Lezione 23

CPU pipeline II: struttura, criticità (*hazards*)

Proff. A. Borghese, F. Pedersini

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

Criticità (hazards) – Sintesi

- **Criticità strutturali**
  - **Necessità della stessa unità funzionale più volte, nello stesso passo.**
  - Le unità funzionali non sono in grado di supportare le istruzioni (nelle diverse fasi) che devono essere eseguite in un determinato ciclo di clock.
- **Criticità di Dato**
  - Dovrei eseguire un'istruzione in cui uno dei dati è il risultato dell'esecuzione di un'istruzione precedente.
  - `lw  $t0, 16($s1)`
  - `add $t1, $t0, $s0`

Criticità strutturali: esec. istruzioni `lw`

Passo esecuzione: `lw`	ALU			Memoria		Register File
	ALU	PC	branch	Dati	Istruzioni	
IF (Fase fetch)	No	Yes	No	No	Yes	No
ID (Decodifica)	No	No	No	No	No	Yes
EX (Esecuzione)	Yes	No	No	No	No	No
MEM (Acc memoria)	No	No	No	Yes	No	No
WB (riscrittura)	No	No	No	No	No	Yes

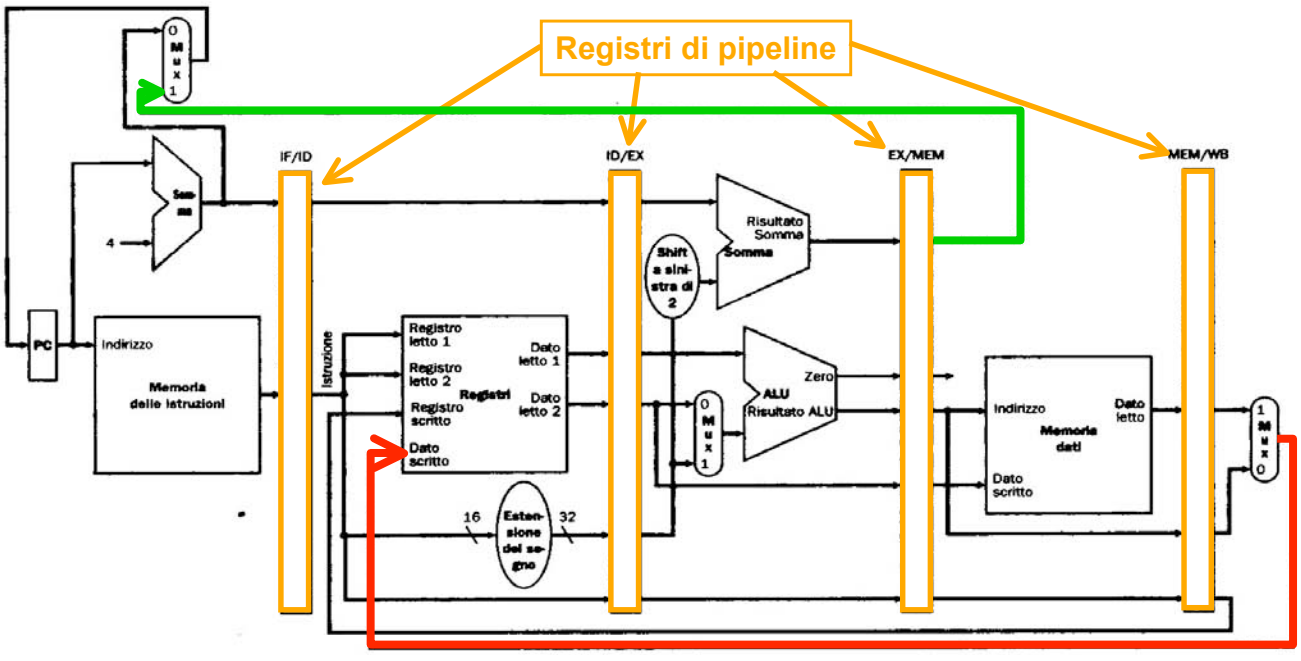
`lw $t0, 8($t2)`	ALU (PC) Mem	RF	ALU	Mem	RF		
`lw $t1, 12($t2)`		ALU (PC) Mem	RF	ALU	Mem	RF	
`lw $t1, 16($t2)`			ALU (PC) Mem	RF	ALU	Mem	RF
`beq $1, $0, +16`				ALU-PC Mem	ALU (addr.)	...	...

