Lezione 24

CPU pipeline – 3
criticità di dati e di controllo

Proff. A. Borghese, F. Pedersini

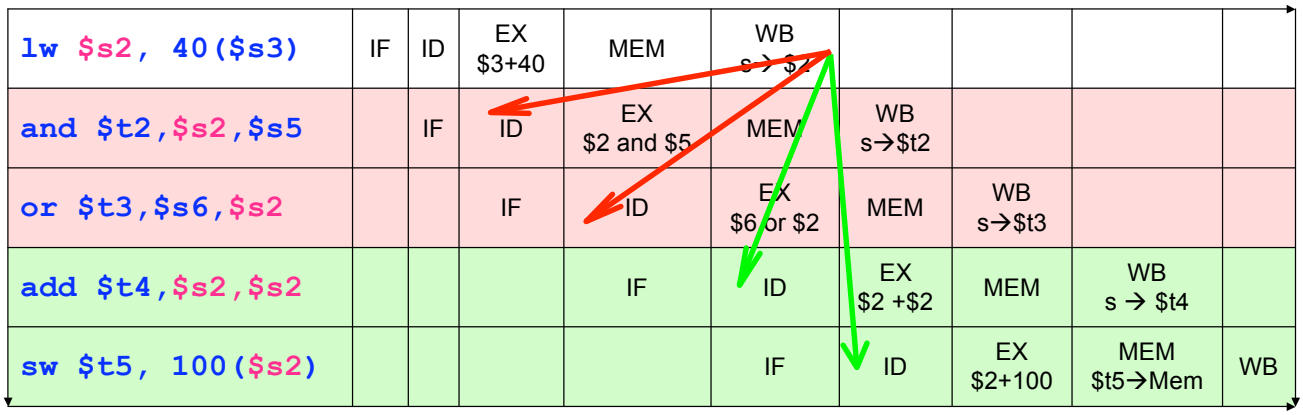
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano

Sommario

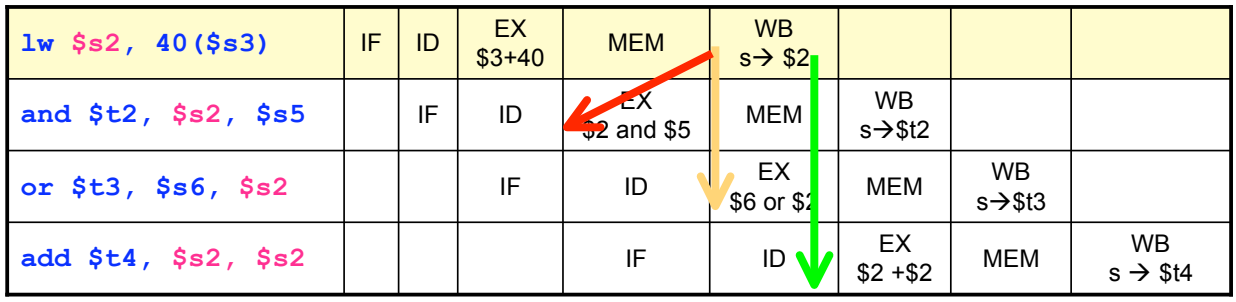
- Criticità nei dati – istruzione `lw`
- Criticità nei branch – Branch hazard

Hazard sui dati: `lw`

- Diagramma delle Dipendenze
  - in verde le dipendenze gestite regolarmente
  - in rosso le criticità (hazards) di dato
- Il dato corretto è pronto nella `lw` alla fine della MEM
  - mentre nel caso di istruzione tipo R era pronto alla fine di EX

