Wykonanie instrukcji poza porządkiem programu

Krzysztof Skrzynecki

5 listopada 2014

Kiedy jest potrzebne?

```
DIV.D F0,F2,F4
ADD.D F10,F0,F8
SUB.D F12,F8,F14
```

Kiedy jest potrzebne?

```
DIV.D F0,F2,F4
ADD.D F10,F0,F8
SUB.D F12,F8,F14
```

`SUB.D` jest zablokowany przez poprzednie instrukcje.

Kiedy jest potrzebne?

```
DIV.D F0,F2,F4
ADD.D F10,F0,F8
SUB.D F12,F8,F14
```

`SUB.D` jest zablokowany przez poprzednie instrukcje. Pozwólmy odejmowaniu wykonwać się razem z dzieleniem.

Czy coś może pójść nie tak?

Czy coś może pójść nie tak?

```
ADD.D F0,F2,F4
MUL.D F6,F0,F8
SUB.D F12,F8,F14
ADD.D F6,F10,F8
```

Czy coś może pójść nie tak?

```
ADD.D F0,F2,F4
MUL.D F6,F0,F8
SUB.D F12,F8,F14
ADD.D F6,F10,F8
```

RAW - read after write - odczyt odbywa się zbyt wcześnie (wczytuje się starą wartość)

Czy coś może pójść nie tak?

```
ADD.D F0,F2,F4
MUL.D F6,F0,F8
SUB.D F12,F8,F14
ADD.D F6,F10,F8
```

RAW - read after write - odczyt odbywa się zbyt wcześnie (wczytuje się starą wartość)
WAW - write after write - nadpisuje się nową wartość starą wartością

Czy coś może pójść nie tak?

```
ADD.D F0,F2,F4
MUL.D F6,F0,F8
SUB.D F12,F8,F14
ADD.D F6,F10,F8
```

RAW - read after write - odczyt odbywa się zbyt wcześnie (wczytuje się starą wartość)
WAW - write after write - nadpisuje się nową wartość starą wartością
WAR - write after read - zapis odbywa się zbyt wcześnie (wczytuje się nową wartość)

Jak radzić sobie z hazardami?

Dzielimy fazę `ID` na dwie fazy:
`Issue` - dekodowanie instrukcji i sprawdzenie hazardów strukturalnych (dostępnośc zasobów)
`Read operands` - odczytaj argumenty dopiero gdy nie będzie hazardów danych

Jak radzić sobie z hazardami?

Dzielimy fazę `ID` na dwie fazy:
`Issue` - dekodowanie instrukcji i sprawdzenie hazardów strukturalnych (dostępnośc zasobów)
`Read operands` - odczytaj argumenty dopiero gdy nie będzie hazardów danych
Pewne instrukcje wykonują się w różnym czasie zatem mogą zakończyć się w kolejności innej niż trafiły do fazy `EX`.

