Architetture Out-of-order

Andrea Bartolini <a.bartolini@unibo.it>

(Architettura dei) Calcolatori Elettronici, 2021/2022

2 TECNICHE: - SCOREBOARDING
- TOMMASULO

Concept of dyanamic scheduling

ESEGUIRE CODICE FUORI ORDINE

eseguita dall'unità floating point divisore

dipendenza di dato

```
1. fdiv.d f0,f2,f4
2. fadd.d f10,f0,f8
3. fsub.d f12,f8,f14
```

eseguite da unità f.p. unitche fanno somme aritmetiche

NON POSSONO ESEGUIRE CONTEMPORANEAMENTE PERCHÉ SFRUTTANO LA STESSA RISORSA

LA TERZA ISTRUZIONE POTREBBE ESSERE ANTICIPATA NEL CODICE, ESSERE MESSA PRIMA DELL'FDIV E DELL'FADD.

TUTTO QUESTO PER ESEGUIRE IN MENO COLPI DI CK.

3 → 1 → 2

√ E' IL compilatore che decide eventualmente di riorganizzare l'ordine. → E QUINDI FARE EXECUTE OUT OF ORDER.

Concept of dyanamic scheduling

	T1	T2	T3	T4	T5	T6	T7	T9	T10	T11	
`fdiv.d f0,f2,f4`	IF	ID	EX	EX							
`fadd.d f10,f0,f8`		IF	ID	ID							
`fsub.d f12,f8,f14`			IF	IF							

DEVO STALLARE X DIP. DI DATO
NON POSSO ENTRARE IN ID PERCHÈ È OCCUPATO DA FADD.

- True dependency on f0
- fsub.d stalls yet isn't dependent on fadd.d nor fdiv.d
- Suppose we let fsub.d move around the stall and execute out of order?

l'unità funzionale che serve a faddd è libera in realtà; il problema è che non ho ancora il dato che mi serve.

Pipeline bloccanti / in-order

✓Simple pipelines fetch an instruction and issue it

χ Unless there is a data dependence between an instruction already in the pipeline and the fetched instruction that cannot be hidden with bypassing or forwarding.

- Forwarding logic reduces the effective pipeline latency so that certain dependences do not result in hazards.
- If unavoidable hazard => the hazard detection hardware stalls the pipeline
- No new instructions are fetched or issued until the dependence is cleared.

- Structural hazards and data hazards are checked in the ID stage.

POSSIBILI SOLUZIONI

1. Static Scheduling: IL COMPILATORE RIORDINA LE ISTRUZIONI.
   - The compiler reorders instructions to avoid/reduce stalls.
2. Dynamic Scheduling: CODICE SEMPRE OTTIMIZZATO
   - The hardware rearranges the instruction execution to avoid/reduce stalls.
   - Handles dependencies not known at compile time
   - The same binary runs efficiently on multiple architectures

Pipeline non bloccanti / out-of-order

We want an instruction to begin execution as soon as its operands are available "even if a predecessor is stalled". → permette all'istruzione di sorpassare un istruzione che sta stallando in ID. → DIVIDO L'ID IN 2.

=> separate the issue process into two parts:

- checking the structural hazards
- waiting for the absence of a data hazard.

1. Decode and issue instructions in order, but
2. Execute instructions as soon as their data operands are available

   => out-of-order execution => out-of-order completion.

ISSUE (in due tck diversi) → READ OPERANDS

ISSUE: verifica se c'è disponibilità dell'unità fu. di cui ha bisogno l'istruzione (controllo delle aree strutturali)

READ OPERANDS: qui leggo gli operandi dal register file.

Si fa DECODE e ISSUE in ordine, ma la eseguiamo fuori ordine non appena gli operandi sono disponibili.

FETCH, DECODE, ISSUE in ordine, ma ESECUZIONE eventualmente fuori ordine se gli operandi sono disponibili più tardi.

