

DCC007 – Organização de Computadores II

Aula 10 – Superescalar 4

Prof. Omar Paranaíba Vilela Neto

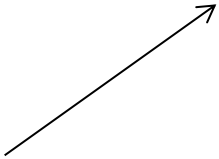


Introdução

- Processadores estudados até aqui estão limitados a $CPI \geq 1$
- Processadores superescalares permitem $CPI < 1$
 - Executam múltiplas instruções em paralelo
- Tipos de processadores superescalares
 - In-order
 - Out-of-order

Introdução

Fetch e Decode



Name	Frontend	Issue	Writeback	Commit	
I4	IO	IO	IO	IO	Fixed Length Pipelines Scoreboard
I2O2	IO	IO	OOO	OOO	Scoreboard
I2O1	IO	IO	OOO	IO	Scoreboard, Reorder Buffer, and Store Buffer
IO3	IO	OOO	OOO	OOO	Scoreboard and Issue Queue
IO2I	IO	OOO	OOO	IO	Scoreboard, Issue Queue, Reorder Buffer, and Store Buffer

IO – In-order

OOO – Out-of-Order

Temas

- Especulação e Branches
- Memory Disambiguation
- Renomeação de Registradores

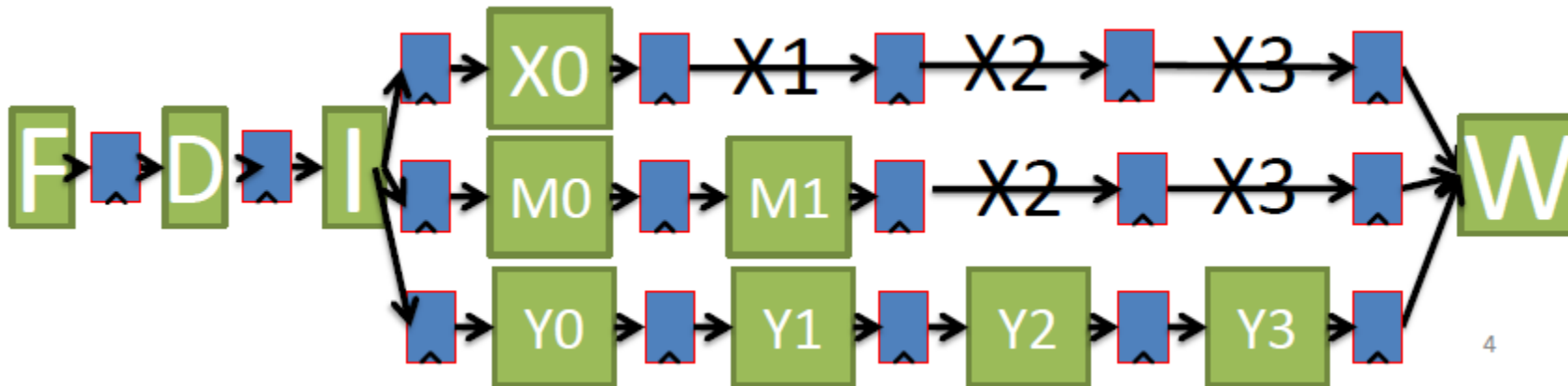
Temas

- Especulação e Branches
- Memory Disambiguation
- Renomeação de Registradores

Especulação e Branch: I4

0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W										
1	ADDIU	R4, R5, 1		F	D	I	X0	X1	X2	X3	W									
2	MUL	R6, R1, R4			F	D	I	I	I	Y0	Y1	Y2	Y3	W						
3	BEQZ	R6, Target				F	D	D	D	I	I	I	I	X0	X1	X2	X3	W		
4	ADDIU	R8, R9 ,1					F	F	F	D	D	D	D	I	--	--	--	--	--	--
5	ADDIU	R10,R11,1								F	F	F	F	D	--	--	--	--	--	--
6	ADDIU	R12,R13,1												F	--	--	--	--	--	--
T															F	D	I	.	.	.

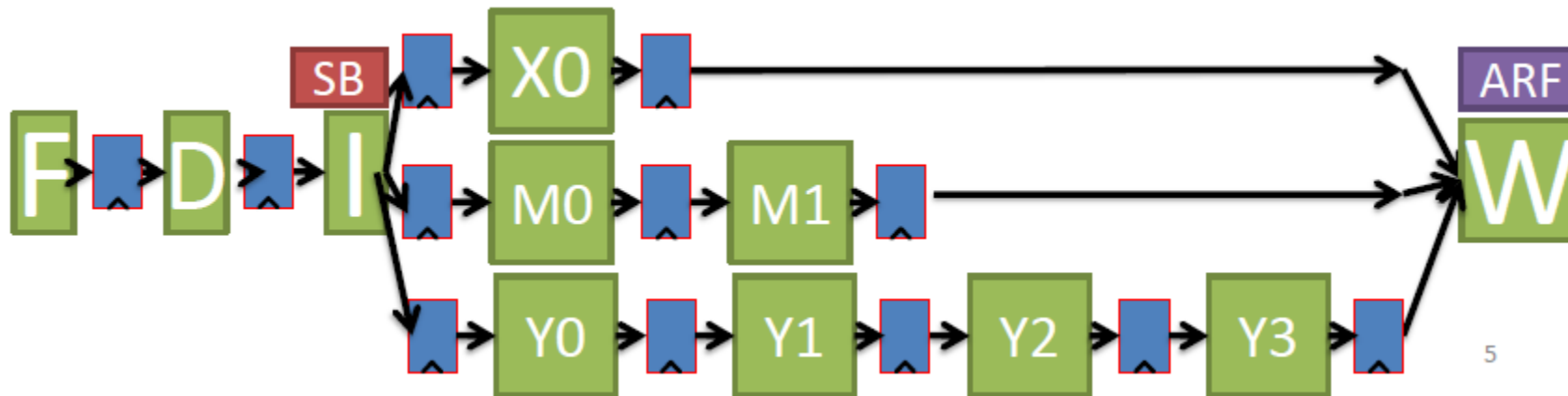
Basta matar as instruções erradas quando desvio é tomado!



Especulação e Branch: I2O2

0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W
1	ADDIU	R4, R5, 1		F	D	I	X0	W		
2	MUL	R6, R1, R4			F	D	I	I	I	Y0 Y1 Y2 Y3 W
3	BEQZ	R6, Target				F	D	D	D	I I I I X0 W
4	ADDIU	R8, R9 ,1					F	F	F	D D D D I -- --
5	ADDIU	R10,R11,1						F	F	F F D -- -- --
6	ADDIU	R12,R13,1								F -- -- -- --
T										F D I . . .

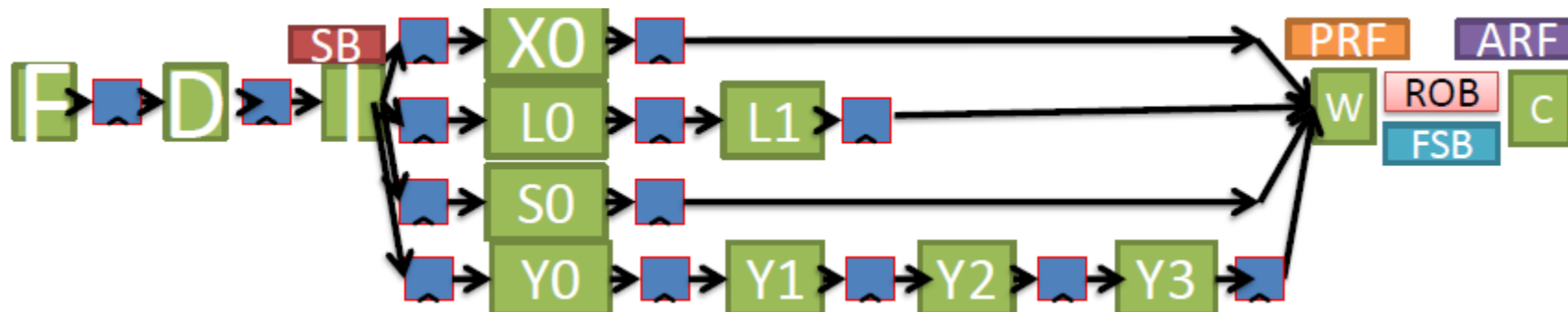
Basta matar as instruções erradas quando desvio é tomado!



Especulação e Branch: I2OI

0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W	C							
1	ADDIU	R4, R5, 1		F	D	I	X0	W	r			C						
2	MUL	R6, R1, R4			F	D	I	I	I	Y0	Y1	Y2	Y3	W	C			
3	BEQZ	R6, Target				F	D	D	D	I	I	I	I	X0	W	C		
4	ADDIU	R8, R9 ,1					F	F	F	D	D	D	D	I	--	--	--	
5	ADDIU	R10,R11,1								F	F	F	F	D	--	--	--	--
6	ADDIU	R12,R13,1												F	--	--	--	--
T															F	D	I	. . .

- Devemos tirar as instruções do pipeline para evitar escrita em PRF;
- Pode remover do ROB imediatamente ou no commit.



