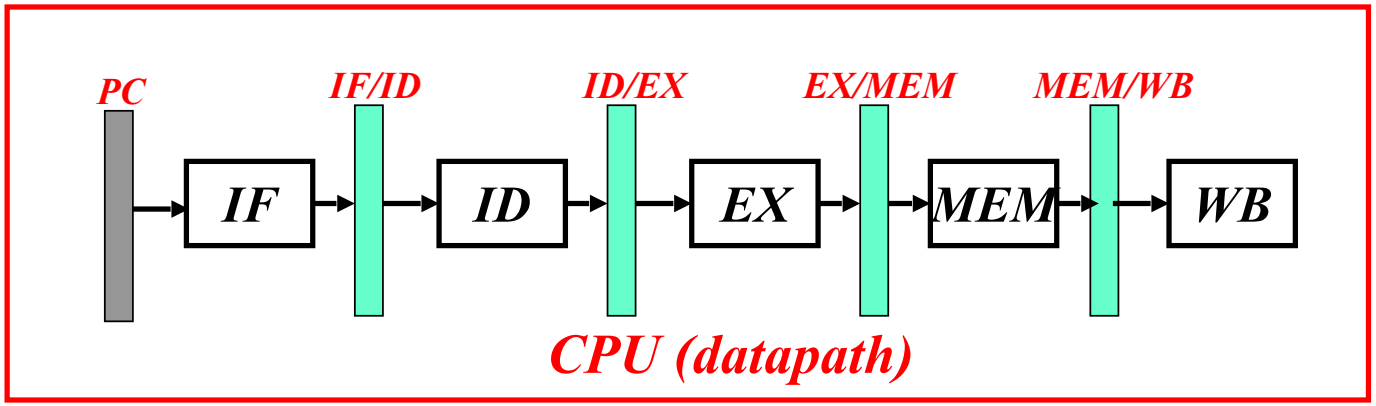
RISC-V FPU Multiciclo

Andrea Bartolini <a.bartolini@unibo.it>

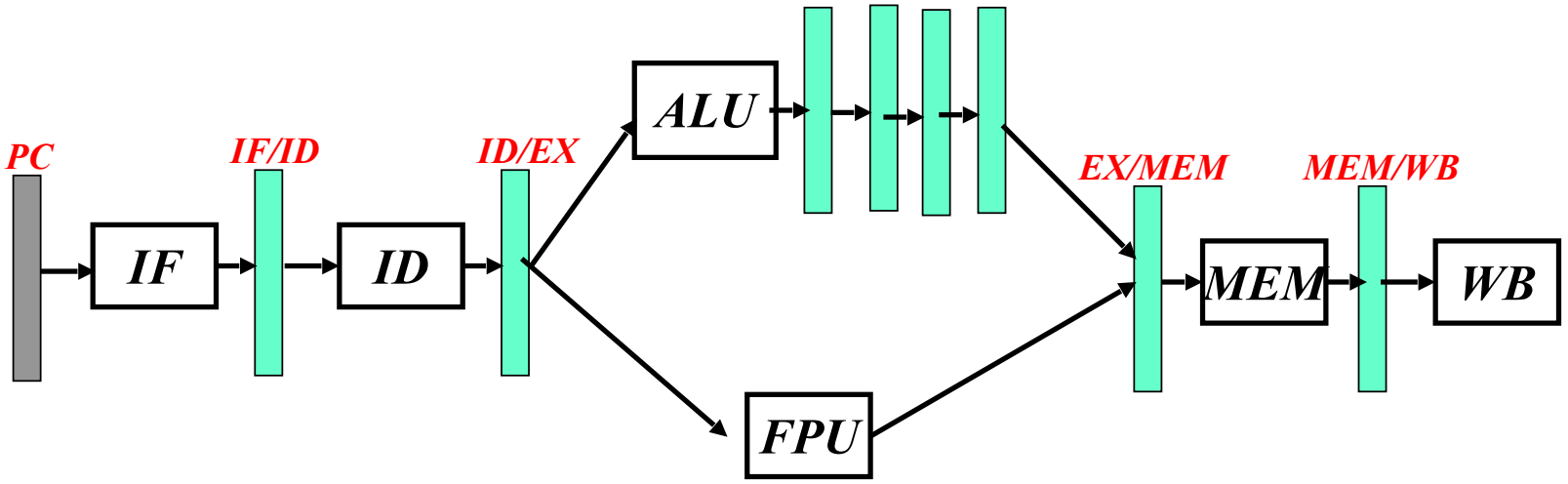
(Architettura dei) Calcolatori Elettronici, 2021/2022

