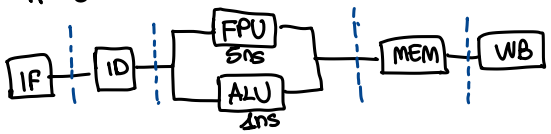
Suppongo di lavorare con una pipeline che ha 1 stadio di IF, uno di ID, poi a valle di questo si ha una FPU e/o una ALU. Dopo uno stadio di MEMORY e poi WB



l'istruzione che va nell'ALU impiega 1 nano sec per fare l'operazione (da ID a MEM).
la FPU invece impiega 5ns per eseguire

Se avessimo solo l'ALU il ck associato sarebbe a freq. 1 giga Hz $= \frac{1}{10^{-9}}$

Se avessimo solo l'FPU, che ha T = 5ns la freq. sarebbe 200 MHz. $\rightarrow = \frac{1}{10^{-9}}$

AVENDO ENTRAMBE LE UNITÀ SI POSSONO AGGIUNGERE STADI DI PIPELINE AGGIUNTIVI NELL'ALU.

	1	2	3	4	5	6	7	8	9	10
ADD $X_1 X_2 X_3$	IF	ID	EX1 ALU	EX2	EX3	EX4	EX5	MEM	WB	
MUL $X_2 X_3 X_4$		IF	ID	EX1	EX2	EX3	EX4	EX5	MEM	WB
ADD $X_4 X_5 X_6$			IF	ID	EX1	EX2	EX3	EX4	EX5	MEM

LATENZA MAGGIORE (più cicli x eseguire un'istruz.)

CPI (di questo esempio senza dip. di dato) = 1

così posso clockare tutto a 1 GHz e posso ricevere un'istruzione intera ogni ciclo

x. Avrò LATENZA maggiore
V. Potrò però fare IF e ID ad ogni ciclo
V. Throughput di 1 istruz./ciclo
V. ck lo posso fare a 1 GHz (e non a 200 MHz)
V. La FPU non ha stadi di pipe e impiegherà i suoi 5 cicli x terminare l'istruzione.

Sono prive di dipendenza di dato per cui tutto va avanti normale

Se avessi una dip. di dato?

RISC-V FPU Multiciclo

Andrea Bartolini <a.bartolini@unibo.it>

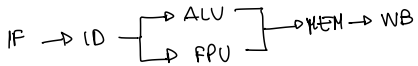
(Architettura dei) Calcolatori Elettronici, 2021/2022

	1	2	3	4	5	6	7	8	9	10
ADD $X_1 X_2 X_3$	IF	ID	EX1 ALU	EX2	EX3	EX4	EX5	MEM	WB	
MUL X_2, X_3, X_4		IF	ID	EX1	EX2	EX3	EX4	EX5	MEM	WB
ADD X_4, X_5, X_6			IF	ID	EX1	EX2	EX3	EX4	EX5	MEM
SUB X_5, X_4, X_6				IF	ID					

della 3° istruzione
V
→ Il risultato della somma è pronto al termine di EX1, quindi potrei avere una forwarding che me lo inietta da EX1.

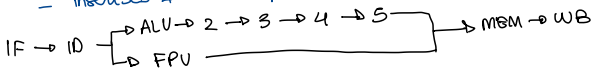
RISC V che può fare istruzioni floating point $f = 1\,GHz$

la FPU impiega 5 ns



Soluzioni: — $f_{clk} = 200\,MHz \rightarrow CPI = 1$ CPI buono, ma frequenza no.

— inserisco 4 stadi di pipeline aggiuntivi



a) $f_{clk} = 1\,GHz$

b) $CPI = 1$ dipendente pipeline lunga

c)

	T1	2	3	4	5	6	7	8	9	10	11	12	13	14
fmul f1,f2,f3	IF	ID	EXf	EXf	EXf	EXf	EXf	MEM	WB					
fADD f4,f5,f6		IF	ID	ID	ID	ID	ID	EXf	EXf	EXf	EXf	EXf	MEM	WB

$CPI = \frac{14}{2} = 7$

Quando eseguiamo f.p. il CPI molto rallentato

PIPELINE MULTICICLO → SI INTRODUCE UNA PIPELINE CHE NON HA STADI AGGIUNTIVI, E SI CONCLUDE L'ISTRUZIONE ALU IN UN SINGOLO CICLO, QUESTO SIGNIFICA CHE LE ISTRUZIONI TERMINANO IN UN ORDINE DIVERSO DA QUELLO PREVISTO DEL CODICE.

↓

può fare commit out of order.

EXf = EXECUTE FLOATING POINT
EXA = EXECUTE ALU

	1	2	3	4	5	6	7	8	9	10
FADD, f1, f2, f3	IF	ID	EXf	EXf	EXf	EXf	EXf	MEM	WB	
ADD X1, X2, X3		IF	ID	EXA	MEM	WB				
MUL X2, X3, X4			IF	ID	EX1	MEM	WB			
ADD X5, X6, X7				IF	ID	EXA	MEM	WB		
fMULd f4, f1, f2					IF	ID	ID	ID	ID	EX

RAW

Non posso anticipare in EX perché il register file ancora non ha il valore corretto di f1.

