
ANNO ACCADEMICO 2024/2025

Architettura degli Elaboratori II

Teoria

Altair's Notes



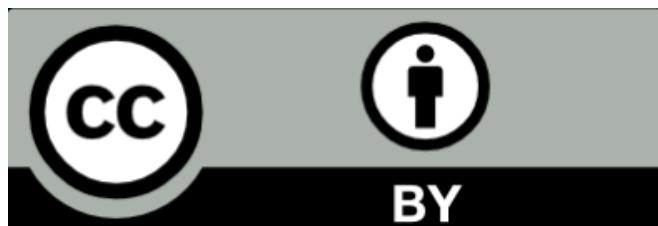
DIPARTIMENTO DI INFORMATICA

CAPITOLO 1	CONCETTI DI BASE	PAGINA 5
1.1	Introduzione Tassonomia delle architetture — 6 • Alcuni concetti fondamentali — 6	5
1.2	Una semplice macchina RISC MIPS - Versione monociclo — 8 • Banco dei registri — 10 • Una semplice Control Unit — 11 • L'esecuzione di un'istruzione — 12	7
1.3	Dal monociclo al multiciclo MIPS - Versione multiciclo — 12 • Control Unit multiciclo — 14 • Macchine a stati finiti e Microprogrammi — 15 • Complex Instruction Set Computer (CISC) — 15 • Reduced Instruction Set Computer (RISC) — 16	12
1.4	Pipeline L'Architettura MIPS Pipelined — 17 • Problemi della Pipeline — 21 • Multiple Pipeline — 23 • Scheduling della Pipeline — 24	17
CAPITOLO 2	INSTRUCTION LEVEL PARALLELISM (ILP)	PAGINA 27
2.1	Introduzione Aumentare la Frequenza del Clock della CPU — 27 • Multiple Issue — 28	27
CAPITOLO 3	CACHING	PAGINA 30
CAPITOLO 4	ARCHITETTURE PARALLELE	PAGINA 32
CAPITOLO 5	QUANTUM COMPUTING	PAGINA 34
CAPITOLO 6	GPU	PAGINA 36

Premessa

Licenza

Questi appunti sono rilasciati sotto licenza Creative Commons Attribuzione 4.0 Internazionale (per maggiori informazioni consultare il link: <https://creativecommons.org/licenses/by/4.0/>).



Formato utilizzato

Box di "Concetto sbagliato":

Concetto sbagliato 0.1: Testo del concetto sbagliato

Testo contenente 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 definizioni

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".

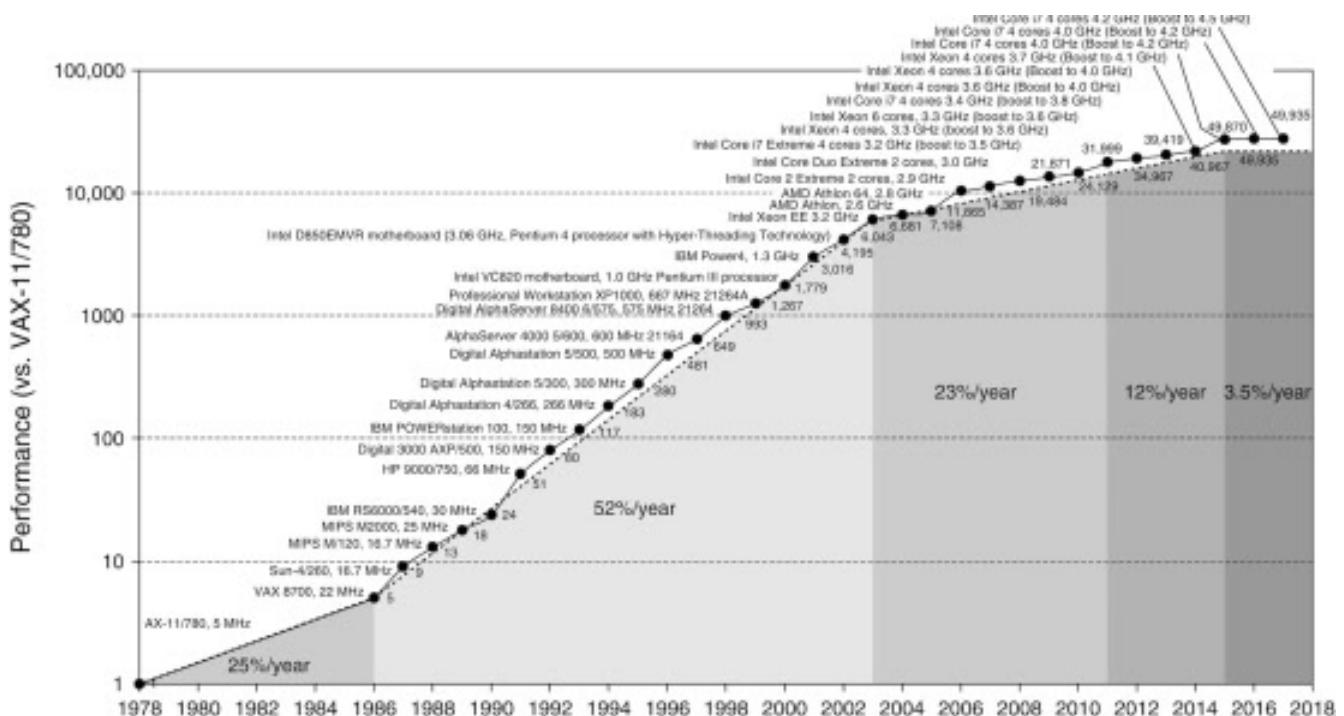


Figure 1.1: Nel 1986 ha inizio la "Rivoluzione RISC", mentre all'inizio degli anni 2000 si inizia a sfruttare l'idea di avere più "core".

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.

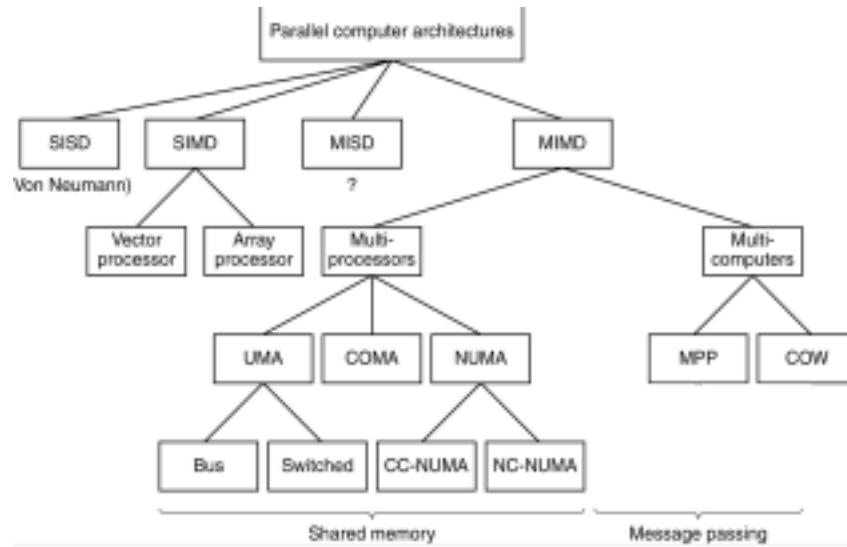


Figure 1.2: La Tassonomia di Flynn

1.1.2 Alcuni concetti fondamentali

Definizione 1.1.2: Microarchitettura

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 d'istruzioni percorrono diverse parti del datapath per venire eseguite.