Pipeline con stadi a ciclo singolo

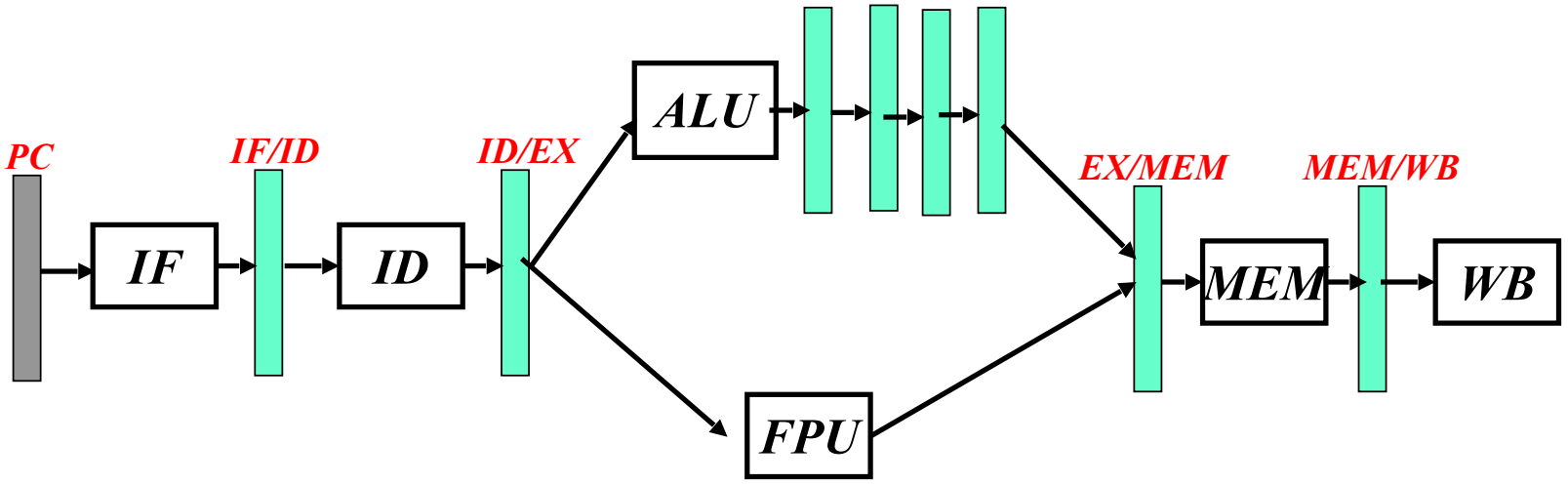
- In assenza di stalli le istruzioni del RISC-V avanzano di uno stadio ad ogni colpo di clock; dunque il *CPI_{ideale} (cioè il $CPI_{senzastalli}$) del RISC-V è 1*



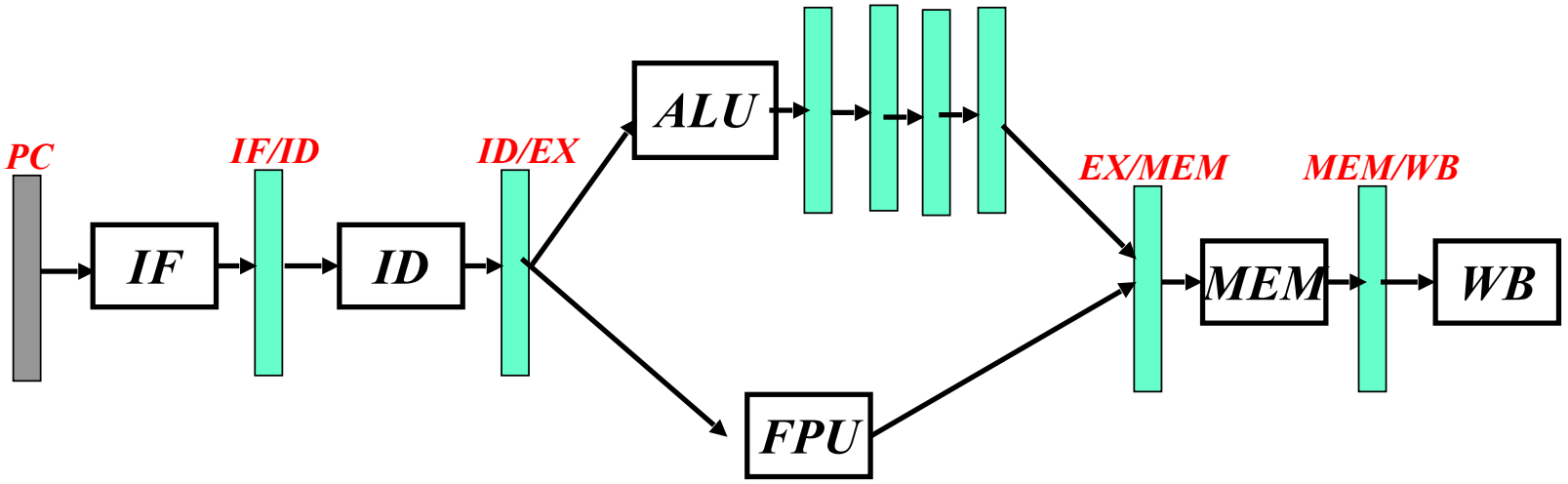
- Facciamo la nuova ipotesi che per eseguire determinate istruzioni i_i lo stadio EX richieda n_i clock; in questo caso la pipeline si comporterebbe come se nello stadio EX si verificassero n_i -1 stalli
- Si verifichi questa proprietà considerando un frammento di codice con istruzioni che si suppone permangono in EX 1, 2 o 3 periodi di clock