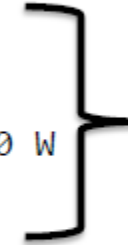
Especulação e Branch: IO3

```

0 MUL    R1, R2, R3 F D I Y0 Y1 Y2 Y3 W
1 ADDIU  R4, R5, 1   F D I X0 W
2 MUL    R6, R1, R4   F D i   I Y0 Y1 Y2 Y3 W
3 BEQZ   R6, Target   F D i           I X0 W
4 ADDIU  R8, R9, 1     F D i I X0 W
5 ADDIU  R10,R11,1     F D i I X0 W
6 ADDIU  R12,R13,1     F D i   I X0 W
7 ???
8 ???
9 ???
10???
11???
T

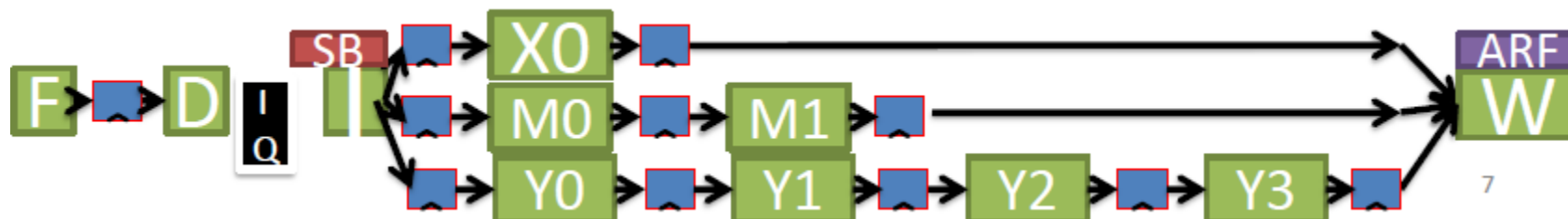
```

X0 W



Instruções
especuladas
escrevem
em ARF

- Sem controle de Especulação no IO3
- Deve parar (stall) no Branch.



Especulação e Branch: IO2I

```

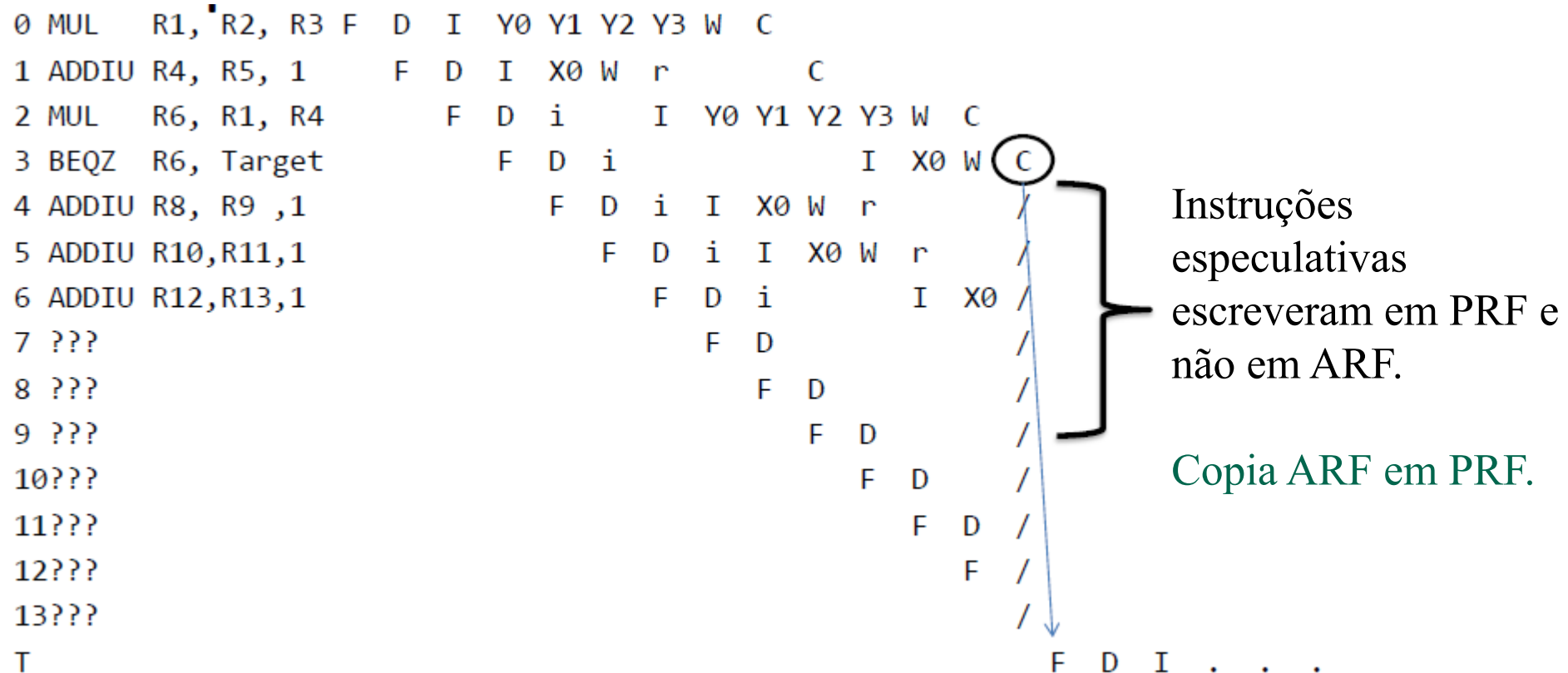
0 MUL    R1, R2, R3 F D I Y0 Y1 Y2 Y3 W C
1 ADDIU  R4, R5, 1   F D I X0 W r      C
2 MUL    R6, R1, R4   F D i   I Y0 Y1 Y2 Y3 W C
3 BEQZ   R6, Target  F D i           I X0 W C
4 ADDIU  R8, R9 ,1    F D i I X0 W r  --
5 ADDIU  R10,R11,1    F D i I X0 W  --
6 ADDIU  R12,R13,1    F D i         --
7 ???
8 ???
9 ???
10???
11???
T

```

Precisa limpar estados
especulativos em PRF.
Necessita Rollback
seletivo.



Especulação e Branch: IO2I



Temas

- Especulação e Branches
- Memory Disambiguation
- Renomeação de Registradores

Memory Disambigation

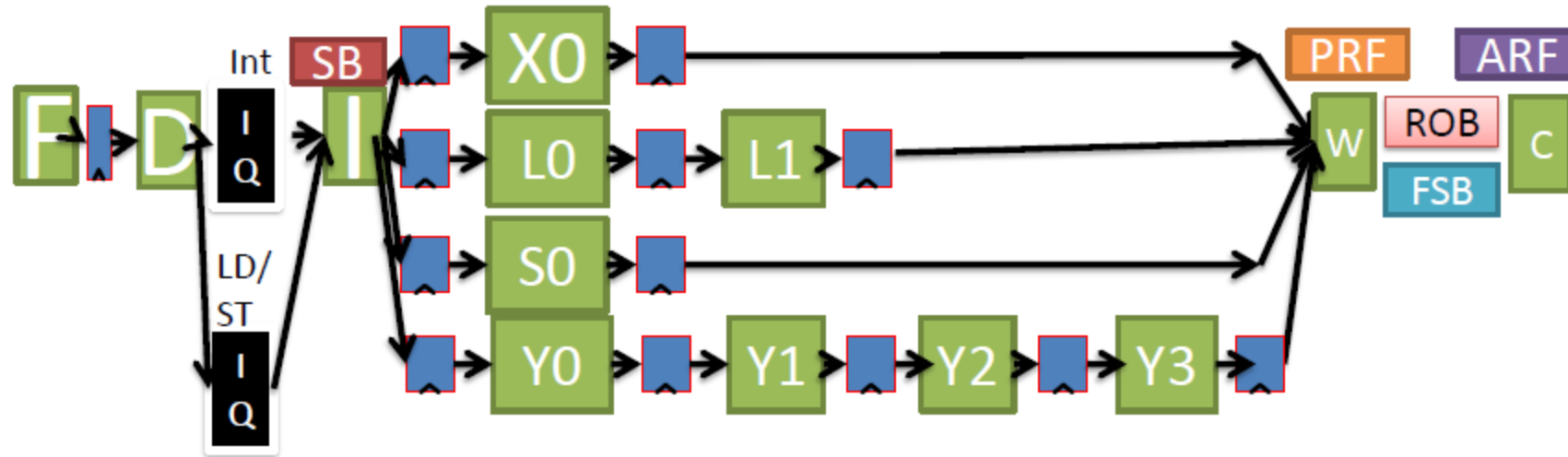
```
ST R1, 0 (R2)  
LD R3, 0 (R4)
```

Quando podemos executar o load?

Fila de Memória em ordem

- Executa todos os **loads e stores** em ordem
 - **Load e store** não podem deixar o IQ para execução até **todos os loads e stores anteriores** tenham **completado** a execução.
- Pode continuar executando **load e store** de forma **especulativa e fora de ordem** com respeito outras (**não-memória**) instruções.
- Necessita de uma **estrutura para ordenar memória**.

Em ordem fila de Memória



Previsão de dependência de Memória

ST R1, 0 (R2)

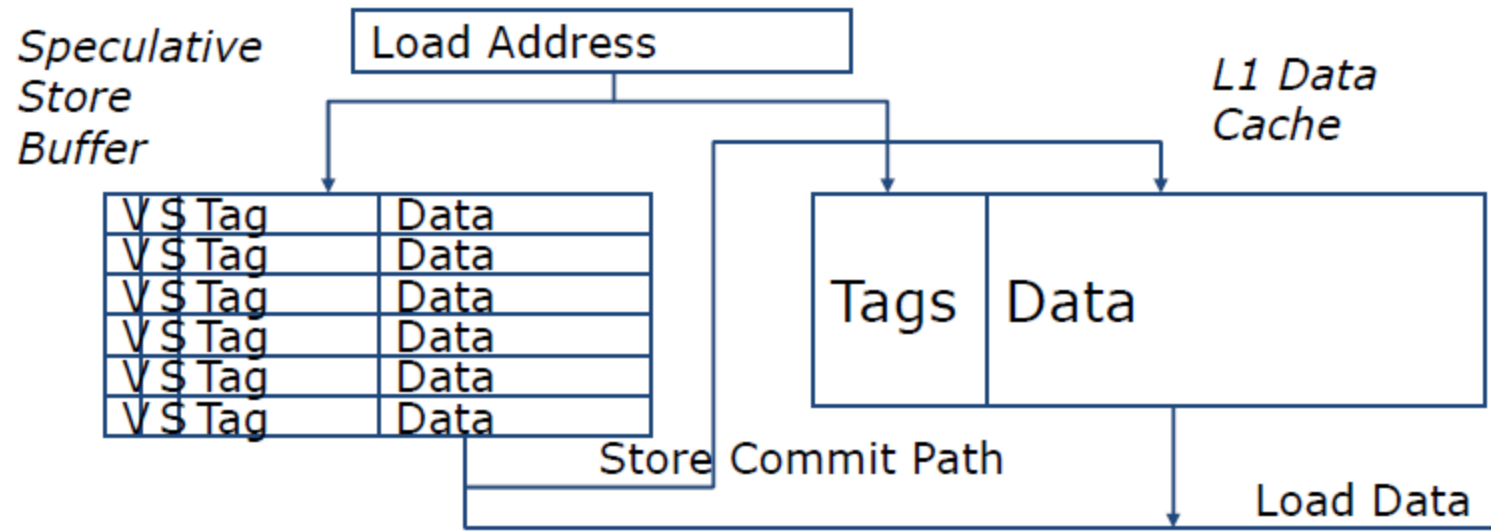
LD R3, 0 (R4)

- “**Chute**” que $r2 \neq r4$ e **execute o load**;
- Se descobrir que $r2 == r4$, **mate o load** e todas as **instruções subsequentes** – **marque o load como store-wait**;
- Execução subsequentes do mesmo load **vão esperar pelos store anteriores**;
- **Limpe o store-wait** periodicamente.