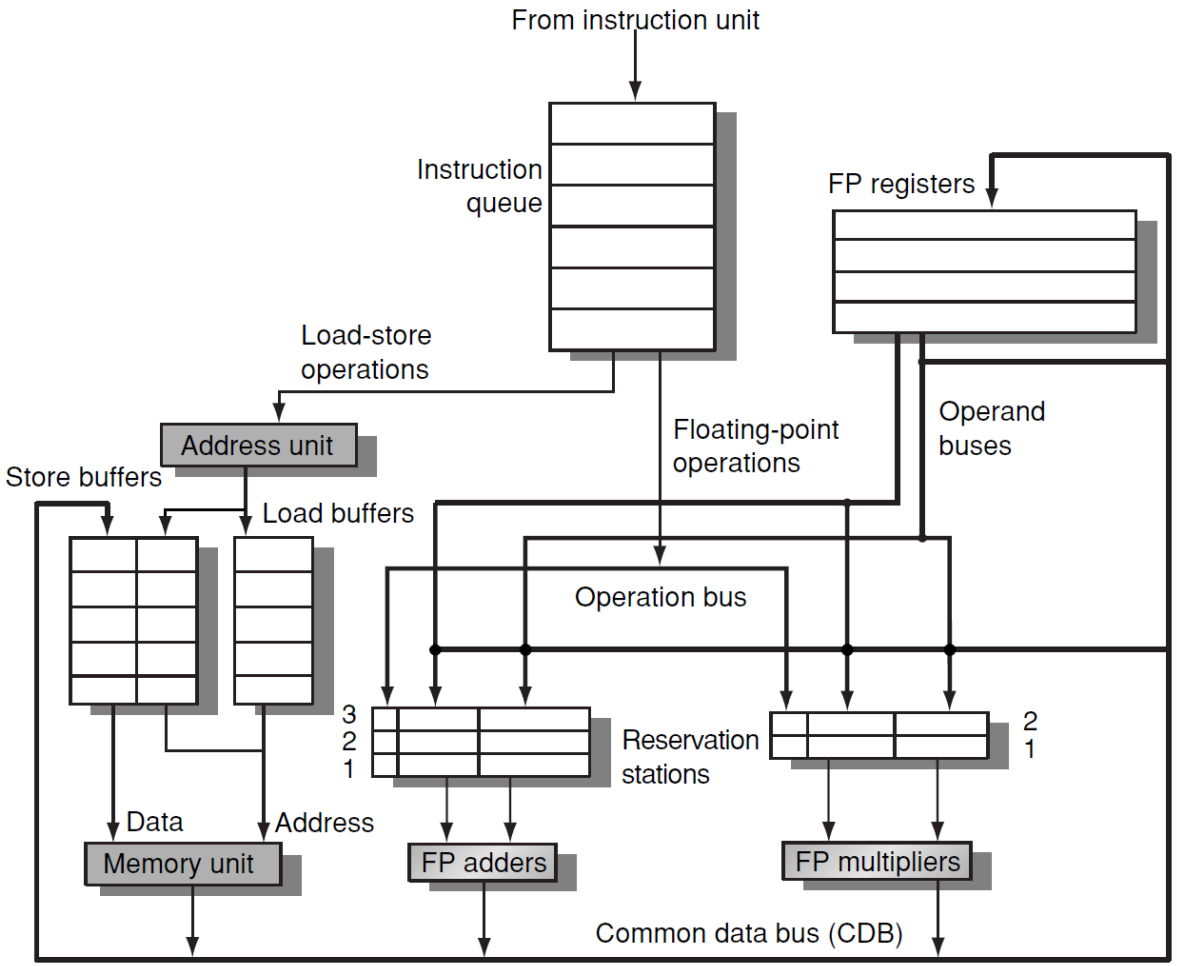
Zmiana nazw rejestów

```
DIV.D F0,F2,F4
ADD.D F6,F0,F8
S.D F6,0(R1)
SUB.D F8,F10,F14
MUL.D F6,F10,F8
```

Zmiana nazw rejestów

```
DIV.D F0,F2,F4
ADD.D S, F0,F8
S.D S, 0(R1)
SUB.D T, F10,F14
MUL.D F6,F10,T
```

Zmodyfikoawny procesor MIPS



Wykonywanie instrukcji

`Issue` - zdekoduj kolejną instrukcję, jeżeli odpowiadająca *reservation station* jest dostępna załaduj tam instrukcję wraz z argumentami z rejestrów. Jeżeli argumentów nie ma w rejestrach monitoruj jednostki funkcyjne aż wyprodukują potrzebne argumetny.
`Execute` - poczekaj aż oba argumeny trafią do *reservation station* i wykonaj operację kiedy będzie odpowiednia jednostka będzie dostępna.
`Write result` - rozgłoś wynik tak by mógł trafić do odpowiednich *reservation station* oraz do pamięci

Przykład pracy procesora

```
L.D F6,32(R2)
L.D F2,44(R3)
MUL.D F0,F2,F4
SUB.D F8,F2,F6
DIV.D F10,F0,F6
ADD.D F6,F8,F2
```

Przykład pracy - stan wewnętrzny

		Instruction status		
Instruction		Issue	Execute	Write Result
L.D	F6,32(R2)	√	√	√
L.D	F2,44(R3)	√	√	
MUL.D	F0,F2,F4	√		
SUB.D	F8,F2,F6	√		
DIV.D	F10,F0,F6	√		
ADD.D	F6,F8,F2	√		

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	yes	Load					45 + Regs[R3]
Add1	yes	SUB		Mem[34 + Regs[R2]]	Load2		
Add2	yes	ADD			Add1	Load2	
Add3	no						
Mult1	yes	MUL		Regs[F4]	Load2		
Mult2	yes	DIV		Mem[34 + Regs[R2]]	Mult1		