	T1	T2	T3	T4	T5	T6	T7	T9	T10	T11	T12	T13	T14	T15	T16
`fmul.d f0,f2,f1`															
`fmul.d f3,f4,f5`															
`fmul.d f6,f7,f8`															



	T1	T2	T3	T4	T5	T6	T7	T9	T10	T11	T12	T13	T14	T15	T16	
`add x0,x2,x1`																
`mul x3,x5,x6`																
`add x4,x7,x8`																

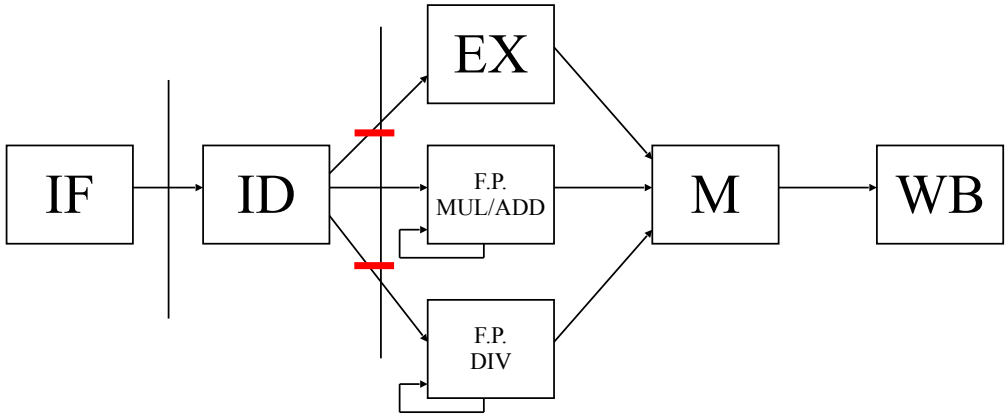


	T1	T2	T3	T4	T5	T6	T7	T9	T10	T11	T12	T13	T14	T15	T16	
`fadd.d f3,f4,f5`																
`add x0,x2,x1`																
`mul x3,x5,x6`																
`add x4,x7,x8`																

Pipeline del RISC-V con stadi multicyle in parallelo (1)

- Attivazione sequenziale => **IN ORDER ISSUE**
- Inizio della Esecuzione sequenziale => **IN ORDER EXECUTION**
- Completamento fuori ordine => **OUT OF ORDER COMPLETION**

La attivazione (issue) è il passaggio dallo stadio ID allo stadio successivo



Ad esempio:

EX	=> 1 tck
MUL/ADD	=> 4 tck (FMUL, FSUB, FADD)
DIV	=> 40 tck (FDIV)

Evoluzione dell'architettura R-R

- estensione dell'ISA: registri e istruzioni in virgola mobile
- Trasformazioni della pipeline
  - Stadio EX con unità funzionali multiciclo (pipeline bloccante)
  - Stadio EX con stazioni di prenotazione (pipeline non bloccante con esecuzione fuori ordine ooo, algoritmo di Tomasulo)
  - Reorder Buffer e stadio di "instruction commit" (pipeline non bloccante con esecuzione speculativa e ooo)