SE NON CI SONO ALTRE STRUTT. e quindi l'unità funzionale è disponibile, metto in esecuzione e l'istruzione iguorando se gli operandi sono disponibili o no.

Pipeline non bloccanti / out-of-order

The pipeline will do out-of-order execution, which implies out-of-order completion.

⇒ ID stage split into 2 stages:

1. Issue: Decode instruction, check for structural hazards.
2. Read Operands: Wait for data hazard to be resolved, read the operands

⇒ IF unchanged.

⇒ EX considered multicycle, Two phases:

- Instruction begin execution
- Instruction complete execution

quando si fa read operands e si mette l'istruzione in esecuzione.

Between two times an instruction is in execution!

termine dell'esecuzione

TECNICA DI SCHEDULING DINAMICO, PERMETTIAMO ALLA PIPE DI MODIFICARE IL FLUSSO DELLE ISTRUZIONI

Scoreboarding

→ Si aggiungono blocchi hw nella pipeline che consentano di tenere parcheggiata un'istruzione in attesa che i suoi operandi siano disponibili.

prima: ID → EX
ora: ISSUE → READ OPERANDS → EX

abbiamo bisogno di una struttura hw che consenta di capire se un'istruzione è in attesa di operandi, ed eventualmente di quali, e quale unità funzionale eseguirà quando i dati saranno disponibili.

Dynamically scheduled pipeline (**in-order issue**)

⇒ all instructions pass through the issue stage in order but,

⇒ they can be stalled or bypass each other in the second stage (read operands) and thus enter execution out of order.

IN READ l'istruzione può essere superata se i suoi operandi ancora non ci sono.

Scoreboarding

⇒allows instructions to execute out of order when sufficient resources & no data dependences

⇒records data dependences

i. Corresponds to instruction issue
ii. Replaces part of the ID step in the RISC V pipeline

⇒determines when the instruction can read its operands and begin execution.

⇒decides if the instruction cannot execute immediately

⇒ it monitors every change in the hardware and decides when the instruction can execute.

⇒controls when an instruction can write its result into the destination register.

⇒All hazard detection and resolution are centralized in the scoreboard.

Scoreboarding

c'è bisogno di blocco hw che tiene traccia dello stato dell'istruzione.

- Keep track of instructions, functional units, and registers to handle hazards

Goal: CPI=1

"CODA VIRTUALE"

- "Pull out" independent instructions later in the "issue window".
- Wait queue after Issue to hold stalled instructions waiting for operands
  - It may not really exist => implemented by the scoreboard!
- Multiple or pipelined functional units
  - recall: during issue, we stall on a structural hazard

LA LOGICA DI SCOREBOARD È INTERPOSTA TRA IL REGISTER FILE E LE UNITÀ FUNZIONALI

Scoreboard:

STRUTTURA DATI CI DICE IN CHE STADIO DELLA PIPELINE SI TROVA L'ISTRUZIONE DI CUI È STATO FATTO L'ISSUE.

1. ***Instruction status*** – current step (of 4) for an instruction

CI DICE QUALE FUNCTIONAL UNIT DOVRÀ PRODURRE UN DETERMINATO RISULTATO.

