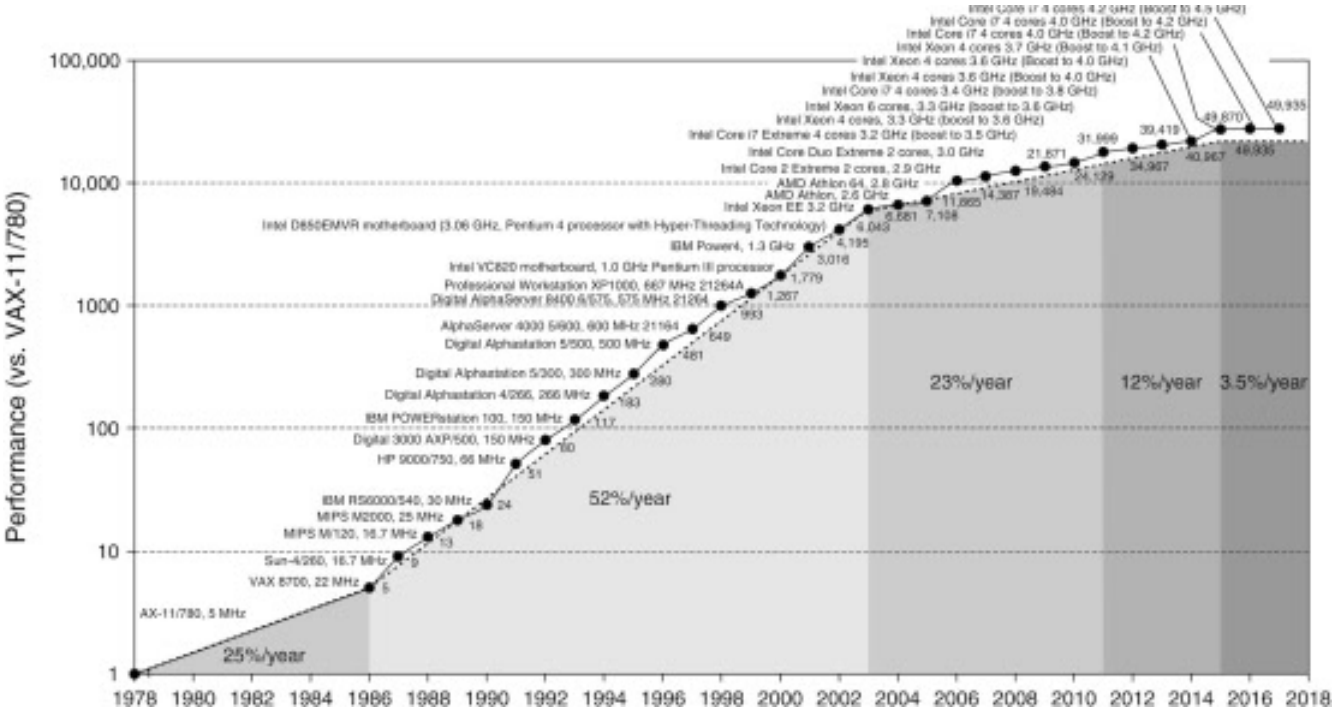

Figure 1.1: Nel 1986 ha inizio la "Rivoluzione RISC", mentre all'inizio degli anni 2000 si inizia a sfruttare l'idea di avere più "core".

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.