Load / Store especulativo

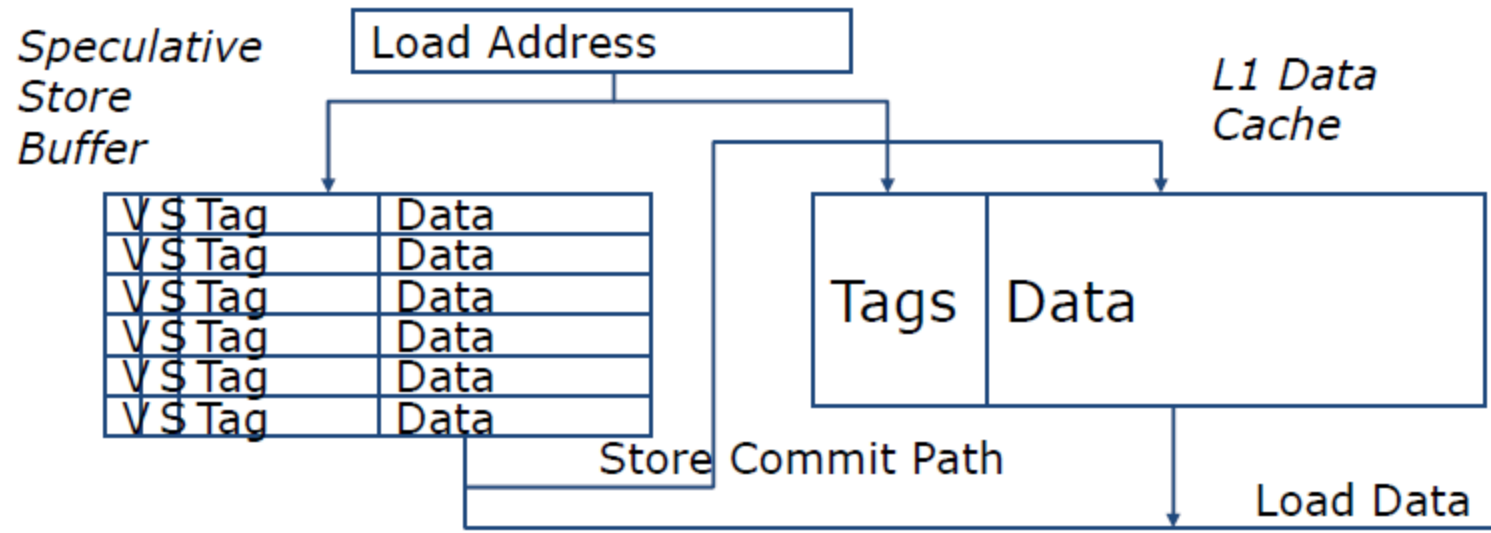
- Como nos registradores, **store não pode modificar a memória** até que a instrução chegue em **commit**;
- Logo temos o **store buffer** para guardar os dados especulativos.

Load / Store especulativo



- **Na execução do Store**
 - Marca a entrada como válida e especulativa, salva o tag e dado;
- **No commit do Store**
 - Limpa bit especulativo e eventualmente envia o dado para a cache;
- **Ao abortar o Store**
 - Limpa o bit de validade.

Load / Store especulativo



- Se o dado estiver no Store Buffer e na cache, qual usar?
 - Use o do Buffer.
- Se o mesmo endereço aparece 2 vezes do store buffer?
 - Use o mais novo.

Temas

- Especulação e Branches
- Memory Disambiguation
- Renomeação de Registradores

WAW e WAR Dependência de “Nome”

- **WAW e WAR** não são dependências reais;
- **RAW** é uma dependência real: leitor necessita o valor do escritor;
- Dependência de “Nome” existe porque **não temos “Nomes”** (registradores e endereço de memória) **suficientes**.

Dependências Reais

0	MUL	R1	R2,	R3	F	D	I	Y0	Y1	Y2	Y3	W	C						
1	MUL	R4	R1,	R5		F	D	i			I	Y0	Y1	Y2	Y3	W	C		
2	ADDIU	R6,	R4	1			F	D	i					I	X0	W	C		
3	ADDIU	R4,	R7,	1				F	D	i		I	X0	W	r			C	

WAW e WAR Dependência de “Nome”

- **WAW e WAR** não são dependências reais;
- **RAW** é uma dependência real: leitor necessita o valor do escritor;
- Dependência de “Nome” existe porque **não temos “Nomes”** (registradores e endereço de memória) **suficientes**.

WAW

0	MUL	R1,	R2,	R3	F	D	I	Y0	Y1	Y2	Y3	W	C				
1	MUL	R4,	R1,	R5		F	D	i			I	Y0	Y1	Y2	Y3	W	C
2	ADDIU	R6,	R4,	1			F	D	i					I	X0	W	C
3	ADDIU	R4,	R7,	1				F	D	i		I	X0	W	r		C

WAW e WAR Dependência de “Nome”

- **WAW e WAR** não são dependências reais;
- **RAW** é uma dependência real: leitor necessita o valor do escritor;
- Dependência de “Nome” existe porque **não temos “Nomes”** (registradores e endereço de memória) **suficientes**.

WAR

0	MUL	R1,	R2,	R3	F	D	I	Y0	Y1	Y2	Y3	W	C						
1	MUL	R4,	R1,	R5		F	D	i				I	Y0	Y1	Y2	Y3	W	C	
2	ADDIU	R6,	R4	1			F	D	i							I	X0	W	C
3	ADDIU	R4	R7,	1				F	D	i			I	X0	W	r			C

Como resolver?

Breaking all "Name" Dependencies

0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W	C				
1	MUL	R4, R1, R5		F	D	i			I	Y0	Y1	Y2	Y3	W	C
2	ADDIU	R6, R4, 1			F	D	i				I	X0	W	C	
3	ADDIU	R4, R7, 1				F	D	i		I	X0	W	r		C

IO2I Microarchitecture Conservatively Stalls

0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W	C							
1	MUL	R4, R1, R5		F	D	i			I	Y0	Y1	Y2	Y3	W	C			
2	ADDIU	R6, R4, 1			F	D	i				I	X0	W	C				
3	ADDIU	R4, R7, 1				F	D	D	D	D	D	D	D	D	I	X0	W	C

Manual Register Renaming.

0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W	C				
1	MUL	R4, R1, R5		F	D	i			I	Y0	Y1	Y2	Y3	W	C
2	ADDIU	R6, R4, 1			F	D	i				I	X0	W	C	
3	ADDIU	R8 , R7, 1				F	D	i		I	X0	W	r		C

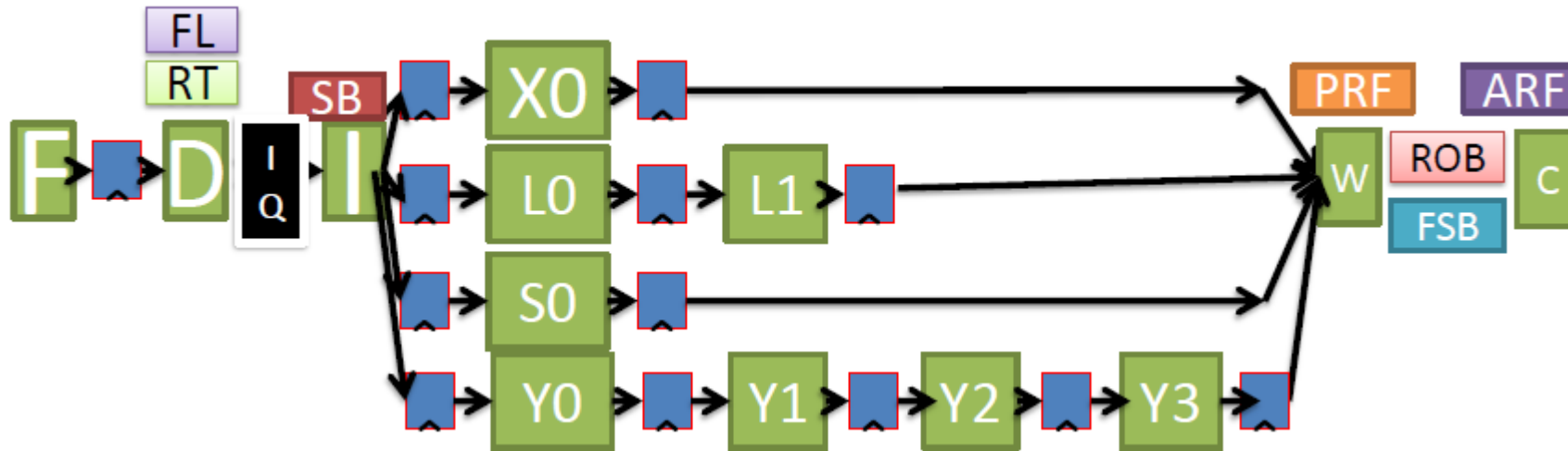
Register Renaming

- Adicionando **mais nomes** (registradores / memórias) **remove as dependências falsas (WAW e WAR)**.
- **Limites de nomes** na arquitetura;
- **Mais registradores** necessita de **mais bits** na instrução:
 - 32 registradores = 5 bits;
 - 128 registradores = 7 bits.

