MO401

Arquitetura de Computadores I

2006

Prof. Paulo Cesar Centoducatte

ducatte@ic.unicamp.br

www.ic.unicamp.br/~ducatte

MO401

Arquitetura de Computadores I

Paralelismo em Nível de Instruções
Exploração Dinâmica: Branch Prediction e Múltiplo Issue

"Computer Architecture: A Quantitative Approach" - (Capítulo 3)

Paralelismo em Nível de Instruções
Exploração Dinâmica

- Scheduling Dinâmico
  - Algoritmo de Tomasulo – continuação
- Algoritmo Tomasulo e Branch Prediction
- Múltiplas Instruções Issuing/Cycle
- Esquemas de Branch Prediction
  1. 1-bit Branch-Prediction Buffer
  2. 2-bit Branch-Prediction Buffer
  3. Correlating Branch Prediction Buffer
  4. Tournament Branch Predictor
  5. Branch Target Buffer
  6. Return Address Predictors

Técnicas para redução de stalls

Capítulo 3

Capítulo 4

Technique	Reduces
Dynamic scheduling	Data hazard stalls
Dynamic branch prediction	Control stalls
Issuing multiple instructions per cycle	Ideal CPI
Speculation	Data and control stalls
Dynamic memory disambiguation	Data hazard stalls involving memory
Loop unrolling	Control hazard stalls
Basic compiler pipeline scheduling	Data hazard stalls
Compiler dependence analysis	Ideal CPI and data hazard stalls
Software pipelining and trace scheduling	Ideal CPI and data hazard stalls
Compiler speculation	Ideal CPI, data and control stalls

Algoritmo de Tomasulo

- IBM 360/91 (1967 - não havia caches; tempo de acesso à memória grande e instruções de FP com grandes latências (delay))

- Idéia: Alto desempenho sem compilador especial
- Um pequeno número de registradores floating point (4 no 360) evita um bom scheduling das operações pelo compilador.
  - Tomasulo: Como ter efetivamente mais registradores ? Como resolver os hazards RAW, WAW e RAW?
  - — seguir quando os operandos estiverem prontos e renaming implementado no hardware!

- Descendentes:
  - Alpha 21264, HP 8000, MIPS 10000, Pentium III, PowerPC 604, …

Algoritmo de Tomasulo

- Controle & buffers distribuído na Function Units (FU)
  - FU buffers chamado de "Reservation Stations"; mantem operandos pendentes

- Substituição dos Registradores nas instruções por valores ou apontadores para a Reservation Stations (RS): denominado register renaming ;
  - Evita os hazards WAR e WAW
  - Se existe mais reservation stations que registradores, então pode-se fazer otimizações não realizadas pelos compiladores

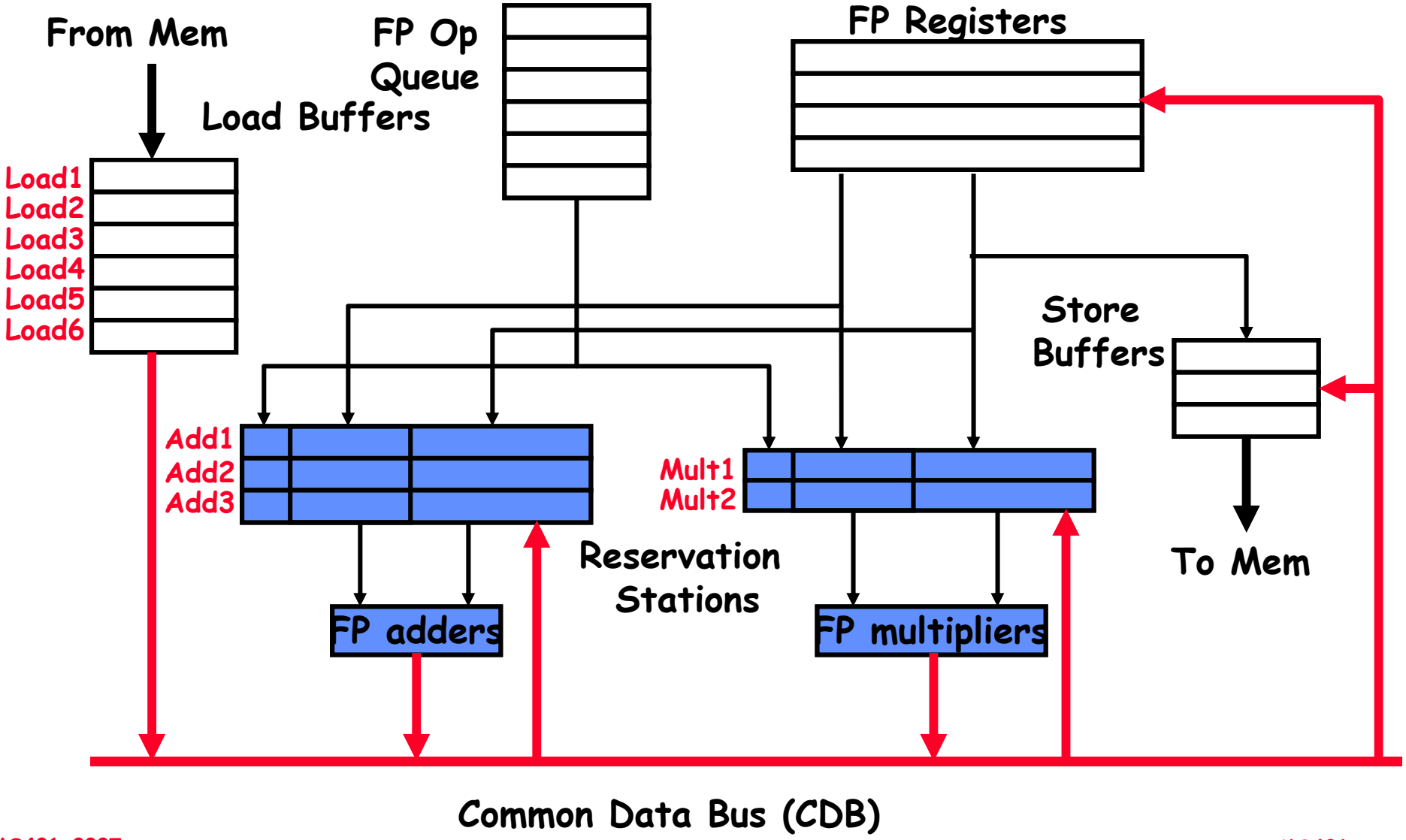
- Resultados da RS para a FU, (sem usar os registradores), broadcasts dos resultados para todas as FUs usando o Common Data Bus