$CPI = \frac{9}{4}$ Buono

ABBIAMO ESEGUITO COMMIT OUT OF ORDER
Il decode e l'execute vengono fatte in ordine, ma il WB no.

Potrei avere alee di dato per il WB, avendo che più istruzioni vogliono fare WB nello stesso ciclo.

se metto f2 lì non c'è rischio di write after read perché gli operandi vengono letti in ID, quindi f2 lo leggo prima che la 1ª istruzione lo vada a perturbare.

WAR √

La WAW può essere un problema. Alla 2ª istr. metto f1
I registri vengono scritti in WB.
Se scrivo di seguito due volte f1, al ciclo 9 verrà sovrascritta dalla prima istruzione; quindi le istruzioni successive non vedranno il valore di f1 che dovrebbe avere dopo la ADD ma vedono ciò che viene scritto in WB dopo la FADD.

WAW X

↓

questi problemi vanno gestiti con una logica aggiuntiva.

RAW → gestita nella pipeline normale stallando o facendo forwarding

Rappresentazione della dinamica di una pipeline

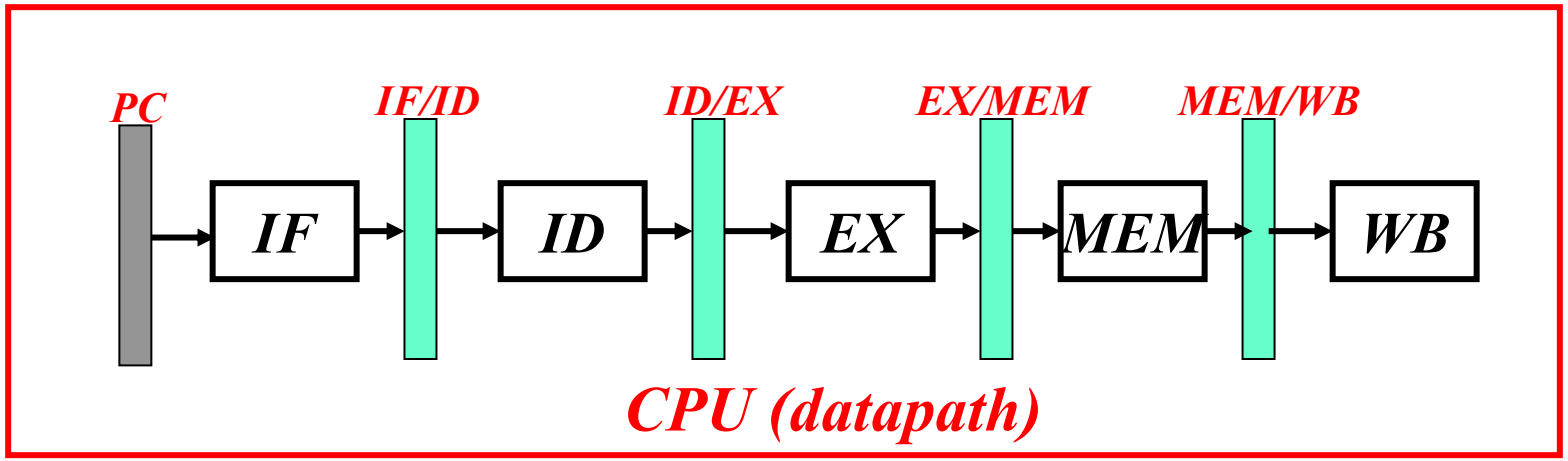
	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18
ADD R1,R2,R3	IF	ID	EX	M	WB													
AND R4,R2,R3		IF	ID	EX	M	WB												
OR R5,R1,R2			IF	ID	EX	M	WB											
ADD R2,R3,R5																		
ADD R1,R9,R10																		

Rappresentazione della dinamica di una pipeline

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18
ADD R1,R2,R3	IF	ID	EX	M	WB													
AND R4,R2,R3		IF	ID	EX	M	WB												
OR R5,R1,R2			IF	ID	ID	EX	M	WB										
ADD R2,R3,R5				IF	IF	IF	IF	ID	EX	M	WB							
ADD R1,R9,R10																		

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_j lo stadio EX richieda n_j clock; in questo caso la pipeline si comporterebbe come se nello stadio EX si verificassero n_j -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

CPU time

- Supponiamo che in assenza di stalli ogni istruzione venga completata $CPI_{senzastalli}$ periodi di clock dopo l'istruzione precedente, con $CPI_{senzastalli}$ costante; allora, fissato un benchmark, applicando la formula precedente e trascurando il termine che tiene conto del riempimento iniziale della pipeline(1) si ottiene:

  $$CPU_{time} = (N_{istruzioni} * CPI_{senzastalli} + N_{stalli}) * T_{ck}$$

Si verifichino le affermazioni precedenti su un frammento di codice di 5 istruzioni eseguito su un RISC-V in pipeline