Definizione 1.1.3: ISA

L'Instruction Set Architettura (ISA) è l'insieme d'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.

- Una generica istruzione MIPS ha il seguente formato:

op	reg_1	reg_2	reg_dest	shift	funct
6 bit	5 bit	5 bit	5 bit	5 bit	6 bit

- op: tipo di operazione di base richiesta
- reg_1: registro sorgente 1
- reg_2: registro sorgente 2
- reg_dest: registro destinazione
- shift: per le eventuali operazioni di shift sui registri
- funct: specifica la variante dell'operazione op richiesta.

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:-

Soltamente 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 d'istruzioni¹" la prossima istruzione da eseguire;
 2. Decodifica l'istruzione e contemporaneamente legge i registri.

Note:-

I passi successivi dipendono dal tipo d'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.

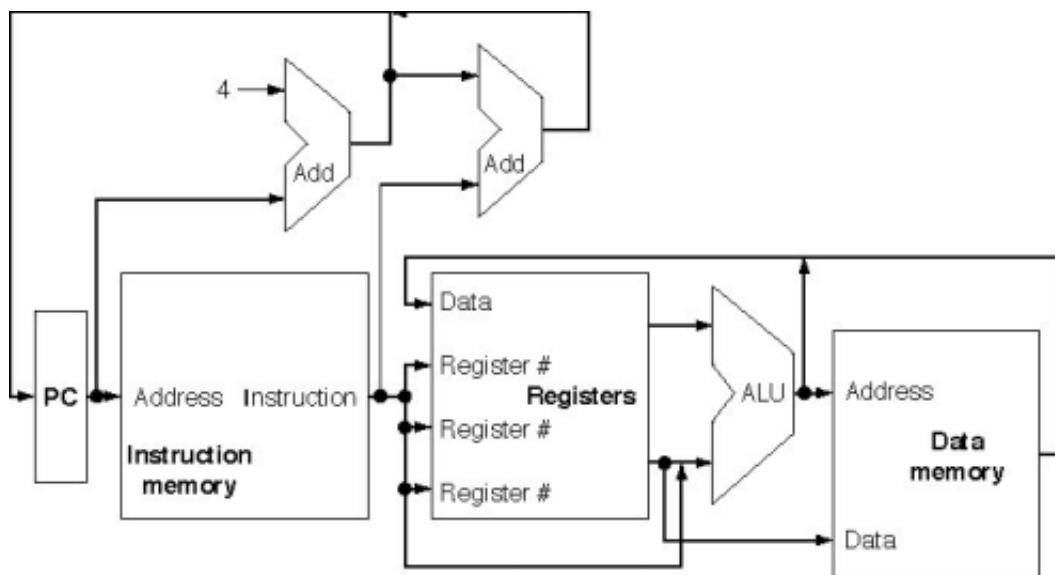


Figure 1.4: Schema ad alto livello del datapath MIPS

Note:-

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

¹Cache di primo livello.

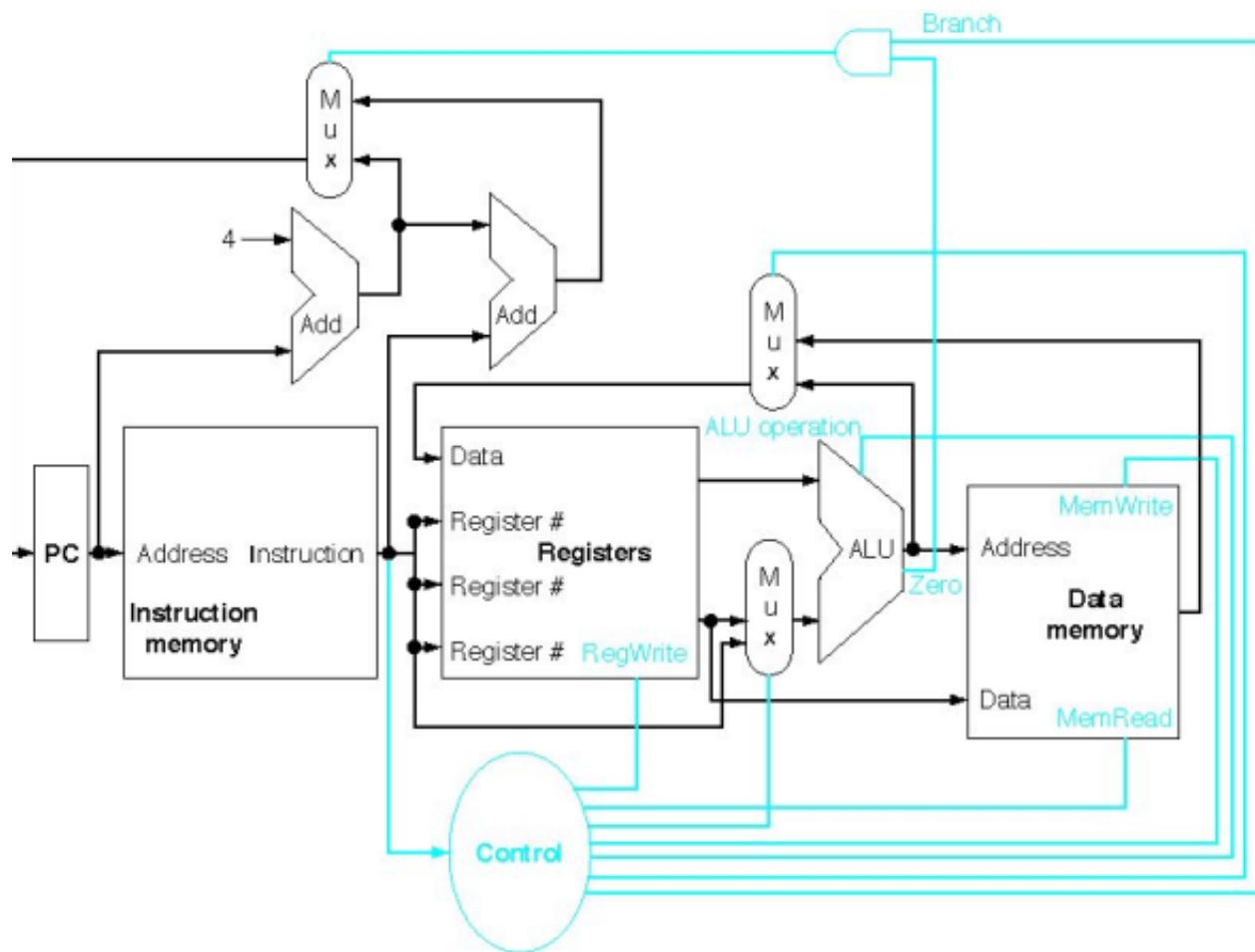


Figure 1.5: Versione modificata del MIPS con una Control Unit

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:-

Soltanamente 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*.