Nei prossimi lucidi

- Si consideri l'isa RV32F con le istruzioni in virgola mobile e con 32 registri da 32 bit destinati agli operandi per le unità di esecuzione in virgola mobile (f[0:31])
- L'esecuzione di una istruzione Floating Point richiede in generale molti periodi di clock
- Pertanto si modifica lo stadio EX della pipeline come segue:
  - Si aggiungono diverse unità funzionali ciascuna destinata all'esecuzione di un sottoinsieme delle istruzioni
  - Ogni nuova unità funzionale richiederà più periodi di clock per eseguire una istruzione
- In sintesi diremo che lo stadio EX della pipeline viene esteso con unità funzionali multiciclo

Pipeline con stadi “Multicycle” (1)

- Uno stadio da n cicli fa stallare la pipeline per n-1 periodi di clock
- Per ridurre o eliminare gli stalli si utilizzano pipeline con più unità multiciclo in parallelo all’unità di esecuzione intera

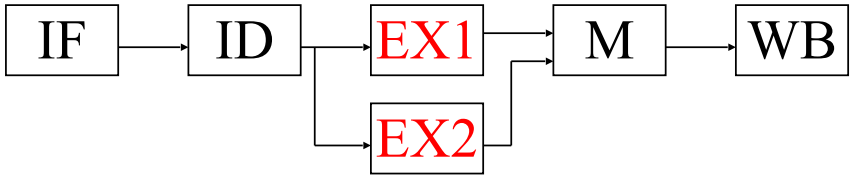
Esempi di pipeline con stadi multicycle

Caso 1: una unita’ di esecuzione con $N_{CK} = 2$

CPI = 2

N_{CK}	1	1	2	1	1

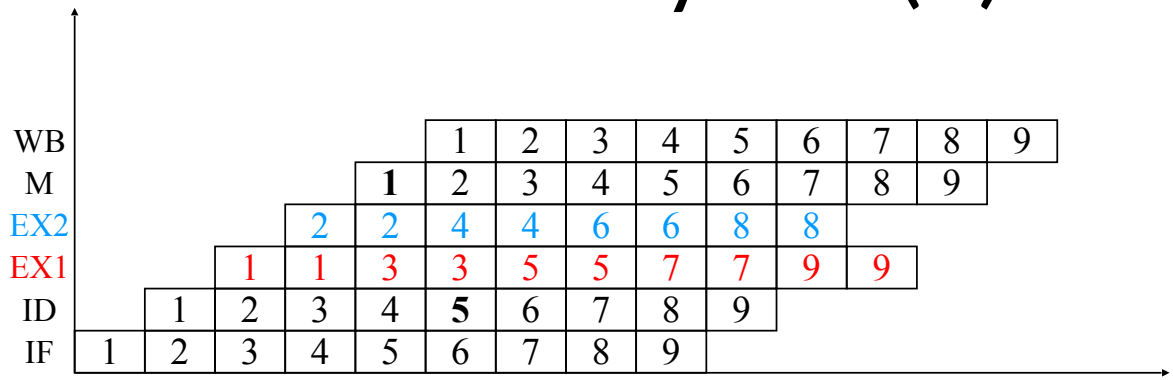
Caso 2: due unita’ di esecuzione con $N_{CK} = 2$



CPI ?

N_{CK}	1	1	2	1	1

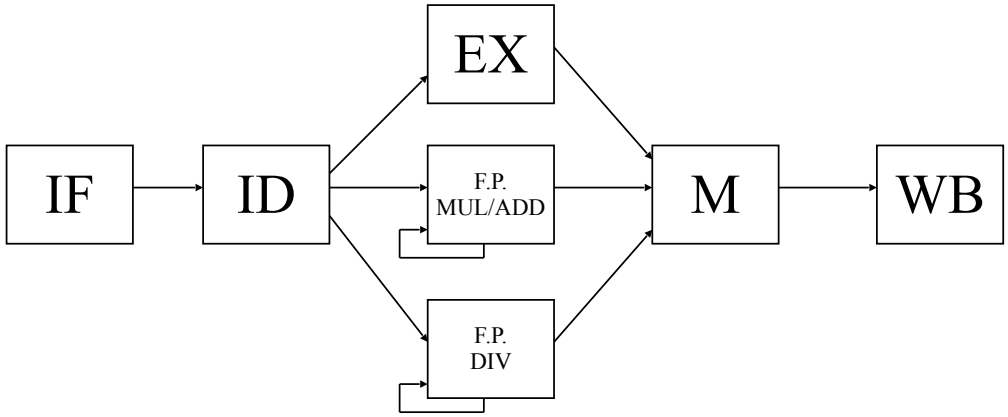
Pipeline con stadi “Multicycle” (2)