(1) Per l'esattezza il termine trascurato e':
latenza della prima istruzione in assenza di stalli - (Tck * CPIsenza stalli)

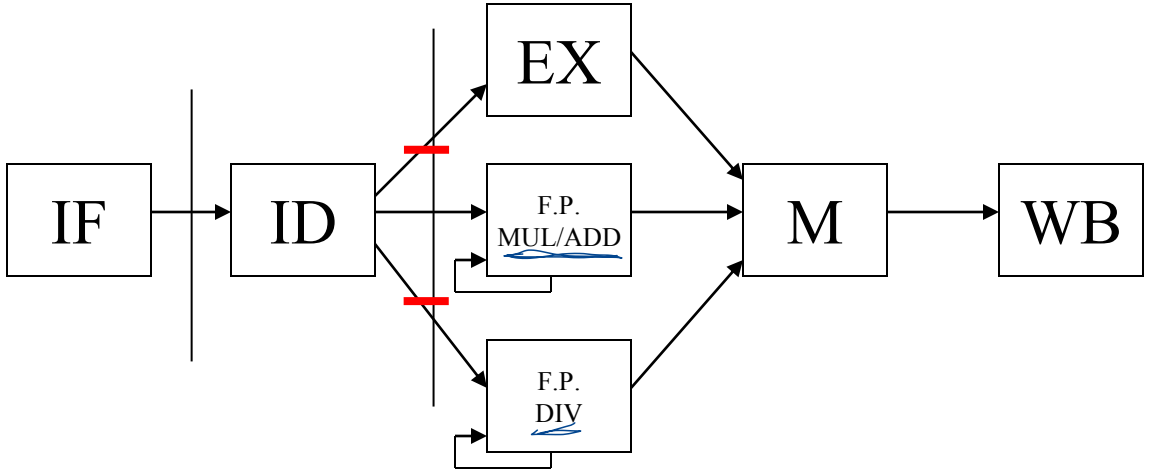
Pipeline del RISC-V con stadi multicyle in parallelo (1)

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

fa il decode e la messa in esecuzione delle istruzioni

seguendo l'ordine delle istruzioni però permette il completamento out of order

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)

eseguite dalla stessa linea all'interno della FPU.

eseguita da un'altra linea nella FPU.

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

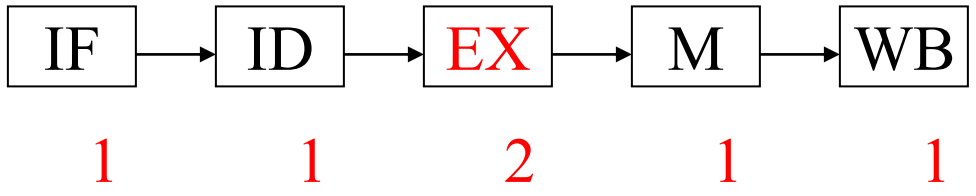
- Si consideri l'isa RV32F con le istruzioni in virgola mobile e con 32 registri da 32 bit destinati agli operandi per le unità di esecuzione in virgola mobile (f[0:31]
- L'esecuzione di una istruzione Floating Point richiede in generale molti periodi di clock
- Pertanto si modifica lo stadio EX della 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