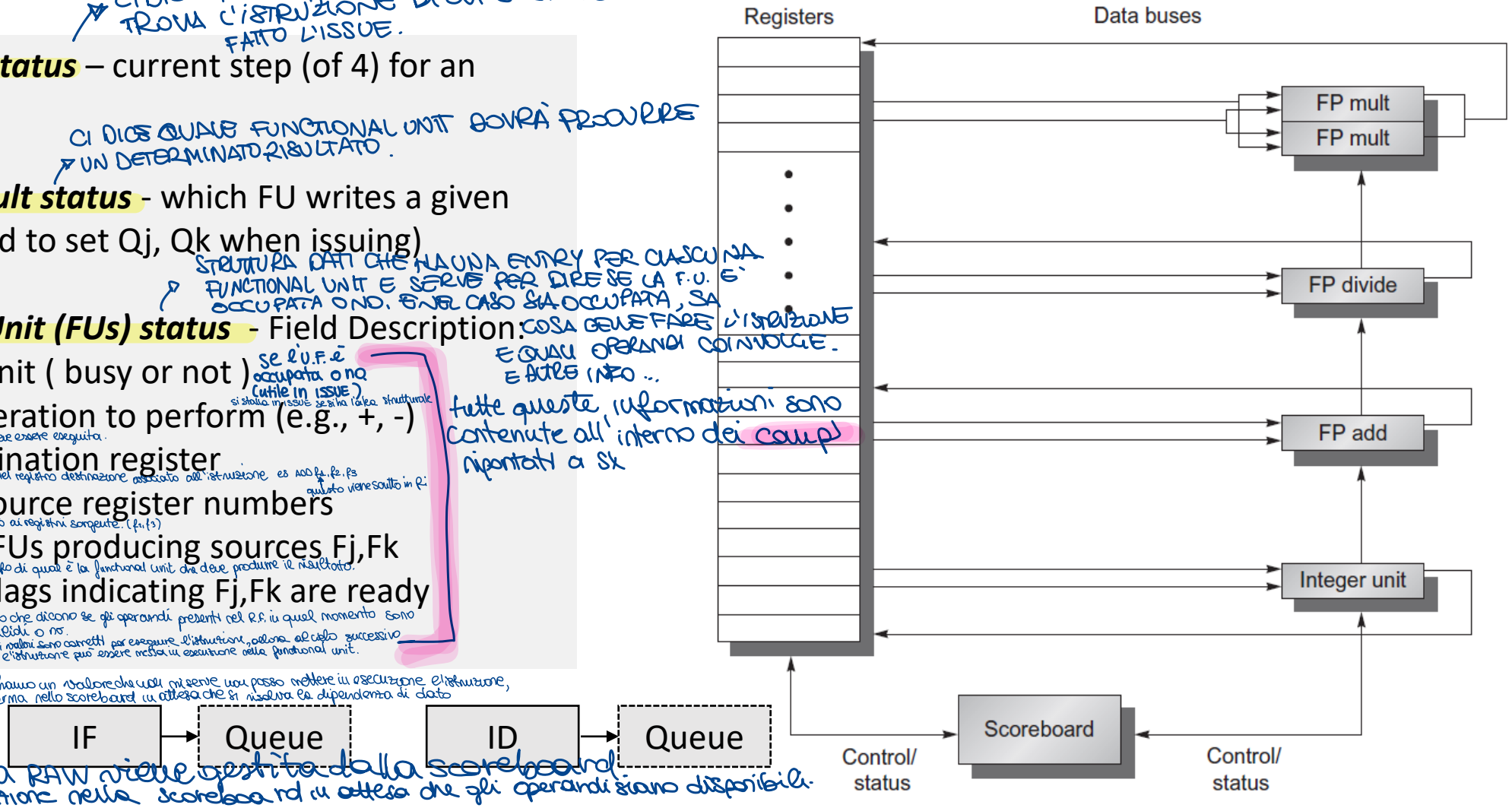
2. ***Register result status*** - which FU writes a given register (used to set Qj, Qk when issuing)

STRUTTURA DATI CHE HA UNA ENTRY PER CIASCUNA FUNCTIONAL UNIT E SERVE PER DIRE SE LA F.U. È OCCUPATA O NO. E NEL CASO SIA OCCUPATA, SA COSA DEVE FARE L'ISTRUZIONE E QUALI OPERANDI COINVOLGE E ALTRE INFO ...