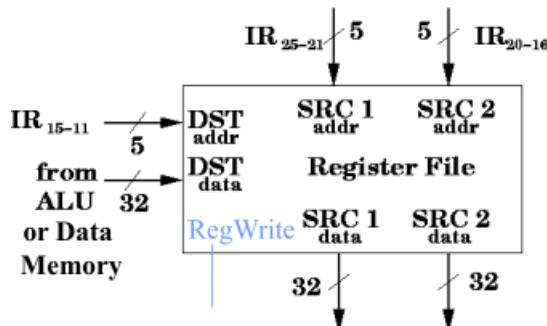


Figure 1.6: Operazione di scrittura

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*.

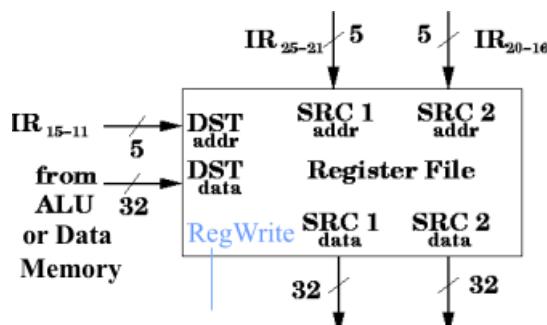


Figure 1.7: Operazione di lettura

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 d'istruzione in esecuzione, specificato nel campo op;
- ⇒ dalla specifica operazione da eseguire, determinata dal campo func.

Esempio 1.2.1 (Istruzioni di tipo-R)

- ⇒ Se $op == \text{load} \parallel op == \text{store}$ allora la ALU deve eseguire una *somma*;
- ⇒ se $op == \text{beq}$ allora la ALU deve eseguire una *sottrazione* (per controllare se il risultato è 0);
- ⇒ Se $op == \text{tipo-R}$ 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.

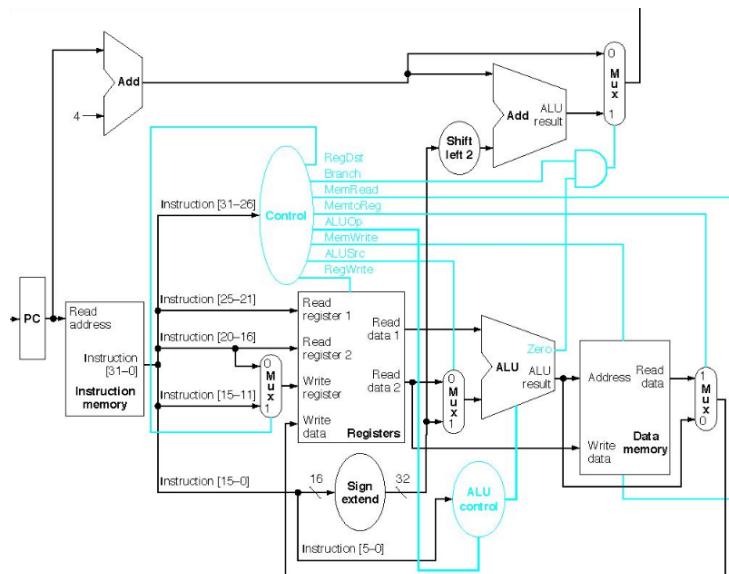


Figure 1.8: Versione modificata del MIPS la ALU Control

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 a una operazione di tipo-R, e in particolare ALUOp=10, che, dati in input alla ALU control insieme al campo *func* 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 scomponete 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

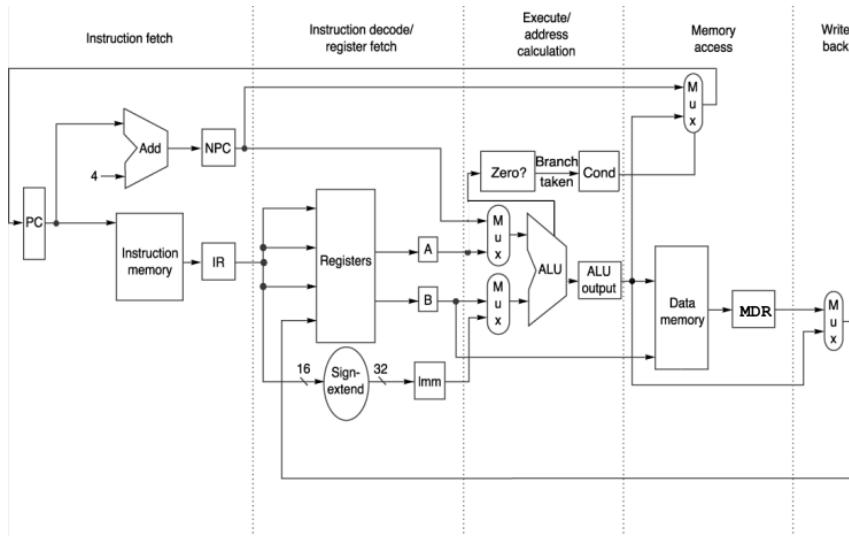


Figure 1.9: Il Datapath suddiviso nelle 5 fasi standard

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 d'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.

1.3.2 Control Unit multiciclo

Definizione 1.3.2: Control Unit - multiciclo

Ogni fase sarà descritta da una tabella di verità i cui input sono:

- ⇒ il tipo d'istruzione in esecuzione;
- ⇒ la fase corrente nella sequenza di esecuzione dell'istruzione.

L'output invece sarà:

- ⇒ l'insieme di segnali da asserire in quella fase;
- ⇒ la fase successiva nella sequenza di esecuzione dell'esecuzione.

Note:-

Gli output della Control Unit sono una descrizione informale di una macchina di Moore.

Corollario 1.3.1 Macchina di Moore

La macchina di Moore è un automa a stati finiti in cui:

- ⇒ a ogni stato è associato un output che dipende solo da quello stato (i segnali da asserire);
- ⇒ la transizione nello stato successivo dipende solo dallo stato corrente e dall'input.