Register Renaming Overview

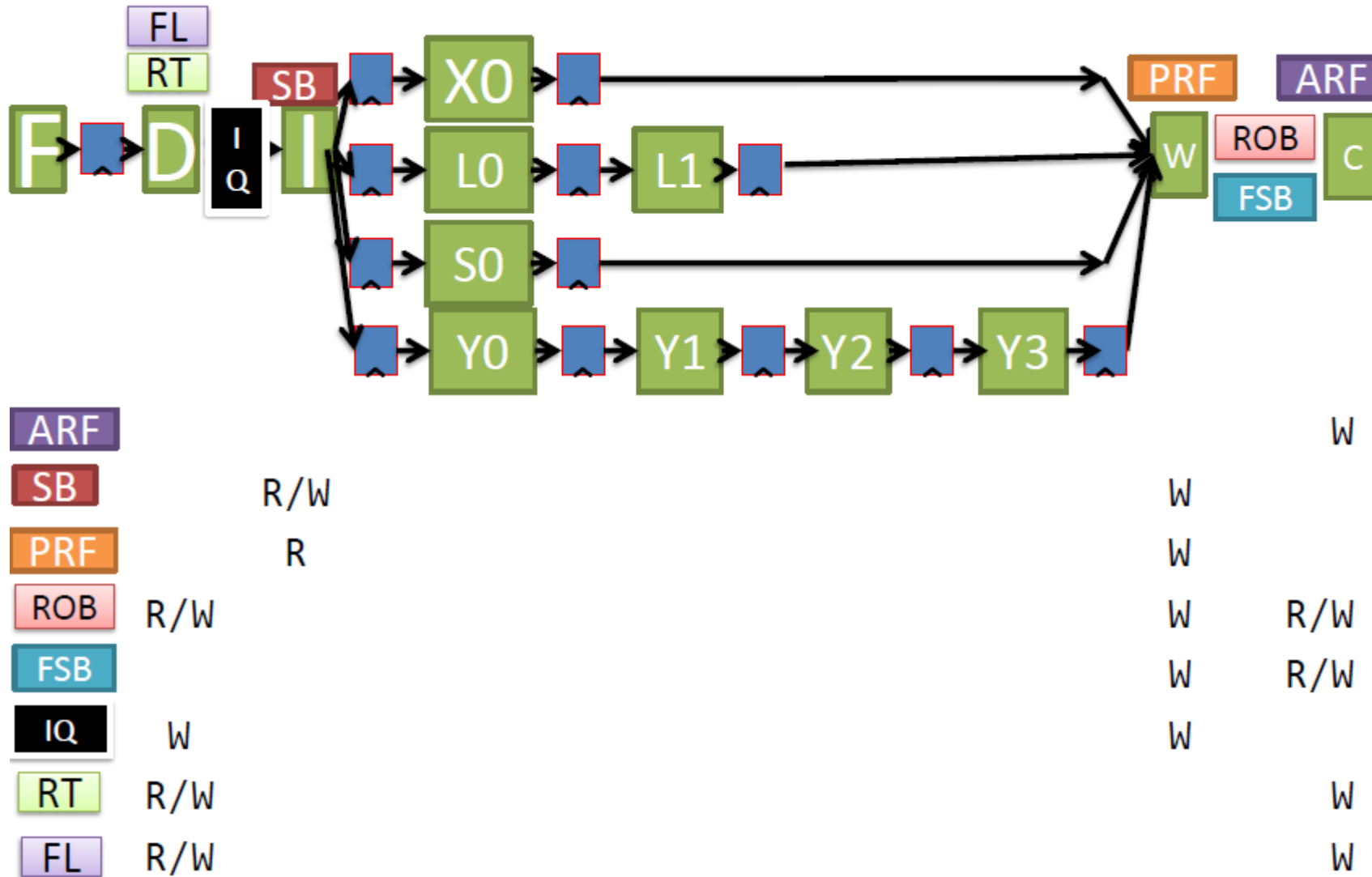
- 2 esquemas possíveis
 - **Ponteiros** no IQ e ROB
 - **Valores** no IQ e ROB
- IO2I usa ponteiros, vamos começar por ele!

IO2I Register Renaming com Ponteiros



- Mesma estrutura de dados do IO2I, exceto:
 - Adiciona 2 campos ao ROB;
 - Adiciona **Rename Table (RT)** e **Free List (FL)** de Registradores;
 - Aumenta o tamanho de **PRF** para ter mais registradores.

IO2I Register Renaming com Ponteiros



ROB Modificado

State	S	ST	V	Preg	Areg	Ppreg
--						
P						
F						
P						
P						
F						
P						
P						
--						
--						

State: (Free, Pending, Finished)

S: especulativo;

ST: Store bit;

V: Destino é Valido;

Preg: Physical Register File Specifier

Areg: Architectural Register File
Specifier

Ppreg: Previous Physical

Rename Table (RT)

	P	Preg
R1		
R2		
R3		
...		
R31		

P: Pendente

Preg: Physical Register Architectural Register
maps to.

Free List (FL)

	Free
p1	
p2	
p3	
...	
pN	

Free: Registrador está livre para renomear.

Se $\text{free} = 0$, physical register está em uso e não pode ser usado para renomear.

```

0 MUL    R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W  C
1 MUL    R4, R1, R5    F  D  i          I  Y0 Y1 Y2 Y3 W  C
2 ADDIU  R6, R4, 1      F  D  i          I  X0 W  C
3 ADDIU  R4, R7, 1      F  D  i          I  X0 W  r          C

```

	RT							FL				IQ				ROB			
Cy	D	I	W	C	R1	R2	R3	R4	R5	R6	R7	0	1	2	3	0	1	2	3
0					p0	p1	p2	p3	p4	p5	p6	p{7,8,9,10}							


```

0 MUL    R1, R2, R3 F D I Y0 Y1 Y2 Y3 W C
1 MUL    R4, R1, R5   F D i           I Y0 Y1 Y2 Y3 W C
2 ADDIU  R6, R4, 1     F D i           I X0 W C
3 ADDIU  R4, R7, 1     F D i           I X0 W C

```

Cy D I W C R1 R2 R3 R4 R5 R6 R7 0
 0 p0 p1 p2 p3 p4 p5 p6 p{7,8,9,10}

RT FL

7 registradores de
arquitetura

IQ ROB
 1 2 3 0 1 2 3

```

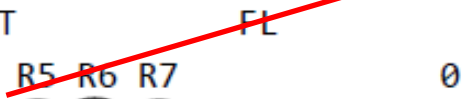
0 MUL    R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W  C
1 MUL    R4, R1, R5    F  D  i          I  Y0 Y1 Y2 Y3 W  C
2 ADDIU  R6, R4, 1      F  D  i          I  X0 W  C
3 ADDIU  R4, R7, 1      F  D  i          I  X0 W  C

```

```

          RT          FL
Cy D I W C R1 R2 R3 R4 R5 R6 R7      0
0          p0 p1 p2 p3 p4 p5 p6 p{7,8,9,10}

```



11 registradores físicos
Círculos – não pendente

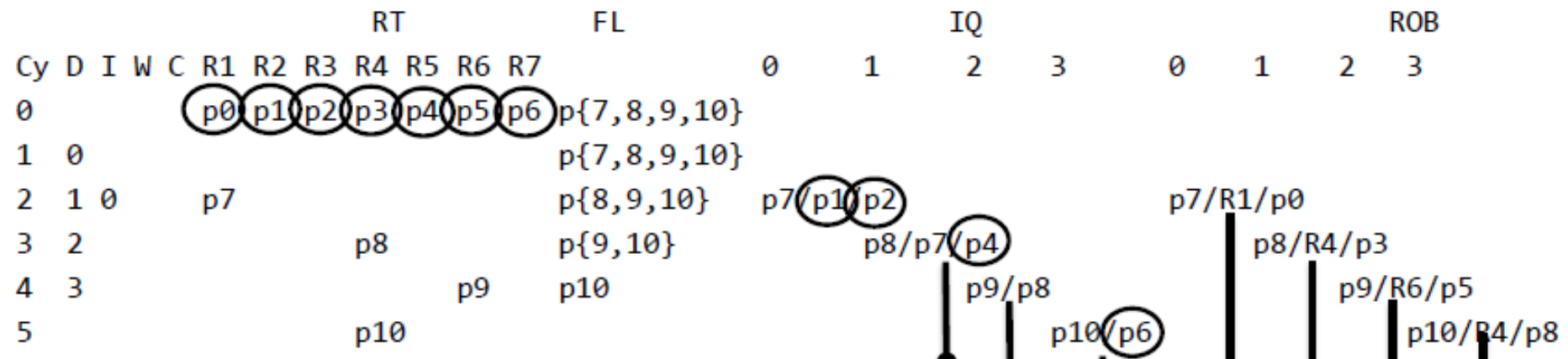
0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W	C					
1	MUL	R4, R1, R5		F	D	i			I	Y0	Y1	Y2	Y3	W	C	
2	ADDIU	R6, R4, 1			F	D	i						I	X0	W	C
3	ADDIU	R4, R7, 1				F	D	i		I	X0	W	r			C