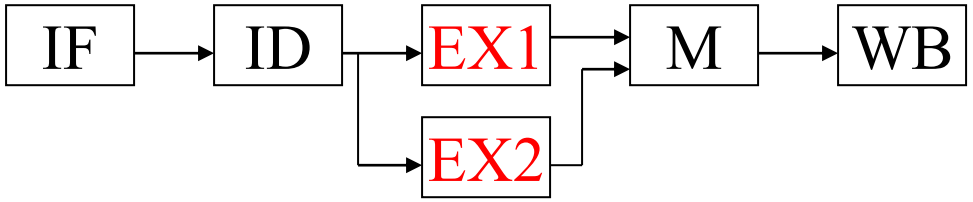
Esempi di pipeline con stadi multicycle

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



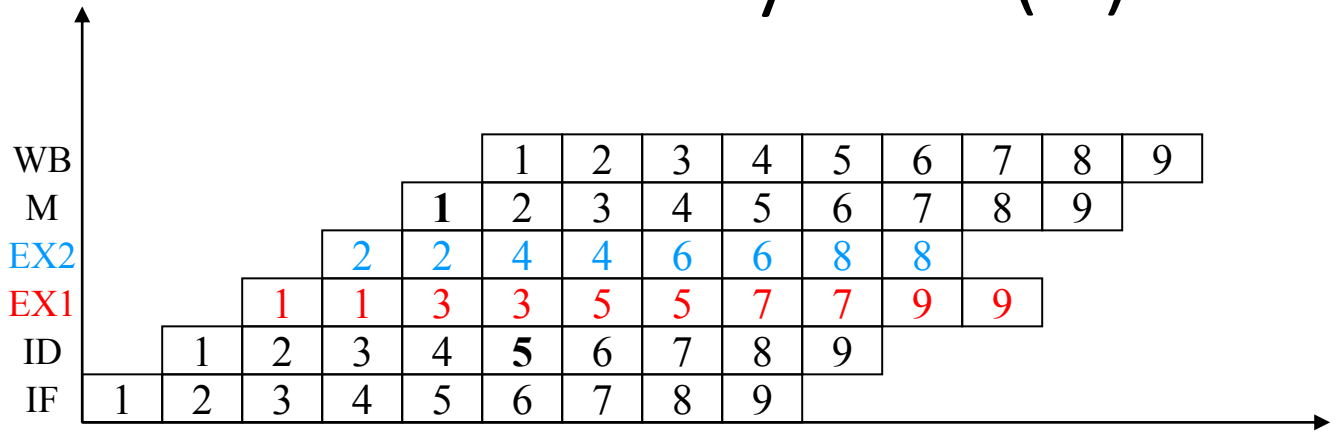
CPI = 2

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



CPI ?

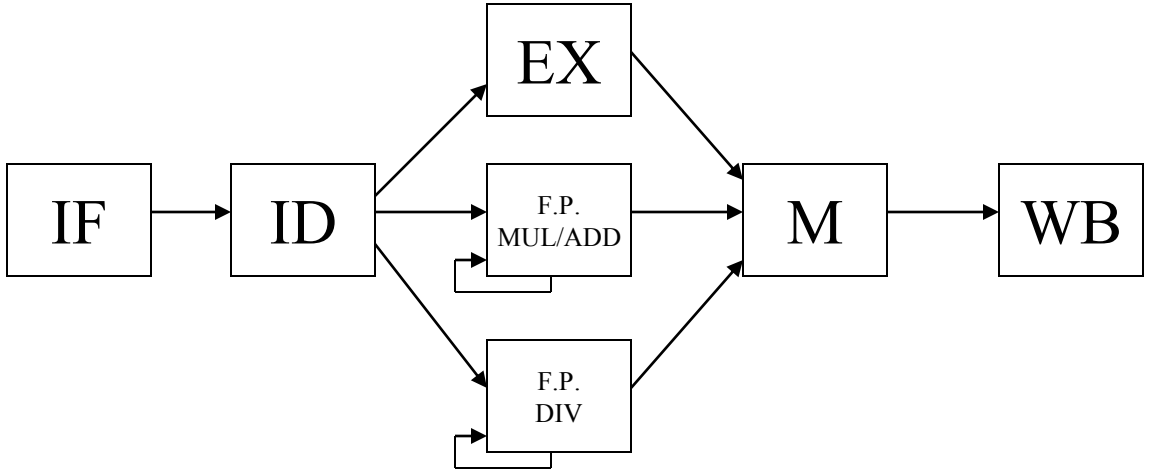
Pipeline con stadi “Multicycle” (2)



- 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
```

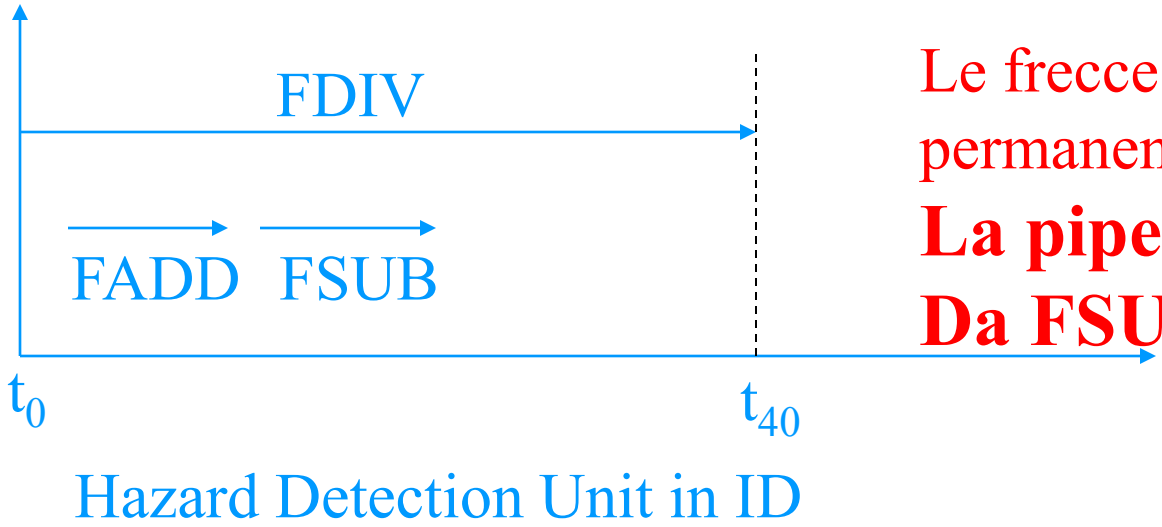
L’istruzione 3 stalla in ID finchè la 2 non libera lo stadio MUL/ADD
(3,2 => 3 STALLI per alea strutturale)

la 1ª istr. occupa l'unità funzionale che serve alla 3ª.

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
```