- CPI_{MIN} = 1 anche se ci sono stadi che richiedono piu’ di un clock
- $CPI_{IDEALE} > CPI_{MIN}$!
- Aumenta la probabilita’ di stalli, nonchè il numero di stalli introdotti
  - **ES:** se alla istruzione **I_i** in ID serve un dato generato in M, allora **I_i** in ID stalla se l’istruzione che genera il dato è **I_{i-1}, I_{i-2} o I_{i-3}**
    - Es: se **I2** ha bisogno in EX2 di un dato prodotto in M da **I1, allora** la istruz. **I2** stalla due volte

- Attivazione sequenziale => IN ORDER ISSUE
- Inizio della Esecuzione sequenziale => IN ORDER EXECUTION
- Completamento fuori ordine => OUT OF ORDER COMPLETION

La attivazione (issue) è il passaggio dallo stadio ID allo stadio successivo



Ad esempio:
EX (integer unit) => 1 tck
MUL/ADD => **4 tck** (FMUL, FSUB, FADD)
DIV => 40 tck (FDIV)

Attivazione delle istruzioni (ISSUE)

In “ID” si deve:

- ✓ Verificare l’assenza di alee strutturali e stallare altrimenti
- ✓ Verificare l’assenza di alee di dato “RAW” e, in presenza di alea RAW:
  - ❖ attivare la logica di forwarding in caso di alee RAW eliminabili
  - ❖ stallare in caso di alea di dato “RAW” non eliminabili (OP SORGENTI in “ID” = OP DESTINAZIONE in una delle unità di esecuzione)

Esempio di CODICE:

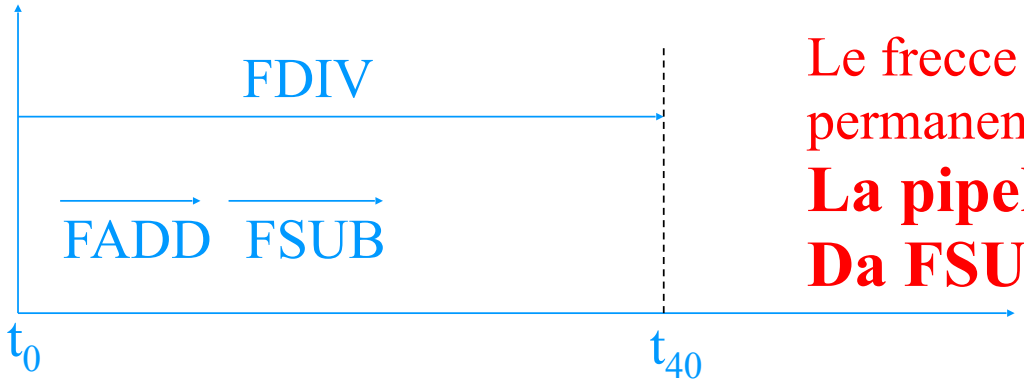
```
1   fdiv.s  f0, f2, f4
2   fadd.s f10, f10, f8
3   fmul.s f12, f12, f14
```

L’istruzione 3 stalla in ID finchè la 2 non libera lo stadio MUL/ADD (3,2 => 3 STALLI per alea strutturale)

Alee strutturali nel RISC-V con
piu' unita' di esecuzione 'multicycle"

ALEA STRUTTURALE DOVUTA
A INDISPONIBILITA' DI
UNITÀ DI ESECUZIONE

```
fdiv.d   f5, f1, f2
fadd.d   f2, f15,f11
fsub.d  f15,f20,f21
```

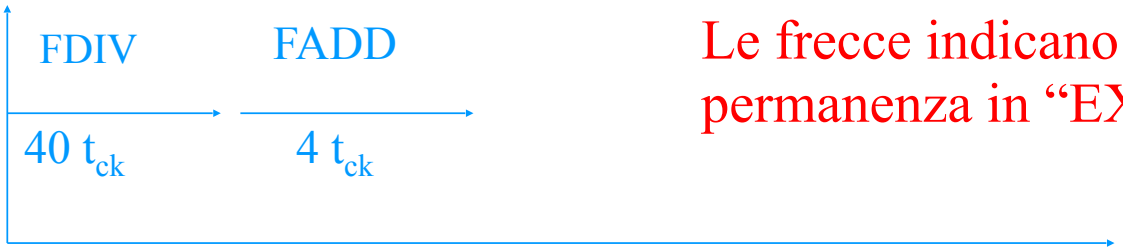


Le frecce indicano la
permanenza in "EX"
La pipeline è bloccata
Da FSUB in ID

Alee di dato nel RISC-V con piu' unita' di esecuzione "multicycle"