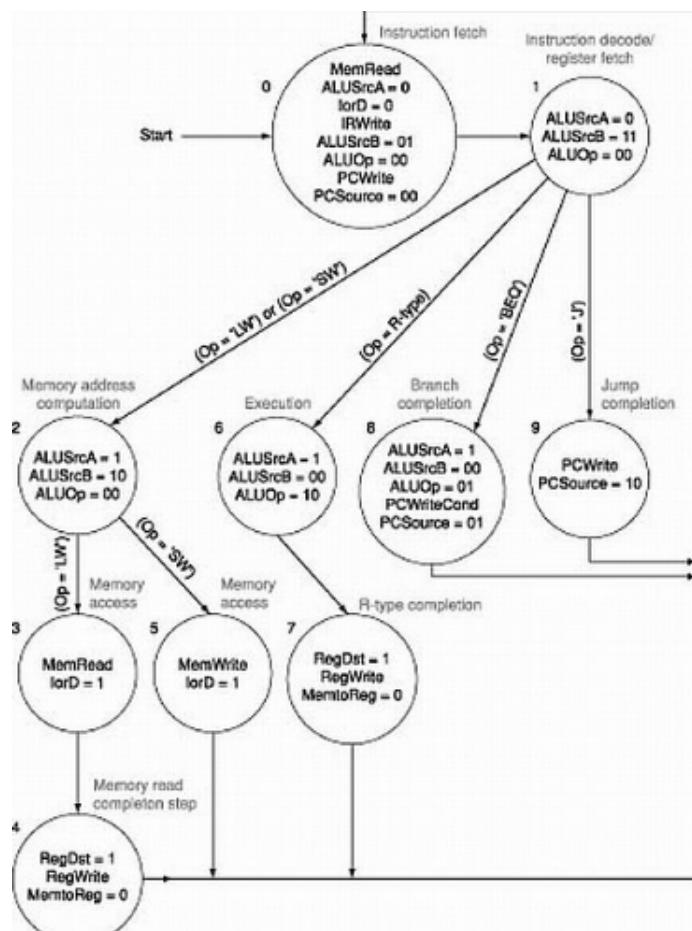


Figure 1.10: Unità di Controllo multiciclo

1.3.3 Macchine a stati finiti e Microprogrammi

Domanda 1.6

Che differenza c'è tra un datapath controllato da un DFA e uno controllato da un Microprogramma?

Risposta: nessuna, è solo un modo di descrivere l'informazione.

Definizione 1.3.3: Microprogrammazione

Tecnica per descrivere il funzionamento della Control Unit mediante una rappresentazione simbolica del controllo in forma di microistruzioni indirizzate da un micro Program Counter.

Note:-

Un microprogramma è solamente una rappresentazione testuale di una macchina di Moore, ogni microistruzione corrisponde a uno stato e il micro PC rappresenta il registro degli stati.

Domanda 1.7

Perché usare una rappresentazione o un'altra?

Risposta: è una questione di convenienza in base alla complessità della funzione (dipende sia dalle istruzioni ISA che dalla loro scomposizione).

Note:-

Per motivi storici le prime Control Unit furono descritte tramite microprogrammazione.

1.3.4 Complex Instruction Set Computer (CISC)

Negli anni '60 e '70 l'instruction set diventa molto grande e, a posteriori, sarà chiamato CISC. In alcuni casi si raggiungevano centinaia d'istruzioni.

Definizione 1.3.4: Completa ortogonalità

Ogni argomento di ogni istruzione poteva indirizzare la memoria usando una qualsiasi modalità d'indirizzamento possibile^a.

^aVisto in "Storia dell'informatica".

Note:-

Questa caratteristica permetteva molta flessibilità, ma al tempo stesso rendeva la Control Unit complicata e molto lenta.

Domanda 1.8

Perché si utilizzava un ISA così complesso?

1. All'epoca l'accesso alla RAM erano molto più elevati di quelli dell'accesso alla ROM che conteneva i microprogrammi per le istruzioni;
2. Non esistevano CPU dotate di cache (introdotte solo nel 1968 e divennero di uso comune solo dopo molti anni);
3. Per semplificare il lavoro del programmatore;
4. La RAM era poca e costosa, istruzioni più espansive generano eseguibili più corti;
5. I microprogramma permettevano di aggiungere istruzioni all'instruction set.

1.3.5 Reduced Instruction Set Computer (RISC)

Tra la fine degli anni '70 e l'inizio degli anni 80' la concezione delle architetture inizia a mutare. Nel 1980, D. Patterson e C. Sequin progettano una CPU la cui Control Unit non è descritta da un microprogramma e coniano i termini CISC e RISC. Il loro progetto si evolverà nelle architetture SPARC (Scalable Processor ARChitecture), usate dalla SUN Microsystem.

Contemporaneamente, J. Hennessy lavora a un'architettura simile per ottimizzare il *pipeline*: MIPS (Microprocessor *without* Interlocked Pipelines Stages), visto precedentemente.

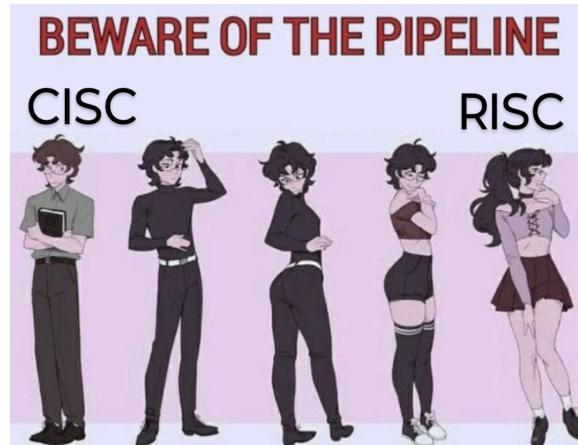


Figure 1.11: The virgin CISC vs. The chad RISC

Domanda 1.9

Qual è l'idea alla base delle architetture RISC?

1. La CPU esegue un numero limitato d'istruzioni macchina semplici (che richiedono datapath più corti e una Control Unit semplice) e quindi avere un ciclo di clock più corto;
2. L'accesso alla RAM va limitato il più possibile;
3. Le istruzioni devono principalmente usare registri come argomenti.

In dettaglio:

- ⇒ Rinunciare a un sofisticato livello di microcodice;
- ⇒ Definire istruzioni di lunghezza fissa e facili da decodificare;
- ⇒ La memoria RAM deve essere indirizzata solo da operazioni di LOAD e STORE;
- ⇒ Avere molti registri;
- ⇒ Sfruttare la pipeline.

Note:-

Questo contribuì al successo dei processori RISC che portò alla scomparsa dei CISC.