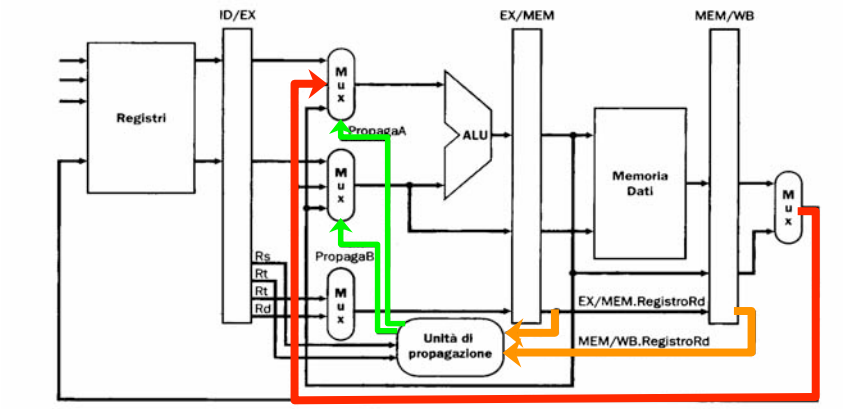


Hazard sui dati: `lw` – *forwarding*



- Il dato corretto per `$s2` è pronto nella `lw` solamente alla fine della **MEM**
  - utilizzabile solamente dall'inizio della fase di WB
- Rilevo la criticità su `or` alla fine della sua fase di **ID**.
  In questo caso il dato corretto si trova all'inizio della fase WB della `lw`
  - Posso risolvere la criticità con il **FORWARDING**
- Rilevo la criticità su `and` alla fine della sua fase di **ID**.
  In questo caso il dato corretto non è ancora stato prodotto dalla `lw`
  - NON posso risolvere la criticità → **STALLO**

Soluzione criticità `or` con forwarding



Risolvo la criticità per la `or` mediante propagazione (forwarding)

- **Datapath:** non cambia rispetto al caso precedente
- **UC** (unità di propagazione): prelievo dalla fase **WB**

```
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
```


Hazard sui dati – `lw`: *stallo*

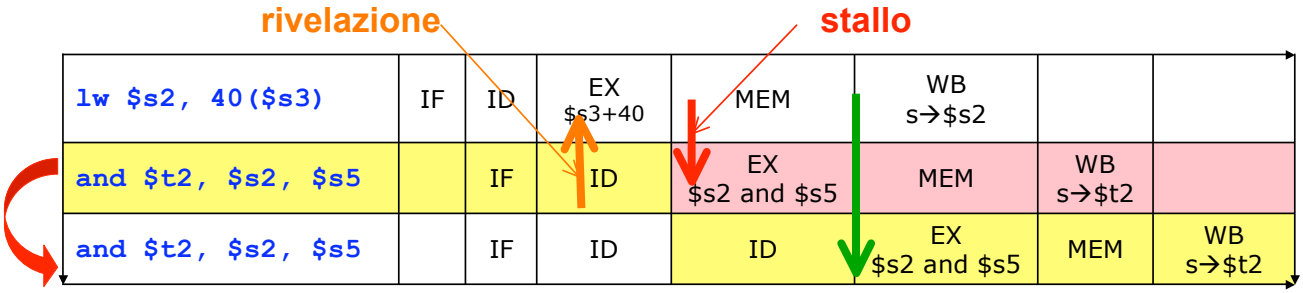
`lw $s2, 40($s3)`	IF	ID	EX $s3+40	MEM	WB s→$s2		
`and $t2, $s2, $s5`		IF	ID	EX $s2 and $s5	MEM	WB s→$t2	
`and $t2, $s2, $s5`			IF	ID	EX $s2 or $s5	MEM	WB s→$t2

- Il dato corretto per `$s2` è pronto nella `lw` solamente alla fine della fase **MEM**, ed è perciò utilizzabile solamente a partire dall'inizio della fase di **WB**
- **Soluzione: 1 ciclo di STALLO + FORWARDING**

  → Stallo della pipeline!