3. ***Functional Unit (FUs) status*** - Field Description:
   - *i.* *Busy* Unit (busy or not) — se l'U.F. è occupata o no (utile in ISSUE) si stalla in ISSUE se si ha l'altro strutturale
   - *ii.* *Op* Operation to perform (e.g., +, -) — L'operazione che deve essere eseguita.
   - *iii.* *Fi* Destination register — Lo prende il numero del registro destinazione associato all'istruzione es ADD Ff, Fk, Fj; quello viene scritto in Ff
   - iv. Fj,Fk Source register numbers — Il numero associato ai registri sorgente. (Fj, Fk)
   - v. Qj,Qk FUs producing sources Fj,Fk — Le contengono l'info di quale è la functional unit che deve produrre il risultato.
   - vi. Rj,Rk Flags indicating Fj,Fk are ready — I bit di servizio che dicono se gli operandi presenti nel R.F. in quel momento sono operandi validi o no. Se Rj = Rk = 0 → i valori sono corretti per eseguire l'istruzione, altrimenti al ciclo successivo l'istruzione può essere messa in esecuzione nella functional unit.

TUTTE QUESTE INFORMAZIONI SONO CONTENUTE ALL'INTERNO DEI CAMPI RIPORTATI A SX

FINCHÉ Rj O Rk HANNO UN VALORE CHE NON RISERVE NON POSSO METTERE IN ESECUZIONE L'ISTRUZIONE, CHE RIMANE FERMA NELLO SCOREBOARD IN ATTESA CHE SI RISOLVA LA DIPENDENZA DI DATO



SE HO UNA DIPENDENZA RAW VIENE GESTITA DALLA SCOREBOARD: CHE TUTTE IN PAUSA L'ISTRUZIONE NELLA SCOREBOARD IN ATTESA CHE GLI OPERANDI SIANO DISPONIBILI.

fDIV f1, f2, f3	F	Issue	Read Op.	EX
fADD f1, f7, f8		F		issue

PROBLEMA: quando ADD deve scrivere il risultato bisogna aspettare che si aggiorni f1 per la divisione.

alca WAW dà problemi? ovvero un'alca (Sì)

Nota DIV: 40 cicli
ADD: 2 cicli

Le alee WAR possono essere stallate prima del WB. finché non viene risolta la dipendenza di dato.

Stage of Scoreboard Control

STADI DELLA SCOREBOARD : MA C'È PIÙ MEMORY

1. Issue (I)

Se c'è una WAW si stalla nello stadio di issue, e le istruzioni che vengono dopo sono bloccate.

- decode instruction, check for structural and WAW hazards, stall when necessary

2. Read Operands (RO)

- wait until no RAW hazards, then read operands, send operation to FU

3. Execution (EX)

- FU starts execution upon receiving the operands & notifies scoreboard when it's completed

4. Write Result (WR)

- Scoreboard checks for WAR hazards and stalls write to register file to avoid them

Hazard Detection:

Dove risolvo le cose

Structural hazards (I): → La risolvo controllando che la entry nello scoreboard associata a quella functional unit sia non busy.

- ensure that a functional unit is available (makes a "reservation")

WAW hazards (I): → Si controlla che nessuna delle precedenti istruzioni di cui è stato fatto issue e che quindi sono presenti nello scoreboard, abbia un registro destinazione identico al reg. dest. dell'istruzione di cui si sta facendo decode in quel momento

- ensure that no previous active instruction has the same destination

RAW hazards (RO): → Si gestiscono verificando che Rj ed Rk abbiano valore vero e non siano in una dipendenza attesa nello Scoreboard di un risultato prodotto da un'altra functional unit.

- check that no previous active instruction writes a source register

WAR hazards (WR): → Si controlla che all'interno dello score. non ci sia corrispondenza tra il reg. destinazione dell'istruzione di cui voglio fare WB e un registro sorgente da qualche parte nello scoreboard.

- before writing a result, check if any previous instructions that haven't gone past RO need that register as a source

(A)
fmul f7, f8, f9
fadd f10, f7, f11
RAW
La RAW viene risolta dallo SB, perché FADD è bloccata dallo SB finché l'fmul non produce il risultato per il registro f7 e poi sblocchi l'esecuzione di FADD.