Definizione 1.3.5: CPI

Il Clock cycles Per Instruction (CPI) è il numero di cicli di clock necessari per eseguire un'istruzione. Indica la velocità alla quale una CPU è in grado di sforzare le istruzioni che esegue.

1.4 Pipeline

1.4.1 L'Architettura MIPS Pipelined

Passando da monociclo a multiciclo si può aumentare l'efficienza, riducendo lo spreco di tempo. Ma c'è un'altra ragione: si può pensare di sovrapporre, parzialmente, istruzioni consecutive. Ciò non sarebbe possibile in una macchina monociclo.

In una macchina multiciclo mentre una certa fase di un'istruzione occupa una certa parte del datapath le altre porzioni di datapath sono libere.

istr. number	1	2	3	4	5	6	7	8	9
istr. i	IF	ID	EX	MEM	WB				
istr. i+1		IF	ID	EX	MEM	WB			
istr. i+2			IF	ID	EX	MEM	WB		
istr. i+3				IF	ID	EX	MEM	WB	
istr. i+4					IF	ID	EX	MEM	WB

Figure 1.12: Rappresentazione di una pipeline.

Note:-

A regime tutte le fasi sono occupate da un'istruzione diversa.

Osservazioni 1.4.1 Accorgimenti

- Si deve evitare di usare le stesse risorse per compiere contemporaneamente operazioni diverse. Per questo motivo spesso si usano più ALU;
- Questo è ancora più importante se si introduce il multiple issue, ossia la possibilità di eseguire istruzioni indipendenti di uno stesso programma;
- In alcuni casi ciò è possibile. Per esempio il banco dei registri può essere usato 2 volte in un ciclo di clock in fasi differenti (nell'immagine sopra cerchiati in giallo), uno nella prima parte del ciclo di clock e l'altro nella seconda.

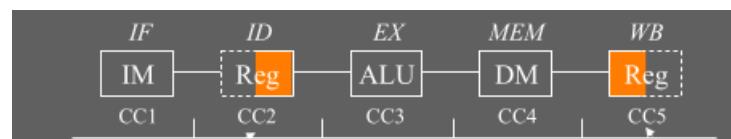


Figure 1.13: Utilizzo del banco dei registri.

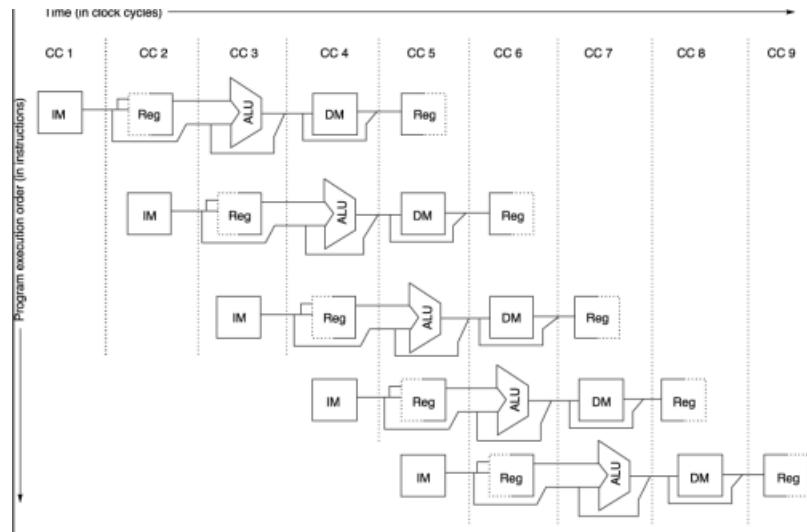


Figure 1.14: MIPS come serie di datapath traslati nel tempo.

Vedendo questo schema si possono fare due osservazioni:

- Si deve ipotizzare l'esistenza di memorie (cache) diverse per dati (DM) e istruzioni (IM) in modo da non avere conflitti tra IF e MEM;
- Il PC deve essere incrementato a ogni ciclo di clock nella fase IF.

Definizione 1.4.1: Registri della Pipeline

In un'implementazione reale ogni stage della pipeline è separato e collegato allo stage successivo da opportuni registri della pipeline^a. Alla fine di un ciclo di clock il risultato viene memorizzato in uno di questi registri.

^aUn po' come una catena di montaggio.

Note:-

Questi registri servono anche a trasportare dati d'istruzioni non adiacenti.

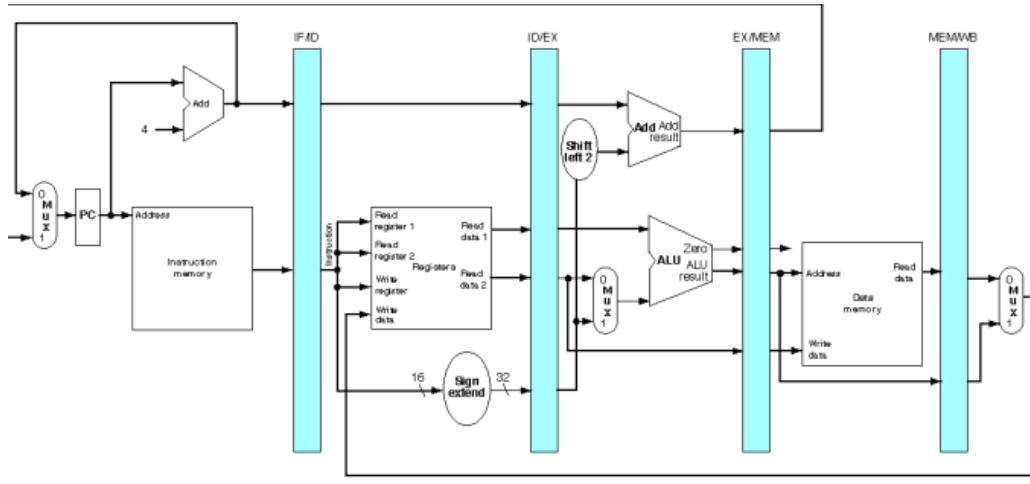


Figure 1.15: Registri della pipeline (in azzurro).