  - Devo bloccare l'esecuzione della `and` e ripeterla un ciclo dopo
    - solo quando è possibile utilizzare il valore corretto del registro `$s2`

Rivelazione della criticità su `lw`



- Devo rivelare la criticità prima possibile, in modo da mettere in stallo in tempo la pipeline
- Il primo momento possibile è:
  - stadio di decodifica (ID) dell'istruzione **`and`**
  - Istruzione **`lw`** (t–1) in stadio di esecuzione (EX)

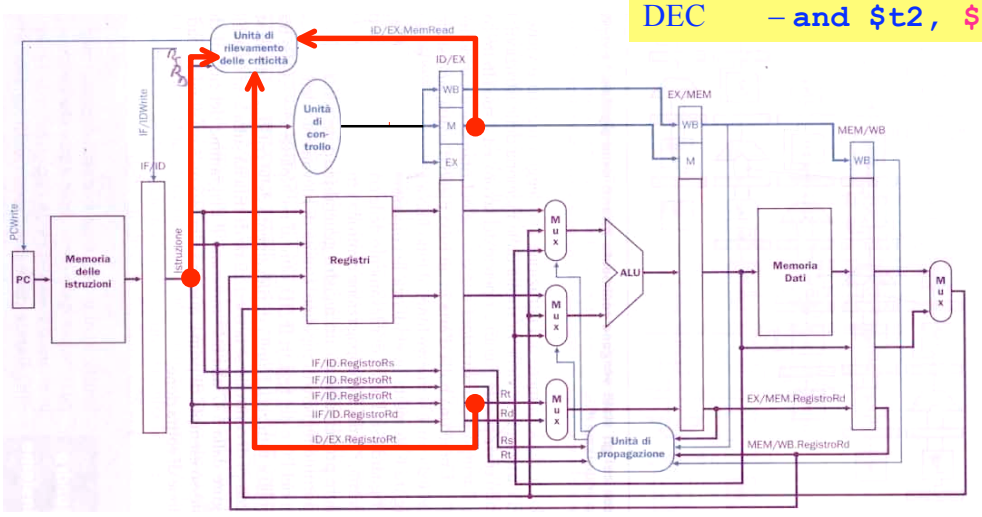
Rivelazione della criticità su `lw`

IF [(ID/EX.MemRead)] → **`lw`** in fase di EX
AND
{[(IF/ID.RegistroRT) == ID/EX.RegistroRD] OR
[(IF/ID.RegistroRS) == ID/EX.RegistroRD]}