(B)
fadd f1, f2, f3
fmul f2, f5, f7
WAR
Qui fadd non deve aspettare il valore di f2 dell'istruzione fmul successiva, il valore che ha è già validato.

Scoreboard pipeline control

Status	Wait until	Bookkeeping
Issue	FU not busy && not result	Busy(FU)←Y; Op(FU)←op; Fi(FU)←D; Fj(FU)←S1; Fk(FU)←S2; Qj←Res(S1); Qk←Res(S2); Rj←!Qj; Rk←!Qk; Res(D)←FU
Read ops	Rj && Rk	Rj←N; Rk←N
Executed	FU done	
Write dest (write result)	∀f((Fj(f)≠Fi(FU)|| Rj(f)=N) && (Fk(f)≠Fi(FU) || Rk(f)=N))	∀f(if Qj(f)=FU, Rj(f)←Y); ∀f(if Qk(f)=FU, Rj(f)←Y); Result(Fi(FU))←0; Busy(FU)←N

quindi in issue si stalla se la functional unit è busy e se c'è una WAW

stalliamo in RO finché Rj ed Rk sono falsi

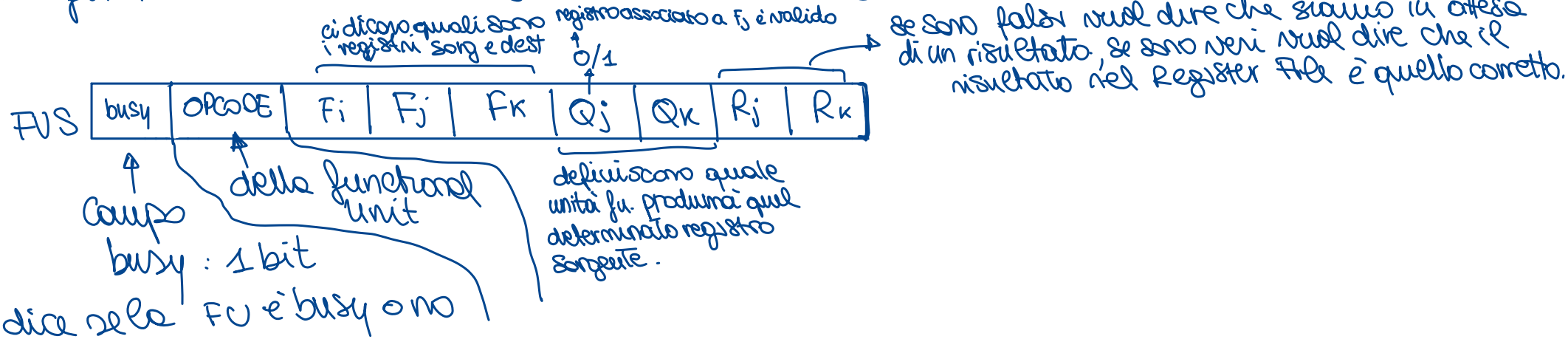
non scriviamo il risultato finché

RES = register result status

in ogni cella è rappresentato un registro.

se il campo è vuoto vuol dire che il valore che è all'interno del register file è un valore valido oppure può esserci un identificativo associato all'unità funzionale.

FUS = functional unit status register: ha una riga per ciascuna functional unit



Write Destination

Wait Until	Bookkeeping
∀f((Fj(f)≠Fi(FU)||	∀f(if Qj(f)=FU, Rj(f)←Y);
Rj(f)=N) &&	∀f(if Qk(f)=FU, Rj(f)←Y);
(Fk(f)≠Fi(FU) ||	Result(Fi(FU))←0; Busy(FU)←N
Rk(f)=N))	

Wait until condition says:

Fj(f)≠Fi(FU)	does this FU write a result used by another FU?
Rj(f)=N	is the other FU waiting for this result?