Simulazione di una LOAD

- **Instruction Fetch:** l'istruzione viene letta dalla Instruction Memory indirizzata dal PC e viene posta nel registro IF/ID. Il PC viene incrementato di 4 e salvato in IF/ID;

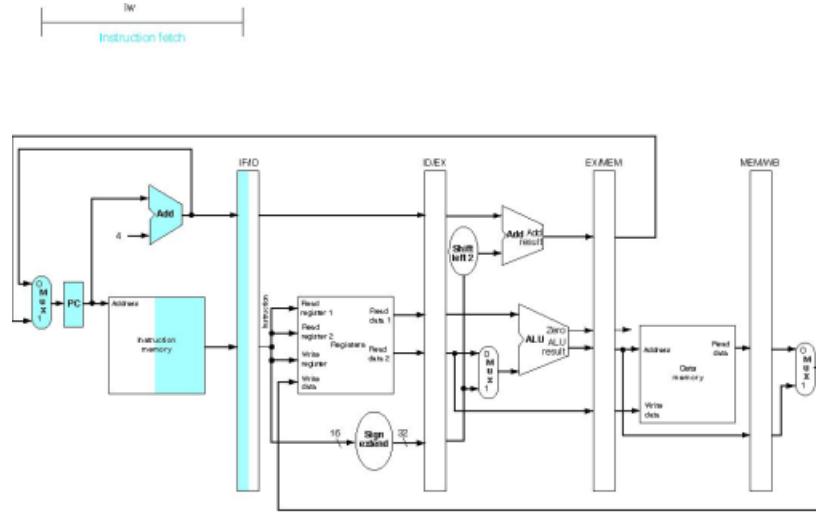


Figure 1.16: IF.

- **Instruction Decode:** il registro IF/ID viene letto per indirizzare il register file. Vengono letti i due registri, indipendentemente dal fatto che se ne userà uno solo oppure entrambi. I 16 bit del valore immediato vengono convertiti a valore 32 bit, e i 3 valori vengono scritti in ID/EX insieme al valore incrementato del PC;

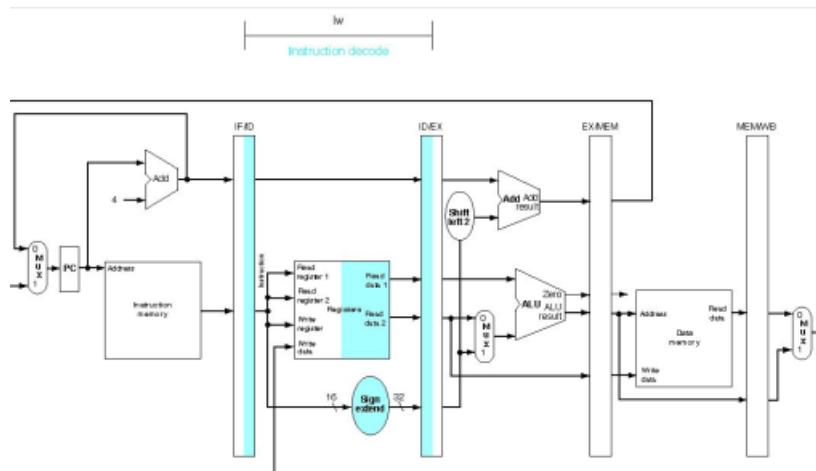


Figure 1.17: ID.

- **EXecute:** la load legge da ID/EX il contenuto del registro 1 e il valore immediato, li somma attraverso l'ALU e scrive il risultato in EX/MEM;

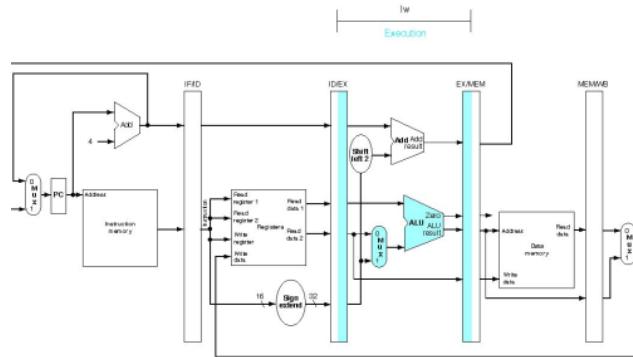


Figure 1.18: EX.

- **MEMory:** avviene l'accesso alla memoria dati, indirizzata dal valore letto nel registro EX/MEM. Il dato letto viene scritto in MEM/WB;

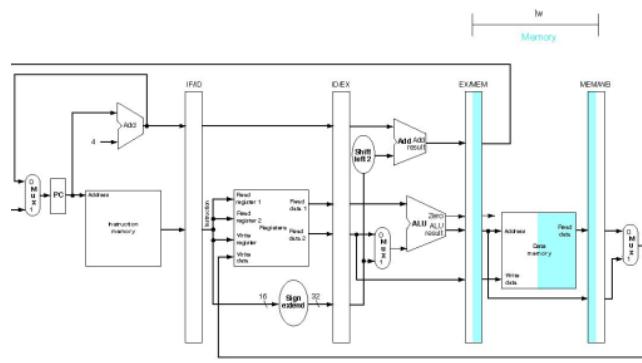


Figure 1.19: MEM.

- **Write Back:** viene scritto il registro di destinazione della load con il valore prelevato dalla memoria dati.

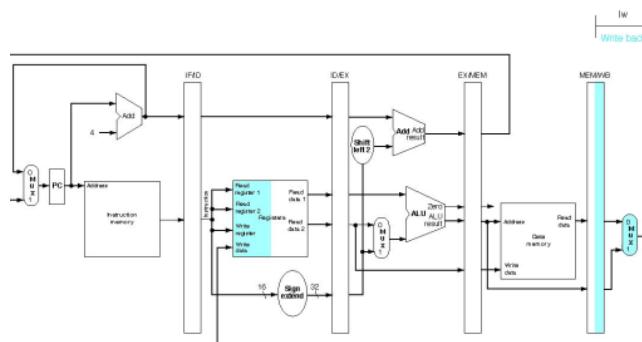


Figure 1.20: WB.

Domanda 1.10

Dov'è il numero del registro da modificare?

Risposta: è andato perso (sovraffatto). Per evitare ciò bisogna far sì che il numero passi attraverso tutti i registri della pipeline fino a WB.

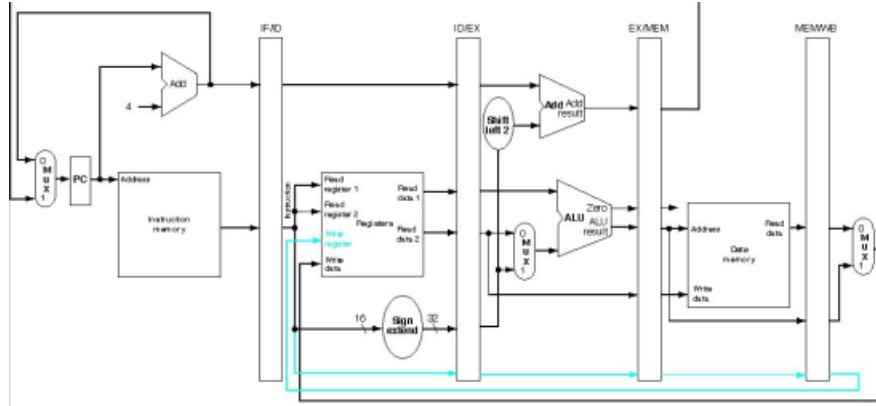


Figure 1.21: Datapath corretto.