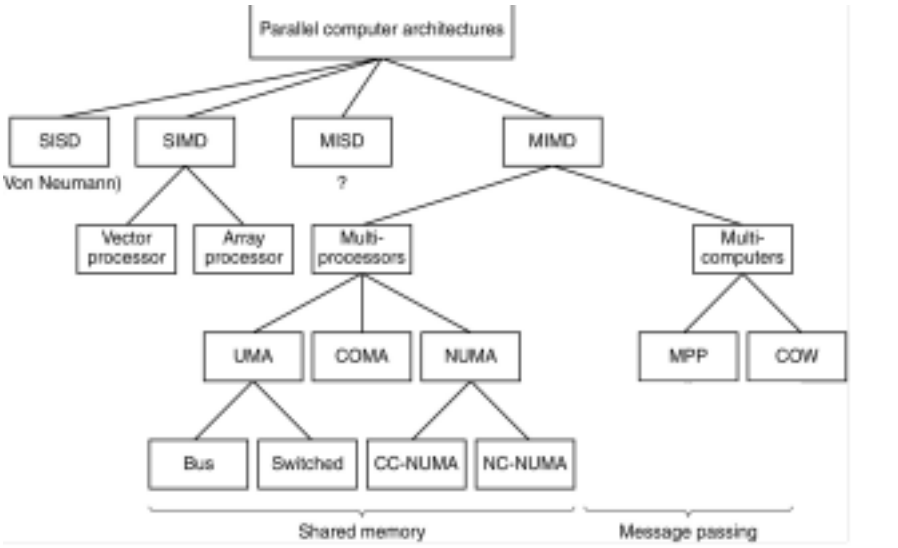

Figure 1.2: La Tassonomia di Flynn

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.

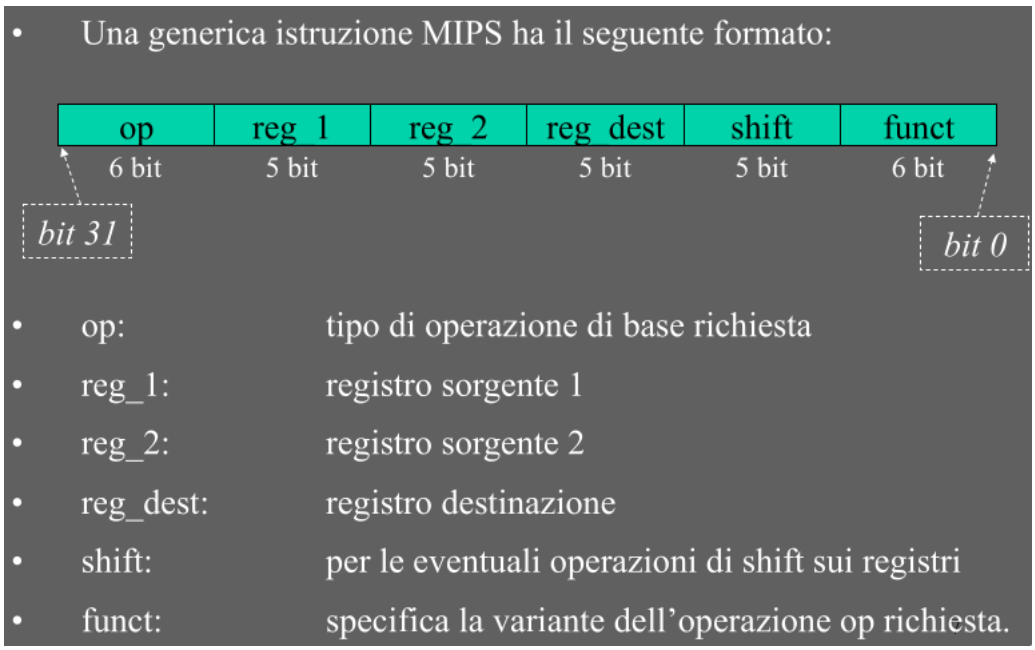

Figure 1.3: Istruzione MIPS

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.

