

Datapath

Qua c'è la teoria necessaria per passare l'esame, non per capire come funziona a fondo il Datapath.

L'istruzione J (Jump) non utilizza i registri A e B e neanche l'ALU. È l'unica istruzione che "fa così".

Tutte le altre istruzioni utilizzano almeno il registro A e sicuramente l'ALU.

La LW e la SW usano l'ALU per calcolare la somma tra l'offset e il contenuto del registro base.

Il blocco/registro B sorgente non è usato solo da LW e jump.

Se nella domanda viene chiesto "quali istruzioni richiedono alla alu di effettuare una sottrazione" sono solo beq e sub.

Mentre "per quali istruzioni l'ALU effettua una sottrazione?" Aggiungere anche slt.

SLT a volte viene considerato come una sottrazione a volte no. Dipende da cosa richiede la domanda.

Inoltre l'indirizzo a cui saltare per una branch viene calcolato nella fase di decode, a prescindere da qualunque istruzione sia stata eseguita.

Un'addizione viene sempre fatta, durante la fase di fetch (PC+4)

Un'addizione viene sempre fatta durante la fase di decode (calcolo indirizzo di branch)

SLT calcola chi dei 2 registri è più piccolo (con una specie di sottrazione).

BEQ utilizza una specie di SLT per fare la sottrazione .

DOMANDA	RISPOSTA
scrittura nel register file	add, and, lw, or, slt, sub
registro B sorgente	add, and, beq, or, slt, sub, sw
scrittura nel registro B	add, and, beq, or, slt, sub, 
registro A sorgente	tutte meno j
due o più somme	tutte le istruzioni
due somme e una (sottrazione)	sub, beq, (slt)
due somme	and, beq, j, or, slt, sub
(sottrazione)	sub, beq, (slt)
tre somme	add, lw, sw
*() = se <u>viene richiesto</u> alla ALU di svolgere una sottrazione quelle tra le parentesi NO, se l'ALU svolge una sottrazione anche quelle tra parentesi	

Infine l'ALU ha un uscita chiamata "zero" perchè, per esempio durante una branch, viene fornito il risultato direttamente al multiplexor.


▼ CONTROL UNIT

La CU (Control Unit) all'interno del datapath multiciclo ha un unico input che è l'opcode dell'istruzione da eseguire (e a volte anche il funct). In base ai bit dell'opcode la CU piloterà i vari segnali di controllo in output.

La CU è a tutti gli effetti una FSM, dunque anche lo stato corrente della macchina è un input e influisce sull'output. Lo stato della macchina non è stato spiegato a lezione ma ha come componenti il PC, IR e lo Stack Pointer.

ChatGPT

A conversational AI system that listens, learns, and challenges

 <https://chat.openai.com/share/6d4f8c22-ad23-4b40-9ada-30a1ebb9aec8>

L'unica cosa che la CU calcola in autonomia, senza l'ausilio dell'Opcode è il registro di controllo PCSrc. Esso viene impostato a 10 SSE c'è una beq e la sottrazione ha dato esito positivo (risultato=0).

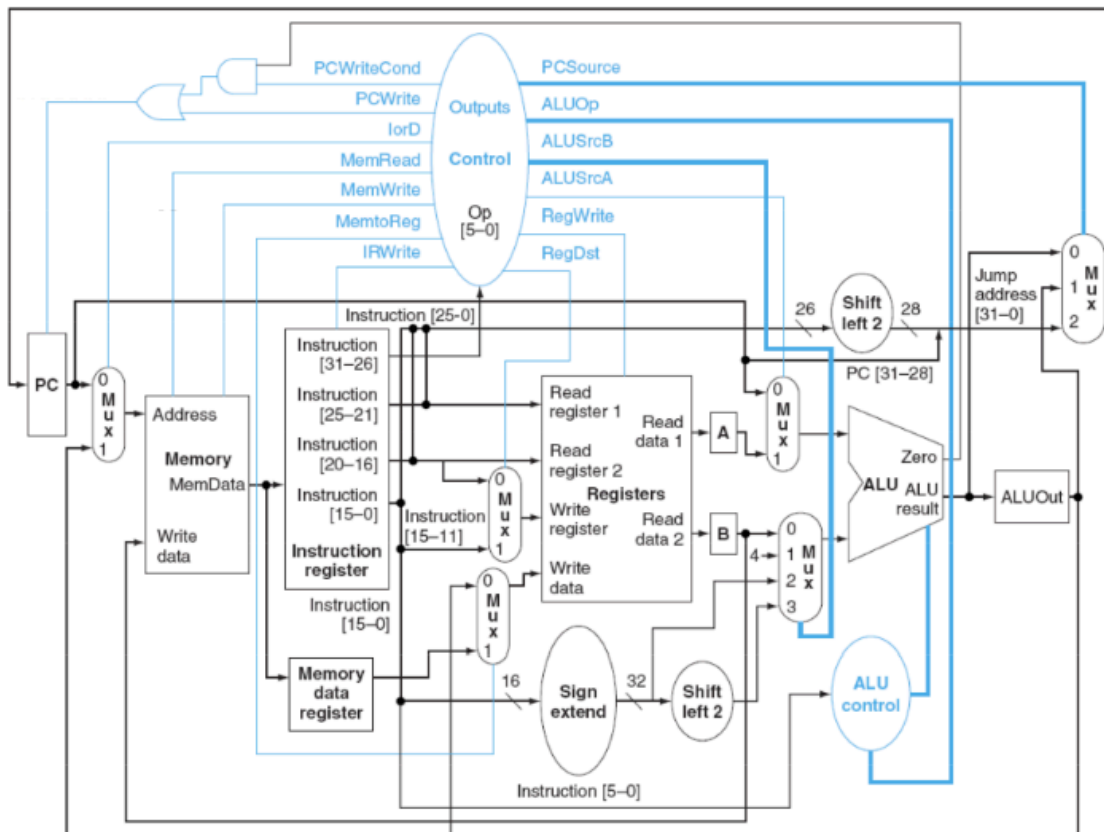
L'opcode (6 bit) serve a inizializzare/modificare ben 9 segnali di controllo, tranne uno che è appunto il PCSrc. Questi segnali sono usati nei multiplexer del datapath.