														RT	FL				IQ				ROB			
Cy	D	I	W	C	R1	R2	R3	R4	R5	R6	R7					0	1	2	3	0	1	2	3			
0					p0	p1	p2	p3	p4	p5	p6	p{7,8,9,10}														
1	0											p{7,8,9,10}														

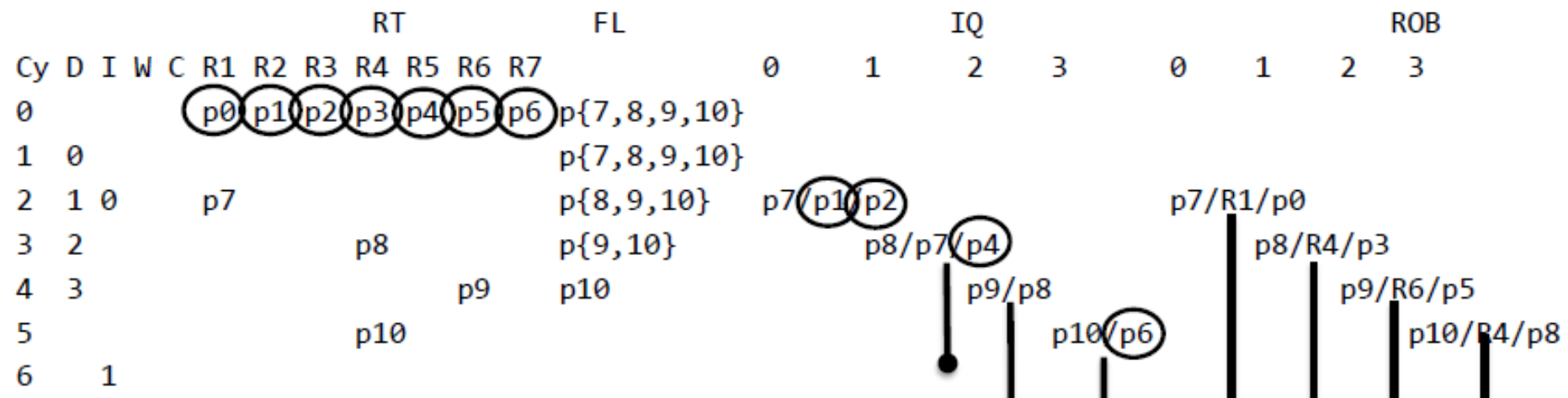
0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W	C				
1	MUL	R4, R1, R5		F	D	i			I	Y0	Y1	Y2	Y3	W	C
2	ADDIU	R6, R4, 1			F	D	i					I	X0	W	C
3	ADDIU	R4, R7, 1				F	D	i		I	X0	W	r		C

RT													FL				IQ				ROB			
Cy	D	I	W	C	R1	R2	R3	R4	R5	R6	R7					0	1	2	3	0	1	2	3	
0					p0	p1	p2	p3	p4	p5	p6	p{7,8,9,10}												
1	0											p{7,8,9,10}												
2	1	0			p7							p{8,9,10}	p7	p1	p2					p7/R1/p0				

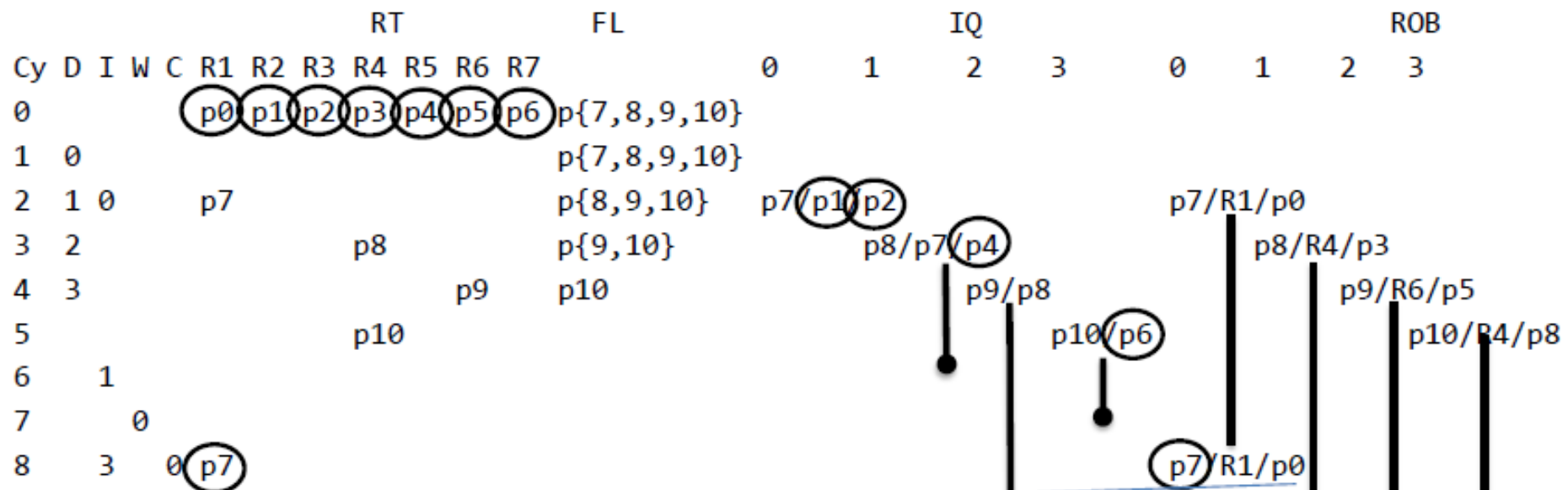
0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W	C
1	MUL	R4, R1, R5		F	D	i			I	Y0	Y1
2	ADDIU	R6, R4, 1			F	D	i			I	X0
3	ADDIU	R4, R7, 1			F	D	i		I	X0	W



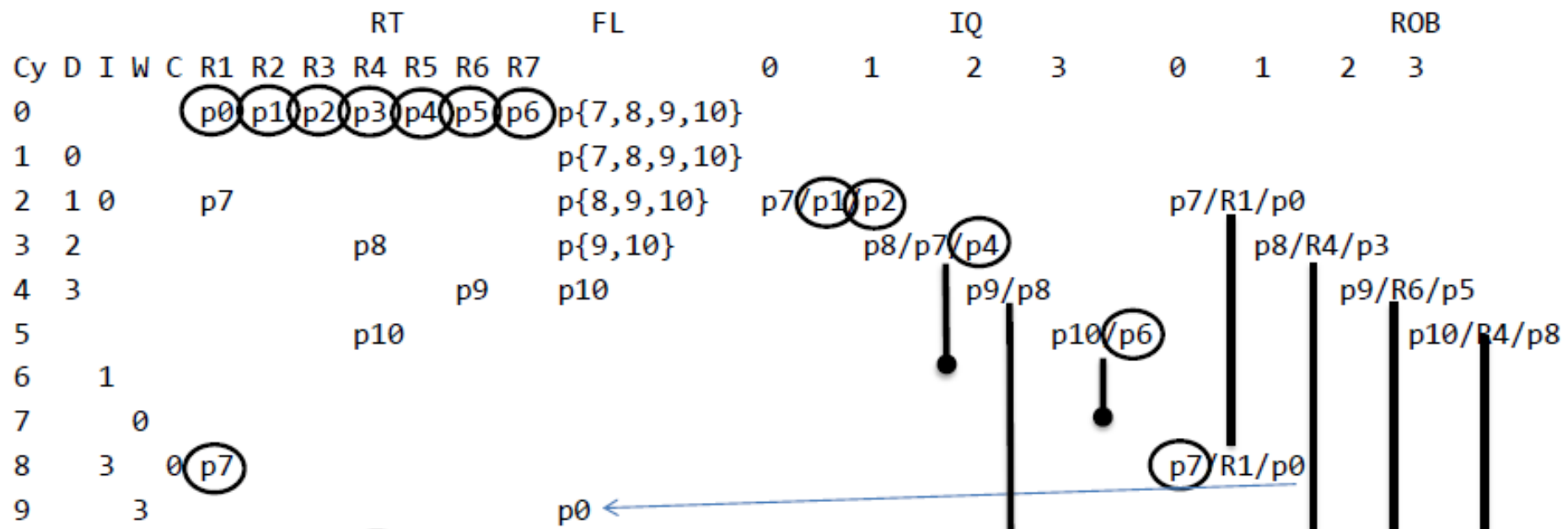
0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W	C				
1	MUL	R4, R1, R5		F	D	i			I	Y0	Y1	Y2	Y3	W	C
2	ADDIU	R6, R4, 1			F	D	i				I	X0	W	C	
3	ADDIU	R4, R7, 1			F	D	i		I	X0	W	r		C	