*THEN "Metti in **stallo** la pipeline"*

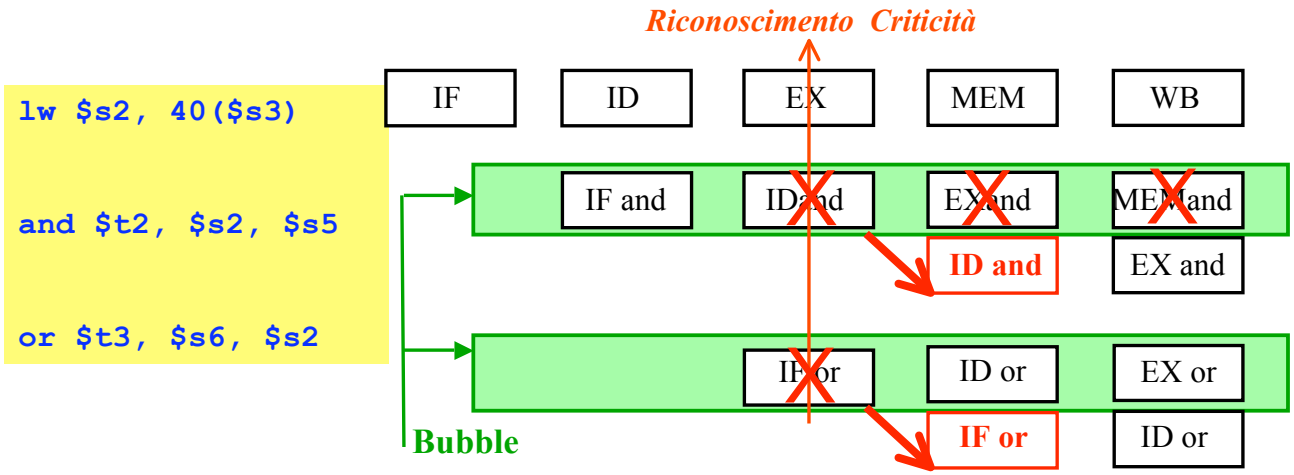
`rd, rt`

EX – **`lw $s2, 40($s3)`**
DEC – **`and $t2, $s2, $s5`**

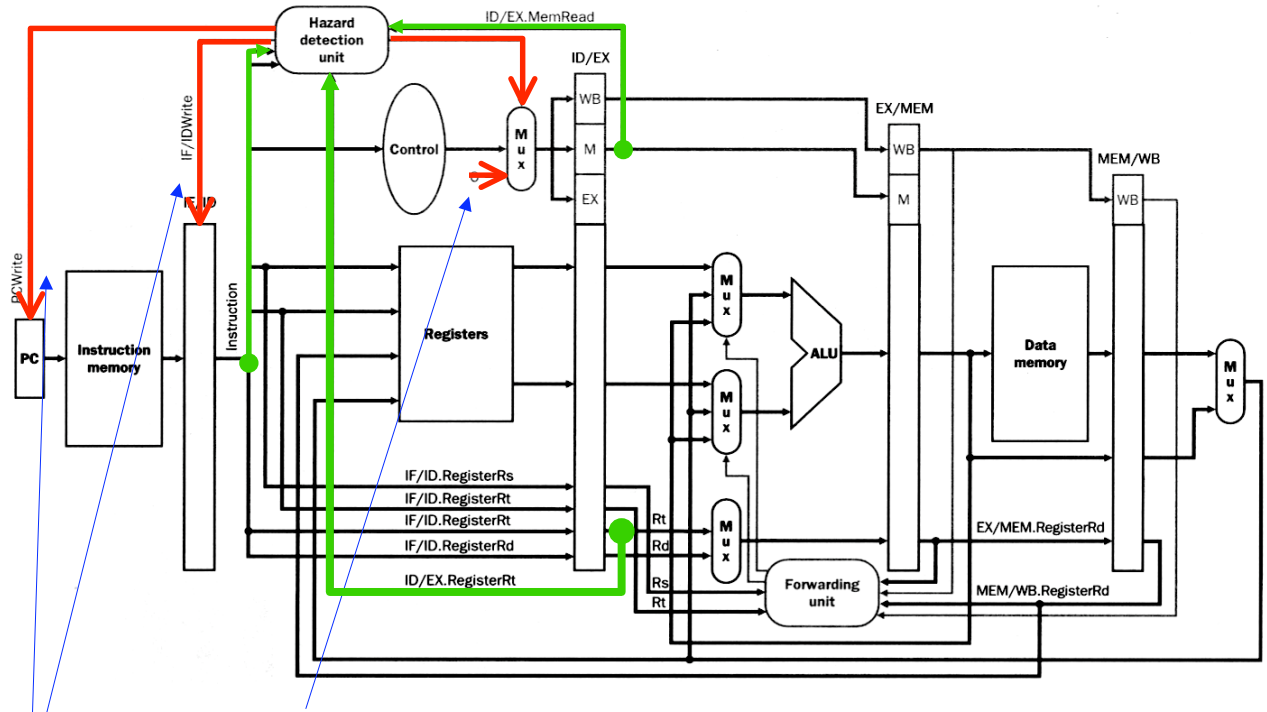


Stallo della pipeline

- **Implementazione dello STALLO** (generazione di una "**BUBBLE**")
  1. **Annullare i segnali di controllo**
     generati nella fase ID per l'esecuzione dell'istruzione successiva alla `lw`
     - Segnali di controllo inattivi ➔ non succede niente
  2. **Ripetere la lettura e la decodifica delle 2 istruzioni successive**
     - Si ripetono le fasi di **fetch(t–1)** e **decodifica(t–2)**