Actions of the 1-bit control signals

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register file destination number for the Write register comes from the rt field.	The register file destination number for the Write register comes from the rd field.
RegWrite	None.	The general-purpose register selected by the Write register number is written with the value of the Write data input.
ALUSrcA	The first ALU operand is the PC.	The first ALU operand comes from the A register.
MemRead	None.	Content of memory at the location specified by the Address input is put on Memory data output.
MemWrite	None.	Memory contents at the location specified by the Address input is replaced by value on Write data input.
MemtoReg	The value fed to the register file Write data input comes from ALUOut.	The value fed to the register file Write data input comes from the MDR.
lorD	The PC is used to supply the address to the memory unit.	ALUOut is used to supply the address to the memory unit.
IRWrite	None.	The output of the memory is written into the IR.
PCWrite	None.	The PC is written; the source is controlled by PCSrc.
PCWriteCond	None.	The PC is written if the Zero output from the ALU is also active.

Actions of the 2-bit control signals

Signal name	Value (binary)	Effect
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs a subtract operation.
	10	The funct field of the instruction determines the ALU operation.
ALUSrcB	00	The second input to the ALU comes from the B register.
	01	The second input to the ALU is the constant 4.
	10	The second input to the ALU is the sign-extended, lower 16 bits of the IR.
	11	The second input to the ALU is the sign-extended, lower 16 bits of the IR shifted left 2 bits.
PCSrc	00	Output of the ALU ($PC + 4$) is sent to the PC for writing.
	01	The contents of ALUOut (the branch target address) are sent to the PC for writing.
	10	The jump target address (IR[25:0] shifted left 2 bits and concatenated with $PC + 4[31:28]$) is sent to the PC for writing.



Controlli a 2 bit → linee blu scure

Controlli ad 1 bit → linee blu chiare

- **Memoria:** contiene istruzioni e dati; sulla base dell'indirizzo in input, restituisce in output l'istruzione o il dato letto
- **Register file:** 32 registri che contengono i dati utilizzati nel corso dell'esecuzione delle istruzioni; restituisce in output il contenuto dei due registri letti (indicati dai due input – readRegister1 e ReadRegister2) e scrive, se il segnale di scrittura è attivo, il dato in input WriteData nel registro indicato dall'indirizzo in input (WriteRegister)
- **ALU:**
 - per istruzioni R-type: esegue operazioni aritmetico-logiche su due operandi in funzione del function_code indicato dai 6 bit meno significativi dell'istruzione
 - per istruzioni di accesso a memoria: calcola l'indirizzo di memoria cui accedere
 - per istruzioni branch: confronta i due registri in input
 - per tutte le istruzioni: incrementa il PC (PC+4) e calcola il valore di branch (che sarà usato solo nel caso di istruzioni di branch)
- **Sign Extended:** opera l'estensione di segno dai 16 bit in input ai 32 bit in output
- **Shift Left 2:** opera uno shift a sinistra dei bit in input (con l'effetto di moltiplicare per 4 l'input)

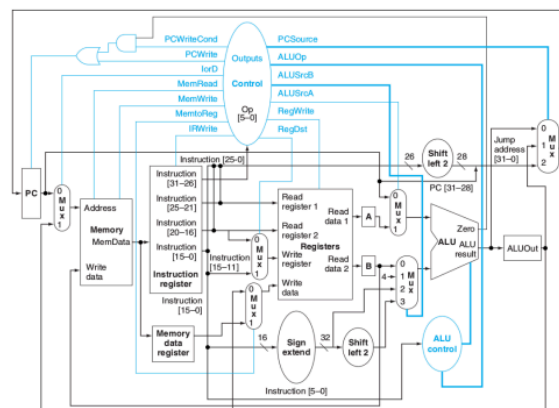


FIGURE 5.28 The complete datapath for the multicycle implementation together with the necessary control lines. The con-