- Presenza di **criticità strutturali: ALU, MEMORIA**
- Soluzione: **replicazione** delle risorse

  ALU / ALU (PC) / ALU (address) Memoria DATI / ISTRUZIONI

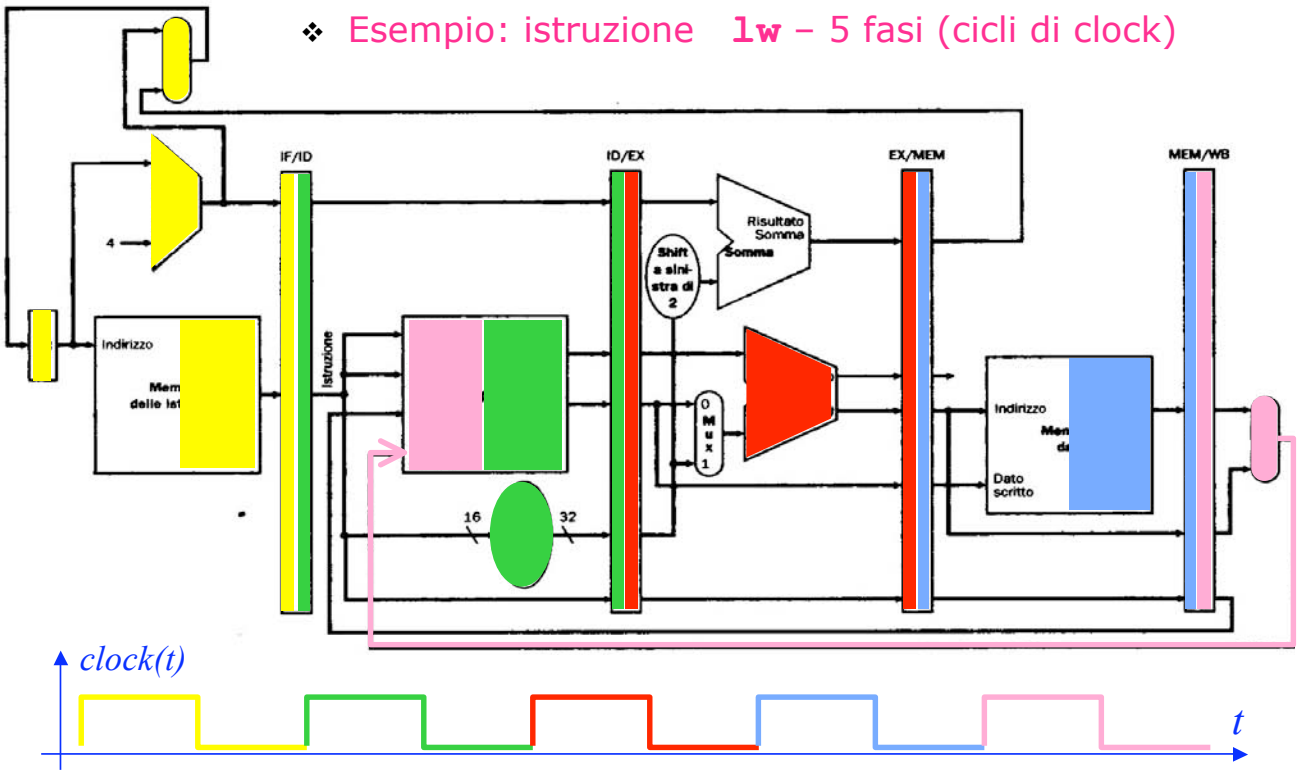
Struttura CPU pipeline

- Duplicazione **memoria** (**Dati / Istruzioni**)
- Triplicazione **ALU** (**ALU, ALU-PC, ALU-address**)



Funzionamento CPU pipeline

- Esempio: istruzione **`lw`** – 5 fasi (cicli di clock)