- Load e Stores tratados como FUs com RSs

Algoritmo de Tomasulo
Exemplo

- Foco: Unidades de ponto-flutuante e load-store
- Cada estágio pode ter um número arbitrário de ciclos
- Multiplas unidades funcionais
- Diferentes instruções possuem tempos diferentes no estágio EX
- Unidades disponíveis: load-store; mult e adder

Estrutura Básica de uma Implementação do Algoritmo de Tomasulo (para o MIPS)



Reservation Station

Op: Operação a ser executada na unidade (e.g., + or –)

Vj, Vk: Valores dos operantos Fontes

- Store buffers tem campos V, resultados devem ser armazenados

Qj, Qk: Reservation stations produzirá os operandos correspondentes (valores a serem escritos)

- Qj,Qk = 0 => ready
- Store buffers tem somente Qi para RS producing result

Busy: Indica que a reservation station e sua FU estão busy

A: Mantém informação sobre o end. de memória calculado para load ou store

Register result status (campo Qi no register file) — Indica para cada registrador a unidade funcional (reservation station) que irá escreve-lo. Em branco se não há instruções pendentes que escreve no registrador.

Exemplo 2: Tomasulo & Loop

```
Loop:LD      F0, 0(R1)
     MULTD   F4, F0,F2
     SD      F4, 0(R1)
     SUBI    R1, R1,#8
     BNEZ    R1  Loop
```

Branch Prediction ~ Taken

- Assuma que Mult gasta 4 clocks
- Assuma que o 1° load gasta 8 clocks (L1 cache miss), 2° load gasta 1 clock (hit)
- Para maior compreenção mostraremos os clocks para SUBI e BNEZ
  - Real: instruções inteiras são tratadas pela unidade de inteiros
  - 2 iterações

Exemplo 2: Tomasulo & Loop

Contador de Iterações

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1			
1	MULTD	F4	F0	F2			
1	SD	F4	0	R1			
2	LD	F0	0	R1			
2	MULTD	F4	F0	F2			
2	SD	F4	0	R1			

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

+ Store Buffers

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Code:			
LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Instruções do Loop

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
0	80	Fu									

Valor do Registrador usado para controle das iterações

Exemplo 2: Tomasulo & Loop: Ciclo 1

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Code:

```
LD    F0   0    R1   <-
MULTD F4   F0   F2
SD    F4   0    R1
SUBI  R1   R1   #8
BNEZ  R1   Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
1	80	Fu	Load1								

Exemplo 2: Tomasulo & Loop: Ciclo 2

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
2	80	Fu	Load1		Mult1						

Exemplo 2: Tomasulo & Loop: Ciclo 3

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*S1 Vk*	*S2 Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

```
LD    F0  0   R1
MULTD F4  F0  F2
SD    F4  0   R1
SUBI  R1  R1  #8
BNEZ  R1  Loop
```

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
3	**80**	*Fu*	Load1		Mult1						

- **Renaming implícito**
- **Porque o Load não está em Exec?**

Exemplo 2: Tomasulo & Loop: Ciclo 4

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
4	80	Fu	Load1		Mult1						

- Dispatching a Instrução SUBI (não está na FP queue)

Exemplo 2: Tomasulo & Loop: Ciclo 5

Instruction status:

					Exec	Write
ITER	Instruction		*j*	*k*	*Issue*	*Comp*	*Result*
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	**R1**		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
5	**72**	*Fu*	Load1		Mult1						

- e instrução BNEZ (não está na FP queue)

Exemplo 2: Tomasulo & Loop: Ciclo 6

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