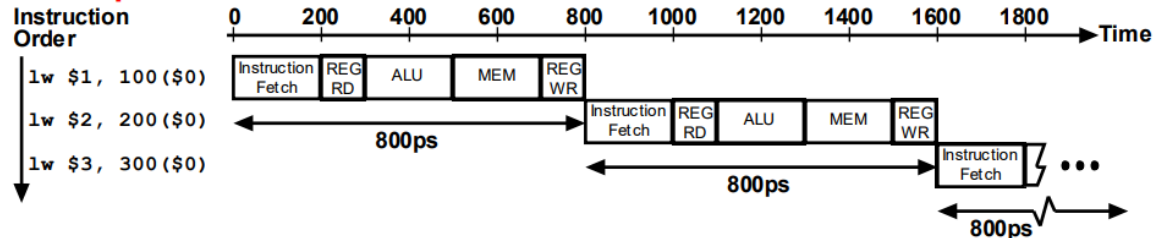
- **PC:** contiene i 32 bit che indicano l'indirizzo dell'istruzione da eseguire
- **Instruction register:** contiene i 32 bit che corrispondono alla codifica dell'istruzione prelevata da memoria per l'esecuzione
- **Memory Data Register:** contiene il dato letto da memoria prima della sua scrittura nel registro destinazione (e.g. nel caso di esecuzione di istruzione load)
- **A e B:** contengono i valori letti dai registri del RegisterFile
- **ALUOut:** contiene l'output della ALU

▼ PIPELINE E HAZARD

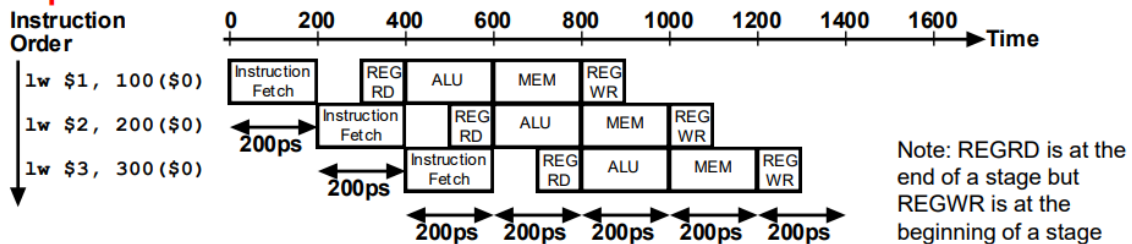
La pipeline permette di sovrapporre l'esecuzione di diverse istruzioni in un modo che mentre un'istruzione viene eseguita in una fase, le altre possano procedere nelle fasi successive. Sfrutta quindi al massimo le risorse della CPU.

Single-Cycle vs. Pipelined Execution

Non-Pipelined

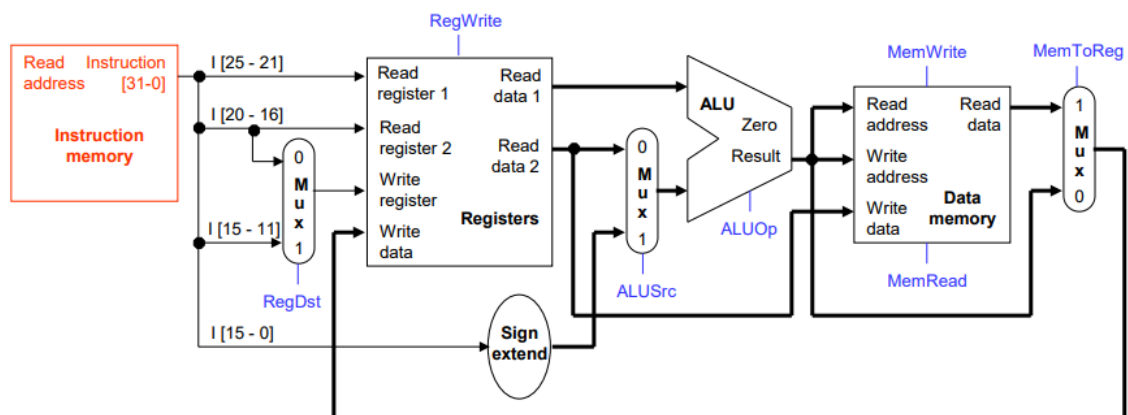


Pipelined



Nel nostro caso vedremo la pipeline con un datapath a singolo ciclo (non c'è scampo). Il datapath a singolo ciclo è differente dal quello multiciclo perché esegue un'operazione alla volta. Nella pipeline però esistono dei Pipeline Registers. Essi salvano ad ogni ciclo di clock i dati necessari per eseguire la prossima fase.

Ogni "fase" è strettamente collegata ad un componente del datapath e ad un Pipeline Register. Vediamoli:



- ❑ Executing a MIPS instruction can take up to five steps.