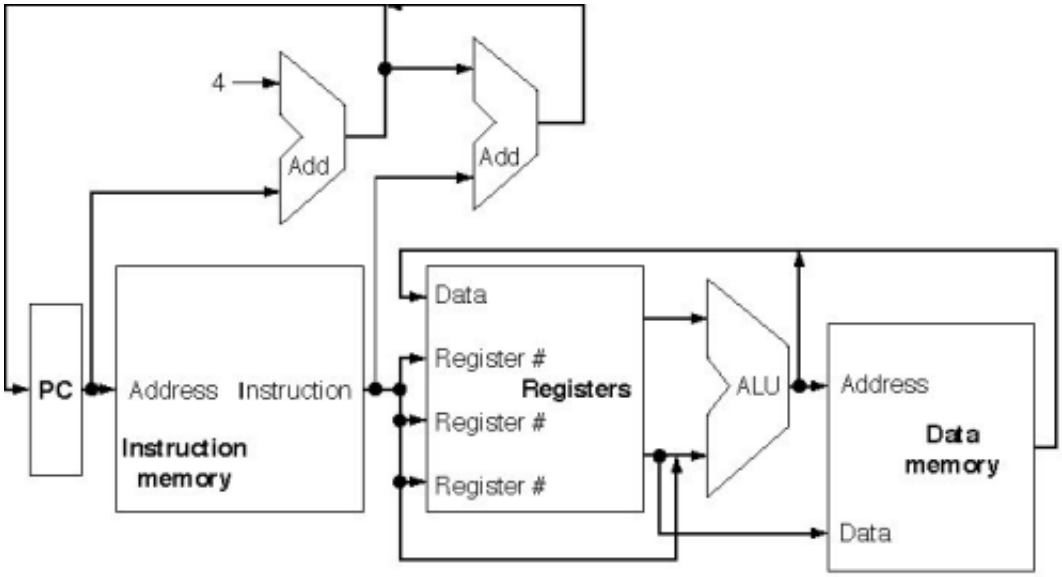

Figure 1.4: Schema ad alto livello del datapath MIPS

Note:-

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

[1] Cache di primo livello.