Segnali di controllo – CPU pipeline

Segnale	Effetto quando è negato (0)	Effetto quando è affermato (1)
RegDst	Il numero del registro destinazione proviene dal campo **`rt`** (bit 20-16)	Il numero del registro destinazione proviene dal campo **`rd`** (bit 15-11)
ALUSrc	Il secondo operando della ALU proviene dalla seconda uscita in lettura del RF	Il secondo operando della ALU è la versione estesa (con segno) del campo offset
Branch	Il valore del PC viene sostituito dall'uscita del sommatore che calcola **PC + 4** (condizionato all'uscita di ALU)	Il valore del PC viene sostituito dall'uscita del sommatore che calcola l'**indirizzo di salto** (condizionato all'uscita di ALU)
MemtoReg	Il valore inviato all'ingresso Dato al RF proviene dalla ALU	Il valore inviato all'ingresso DatoScritto al RF proviene dalla memoria
MemRead	Nessuno	Il contenuto della cella di memoria dati indirizzata dal MAR è posto nel MDR
MemWrite	Nessuno	Il contenuto in ingresso al MDR viene memorizzato nella cella il cui indirizzo è caricato nel MAR
RegWrite	Nessuno	Nel registro specificato a #RegWrite viene scritto il valore presente all'ingresso DatoScritto

- Scrittura di PC e dei registri di pipeline avviene ad ogni fronte di clock

CPU pipeline: progetto unità di controllo

Unità di controllo – CPU pipeline:

Ipotesi:

- Ogni istruzione necessita dei propri segnali di controllo
- Ogni segnale di controllo è utilizzato in una sola fase
- Segnali per fase EX, MEM o WB

Soluzione:

- una volta generati i segnali di controllo di un'istruzione...
  - nella fase di decodifica
- ...li trasporto attraverso la pipeline insieme all'istruzione stessa
  - nelle fasi successive (EX, MEM, WB)

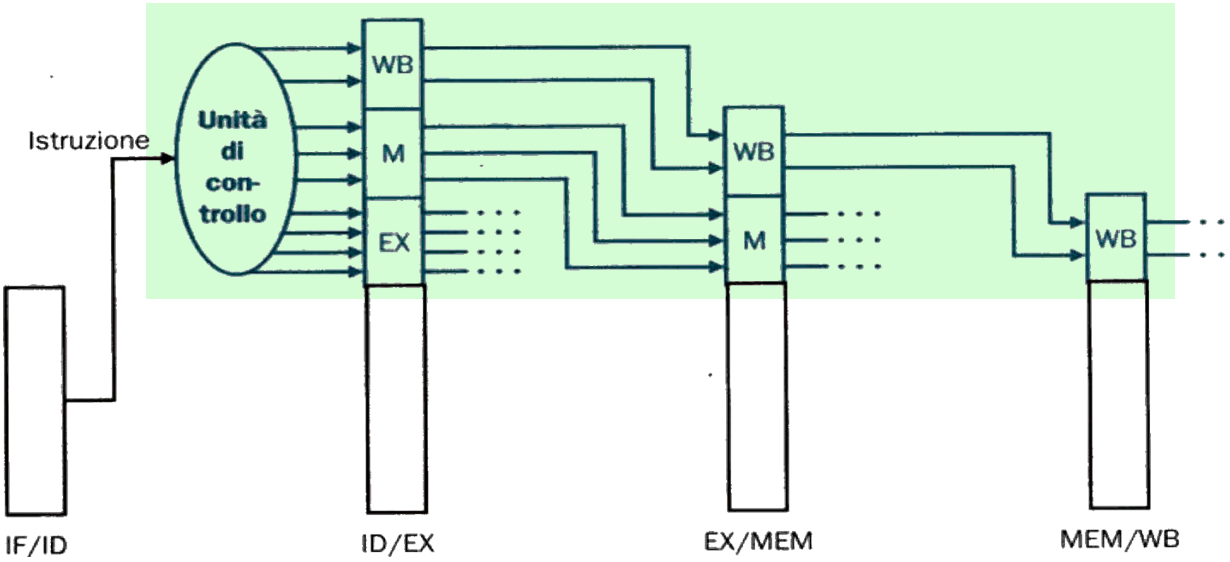
Osservazioni

- I segnali di controllo particolari (legati alle diverse istruzioni) si possono così raggruppare:
  - Nelle fasi di fetch e decodifica non esistono segnali di controllo
  - Il contenuto di **rt** ed il numero di **rd** vengono portati attraverso i vari stadi