Note:-

La presenza di una pipeline rende l'esecuzione di un'istruzione leggermente più lenta di una versione senza, ma i programmi girano più velocemente.

1.4.2 Problemi della Pipeline

Purtroppo la pipeline presenta alcuni problemi che ne limitano la produttività:

- **Problemi Strutturali:** alcune combinazioni d'istruzioni non possono essere eseguite simultaneamente (e.g. se si ha una sola ALU non la si può usare nello stesso ciclo di clock per fare più cose diverse);
- **Problemi sui Dati:** se un'istruzione A ha bisogno del risultato di un'istruzione B precedente che non ha ancora terminato;
- **Problemi di Controllo:** quando vengono eseguiti salti che possono cambiare il valore di PC.

Definizione 1.4.2: Stall

Se si verifica uno di questi problemi è necessario fermare la pipeline (stall). Se un'istruzione generica I genera un problema:

- le istruzioni avviate prima dell'istruzione I possono proseguire fino a essere completate;
- le istruzioni avviate dopo I devono essere fermate, fino a che non viene risolto il problema dell'istruzione I.

Problemi Strutturali

Definizione 1.4.3: Problemi Strutturali

In generale i problemi strutturali si verificano perché, per ragioni di complessità o di costi di produzione, alcune risorse hardware all'interno del datapath non sono duplicate.

Note:-

Una cache L1 unica per dati e istruzioni genererebbe molti problemi strutturali, per questo solitamente ve ne sono 2 separate.

Problemi sui Dati

Definizione 1.4.4: Problemi sui Dati

I problemi sui dati sono causati dal fatto che le istruzioni di un programma non sono scollegate tra di loro e alcune devono usare output generati da istruzioni precedenti.

Corollario 1.4.1 Forwarding

Il forwarding è una tecnica per bypassare questi problemi. L'idea è quella di rendere disponibile un risultato il prima possibile, per esempio in una ADD il valore di output è generato prima della fase di WB. Quindi si può usare il valore memorizzato in EX/MEM dell'istruzione interessata.

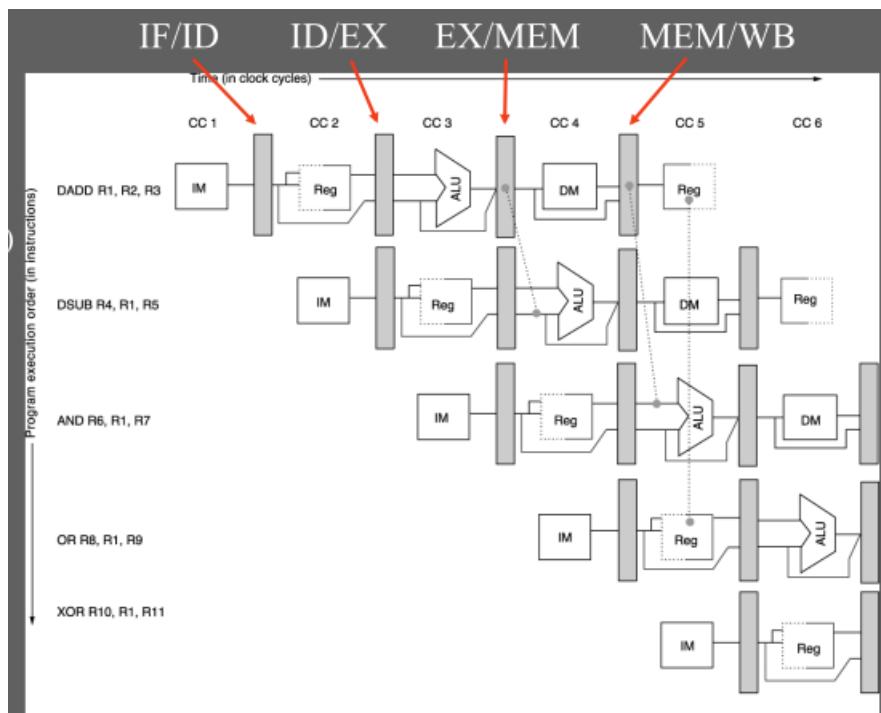


Figure 1.22: Forwarding.

Problemi di Controllo

Definizione 1.4.5: Problemi di Controllo

I problemi di controllo sono dovuti al fatto che, nel caso d'istruzioni di salto, il PC può cambiare.

Note:-

Si sprecano cicli di clock quando si salta. Per mitigare questo problema si usa una predizione statica: si dà per scontato che i salti in indietro vengano sempre presi e i salti in avanti no.

Corollario 1.4.2 Delayed Branch

Il Delayed Branch consiste nello spostare dopo un branch una istruzione I che è comunque necessario eseguire indipendentemente dall'esito. In questo modo si dà tempo al datapath di valutare se il salto debba essere eseguito o meno.

Note:-

Questa tecnica richiede l'intervento del compilatore e la CPU deve essere progettata apposta.

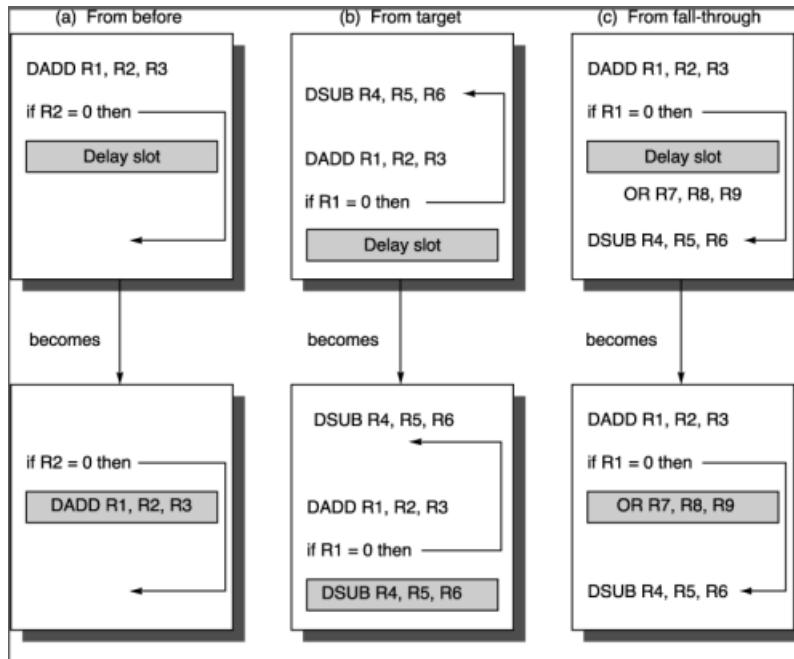


Figure 1.23: Delayed Branch.