ALEA DI TIPO RAW DOVUTA A DIPENDENZA DI DATO

```
fdiv.d f5,f1,f2
fadd.d f2,f5,f11
```



Le frecce indicano la permanenza in "EX"

Hazard Detection Unit in ID

fadd.d stalla in ID per 39 clock

Assenza di alee Write after Read - “WAR” nel RISC-V con stadi multicycle in parallelo

Richiamo:
si parla di alea “WAR” quando il flusso nella pipeline non può proseguire in quanto si deve scrivere su un registro che una istruzione precedente deve leggere ma non l’ha ancora letto

ALEE “WAR” non sono possibili in quanto gli operandi sono sempre letti in “ID”, quindi e’ impossibile che quando si legge un operando l’istruzione successiva l’abbia gia’ aggiornato

Esempi:

```
fadd.d f2, f15, f11
fdiv.d  f15, f20, f21
```

```
fdiv.d   f2, f15, f11
fsub.d  f15, f20, f21
```

Nonostante l’antidipendenza su F15, nel RISC-V non c’è alea WAR: si verifichi questa affermazione disegnando la dinamica della pipeline nei due esempi

Alee Write after Write – “WAW” nel RISC-V con stadi multicycle in parallelo

In alcuni casi le Alee “WAW” si possono eliminare inibendo il completamento dell’istruzione che determina il malfunzionamento

.

ES: *fdiv.d f0, f2, f4*
fsub.d f0, f10, f12

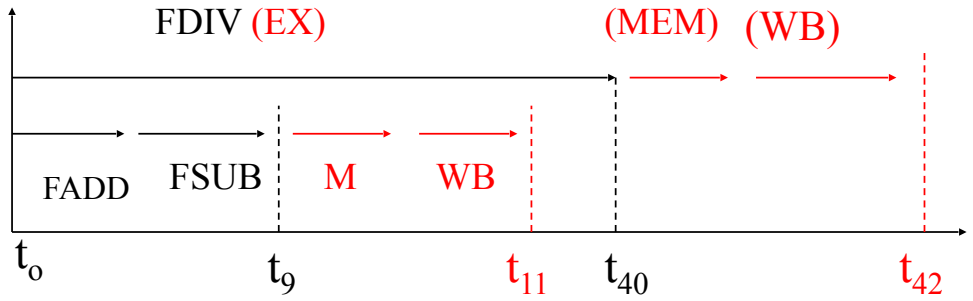
In questo caso, nel periodo di clock in cui la FSUB esce dallo stadio EX si può abortire la esecuzione della FDIV eliminandola dalla pipeline

Richiamo:
si parla di alea “WAW” quando il flusso nella pipeline non può proseguire in quanto una istruzione precedente deve ancora scrivere su un registro che anche l’istruzione corrente deve aggiornare (come si vedrà più avanti, l’origine delle alee WAW sono le “dipendenze di uscita”)

Visualizzazione dell'esecuzione in caso di alee WAW

Questo lucido mostra che dipendenze di uscita possono originare alee WAW in architetture caratterizzate da:

- IN ORDER ISSUE
- IN ORDER EXECUTION
- OUT OF ORDER COMPLETION



Soluzione 1:
Quando FSUB esce da EX si inibisce la scrittura del di FDIV

Soluzione 2:
Stall FSUB.

FDIV F5,F1,F2
FADD F20,F20,F21
FSUB F5,F10,F12

Alea:
Da t_{42} in poi F5 ha un valore errato

Interrupt imprecisi
nel RISC-V con stadi multicycle in parallelo

- Il completamento fuori ordine può dare origine ad eccezioni imprecise.
- Esempio: si puo' verificare una F.P. exception su FDIV quando le istruzioni successive sono gia' state eseguite
- Varie soluzioni:
    1. Ignorare il problema. Non possible in caso di VM e eccezioni IEEE FP
    2. Dual mode processor: Veloce con interrupt imprecise, Lento con interrupt precise.
    3. Memorizzare (buffer) il risultato finchè tutte le operazioni precedenti non hanno completato.
        1. Oneroso se istruzioni di durata molto diversa.
        2. Servono comparatori e multiplexer per bypassare i risultati memorizzati nel buffer come operandi di istruzioni nuove.
    4. Supportare interrupt imprecise, ma passare all'interrupt handler abbastanza informazioni per ricostruire una sequenza precisa (conoscere quali istruzioni erano in esecuzione ed il loro PC e terminarle prima di uscire dall'handler.)
    5. Issue istruzione solo se si rileva che l'istruzione prima dell'issue non può causare eccezioni. Rilevare eccezioni FP nei primi cicli dell'EX stage