Gestione dello stallo

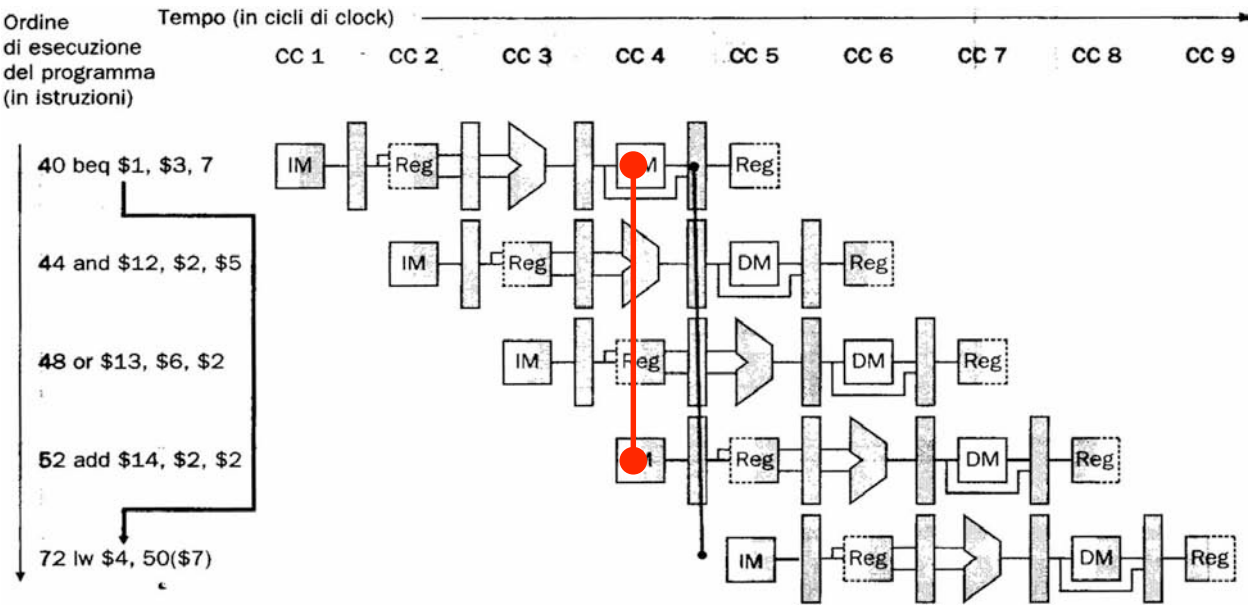


- **Annullamento** dei segnali di controllo associati → formaz. BUBBLE
- **Disabilitazione** della scrittura del **PC** e del registro **IF/ID**

Sommario

- Criticità nei dati – istruzione **`lw`**
- Criticità di salto: Branch hazard

Hazard: istruzione **`lw`** dopo **`branch`**



- Criticità di controllo (Control Hazard)

Criticità di salto (control hazard)

- Criticità di salto:
  - L'indirizzo di salto eventuale è pronto al termine della fase di EX
  - Il Program Counter può venire aggiornato durante la fase di MEM: in questo istante può iniziare il FETCH dell'istruzione all'indirizzo di salto.

    ➜ ho DUE istruzioni da eliminare.

`sub $s2,$s1,$s3`	IF	ID	EX $1–$3	MEM	WB ris.➜ $s2			
`beq $t2,$t6,24`		IF	ID	EX zero IF $t2=$t6	MEM	WB		
`or $t7,$s6,$s7`			IF	ID	EX $s6 or $s7	MEM	WB ris.➜ $t7	
`add $t4,$s8,$s8`				IF	ID	EX $s8+$s8	MEM	WB ris.➜ $t4
`add $t0,$t1,$t2`					IF	ID	EX	MEM

Soluzioni alla criticità nel controllo

- Soluzioni possibili...
- ...per aggirare la criticità,
  - Riordinamento del codice – **Delayed branch**
- ...per minimizzare le conseguenze della criticità
  - Modifiche strutturali della CPU per **anticipazione dei salti**
  - Minimizzazione delle bolle in pipeline dovute ai branch: **Branch Prediction Table**

Soluzioni: Delayed branch

- **Delayed branch:** decisione ritardata
  - ci si affida al compilatore
- Si aggiunge un: "**branch delay slot**"
  - l'istruzione successiva ad un salto condizionato viene sempre eseguita
  - contiamo sul compilatore/assemblatore per mettere dopo l'istruzione di salto una istruzione che andrebbe comunque eseguita indipendente-mente dal salto

Delayed branch: esempio

```
      add $s4, $t0, $t1
      beq $s5, $s6, salto
      add $s0, $s0, $s1
salto:
      add $t5, $t4, $t3
      add $t6, $t7, $t7
```



```
      add $s4, $t0, $t1
      beq $s5, $s6, salto
      add $t5, $t4, $t3
      add $s0, $s0, $s1
salto:
      add $t6, $t7, $t7
```

- L'istruzione `add $t5, $t4, $t3` deve essere comunque eseguita.
- Il salto (se richiesto) avviene dopo.