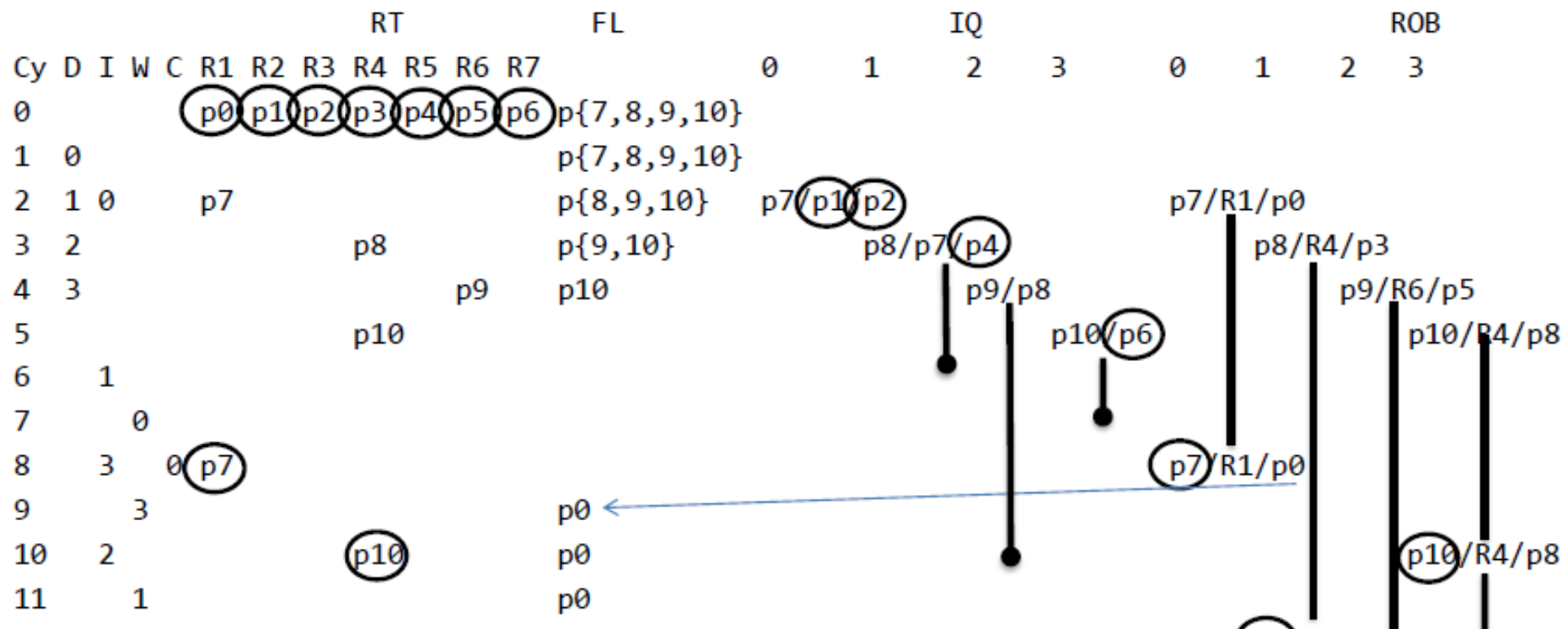
0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W	C
1	MUL	R4, R1, R5		F	D	i			I	Y0	Y1 Y2 Y3 W C
2	ADDIU	R6, R4, 1			F	D	i			I	X0 W C
3	ADDIU	R4, R7, 1			F	D	i		I	X0	W r C



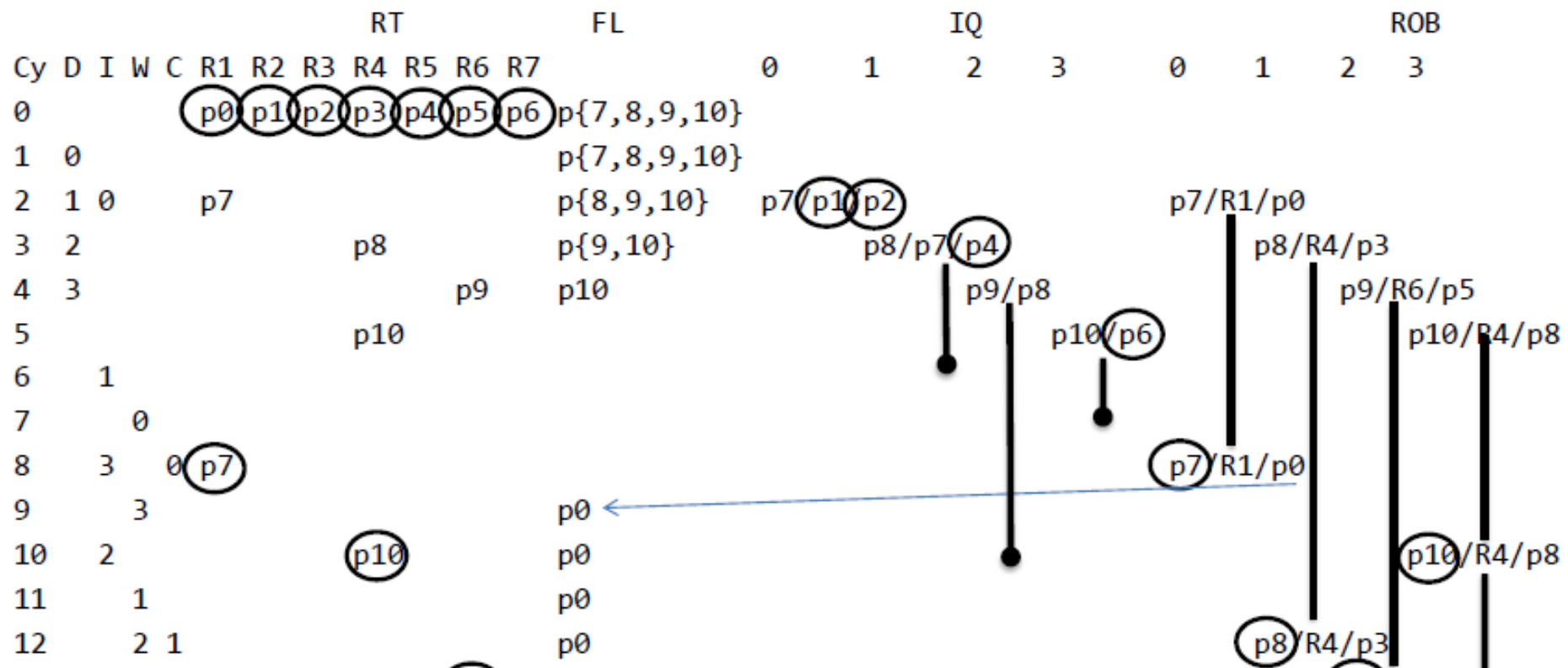
0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W	C	
1	MUL	R4, R1, R5		F	D	i			I	Y0	Y1	Y2
2	ADDIU	R6, R4, 1			F	D	i			I	X0	W
3	ADDIU	R4, R7, 1			F	D	i		I	X0	W	r



0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W	C			
1	MUL	R4, R1, R5		F	D	i		I	Y0	Y1	Y2	Y3	W	C
2	ADDIU	R6, R4, 1			F	D	i			I	X0	W	C	
3	ADDIU	R4, R7, 1			F	D	i		I	X0	W	r		C



0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W	C			
1	MUL	R4, R1, R5		F	D	i		I	Y0	Y1	Y2	Y3	W	C
2	ADDIU	R6, R4, 1			F	D	i			I	X0	W	C	
3	ADDIU	R4, R7, 1			F	D	i		I	X0	W	r		C

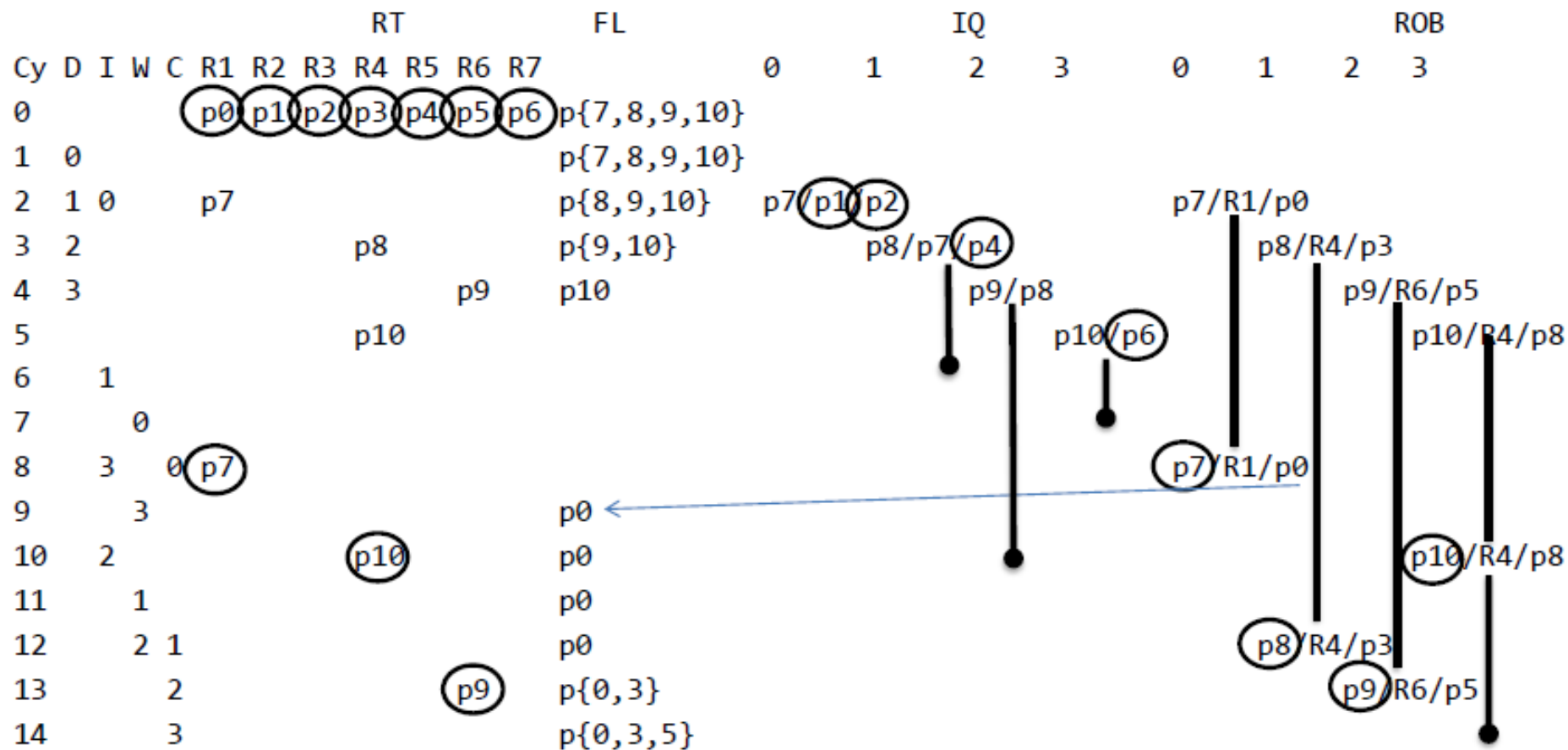


C



0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W	C			
1	MUL	R4, R1, R5		F	D	i		I	Y0	Y1	Y2	Y3	W	C
2	ADDIU	R6, R4, 1			F	D	i			I	X0	W	C	
3	ADDIU	R4, R7, 1			F	D	i		I	X0	W	r		