Bookkeeping says:

if Qj(f)=FU, Rj(f)←Y	set register ready for all consumer FUs
Result(Fi(FU))←0;	clear entry in the register result table

Scoreboard example

Code Sequence:

```
ld.d    f6, 34(R2)
ld.d    f2, 45(R3)
fmul.d  f0, f2, f4
fsub.d  f8, f6, f2
fdiv.d  f10, f0, f6
fadd.d  f6, f8, f2
```

Functional Units: UNITÀ FUNZIONALI DISPONIBILI

- 1 Integer, 2 FP multipliers, 1 FP adder, 1 FP divider

1 unità ALU (interi) 2 FP moltiplic.

Pipeline Latencies:

- ld: 1 cycle <= uses Integer unit → la load la fa l'unità intera e ci mette 1 ciclo
- fmul: 10 cycles <=used FP multiplier →
- fsub, fadd: 2 cycles <= uses FP adder →
- fdiv: 40 cycles <= uses FP divider →

```
ld.d    f6, 34(R2)
ld.d    f2, 45(R3)
fmul.d  f0, f2, f4
fsub.d  f8, f6, f2
fdiv.d  f10, f0, f6
fadd.d  f6, f8, f2
```

Instruction status : SERVE PER VISUALIZZARE IL FLUSSO DELLE ISTRUZIONI

Op	d	j	k	Issue	Read	Finish	Write
ld.d	F6	34+	R2	1	2		
ld.d	F2	45+	R3				
fmul.d	F0	F2	F4				
fsub.d	F8	F6	F2				
fdiv.d	F10	F0	F6				
fadd.d	F6	F8	F2				

Qui ISSUE deve verificare la presenza di alee WAW e alee strutturali.
busy: no la riga del FUS associata alla functional unit che può eseguire quell'istruzione

Functional unit status : ha una riga per ciascuna unità funzionale

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	No	LD	F6	34+	R2	0	0	1	1
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

entrambi validi

ho fatto issue della load in questo ciclo nel prox ciclo potrà essere messo in esecuz.

Register result status : contiene un campo per ciascun registro floating point

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU				INT					

F6 verrà prodotto dall'unità intera

CLOCK:

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1			
ld.d	F2	45+	R3				
fmul.d	F0	F2	F4				
fsub.d	F8	F6	F2				
fdiv.d	F10	F0	F6				
fadd.d	F6	F8	F2				

Functional unit status

Time	*Name*	*Busy*	*Op*	D *Fi*	S1 *Fj*	S2 *Fk*	F1 *Qj*	F2 *Qk*	R1 *Rj*	R2 *Rk*
	Integer	Yes	Ld	F6	34+	R2	0	0	1	Yes 1
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU				Int					

CLOCK: 1) Fill in FU status, mark FU producing results.

Posso fare l'issue della 2° istruzione?

la ld.d 1 è entrata in read operands, posso fare l'issue della 2° istruz.?

Instruction status

Op	d	j	k	Issue	Read	Finish	Write
ld.d	F6	34+	R2	1 (Tck)	2	3	4
ld.d	F2	45+	R3				
fmul.d	F0	F2	F4				
fsub.d	F8	F6	F2				
fdiv.d	F10	F0	F6				
fadd.d	F6	F8	F2				

serve l'unità integer, che però è busy, quindi stalliamo in issue

al 4° ciclo abbiamo il risultato

→ al 5° ciclo l'unità integer non è più busy e l'issue dell'istruz. 2 sarà terminata.

Functional unit status

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	Ld	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU				Int					

CLOCK: 2) Issue 2° ld ?

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	
ld.d	F2	45+	R3				
fmul.d	F0	F2	F4				
fsub.d	F8	F6	F2				
fdiv.d	F10	F0	F6				
fadd.d	F6	F8	F2				

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	Yes	Ld	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU				Int					

CLOCK: 3) Single cycle load completes. What if we have a multi-cycle load?

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3				
fmul.d	F0	F2	F4				
fsub.d	F8	F6	F2				
fdiv.d	F10	F0	F6				
fadd.d	F6	F8	F2				

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	Yes	Ld	F6		R2				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU				Int					

CLOCK: 4) load completes, write result (WAR?) .