Gestione con modifiche strutturali della CPU

- Anticipazione dei salti:
  - diminuire il numero di passi necessari per valutare la branch
  - da 3 (fase MEM) a 1 (fase DEC)
- Guessing (branch prediction buffer):
  - la pipeline cerca di indovinare se il salto debba essere eseguito
    - Es: si comporta come se il salto non dovesse essere eseguito
  - Se "indovino": non ho vuotato la pipeline (50% dei casi)
  - Se "sbaglio": devo scartare le istruzioni in corso

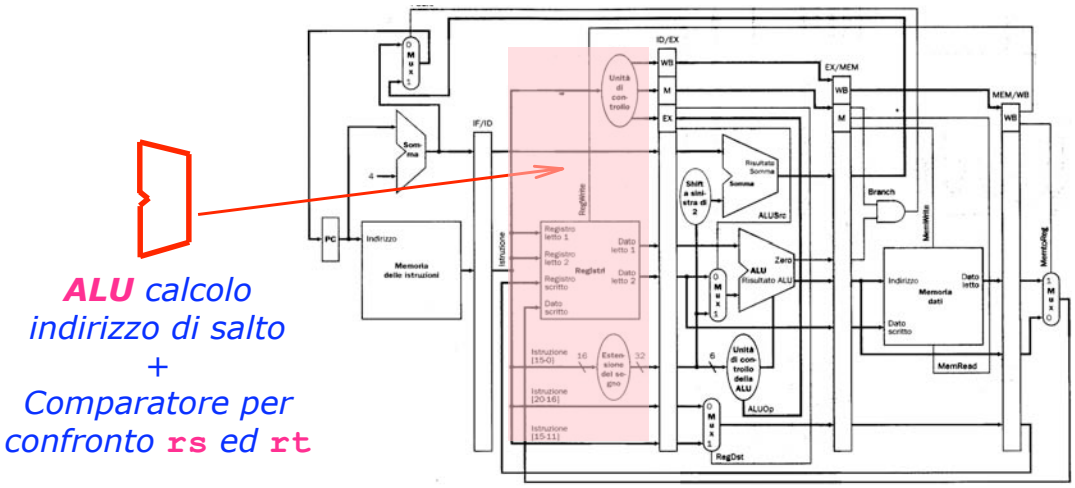
  - Modifiche architetturali per scartare istruzioni già in corso

Modifica della CPU

- Anticipazione del salto:
  - Identificare l'hazard durante la fase **ID** di esecuzione della branch.
  - Necessario scartare **una sola** istruzione.