Step	Name	Description
Instruction Fetch	IF	Read an instruction from memory.
Instruction Decode	ID	Read source registers and generate control signals.
Execute	EX	Compute an R-type result or a branch outcome.
Memory	MEM	Read or write the data memory.
Writeback	WB	Store a result in the destination register.

- ❑ However, as we saw, not all instructions need all five steps.

Instruction	Steps required				
beq	IF	ID	EX		
R-type	IF	ID	EX		WB
sw	IF	ID	EX	MEM	
lw	IF	ID	EX	MEM	WB

Possono verificarsi dunque 3 tipi di Hazards (Errori). Essi si suddividono in base A COSA scatena l'errore.

DATA HAZARDS

Riguardano i registri. Quando un'istruzione dipende dal risultato di un'altra istruzione che però è ancora in esecuzione.

```
lw $r0, 10($r2)
sw $r3, 30($r0)
```

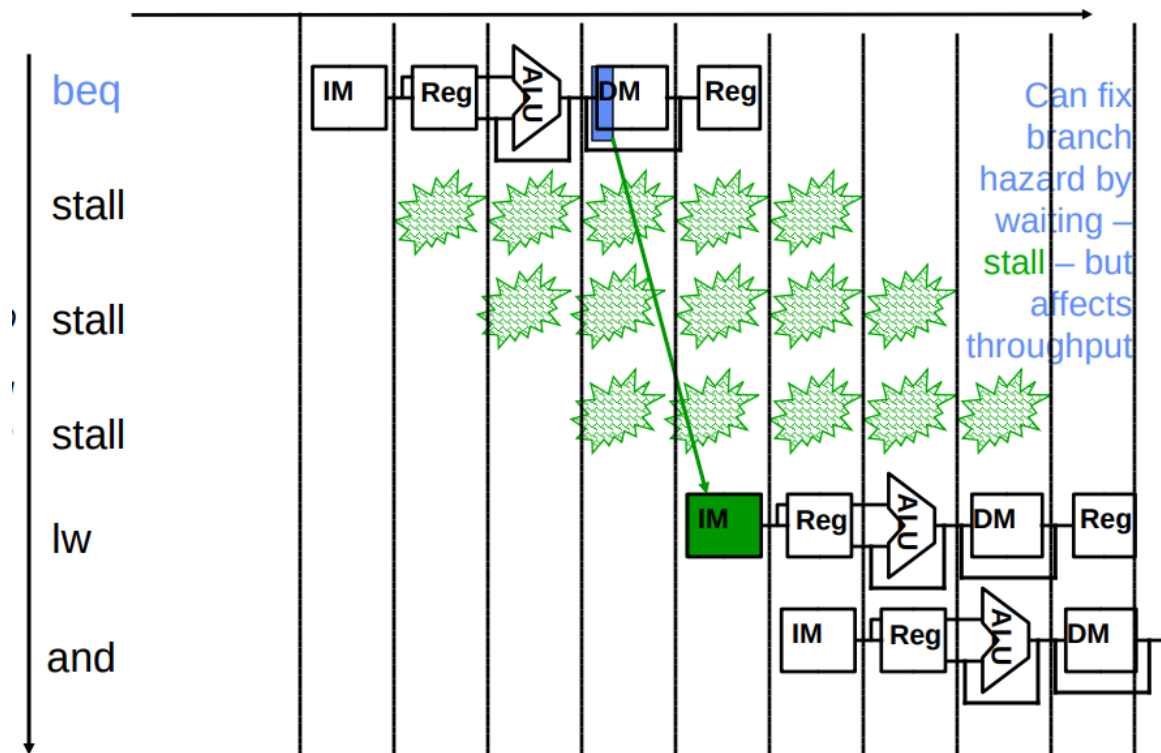
L'istruzione Load Word deve ancora caricare il contenuto in \$r0, dunque se eseguiamo una Store Word caricheremmo un valore sbagliato.

CONTROL HAZARDS

Al 99% avviene con le istruzioni di salto condizionate (istruzioni che cambiano il pc), succede quando hai un'istruzione che in base al suo output deciderà se bisogna fare le istruzioni dopo oppure saltarle. Il problema avviene

perchè con la pipeline le istruzioni vengono eseguite una dopo l'altra e bisogna aspettare la fine dell'esecuzione dell'istruzione di branch prima di sapere se eseguire le istruzioni dopo.

Essendo che una branch è "riconosciuta" solo nella fase di MEM, ogni qualvolta ci sia un'istruzione con salto condizionato la CPU dovrebbe bloccarsi di 3 cicli di clock.