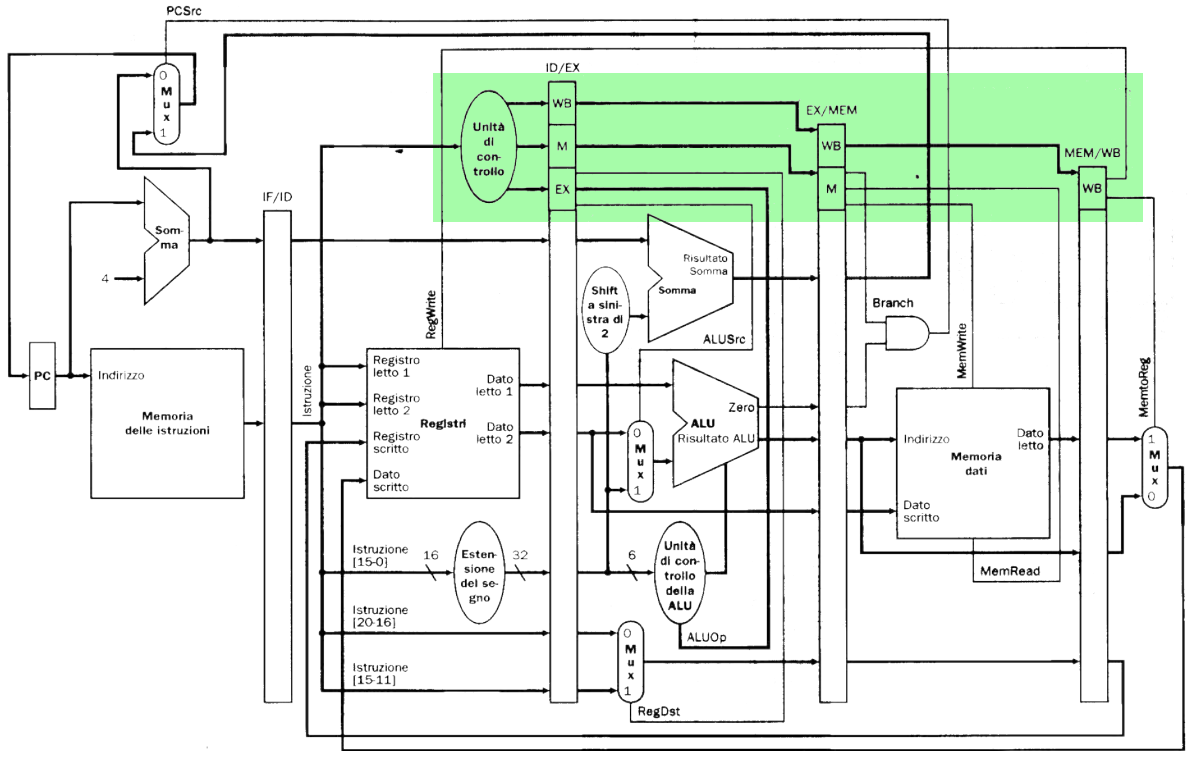
Fase →	Exec				Memory			WB	
Segnali → ↓ Istruz.	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem-Read	Mem-Write	Reg-Write	MemTo-Reg
Tipo R	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	x	0	0	1	0	0	1	0	x
beq	x	0	1	0	1	0	0	0	x

Generazione dei segnali di controllo

- I segnali di controllo vengono generati nello stadio di decodifica e poi propagati agli stadi successivi.



UC per CPU Pipeline



Sommario

- Struttura CPU Pipe-line
- Gestione delle criticità
  - Hazard nei dati

Criticità nella CPU

- **Criticità strutturali:**
  - risolte con la duplicazione (suddivisione) delle unità funzionali
  - **3 ALU, 2 memorie**, come nella CPU singolo ciclo → criticità risolta!

- **Criticità di dati:** Devo eseguire un'istruzione in cui uno dei dati è il risultato dell'esecuzione di un'istruzione precedente
  - I prelievi **MEM** e **WB** portano informazioni a ritroso → possibile causa di problemi.
  - prelievo nello stadio **WB** del risultato dell'operazione
- *Soluzione mediante due tecniche:*
  - SW: Riorganizzazione del codice (compilatore).
  - **HW: Propagazione *(forwarding)***

Esempio: criticità di dati – istruzioni A/L tipo R

- Il dato in **$2** diviene disponibile nel RF nella fase **WB** della **sub**.
  - Non è ancora pronto quando viene effettuata la decodifica della **and** e della **or** successive.
- Hazards: tra **sub** e **and** e tra **sub** e **or**
  - Con le frecce sono indicate le dipendenze, in rosso gli hazard