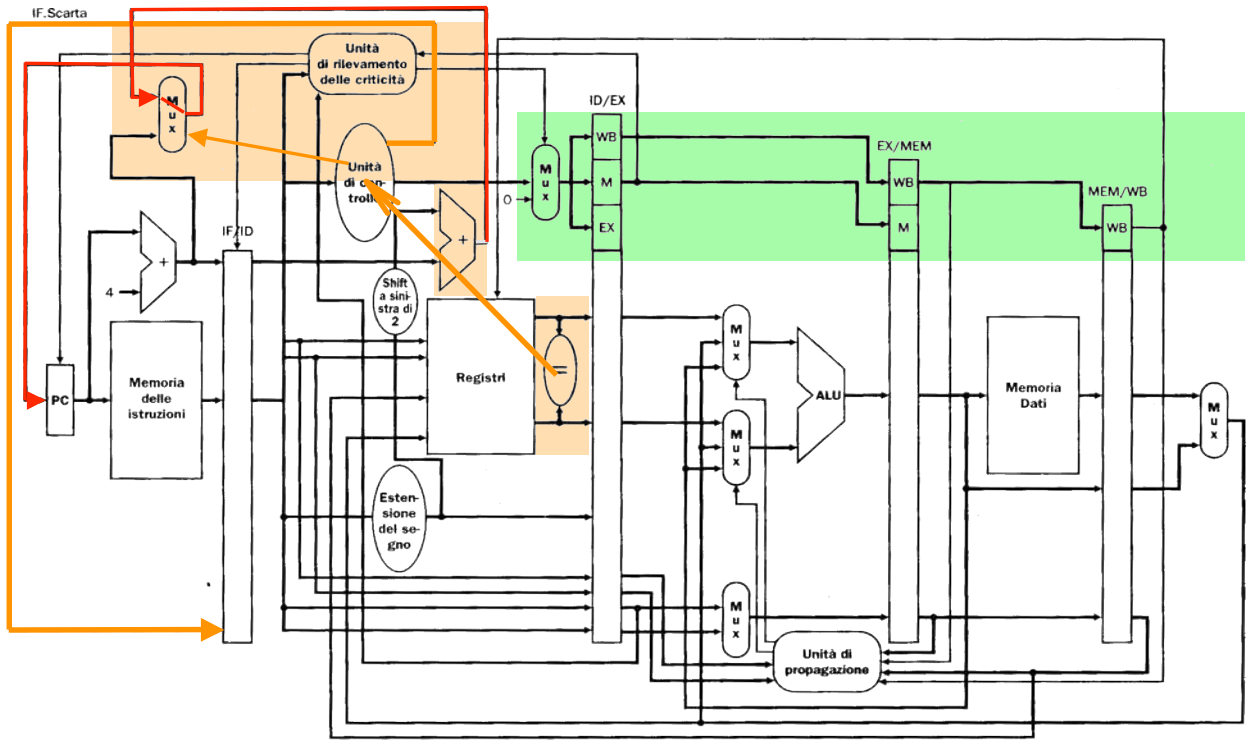
`sub $s2,$s1,$s3`	IF	ID	EX $s1−$s3	MEM	WB ris.➔ $s2			
`beq $t2,$t5, 24`		IF	ID	EX zero IF $t2=$t5	MEM	WB		
`or $t7,$s6,$s7`			IF	ID	EX $s6 **or** $s7	MEM	WB ris.➔ $t7	
`add $t4,$s8,$s8`				IF	ID	EX $s8+$s8	MEM	WB ris.➔ $t4
`add $t0,$t1,$t2`					IF	ID	EX	MEM

Modifica **HW**: ***Anticipazione dei salti***



- Anticipazione della valutazione della branch:
  - Modifica della CPU nella gestione dei salti: anticipazione del calcolo dell'indirizzo di salto.
  - HW addizionale:
    - un **comparatore** all'uscita del **Register File**.
    - Modifica dell'**Unità di Controllo**.

CPU pipeline con gestione degli hazard



Ottimizzazione Criticità di controllo

Branch prediction:

- Faccio una predizione (tentando di indovinare) della necessità di effettuare il salto
- Se indovino, non dovrò fermare la pipeline

- Algoritmi e strutture per la predizione ottima del salto:

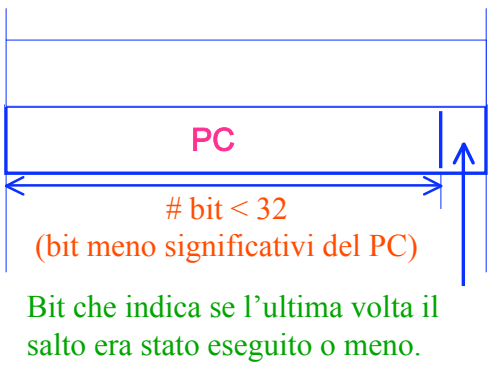
Branch Prediction Buffer:

- Buffer di memoria in cui, associata all'indirizzo di salto, è presente l'informazione se debba essere eseguito o meno.
  - Intel Pentium 4:
    Branch Prediction Buffer con capacità di 4 **kByte**

Branch Prediction buffer

- Problema: previsione relativa ad una **`beq`** (con gli stessi bit meno significativi del PC)
- Suppongo di non dovere saltare → procedo in sequenza
  PC* = PC+4: **`add`**
- Se la previsione è sbagliata, devo:
  - annullare la **`add`**
  - saltare a **`SALTA`**.