Questo, però, non avviene con le Jump (a parte che non è istruzione di salto condizionato). Con le Jump potrebbero esserci dei data Hazard.

```
add $t2,$t2,$t0
add $t5,$t7,$t3
```

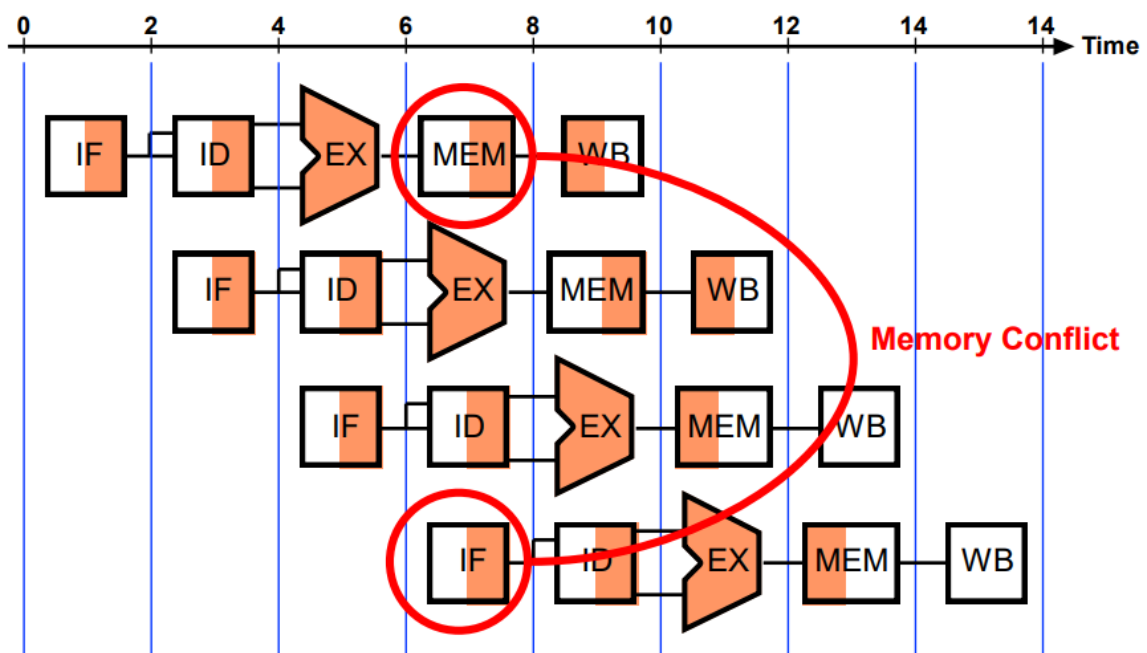
Questo invece calcola ciò che sta sotto il branch a prescindere dal fatto che l'esecuzione del branch sia finita o no. Solo quando viene finita allora se è giusto che doveva calcolare ciò che stava sotto continua a farlo. Altrimenti "butta" via tutto e fa il salto.

Branch Prediction

- Resolve branch hazards by assuming a given outcome and proceeding without waiting to see the **actual** branch outcome
- 1. **Predict not taken** – always predict branches will **not** be taken, continue to fetch from the sequential instruction stream, only when branch is taken does the pipeline stall
 - **If taken, flush instructions in the pipeline after the branch**
 - in IF, ID, and EX if branch logic in MEM – **three stalls**
 - in IF if branch logic in ID – **one stall**
 - **ensure that those flushed instructions haven't changed machine state**– automatic in the MIPS pipeline since machine state changing operations are at the tail end of the pipeline (MemWrite or RegWrite)
 - **restart the pipeline at the branch destination**

STRUCTURAL HAZARDS

Due o più istruzioni necessitano della stessa componente del datapath (ALU, MEM ...). Si genera dunque un conflitto di memoria.