`sub $2, $1, $3`	IF	ID	EX $1–$3	MEM	WB s→$2				
`and $12, $2, $5`		IF	ID	EX $2 **and** $5	MEM	WB s→$12			
`or $13, $6, $2`			IF	ID	EX $6 **or** $2	MEM	WB (s→$13)		
`add $14, $2, $2`				IF	ID	EX $2+$2	MEM	WB s→$14	
`sw $15, 100($s2)`					IF	ID	EX $2+100	MEM $15→Mem	WB

Soluzione *SW*: riorganizzazione del codice

Soluzioni:

- IDEA: inserimento **nop**

```
sub $2, $1, $3
nop
nop
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($s2)
```

  - **Spreco di 2 cicli di clock**, in modo che la fase **ID** dell'istruzione:
    `and $12, $2, $5`
    vada a coincidere con la fase di **WB** della:
    `sub $2, $1, $3`
- Situazione troppo frequente perché la soluzione sia accettabile.

Soluzione **HW**: **forwarding**

`sub $2, $1, $3`	IF	ID	EX $1–$3	MEM	**WB s→$2**				
`and $12, $2, $5`		IF	ID	EX **$2 and** $5	MEM	WB s→$12			
`or $13, $6, $2`			IF	ID	EX $6 **or** $2	MEM	WB (s→$13)		
`add $14, $2, $2`				IF	ID	EX $2+$2	MEM	WB s→$14	
`sw $15, 100($s2)`					IF	ID	EX $2+100	MEM $15→Mem	WB

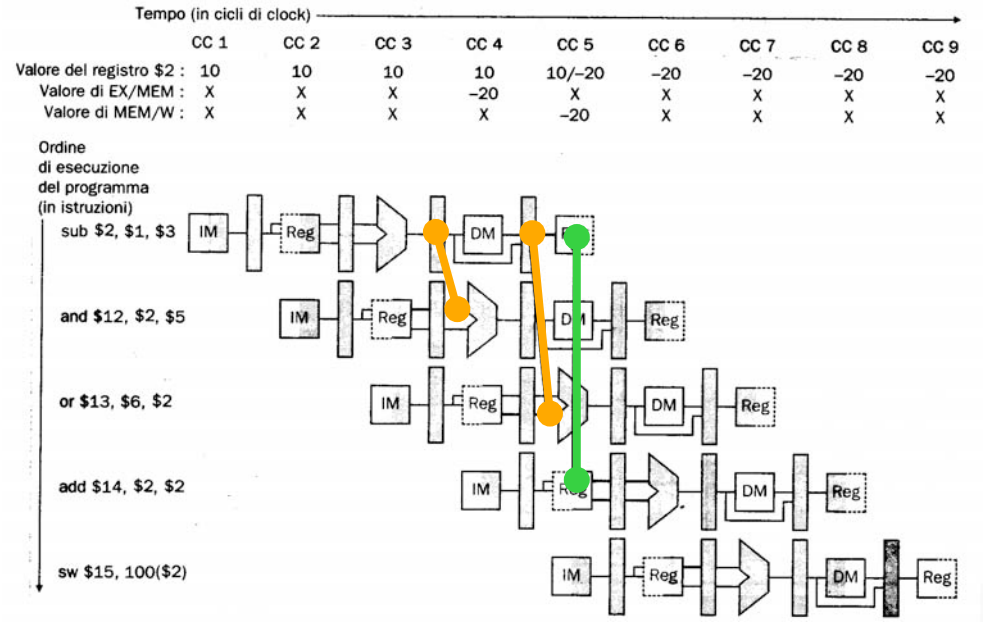
- Prendo il risultato della sottrazione all'inizio dello stadio **MEM** (per le istruzioni R, lo stadio M è uno stadio idle): `$1-$3` è già contenuto nel registro EX/MEM e si trova all'uscita nella fase bassa del clock.
- Sovrascrivo il primo operando della **`and`** successiva (`$2`) con il risultato della sottrazione eseguita in precedenza (contenta all'uscita del registro EX/MEM), senza attendere la fase di **WB**.