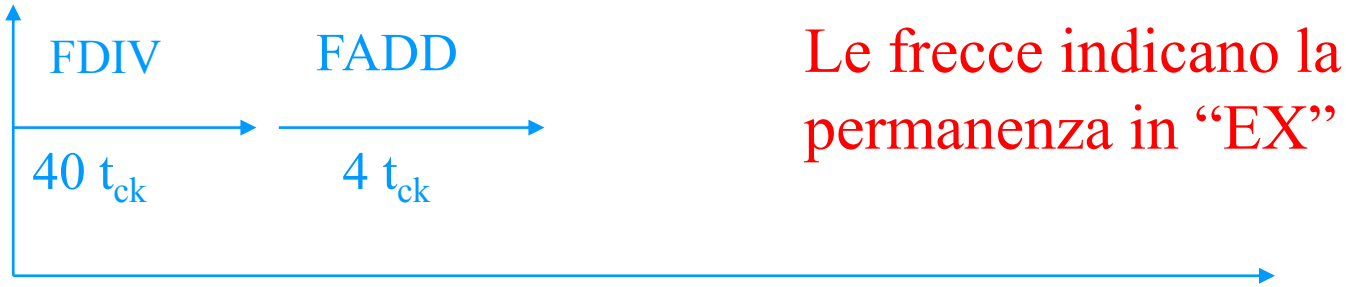
Le frecce indicano la permanenza in "EX"

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

RARO

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

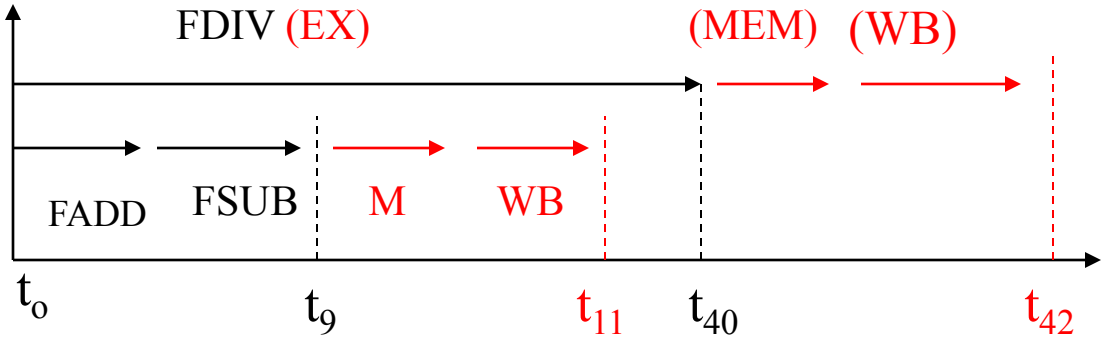
In questo caso, nel periodo di clock in cui la FSUB esce dallo stadio EX si può abortire la esecuzione della FDIV eliminandola dalla pipeline

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 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 ed FADD iniziereranno l'esecuzione mentre FSUB invece dovrà aspettare che FADD liberi l'unità strutturale, nel frattempo stalla

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

entrambe usano la stessa unità funzionale → DIPENDENZA STRUTTURALE

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

Quando termina FSUB, in WB scrivo prima che lo faccia l'istruz. FDIV. Risolvo stallando FSUB fino al ciclo T_{42}.

ECCEZIONI: es. divisione per zero.
MPU.
Memoria virtuale.

Interrupt imprecisi
nel RISC-V con stadi multicycle in parallelo

- Il completamento fuori ordine può dare origine a 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

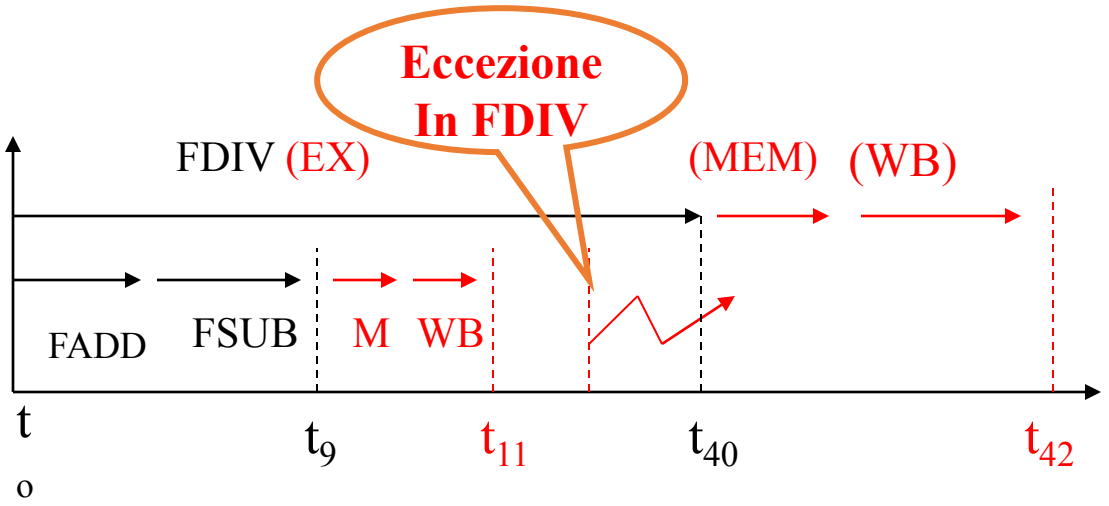
Il completamento fuori ordine può rendere "imprecisa" la gestione delle eccezioni; ne può consegue una esecuzione non corretta del codice, con possibilità di perdere informazioni in modo irrecuperabile