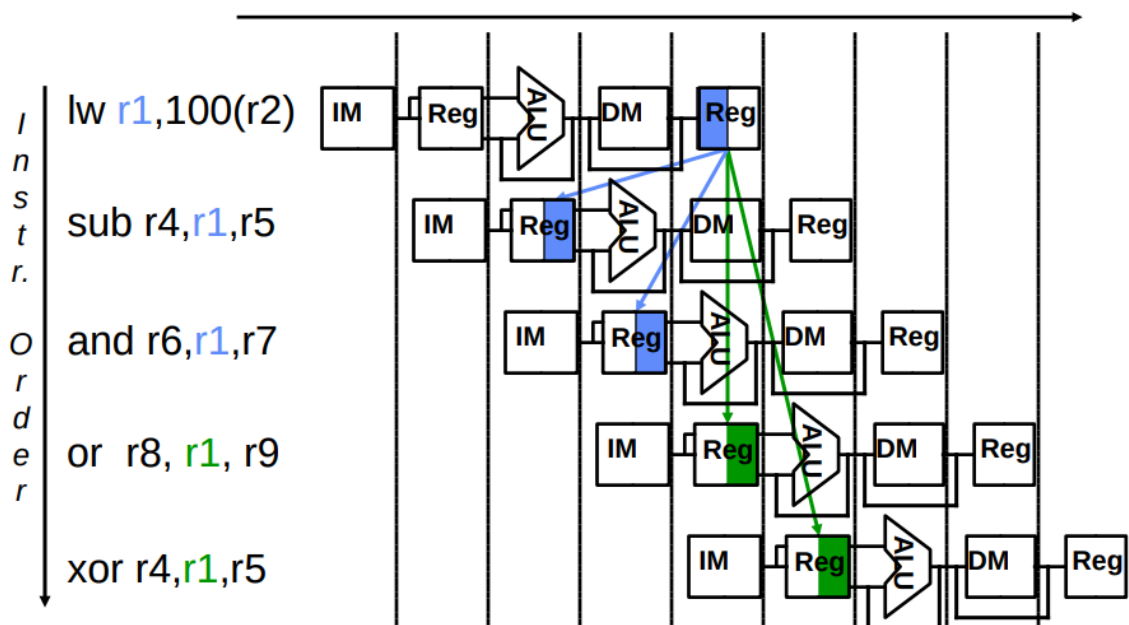


Esistono infatti delle soluzioni per risolvere gli hazards.
Le tecniche hardware per risolvere gli hazards sono principalmente due:

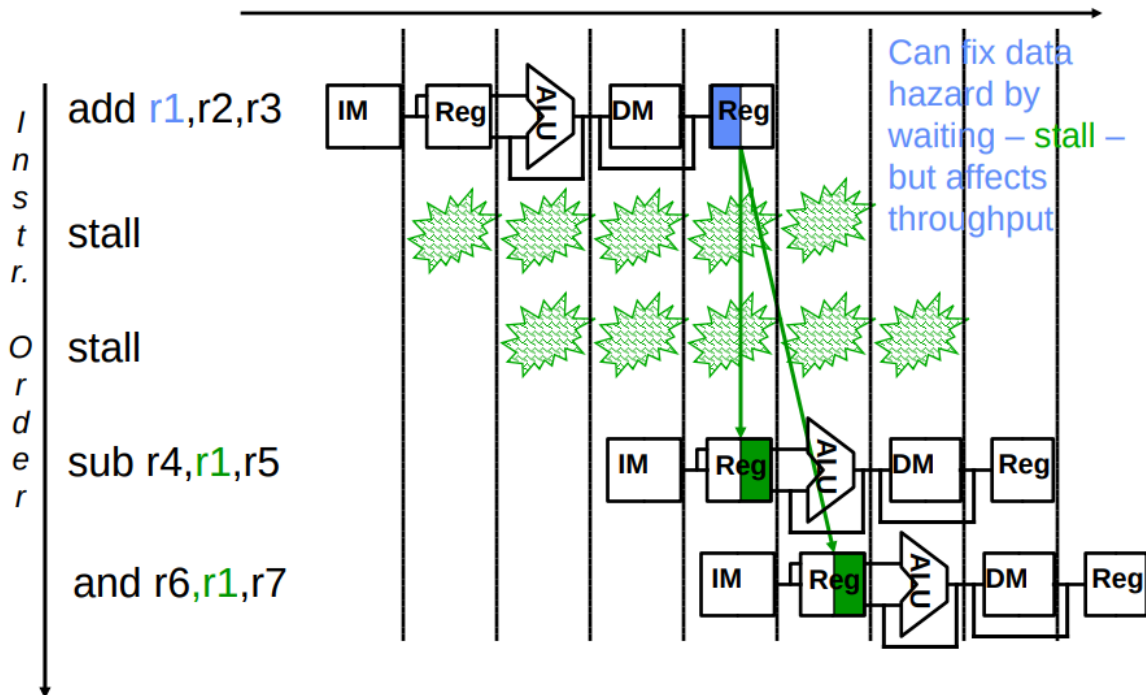
- STALLI
- FORWARDING
- DIVISIONE DEL CLOCK

STALLI - NOP (No Operation)

Soluzione più facile. Si può sostanzialmente applicare a qualunque tipo di errore. L'unico problema è che spreca tempo.



Soluzione:

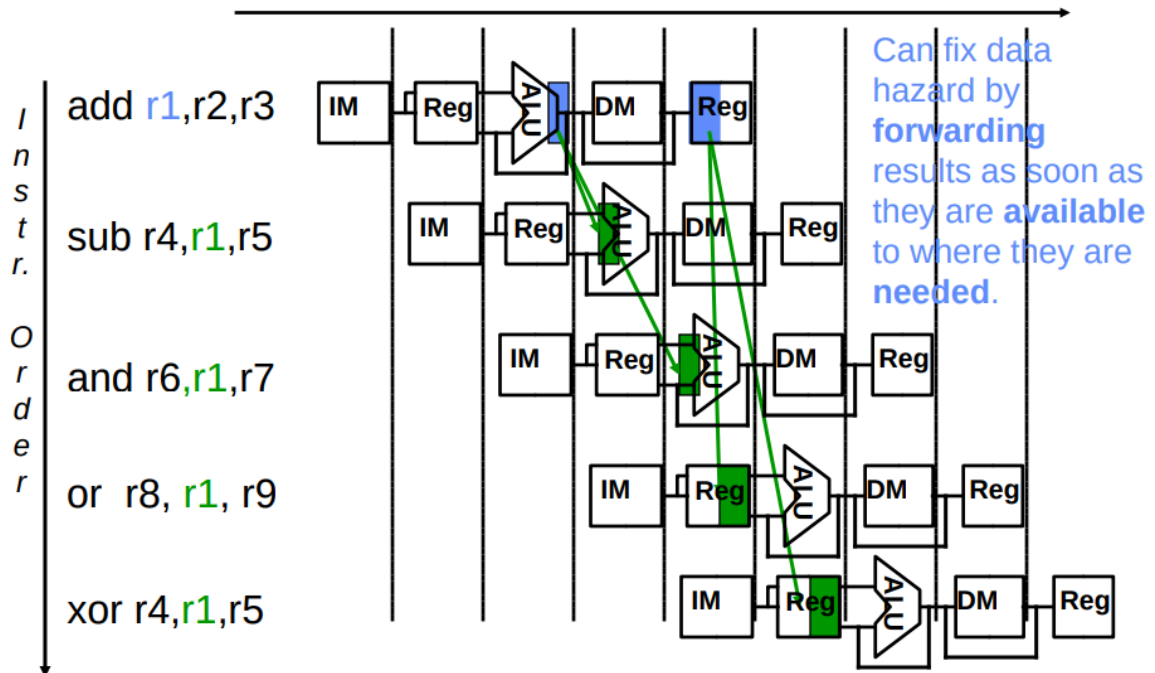


FORWARDING

Il forwarding è una tecnica particolare. Invece che aspettare che un risultato venga salvato in memoria, lo passo direttamente all'istruzione successiva.

Ovvero il forwarding consiste nel collegare l'output dell'ALU con l'input dell'ALU della prossima istruzione. Attenzione, non è detto che le "ALU" debbano trovarsi per forza una dietro l'altra. È possibile "saltare" anche per più cicli di clock.

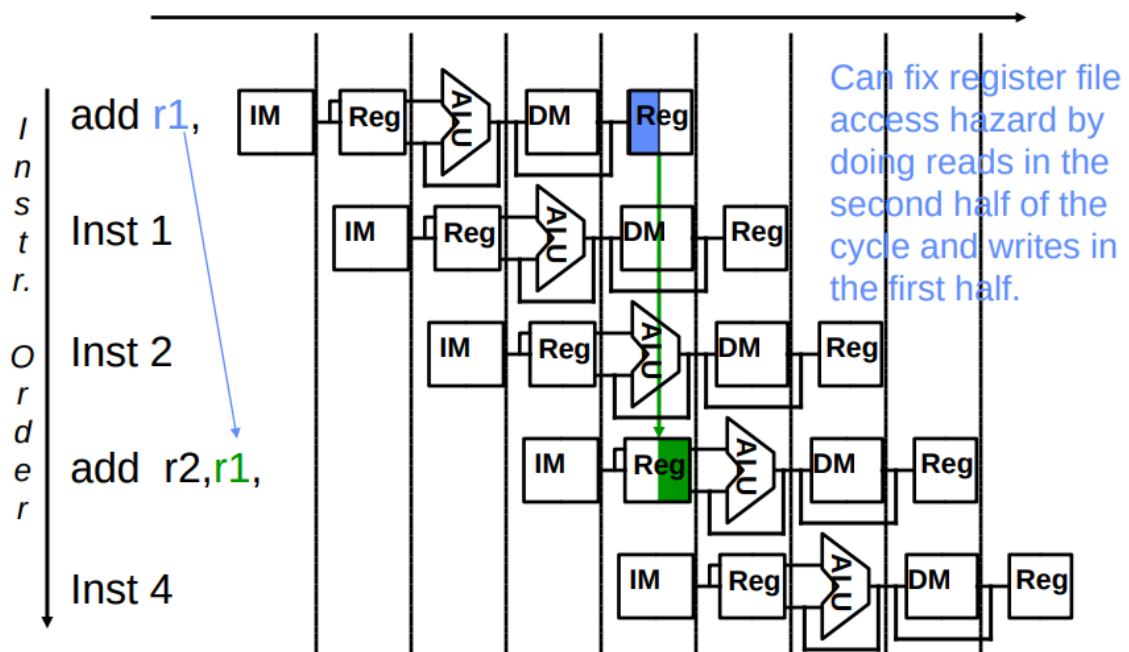
Another Way to “Fix” a Data Hazard



DIVISIONE DEL CLOCK

È la soluzione migliore se è possibile usarla. Sfrutta la proprietà **level-triggered**, questa proprietà infatti permette di eseguire operazioni quando il clock scende e quando il clock sale. Dunque nella prima metà di un ciclo di clock posso per esempio aggiornare e nell'altra posso leggere il dato che ho appena aggiornato.