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5			
fmul.d	F0	F2	F4				
fsub.d	F8	F6	F2				
fdiv.d	F10	F0	F6				
fadd.d	F6	F8	F2				

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	Yes	Ld	F2		R3				Yes
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU		Int							

CLOCK: 5) structural hazard for Int unit resolved. 2nd load can issue, FMUL is fetched.

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6		
fmul.d	F0	F2	F4	6			
fsub.d	F8	F6	F2				
fdiv.d	F10	F0	F6				
fadd.d	F6	F8	F2				

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	Yes	Ld	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Int		No	Yes
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1	Int							

CLOCK: 6) - LD operands available, issue FMUL. Can the FSUB issue on the next cycle?

Instruction status

Op	d	j	k	Issue	Read	Finish	Write
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	
fmul.d	F0	F2	F4	6			
fsub.d	F8	F6	F2	7			
fdiv.d	F10	F0	F6				
fadd.d	F6	F8	F2				

NON POSSO perché Rj è NO devono essere entrambi SI per fare READ

Functional unit status

Time	Name	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
	Integer	Yes	Ld	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Int		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Int	Yes	No
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1	Int			Add				

CLOCK: 7) - LD finishes, FMUL stalls (why?)

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	
fmul.d	F0	F2	F4	6			
fsub.d	F8	F6	F2	7			
fdiv.d	F10	F0	F6	8			
fadd.d	F6	F8	F2				

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	Yes	Ld	F2		R3				Yes
	Mult1	Yes	Mult	F0	F2	F4	Int		No	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2		Int	Yes	No
	Divide	Yes	Div	F10	F0	F6	M1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1	Int			Add	Div			

CLOCK: 8(a) - immediately before LD completes.

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	8
fmul.d	F0	F2	F4	6			
fsub.d	F8	F6	F2	7			
fdiv.d	F10	F0	F6	8			
fadd.d	F6	F8	F2				

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	M1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1				Add	Div			

CLOCK: 8(b) (the LD completes. FMUL and FADD ready.)

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	8
fmul.d	F0	F2	F4	6	9		
fsub.d	F8	F6	F2	7	9		
fdiv.d	F10	F0	F6	8			
fadd.d	F6	F8	F2				

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
10	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
2	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	M1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1				Add	Div			

CLOCK: 9) - FMUL and FSUB's operands ready, go! Can the FADD issue next cycle? What happens?

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	8
fmul.d	F0	F2	F4	6	9		
fsub.d	F8	F6	F2	7	9	11	
fdiv.d	F10	F0	F6	8			
fadd.d	F6	F8	F2				

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
8	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
0	Add	Yes	Sub	F8	F6	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	M1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1				Add	Div			

CLOCK: 11) - FSUB finishes before FMUL. Can FSUB write its result?

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	8
fmul.d	F0	F2	F4	6	9		
fsub.d	F8	F6	F2	7	9	11	12
fdiv.d	F10	F0	F6	8			
fadd.d	F6	F8	F2				

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
7	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	No								
	Divide	Yes	Div	F10	F0	F6	M1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1					Div			

CLOCK: 12) - FSUB completes before FMUL. Can we read operands for FDIV? What about FADD?

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	8
fmul.d	F0	F2	F4	6	9		
fsub.d	F8	F6	F2	7	9	11	12
fdiv.d	F10	F0	F6	8			
fadd.d	F6	F8	F2	13			

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
6	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	M1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1			Add		Div			

CLOCK: 13) - FADD structural hazard cleared.

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	8
fmul.d	F0	F2	F4	6	9		
fsub.d	F8	F6	F2	7	9	11	12
fdiv.d	F10	F0	F6	8			
fadd.d	F6	F8	F2	13	14		

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
5	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
2	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	M1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1			Add		Div			

CLOCK: 14) - FADD reads operands.