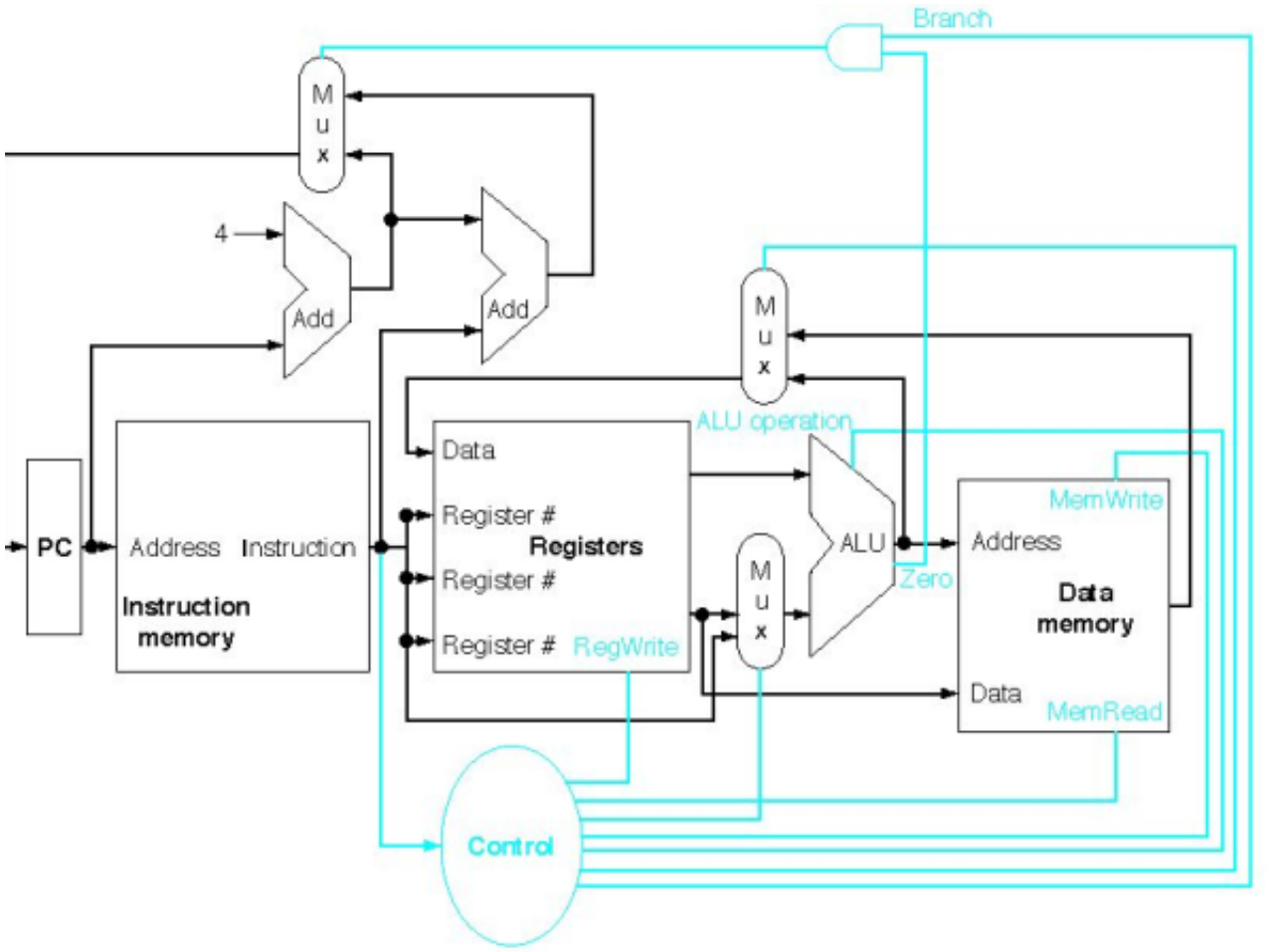

Figure 1.5: Versione modificata del MIPS con una Control Unit

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*.

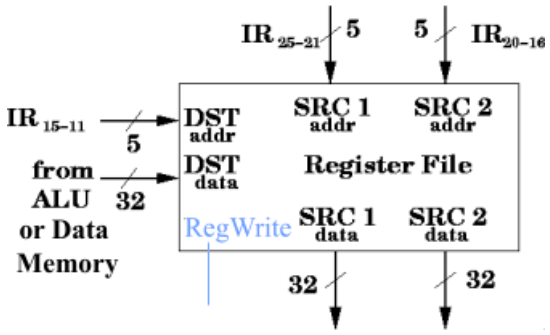

Figure 1.6: Operazione di scrittura

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*.

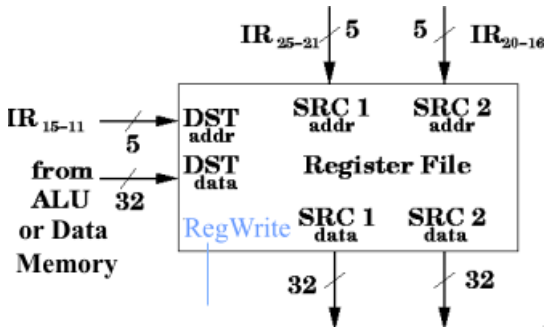

Figure 1.7: Operazione di lettura

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.

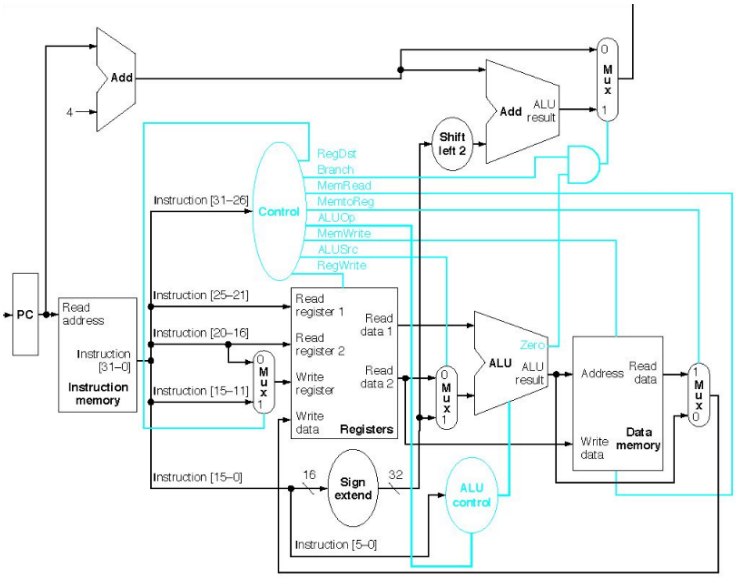

Figure 1.8: Versione modificata del MIPS la ALU Control