Questo rende la divisione del clock utile sicuramente per i STRUCTURAL HAZARDS e anche per gli altri hazards, ma solo se le condizioni lo permettono.



Domande


Una lunga sequenza di istruzioni, priva di salti condizionati, da luogo ad una migliore o peggiore performance in una cpu con pipelining?

Aldilà delle ovvie migliorie in termini di velocità che derivano dalla pipeline, la mancanza di salti condizionati permette una performance migliore perchè ogni volta che in un sistema con pipeline si verifica un salto potrebbe verificarsi un Control Hazard. ogni volta che si verifica un salto la pipeline va svuotata per permettere l'acquisizione dei nuovi dati contenuti nel blocco a cui si è saltato. Questo spreca dei cicli di clock.

Il pipelining consente ad un maggior numero di istruzioni di raggiungere il completamento per unità di tempo ma non diminuisce il tempo perchè un'istruzione venga completata?

Ovvio, il pipeline non rende l'esecuzione delle singole istruzioni più veloci ma rende più veloce l'esecuzione complessiva di un blocco di istruzioni in quanto vengono eseguita quasi contemporaneamente.

Fammi 3 esempi di DATA HAZARDS.



I "data hazards" (pericoli di dati) si verificano quando ci sono dipendenze di dati tra le istruzioni che possono causare conflitti nell'accesso o nell'utilizzo dei dati. Ci sono tre tipi comuni di data hazards:

1. Data Hazard di lettura-dopo-scrittura (Read-After-Write Hazard): Si verifica quando un'istruzione legge un registro dopo che un'istruzione precedente ha scritto nello stesso registro. Ad esempio:
Istruzione 1: ADD R1, R2, R3
Istruzione 2: SUB R4, R1, R5
Qui, l'istruzione 2 dipende dal risultato dell'istruzione 1, che scrive nel registro R1. Se non viene gestito correttamente, questo hazard può causare risultati errati.
2. Data Hazard di scrittura-dopo-scrittura (Write-After-Write Hazard): Si verifica quando due istruzioni cercano di scrivere nello stesso registro contemporaneamente. Ad esempio:
Istruzione 1: ADD R1, R2, R3
Istruzione 2: SUB R1, R4, R5
Entrambe le istruzioni cercano di scrivere nel registro R1. Se non viene gestito correttamente, l'output finale può essere imprevedibile.
3. Data Hazard di scrittura-dopo-lettura (Write-After-Read Hazard): Si verifica quando un'istruzione cerca di scrivere in un registro mentre un'istruzione successiva cerca di leggerlo. Ad esempio:
Istruzione 1: ADD R1, R2, R3
Istruzione 2: SUB R4, R5, R1
Qui, l'istruzione 2 legge il valore dal registro R1 mentre l'istruzione 1 potrebbe non aver ancora scritto il suo risultato. Se non viene gestito correttamente, l'istruzione 2 potrebbe leggere un valore errato.

Regenerate response

BOH altra roba

23) A cosa servono i selettori che fanno sì che un dato venga memorizzato o meno:

I selettori consentono di evitare scritture accidentali dei dati. Si pensi, ad esempio, che qualsiasi valore abbiano i bit in ingresso se ad ogni ciclo di clock essi venissero scritti si perderebbero tutte le informazioni.

30) Perché un datapath multi-ciclo risulta più veloce di uno a ciclo singolo?

Perché in un datapath a ciclo singolo il clock è determinato dal tempo di esecuzione della più lunga delle istruzioni (LW) mentre invece in un multi-ciclo le istruzioni vengono spezzate su più cicli di clock (ma con un periodo notevolmente più basso rispetto al ciclo singolo).

23) Facendo riferimento alla CPU multiciclo vista a lezione, riportata per comodità in figura, e alla relativa logica di controllo per la implementazione delle istruzioni add, and, beq, j, lw, or, slt, sub, sw, si chiede quali siano le istruzioni durante la cui realizzazione (fetch + execute) alla ALU sia richiesto, attraverso i suoi controlli in ingresso, di effettuare esattamente due somme (e non tre).

Le istruzioni che richiedono di effettuare esattamente 2 somme alla ALU sono: add, lw, sw, beq, j

consideriamo una implementazione dell'architettura MIPS con pipeline a 5 stadi (fetch, decode, execute, read/write, register write); date le seguenti istruzioni:

```
...  
beq $t0, $t1, dopo  
add $t2, $t1, $t0  
j loop  
...  
dopo: ...
```

una soluzione che affronta le alee di controllo mediante esecuzione predittiva dei salti condizionati (non presi per default) ed eventuale annullamento degli effetti delle due istruzioni successive, può funzionare in questo caso ?

Scegli un'alternativa:

- ☒ a. no, perchè l'effetto della jump non è annullabile ✓
- ☐ b. sì, perchè l'effetto delle jump è ovviamente annullabile
- ☐ c. no, perchè comunque il registro \$t0 deve venire utilizzato nella istruzione intermedia

ma che vuol dire???