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	8
fmul.d	F0	F2	F4	6	9		
fsub.d	F8	F6	F2	7	9	11	12
fdiv.d	F10	F0	F6	8			
fadd.d	F6	F8	F2	13	14		

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
4	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
1	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	M1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1			Add		Div			

CLOCK: 15)

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	8
fmul.d	F0	F2	F4	6	9		
fsub.d	F8	F6	F2	7	9	11	12
fdiv.d	F10	F0	F6	8			
fadd.d	F6	F8	F2	13	14	16	

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
2	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	M1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1			Add		Div			

CLOCK: 16) - FADD finishes.

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	8
fmul.d	F0	F2	F4	6	9		
fsub.d	F8	F6	F2	7	9	11	12
fdiv.d	F10	F0	F6	8			
fadd.d	F6	F8	F2	13	14	16	

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
1	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	M1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1			Add		Div			

CLOCK: 18) - Yikes! Add FU still occupied?

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	8
fmul.d	F0	F2	F4	6	9	19	
fsub.d	F8	F6	F2	7	9	11	12
fdiv.d	F10	F0	F6	8			
fadd.d	F6	F8	F2	13	14	16	

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
0	Mult1	Yes	Mult	F0	F2	F4			Yes	Yes
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6	M1		No	Yes

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU	Mult1			Add		Div			

CLOCK: 19) - FMUL finishes after FSUB.

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	8
fmul.d	F0	F2	F4	6	9	19	20
fsub.d	F8	F6	F2	7	9	11	12
fdiv.d	F10	F0	F6	8			
fadd.d	F6	F8	F2	13	14	16	

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU				Add		Div			

CLOCK: 20) - FMUL completes, FDIV ready to go

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	8
fmul.d	F0	F2	F4	6	9	19	20
fsub.d	F8	F6	F2	7	9	11	12
fdiv.d	F10	F0	F6	8	21		
fadd.d	F6	F8	F2	13	14	16	

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	Yes	Add	F6	F8	F2			Yes	Yes
	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU				Add		Div			

CLOCK: 21 - FDIV has operands. Can FADD now complete on next cycle?

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	8
fmul.d	F0	F2	F4	6	9	19	20
fsub.d	F8	F6	F2	7	9	11	12
fdiv.d	F10	F0	F6	8	21		
fadd.d	F6	F8	F2	13	14	16	22

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
40	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU						Div			

CLOCK: 22) FDIV executes, FADD writes result .

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	8
fmul.d	F0	F2	F4	6	9	19	20
fsub.d	F8	F6	F2	7	9	11	12
fdiv.d	F10	F0	F6	8	21	61	
fadd.d	F6	F8	F2	13	14	16	22

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
0	Divide	Yes	Div	F10	F0	F6			Yes	Yes

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU						Div			

CLOCK: 61) FDIV finishes

Instruction status

Op	d	j	k	*Issue*	*Read*	*Finish*	*Write*
ld.d	F6	34+	R2	1	2	3	4
ld.d	F2	45+	R3	5	6	7	8
fmul.d	F0	F2	F4	6	9	19	20
fsub.d	F8	F6	F2	7	9	11	12
fdiv.d	F10	F0	F6	8	21	61	62
fadd.d	F6	F8	F2	13	14	16	22

Functional unit status

Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Integer	No								
	Mult1	No								
	Mult2	No								
	Add	No								
	Divide	No								

Register result status

	F0	F2	F4	F6	F8	F10	F12	...	F30
FU									

CLOCK: 62 (FDIV writes its result back)

Scoreboarding: Pro and Contra

Structural hazards possible on buses

- Buses to/from register file may be limited
- Ensure there are not more instructions “in flight” than can successfully fetch registers

No forwarding

- Operands are read only when both are available in the register file
- Fortunately, instructions write into the register file right away

Summary:

- Stalls on I if WAW or structural hazards.
- Stalls on RO if RAW
- Stalls on WR if WAR