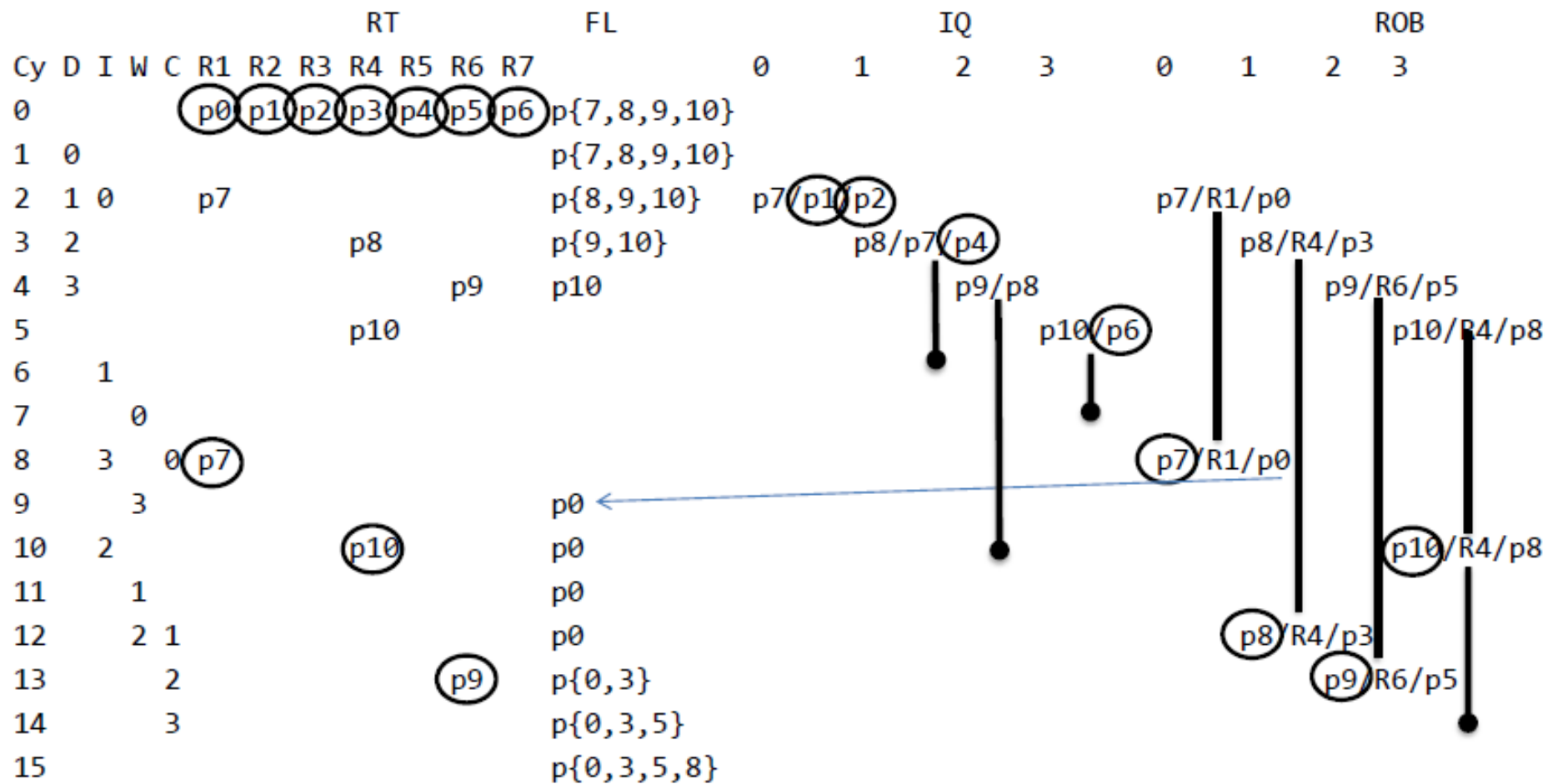
C



```

0 MUL    R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W  C
1 MUL    R4, R1, R5    F  D  i          I  Y0 Y1 Y2 Y3 W  C
2 ADDIU  R6, R4, 1      F  D  i          I  X0 W  C
3 ADDIU  R4, R7, 1      F  D  i          I  X0 W  r          C

```

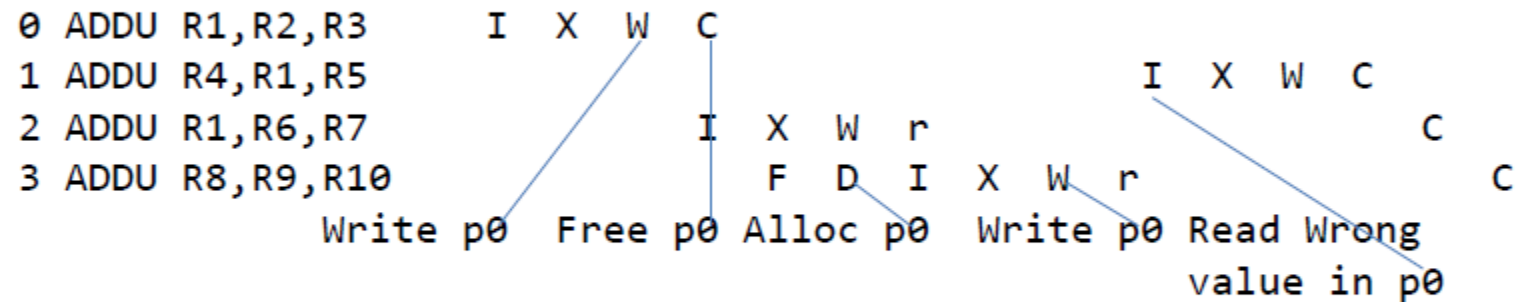


Liberando os registradores físicos

```
ADDU R1,R2,R3    <-Assume Arch. Reg R1 maps to Phys. Reg p0
ADDU R4,R1,R5
ADDU R1,R6,R7     <-Next write of Arch Reg R1, Mapped to Phys. Reg p1
ADDU R8,R9,R10
```

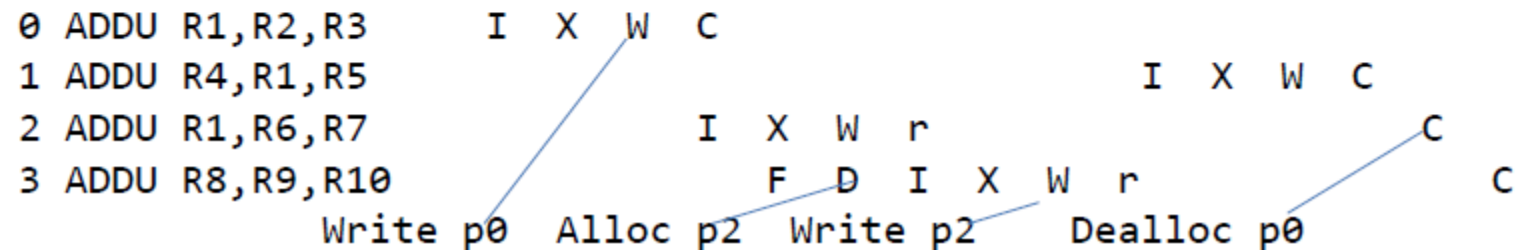
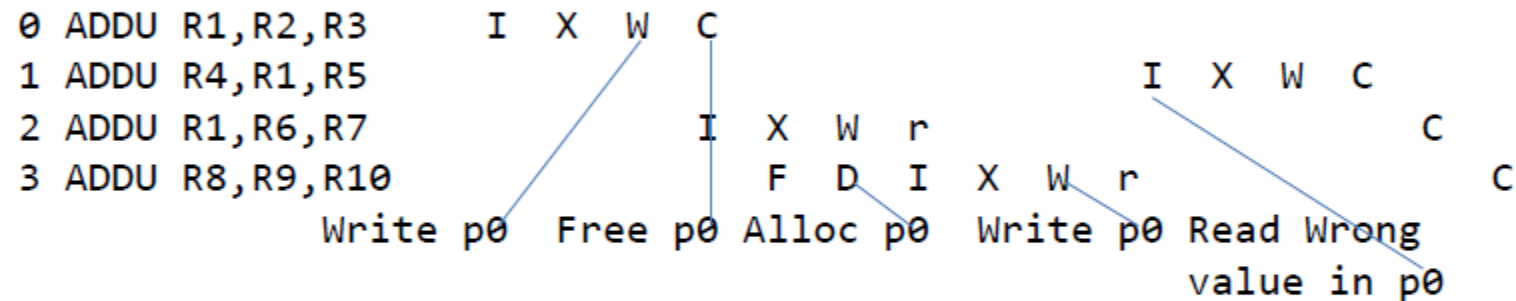
Liberando os registradores físicos

ADDU R1,R2,R3 <-Assume Arch. Reg R1 maps to Phys. Reg p0
ADDU R4,R1,R5
ADDU R1,R6,R7 <-Next write of Arch Reg R1, Mapped to Phys. Reg p1
ADDU R8,R9,R10