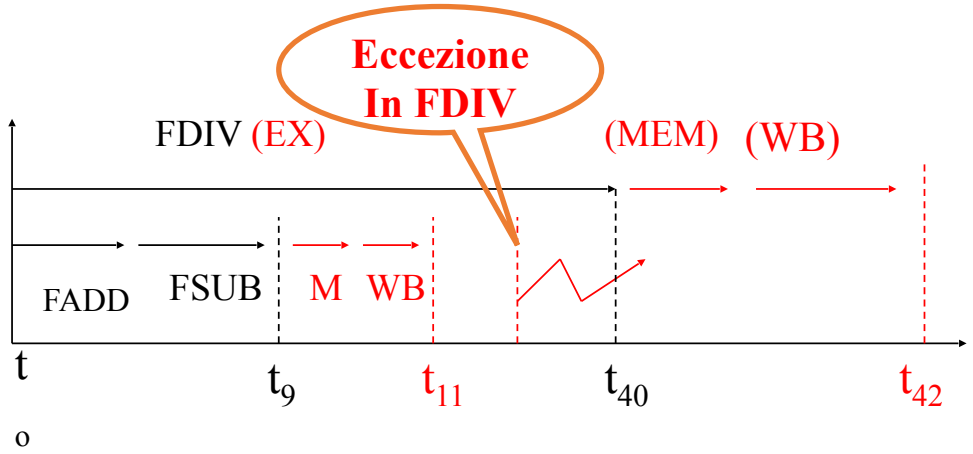
Richiamo:
si dice che una pipeline gestisce le eccezioni in modo preciso se la pipeline può essere fermata in modo che tutte le istruzioni precedenti l'istruzione in cui l'eccezione si è verificata siano completate, mentre tutte le istruzioni successive non modificano lo stato della CPU prima che l'eccezione sia stata servita.
Se l'eccezione è un interrupt esterno, l'interrupt è gestito in modo preciso se esiste una istruzione nella pipeline tale per cui tutte le istruzioni precedenti sono completate prima che l'interrupt sia servito mentre tutte le istruzioni successive ripartono da zero dopo il servizio della interruzione stessa

Esempio di eccezione imprecisa che porta a errori nell'esecuzione del codice

Il completamento fuori ordine può rendere "imprecisa" la gestione delle eccezioni; ne consegue una esecuzione non corretta del codice, con possibilità di perdere informazioni in modo irrecuperabile

FDIV F15,F1,F2
FADD F20,F20,F21
FSUB F5,F10,F12

Supponiamo che l'eccezione in FDIV si verifichi in t_{20}:
Quando si serve l'eccezione la CPU non si trova nello stato corretto (F20 e' perso)



Rappresentazione della dinamica di una pipeline

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18
fadd.d. F1,F2,F3	IF	ID																
fdiv.d F4,F2,F3		IF																
fmul.d F5,F1,F2																		
fadd.d F2,F3,F5																		
fmul.d F1,F9,F10																		