Register status

Field	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Mult1	Load2		Add2	Add1	Mult2			

Przykład pracy - stan wewnętrzny

		Instruction status		
Instruction		Issue	Execute	Write Result
L.D	F6,32(R2)	√	√	√
L.D	F2,44(R3)	√	√	√
MUL.D	F0,F2,F4	√	√	
SUB.D	F8,F2,F6	√	√	√
DIV.D	F10,F0,F6	√		
ADD.D	F6,F8,F2	√	√	√

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	no						
Add1	no						
Add2	no						
Add3	no						
Mult1	yes	MUL	Mem[45 + Regs[R3]]	Regs[F4]			
Mult2	yes	DIV		Mem[34 + Regs[R2]]	Mult1		

Register status

Field	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Mult1					Mult2			

Możliwość obsługi pętli

```
Loop:
L.D     F0,0(R1)
MUL.D   F4,F0,F2
S.D     F4,0(R1)
DADDIU  R1,R1,-8
BNE     R1,R2,Loop; branches if R1!=R2
```

Przykład pracy - pętla

		Instruction status			
Instruction		From iteration	Issue	Execute	Write Result
L.D	F0,0(R1)	1	√	√	
MUL.D	F4,F0,F2	1	√		
S.D	F4,0(R1)	1	√		
L.D	F0,0(R1)	2	√	√	
MUL.D	F4,F0,F2	2	√		
S.D	F4,0(R1)	2	√		

	Reservation stations						
Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	yes	Load					Regs[R1] + 0
Load2	yes	Load					Regs[R1] – 8
Add1	no						
Add2	no						
Add3	no						
Mult1	yes	MUL		Regs[F2]	Load1		
Mult2	yes	MUL		Regs[F2]	Load2		
Store1	yes	Store	Regs[R1]			Mult1	
Store2	yes	Store	Regs[R1] – 8			Mult2	

	Register status								
Field	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Load2		Mult2						

Commit

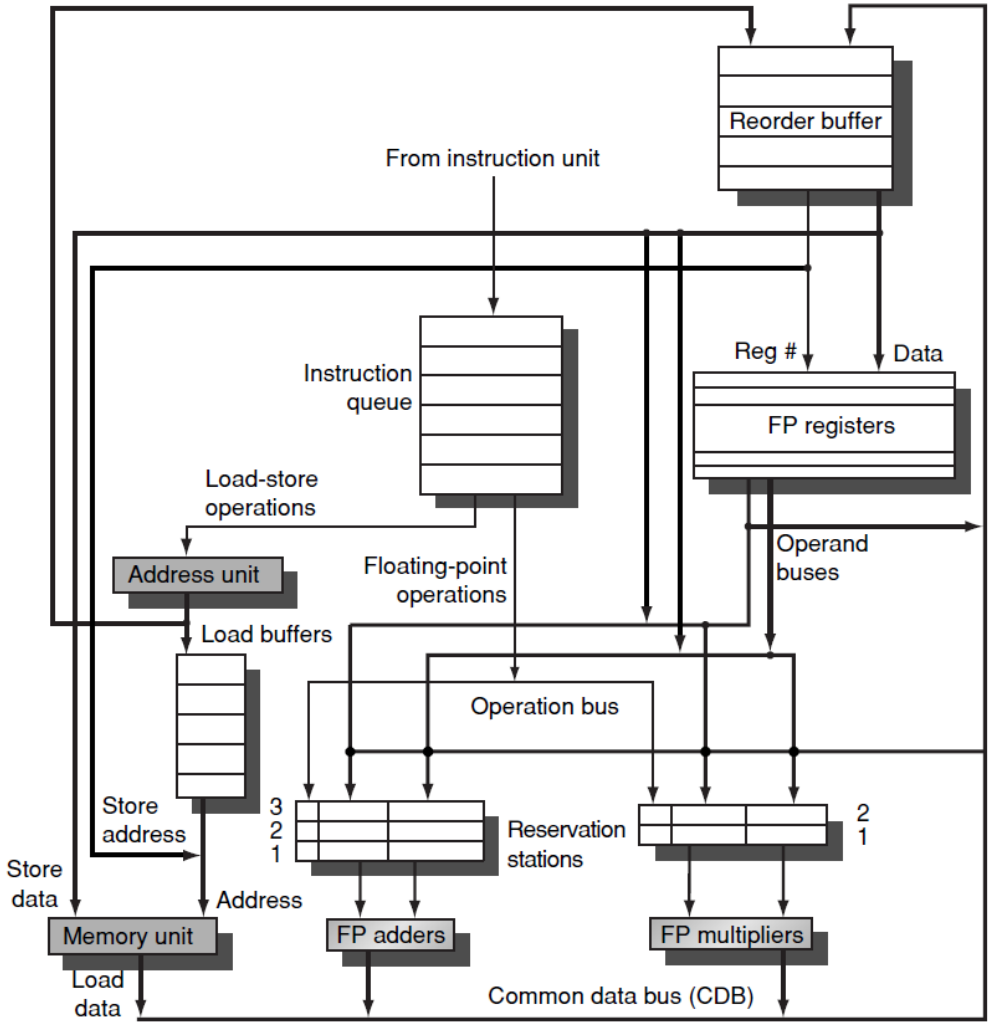
Operacje mogą zakończyć się niepowodzeniem - konieczność umożliwienia cofnięcia operacji.
Efekty pracy instrukcji powinny pojawiać się w takiej kolejności w jakiej pojawiały się instrukcje.