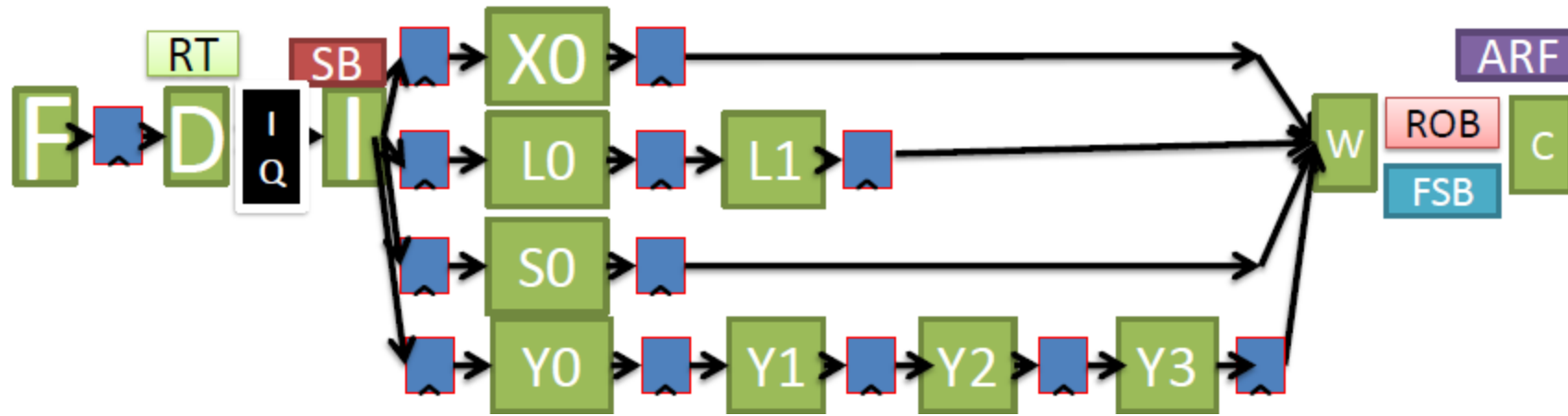
Liberando os registradores físicos

ADDU R1,R2,R3 <-Assume Arch. Reg R1 maps to Phys. Reg p0
ADDU R4,R1,R5
ADDU R1,R6,R7 <-Next write of Arch Reg R1, Mapped to Phys. Reg p1
ADDU R8,R9,R10



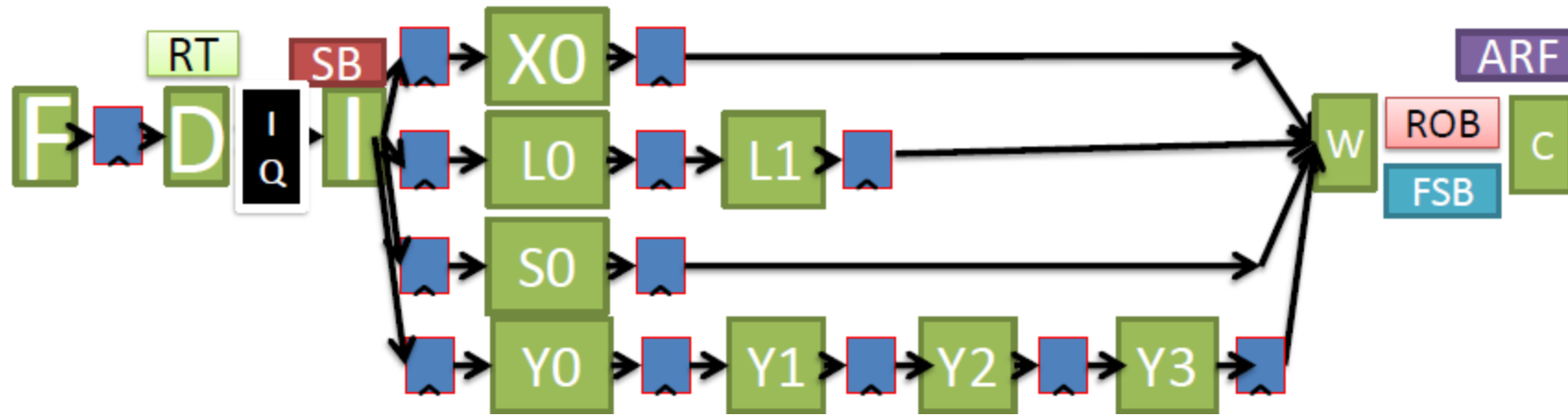
- Se Arch. Reg Ri é mapeado para Phys. Reg. Pj, nós só podemos liberar Pj quando a próxima instrução que escrever Ri commit.

IO2I Register Renaming com Valores



- Mesma estrutura de dados do IO2I, exceto:
 - **ROB** Modificado (Valores ao invés de Espec. de Reg.);
 - **RT** Modificado;
 - **IQ** Modificado;
 - Sem **FL**;
 - Sem **PRF**, valores em ROB.

IO2I Register Renaming com Valores



Cache	Processor	Cache	Cache	Cache
ARF	R			W
SB		R/W	W	
ROB	R/W		W	R/W
FSB			W	R/W
IQ	W		W	
RT	R/W			W

ROB Modificado

State	S	ST	V	Value	Areg
--					
P					
F					
P					
P					
F					
P					
P					
--					
--					

State: (Free, Pending, Finished)

S: especulativo;

ST: Store bit;


V: Destino é Valido;

Value: Valor atual do registrador

Areg: Architectural Register File
Specifier

IQ Modificado

Op	Imm	S	V	Dest	V	P	Src0	V	P	Src1



Op: Opcode

Imm: Imediato;

S: Especulativo;

V: Valido;

P: Pendente

Se pendente, Src contém index em ROB. Como um indentificador Preg .

Rename Table (RT)

	V	P	Preg
R1			
R2			
R3			
...			
R31			

V: Bit validade

P: Pendente

Preg: Index no ROB.

V:

if V==0

Valores em ARF estão atualizados;

if V==1

Valores *in-flight* ou em ROB

P:

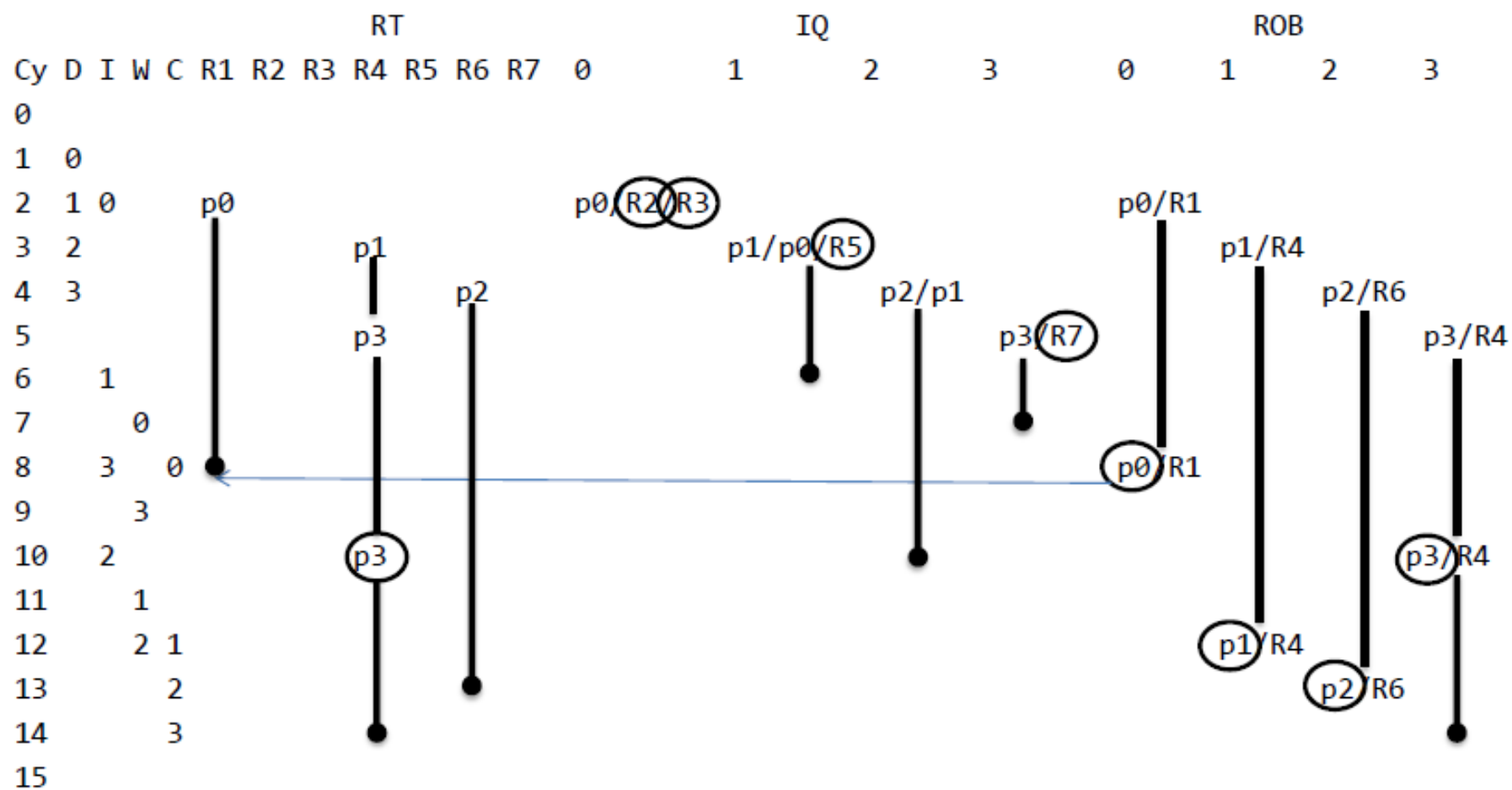
if P==0

Valores ROB;

if P==1

Valores *in-flight*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	MUL	R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W	C					
1	MUL	R4, R1, R5	F	D	i			I	Y0	Y1	Y2	Y3	W	C		
2	ADDIU	R6, R4, 1		F	D	i					I	X0	W	C		
3	ADDIU	R4, R7, 1			F	D	i		I	X0	W	r			C	



Agradecimiento

David Wentzlaff (Princeton University)