Hazard tra **`sub`** e **`or`**: **forwarding**

`sub $2, $1, $3`	IF	ID	EX $1–$3	MEM	**WB s→$2**				
`and $12, $2, $5`		IF	ID	EX $2 **and** $5	MEM	WB s→$12			
`or $13, $6, $2`			IF	ID	EX $6 **or** **$2**	MEM	WB (s→$13)		
`add $14, $2, $2`				IF	ID	EX $2+$2	MEM	WB s→$14	
`sw $15, 100($s2)`					IF	ID	EX $2+100	MEM $15→Mem	WB

- Prendo il risultato della sottrazione (**`$1-$3`**) all'inizio dello stadio **WB**, che è presente all'uscita del registro **MEM/WB** nella fase bassa del clock.
- Sovrascrivo il primo operando della **`and`** successiva (`$2`) con il risultato della sottrazione eseguita in precedenza (presente all'uscita del registro MEM/WB), senza attendere la fase di **WB**.

Hazard dati: **Forwarding**



Forwarding: si invia il **risultato dell'istruzione precedente** ad un **passo intermedio dell'istruzione attuale** (in pipe).

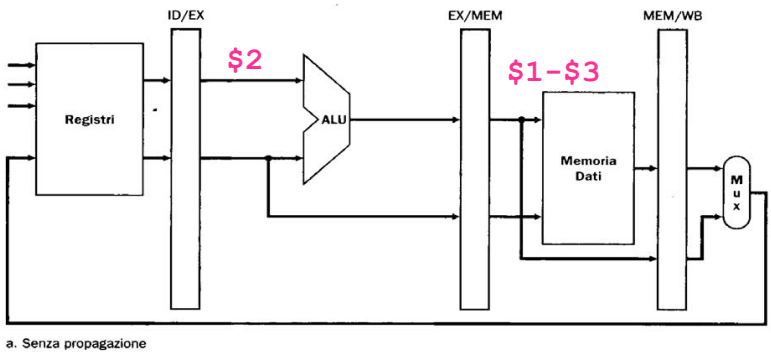
- Il contenuto dei registri **EX/MEM** e **MEM/WB** anticipa il contenuto del Register File.

*Forwarding e contenuto del registro **ID/EX***

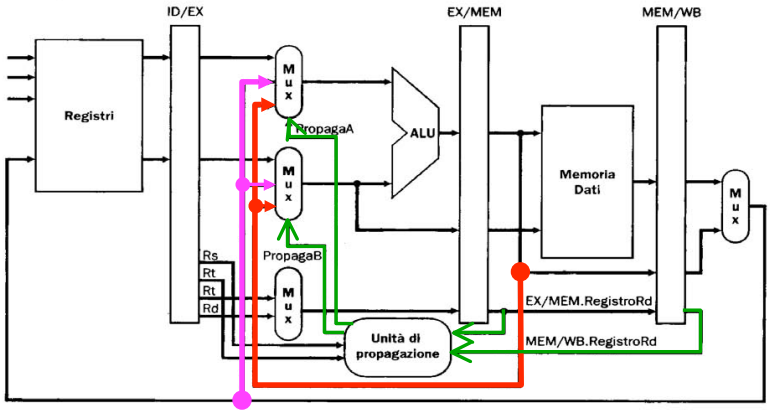
- Nel normale funzionamento, il registro **ID/EX** contiene quanto letto dal Register File.
- Quando abbiamo forwarding, quello che viene letto dal registro **ID/EX** nella fase di esecuzione deve essere sovrascritto da quanto letto dal registro **EX/MEM** o **MEM/WB**.
- Nel registro **EX/MEM** è contenuto il risultato dell'operazione eseguita all'istante precedente.
- Nel registro **MEM/WB** è contenuto il risultato dell'operazione eseguita 2 istanti precedenti.

Hazard nei dati, **progetto soluzione HW**

- Datapath:
  - collegamenti da **ALU:out** ad ALU:in
  - e da MEM:out a ALU:in
- Controllore:
  - Unità di propagazione



a. Senza propagazione



Soluzione architetturale per **AND**

Hazard su AND:

- Occorre implementare la seguente funzione logica nella fase ID dell'esecuzione dell'istruzione:

```
t-1:   sub $2, $1, $3
t:     and $12, $2, $5
```

> **IF (RD_{t-1} == RS_t)**
> **then "collega": RD_{t-1} → RS_t**
> **IF (RD_{t-1} == RT_t)**
> **then "collega": RD_{t-1} → RT_t**

- $RD_{t-1} = RD_{sub} = \$2$
  è il contenuto del registro destinazione dell'istruzione precedente la `and` (`sub $2, $1, $3`) contenuto nel registro EX/MEM

Soluzione architetturale per **OR**

Hazard su OR:

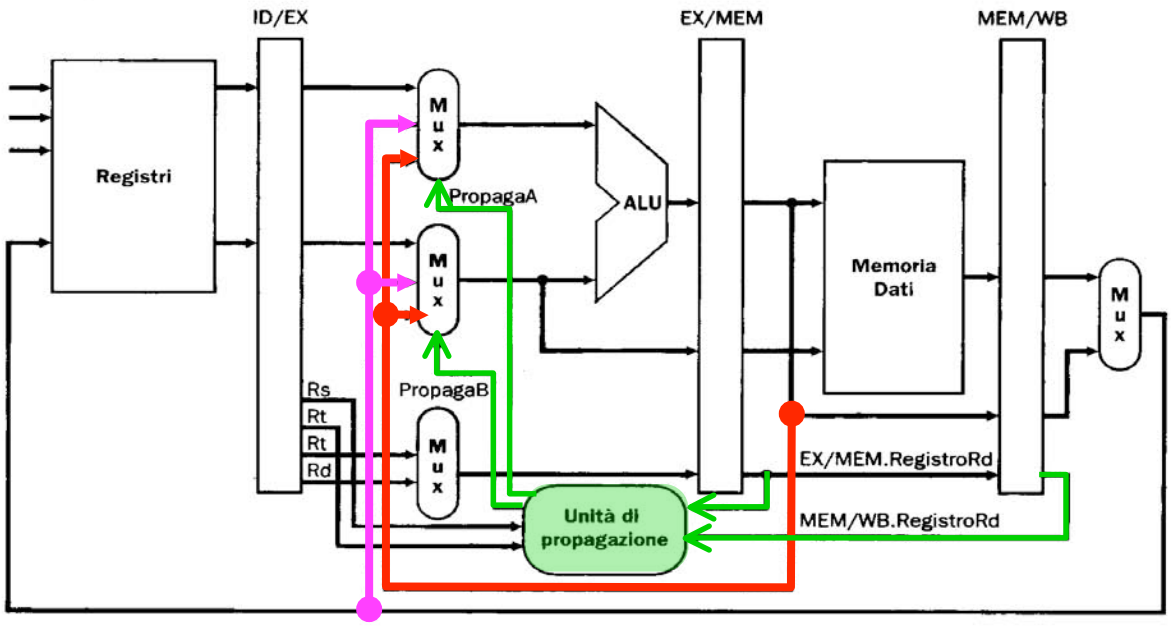
- Occorre implementare la seguente funzione logica nella fase ID dell'esecuzione dell'istruzione:

```
t-2:   sub $2, $1, $3
t-1:   ...
t:     or $13, $6, $2
```

> if (RD_{t-2} = RS_t)
> then "collega" RD_{t-2} con RS_t
> if (RD_{t-2} = RT_t)
> then "collega" RD_{t-2} con RT_t

- $RD_{t+2} = RD_{sub} = \$2$
  è il contenuto del registro destinazione di
  2 istruzioni precedenti la `or:` (`sub $2, $1, $3`)
  che è copiato nella fase MEM.

CPU con forwarding: schema circuitale



Controllo MUX ingresso alla ALU

- Nuovi segnali di controllo: PropagaA, PropagaB
  - generati dall'unità di propagazione

Controllo Multiplexer	Registro Sorgente	Funzione
PropagaA = 00	ID/EX	Il primo operando della ALU proviene dal Register File
PropagaA = 10	EX/MEM	Il primo operando della ALU è propagato dal risultato della ALU per l'istruzione precedente.
PropagaA = 01	MEM/WB	Il primo operando della ALU è propagato dalla memoria o da un'altra istruzione precedente.
PropagaB = 00	ID/EX	Il secondo operando della ALU proviene dal Register File
PropagaB = 10	EX/MEM	Il secondo operando della ALU è propagato dal risultato della ALU per l'istruzione precedente.
PropagaB = 01	MEM/WB	Il secondo operando della ALU è propagato dalla memoria o da un'altra istruzione precedente.

Data Hazard: soluzioni

Soluzioni DATA HAZARD:

SW: Buona scrittura del codice

- il programmatore deve conoscere la macchina per scrivere un buon codice!

- Compilatore efficiente
  - che riordini il codice

HW: FORWARDING: Architettura che renda disponibile i dati appena pronti alla fase di esecuzione.

- Inserire una **nop**
- Accettare uno stallo
  - non sempre si può evitare…