Commit

Operacje mogą zakończyć się niepowodzeniem - konieczność umożliwienia cofnięcia operacji.
Efekty pracy instrukcji powinny pojawiać się w takiej kolejności w jakiej pojawiały się instrukcje.

- Dodanie fazy commit za fazą write oraz bufora umożliwiającego przywrócenie kolejności (reorder buffer - ROB).

Struktura procesora



Przykład pracy

```
L.D F6,32(R2)
L.D F2,44(R3)
MUL.D F0,F2,F4
SUB.D F8,F6,F2
DIV.D F10,F0,F6
ADD.D F6,F8,F2
```

Przykład pracy

Reorder buffer

Entry	Busy	Instruction		State	Destination	Value
1	no	L.D	F6,32(R2)	Commit	F6	Mem[34 + Regs[R2]]
2	no	L.D	F2,44(R3)	Commit	F2	Mem[45 + Regs[R3]]
3	yes	MUL.D	F0,F2,F4	Write result	F0	#2 × Regs[F4]
4	yes	SUB.D	F8,F2,F6	Write result	F8	#2 – #1
5	yes	DIV.D	F10,F0,F6	Execute	F10	
6	yes	ADD.D	F6,F8,F2	Write result	F6	#4 + #2

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Load1	no							
Load2	no							
Add1	no							
Add2	no							
Add3	no							
Mult1	no	MUL.D	Mem[45 + Regs[R3]]	Regs[F4]			#3	
Mult2	yes	DIV.D		Mem[34 + Regs[R2]]	#3		#5	

FP register status

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
Reorder #	3						6		4	5
Busy	yes	no	no	no	no	no	yes	...	yes	yes

Obsługa pętli - przewidywanie skoków

```
Loop:
L.D     F0,0(R1)
MUL.D   F4,F0,F2
S.D     F4,0(R1)
DADDIU  R1,R1,-8
BNE     R1,R2,Loop; branches if R1!=R2
```

Przykład pracy - pętla

Reorder buffer

Entry	Busy	Instruction	State	Destination	Value
1	no	L.D F0,0(R1)	Commit	F0	Mem[0 + Regs[R1]]
2	no	MUL.D F4,F0,F2	Commit	F4	#1 × Regs[F2]
3	yes	S.D F4,0(R1)	Write result	0 + Regs[R1]	#2
4	yes	DADDIU R1,R1,#-8	Write result	R1	Regs[R1] – 8
5	yes	BNE R1,R2,Loop	Write result		
6	yes	L.D F0,0(R1)	Write result	F0	Mem[#4]
7	yes	MUL.D F4,F0,F2	Write result	F4	#6 × Regs[F2]
8	yes	S.D F4,0(R1)	Write result	0 + #4	#7
9	yes	DADDIU R1,R1,#-8	Write result	R1	#4 – 8
10	yes	BNE R1,R2,Loop	Write result		

FP register status

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8
Reorder #	6				7				
Busy	yes	no	no	no	yes	no	no	...	no

Błędnie przewidziany skok?

W takim przypadku wystarczy wyczyścić ROB i kontynuować pracę od miejsca w którym nastąpiło błędne przewidzenie skoku lub wyjątek.

Dziękuję

Pytania?