1.4.3 Multiple Pipeline**Domanda 1.11**

Se una pipeline è efficiente, perché non usarne 2?

Bisogna che:

- Si riescano a prelevare due istruzioni dalla instruction memory.
- Le due istruzioni non generino conflitti.
- Le due istruzioni sono indipendenti.

Un'architettura a due pipeline era adottata dal primo pentium (80586). Una pipeline u poteva eseguire qualsiasi istruzione macchina, mentre la pipeline v solo istruzioni intere.

Le istruzioni venivano eseguite *in-order*, ossia nell'ordine in cui comparivano nell'eseguibile e alcune regole stabilivano se erano compatibili. Esistevano specifici compilatori per il pentium in grado di generare codice con un alto numero di istruzioni compatibili.

Note:-

Ovviamente è possibile avere più di due pipeline in parallelo.

Tuttavia alcune fasi di execute sono molto lunghe rispetto alla semplice addizione tra interi:

- Somma/sottrazione di numeri floating point.
- Moltiplicazione/divisione tra numeri interi.
- Moltiplicazione/divisione tra numeri floating point.

Note:-

Si potrebbe adottare un ciclo di clock sufficientemente lungo per le operazioni più lente, ma ciò penalizzerebbe le operazioni più veloci. Conviene suddividere la execute in più fasi.

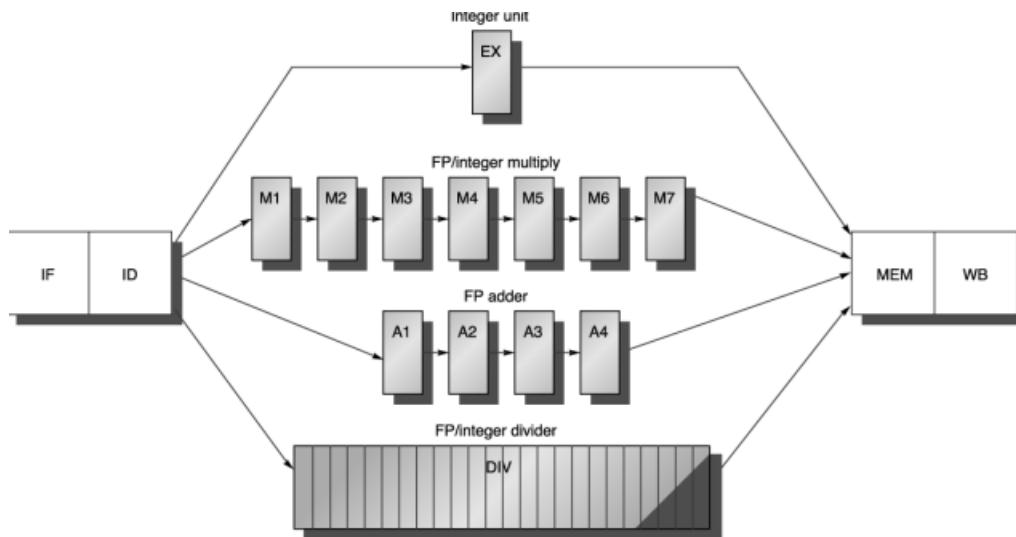


Figure 1.24: Pipeline con più unità funzionali.

In questo schema si vede:

1. Il numero di stage della pipeline dipende dalla complessità dell'operazione.
2. Ci possono essere più istruzioni in fase di execute contemporaneamente.

1.4.4 Scheduling della Pipeline

Fin'ora si è sempre assunta l'esecuzione delle istruzioni in-order, però se si verifica un problema strutturale o sui dati la pipeline viene temporaneamente fermata. Questo tipo di pipeline è detto *schedulata staticamente*.

Le CPU moderne implementano una qualche forma di *scheduling dinamico*, per limitare gli stall sulla pipeline.

Definizione 1.4.6: IPC

Instruction Per Clock (cycle): il numero medio di istruzioni eseguite per ciclo di clock.

Note:-

Si calcola facendo girare un benchmark di N istruzioni in C cicli di clock.

$$IPC = N/C$$

Definizione 1.4.7: CPI

Clock Per Instruction ossia il numero medio di cicli di clock necessari per eseguire un'istruzione.

$$CPI = 1/IPC$$

2

Instruction Level Parallelism (ILP)

2.1 Introduzione

Definizione 2.1.1: Instruction Level Parallelism (ILP)

I processori che tratteremo sono pipelined e superscalari:

1. Eseguono le istruzioni in pipeline.
2. Avviano all'esecuzione in parallelo più istruzioni per ciclo di clock.

2.1.1 Aumentare la Frequenza del Clock della CPU

Ciò significa un ciclo di clock più corto con una divisione in un maggiore numero di fasi. Se aumenta il numero di fasi (*profondità*) allora ci saranno più istruzioni in esecuzione contemporaneamente.

Note:-

Questa relazione tra numero di fasi e ciclo di clock fu pesantemente sfruttata nel pentium IV in cui si sfioravano i 4 GHz con pipeline di quasi 30 stadi.

Tuttavia non è possibile sfruttare all'infinito questa tecnica perché:

1. Maggiore è il numero di fasi, maggiore è la complessità della pipeline e quindi la sua control unit.
2. Frequenze di clock maggiori producono interferenze tra le piste, consumi e conseguenti problemi di dissipazione del calore.

Definizione 2.1.2: Overclocking

Il progettista di una CPU non tara il ciclo di clock sulla durata esatta del tempo necessario all'impulso elettrico per attraversare una parte del datapath, ma lo rende un po' più lungo. Su quella differenza gli smanettoni possono "giocare" per aumentare le prestazioni della CPU.

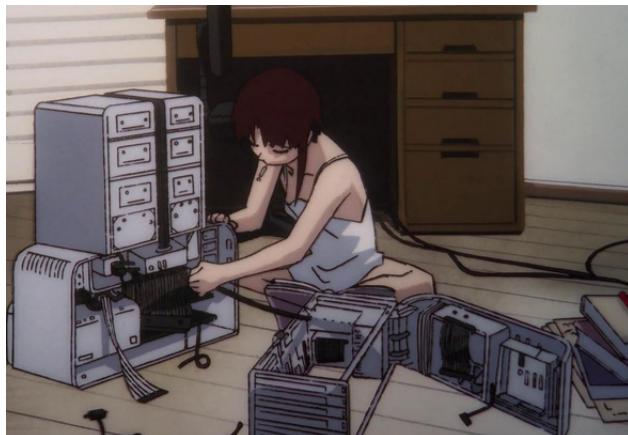


Figure 2.1: Io che faccio overclock del case.

2.1.2 Multiple Issue

3

Caching

4

Architetture Parallele

5

Quantum Computing

6
GPU