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)

cioè dopo che sia FADD che FSUB hanno fatto WB.

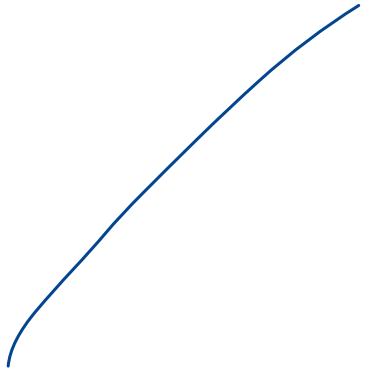
il valore che era in F20 è perso se torno dall'Interrupt service routine perchè è stato già modificato



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

Intervallo tra un'istruzione e la successiva

POSSO ESEGUIRE 1 ISTRUZIONE OGNI CICLO E LA LATENZA È ZERO PERCHÈ ESEGUO NEL CICLO.

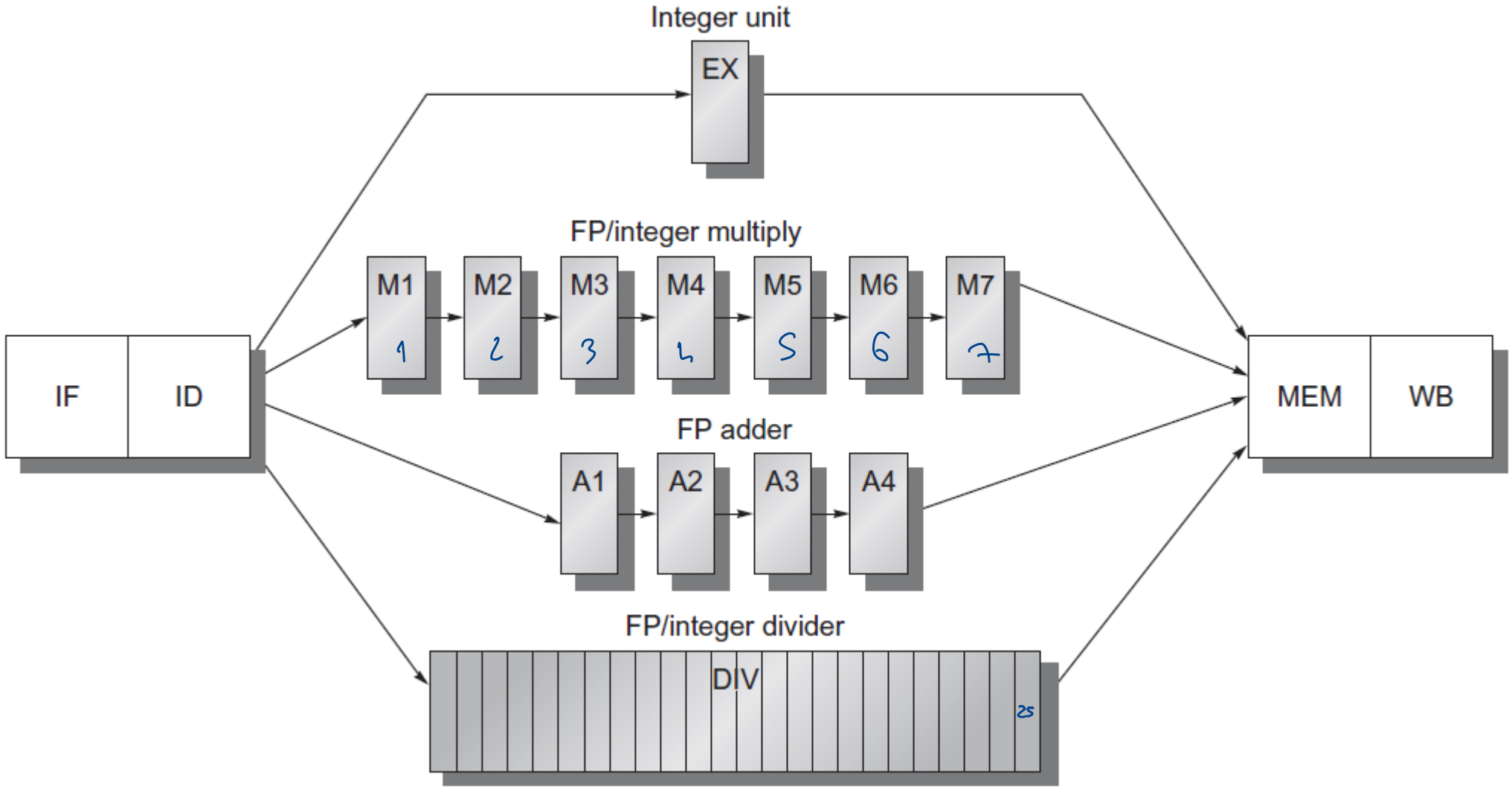
PERCHÈ AVVIENE NELLO STADIO DI MEMORY

INIZIA UN'ISTRUZIONE OGNI 25 TCK.