→ algoritmi di ottimizzazione per predizione ottima del salto.

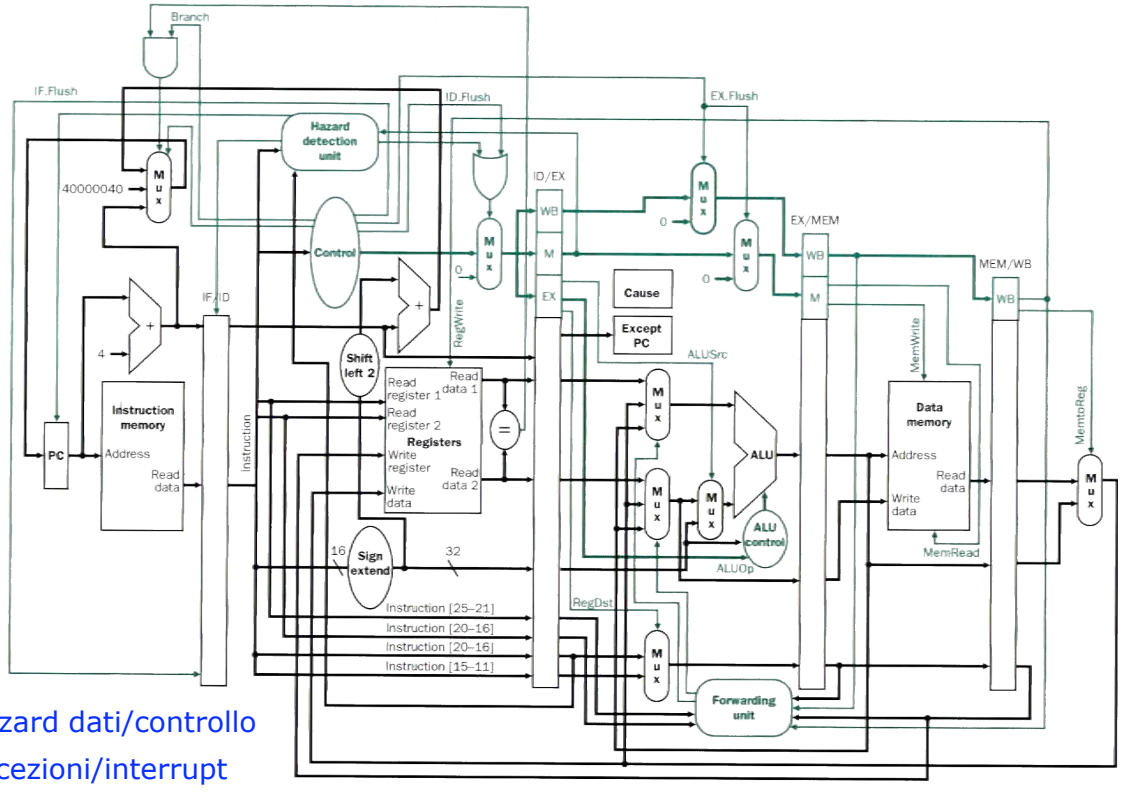



```
        beq $t0, $t1, SALTA
        add $s0, $s1, $s2
        sub $s3, $s4, $s5
        or   $s6 $s7, $s8
SALTA:  and $t2, $t3, $t4
```

Come scartare le istruzioni

- Modifica architettura CPU: introduzione del meccanismo di eliminazione istruzioni:
  - Dipende dallo stadio in cui si trova l'istruzione da scartare:
- In fase di fetch:
  - sovrascrivo IF/ID con l'istruzione **nop**
- In fase esecuzione in un altro stadio:
  - metto a ZERO i segnali di controllo nel registro di pipeline di quello stadio.

CPU pipeline: struttura finale



CPU completa:

Gestione di hazard dati/controllo

Gestione di eccezioni/interrupt