More realistic FP Pipeline

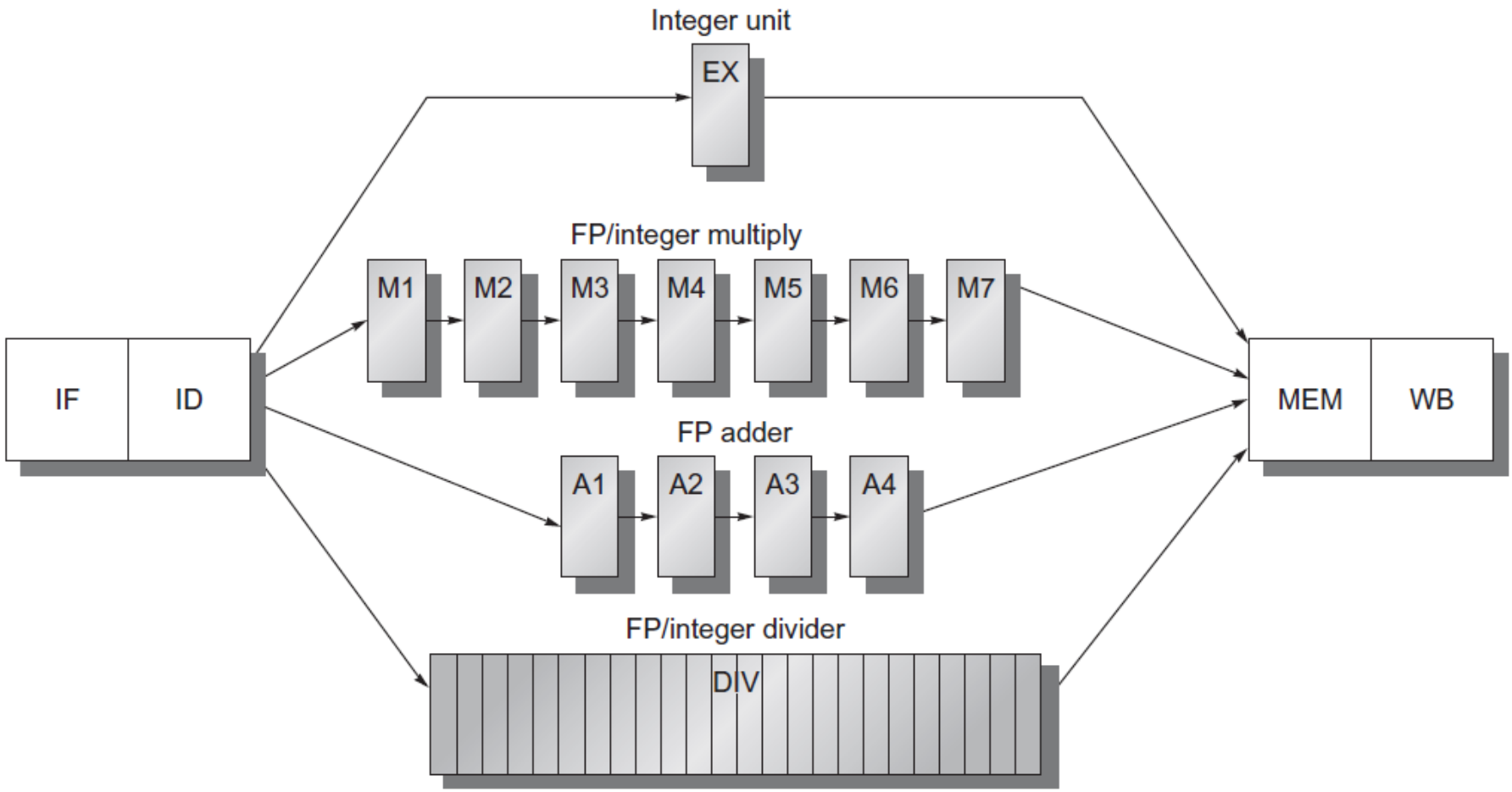
More realistic FP pipeline

Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data memory (integer and FP loads)	1	1
FP add	3	1
FP multiply (also integer multiply)	6	1
FP divide (also integer divide)	24	25

Latency: the number of intervening cycles between an instruction that produces a result and an instruction that uses the result.

Initiation Interval: the number of cycles that must elapse between issuing two operations of a given type.

More realistic FP pipeline



RAW

	Clock cycle number																
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
`fld f4,0(x2)`																	
`fmul.d f0,f4,f6`																	
`fadd.d f2,f0,f8`																	
`fsd f2,0(x2)`																	

RAW

	Clock cycle number																
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
`fld f4,0(x2)`	IF	ID	EX	MEM	WB												
`fmul.d f0,f4,f6`		IF	ID	Stall	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
`fadd.d f2,f0,f8`			IF	Stall	ID	Stall	Stall	Stall	Stall	Stall	Stall	A1	A2	A3	A4	MEM	WB
`fsd f2,0(x2)`					IF	Stall	Stall	Stall	Stall	Stall	Stall	ID	EX	Stall	Stall	Stall	MEM

WB conflict

	Clock cycle number										
Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
`fmul.d f0,f4,f6`	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
...		IF	ID	EX	MEM	WB					
...			IF	ID	EX	MEM	WB				
`fadd.d f2,f4,f6`				IF	ID	A1	A2	A3	A4	MEM	WB
...					IF	ID	EX	MEM	WB		
...						IF	ID	EX	MEM	WB	
`fld f2,0(x2)`							IF	ID	EX	MEM	WB

Summary

1. Divide unit is not fully pipelined,
   => Possible structural hazards
   => Needs to be detected + issuing instructions will need to be stalled.
2. Instructions have varying running times
   => The number of register writes required in a cycle can be larger than 1.
3. Instructions no longer reach WB in order
   => Write after write (WAW) hazards are possible.
4. Register reads always occur in ID
   => Write after read (WAR) hazards are not possible,
5. Instructions can complete in a different order than they were issued
   => problems with exceptions.
6. Longer latency of operations
   => stalls for RAW hazards will be more frequent.