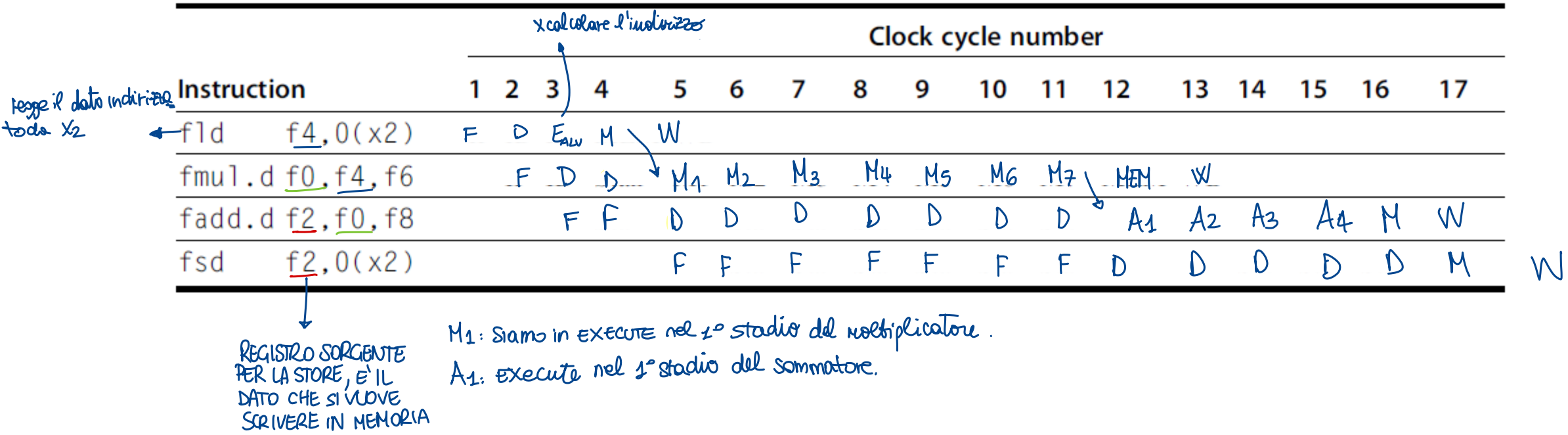
Latency: the number of intervening cycles between an instruction that produces a result and an instruction that uses the result.

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

More realistic FP pipeline



RAW


x calcolare l'indirizzo
Clock cycle number
Instruction
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
legge il dato indirizzato da X2
fld f4,0(x2)
F D E_ALU M W
fmul.d f0,f4,f6
F D D M1 M2 M3 M4 M5 M6 M7 MEM W
fadd.d f2,f0,f8
F F D D D D D D D A1 A2 A3 A4 M W
fsd f2,0(x2)
F F F F F F F D D D D D M W
REGISTRO SORGENTE PER LA STORE, E' IL DATO CHE SI VUOLE SCRIVERE IN MEMORIA
M1: Siamo in EXECUTE nel 1° stadio del moltiplicatore.
A1: Execute nel 1° stadio del sommatore.

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

+ ISTRUZIONI CHE VOGLIONO FARE WB CONTEMPORANEAMENTE

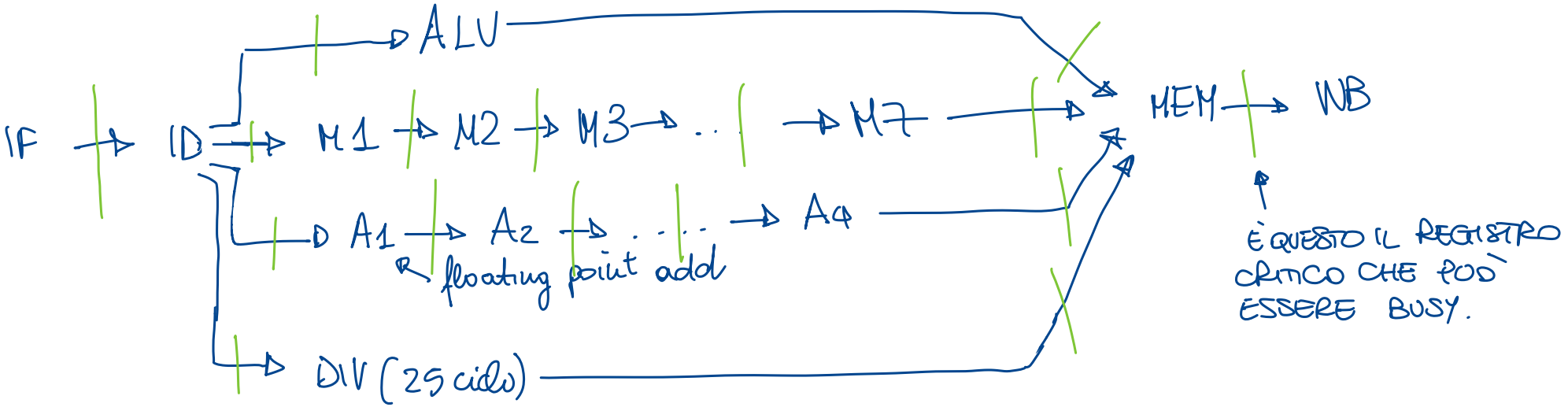
CONVIENE STALLARE IN DECODE, AGGIUNGENDO UNA LOGICA CHE GESTISCA GLI STALLI IN ID

	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

IN BASE ALLA DURATA DELL'ISTRUZIONE PRECEDENTE (es. 7 cicli o 4 cicli)

Questa logica in ID prevede se mettendo l'istruzione in esecuzione, avrà il WB che concilierebbe con il WB di qualche altra istruzione e nel caso stalla.

NOTA: lo stadio di MEM



1 registro IF/ID

1 registro ID/EX per ogni ramo (in questo caso 4)

1 registro EX/MEM 4 rami (in questo caso 4)

1 registro MEM/WB

Quindi il fatto che ci siano 4 registri EX/MEM risolve in buona parte la contesa.

Esempio:

	1	2	3	4	5	6	7
fDIVD f1, f2, f3	F	D	E				
fADD f4, f5, f6		F	D				

L'unità floating point divisione non è divisibile in più stadi di pipeline e quindi è bloccante da origine ad hazard strutturali nell'ID.

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.

se ho + richieste di WB simultanee

WB out of order → però dare problemi di WAW che vanno gestite stallando in ID.

→ perché facciamo issue in order, cioè IF e ID e l'inizio di EX in ordine come sono scritte

→ ECCEZIONI IMPRECISE

Compiler Techniques for Exposing ILP

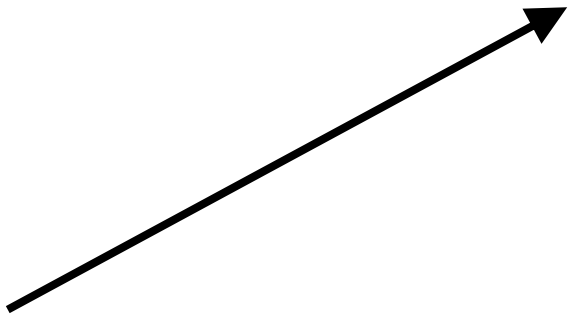
- Pipeline scheduling
  - Separate dependent instruction from the source instruction by the pipeline latency of the source instruction
- Example:

```
for (i=999; i>=0; i=i-1)
  x[i] = x[i] + s;
```

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Pipeline Stalls

```
Loop:   fld      f0,0(x1)
        stall
        fadd.d f4,f0,f2
        stall
        stall
        fsd f4,0(x1)
        addi x1,x1,-8
        bne x1,x2,Loop
```



```
Loop:   fld      f0,0(x1)
        addi x1,x1,-8
        fadd.d f4,f0,f2
        stall
        stall
        fsd f4,0(x1)
        bne x1,x2,Loop
```

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Loop Unrolling

- Loop unrolling
  - Unroll by a factor of 4 (assume # elements is divisible by 4)
  - Eliminate unnecessary instructions

```
Loop:   fld f0,0(x1)
        fadd.d f4,f0,f2
        fsd f4,0(x1) //drop addi & bne
        fld f6,-8(x1)
        fadd.d f8,f6,f2
        fsd f8,-8(x1) //drop addi & bne
        fld f0,-16(x1)
        fadd.d f12,f0,f2
        fsd f12,-16(x1) //drop addi & bne
        fld f14,-24(x1)
        fadd.d f16,f14,f2
        fsd f16,-24(x1)
        addi x1,x1,-32
        bne x1,x2,Loop
```

- note: number of live registers vs. original loop

Loop Unrolling/Pipeline Scheduling

- Pipeline schedule the unrolled loop:

```
Loop:   fld f0,0(x1)
        fld f6,-8(x1)
        fld f8,-16(x1)
        fld f14,-24(x1)
        fadd.d f4,f0,f2
        fadd.d f8,f6,f2
        fadd.d f12,f0,f2
        fadd.d f16,f14,f2
        fsd f4,0(x1)
        fsd f8,-8(x1)
        fsd f12,-16(x1)
        fsd f16,-24(x1)
        addi x1,x1,-32
        bne x1,x2,Loop
```

- 14 cycles
- 3.5 cycles per element