```
LD    F0  0  R1   <-
MULTD F4  F0 F2
SD    F4  0  R1
SUBI  R1  R1 #8
BNEZ  R1  Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
6	72	Fu	Load2		Mult1						

- Note que F0 não recebe o valor carregado da posição location 80

Exemplo 2: Tomasulo & Loop: Ciclo 7

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		
2	MULTD	F4	F0	F2	7		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

```
LD    F0  0  R1
MULTD F4  F0 F2   <-
SD    F4  0  R1
SUBI  R1  R1 #8
BNEZ  R1  Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
7	72	Fu	Load2		Mult2						

- Register file completamente desacoplado da computação
- A primeira e segunda iteração completamente sobrepostas

Exemplo 2: Tomasulo & Loop: Ciclo 8

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

```
LD    F0  0  R1
MULTD F4  F0 F2
SD    F4  0  R1   <-
SUBI  R1  R1 #8
BNEZ  R1  Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
8	72	Fu	Load2		Mult2						

Exemplo 2: Tomasulo & Loop: Ciclo 9

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

```
LD    F0  0  R1
MULTD F4  F0 F2
SD    F4  0  R1
SUBI  R1  R1 #8
BNEZ  R1  Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
9	72	Fu	Load2		Mult2						

- Load1 completa: quem está esperando?
- Nota: Dispatching SUBI

Exemplo 2: Tomasulo & Loop: Ciclo 10

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	Multd	M[80]	R(F2)		
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

```
LD    F0  0   R1
MULTD F4  F0  F2
SD    F4  0   R1
SUBI  R1  R1  #8
BNEZ  R1  Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
10	64	Fu	Load2		Mult2						

- Load2 completa: quem está esperando?

Nota: Dispatching BNEZ

Exemplo 2: Tomasulo & Loop: Ciclo 11

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
3	Mult1	Yes	Multd	M[80]	R(F2)		
4	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

```
LD    F0  0  R1   <-
MULTD F4  F0 F2
SD    F4  0  R1
SUBI  R1  R1 #8
BNEZ  R1  Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
11	64	Fu	Load3		Mult2						

- Próximo load em seqüencia

Exemplo 2: Tomasulo & Loop: Ciclo 12

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
2	Mult1	Yes	Multd	M[80]	R(F2)		
3	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
12	64	Fu	Load3		Mult2						

- Por que não há issue do 3° mult?

Exemplo 2: Tomasulo & Loop: Ciclo 13

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
1	Mult1	Yes	Multd	M[80]	R(F2)		
2	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

```
LD    F0   0    R1
MULTD F4   F0   F2   <-
SD    F4   0    R1
SUBI  R1   R1   #8
BNEZ  R1   Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
13	64	Fu	Load3		Mult2						

- Por que não há issue do 3° store?

Exemplo 2: Tomasulo & Loop: Ciclo 14

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7		
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
0	Mult1	Yes	Multd	M[80]	R(F2)		
1	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

```
LD    F0   0    R1
MULTD F4   F0   F2   <-
SD    F4   0    R1
SUBI  R1   R1   #8
BNEZ  R1   Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
14	64	Fu	Load3		Mult2						

- Mult1 completa. Quem está esperando?

Exemplo 2: Tomasulo & Loop: Ciclo 15

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	Yes	Multd	M[72]	R(F2)		

Code:

```
LD    F0   0    R1
MULTD F4   F0   F2   <-
SD    F4   0    R1
SUBI  R1   R1   #8
BNEZ  R1   Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
15	64	Fu	Load3		Mult2						

- **Mult2 completa. Quem está esperando?**

Exemplo 2: Tomasulo & Loop: Ciclo 16

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	[72]*R2
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

```
LD     F0   0    R1
MULTD  F4   F0   F2
SD     F4   0    R1
SUBI   R1   R1   #8
BNEZ   R1   Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
16	64	Fu	Load3		Mult1						

Exemplo 2: Tomasulo & Loop: Ciclo 17

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	[72]*R2
Store3	Yes	64	Mult1

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

```
LD    F0  0   R1
MULTD F4  F0  F2
SD    F4  0   R1   <-
SUBI  R1  R1  #8
BNEZ  R1  Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
17	64	Fu	Load3		Mult1						

Exemplo 2: Tomasulo & Loop: Ciclo 18

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3	18	
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8		

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	Yes	80	[80]*R2
Store2	Yes	72	[72]*R2
Store3	Yes	64	Mult1

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

```
LD    F0  0   R1
MULTD F4  F0  F2
SD    F4  0   R1
SUBI  R1  R1  #8   <-
BNEZ  R1  Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
18	64	Fu	Load3		Mult1						

Exemplo 2: Tomasulo & Loop: Ciclo 19

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3	18	19
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8	19	

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	Yes	64	
Store1	No		
Store2	Yes	72	[72]*R2
Store3	Yes	64	Mult1

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

```
LD    F0  0  R1
MULTD F4  F0 F2
SD    F4  0  R1
SUBI  R1  R1 #8
BNEZ  R1  Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
19	56	Fu	Load3		Mult1						

Exemplo 2: Tomasulo & Loop: Ciclo 20

Instruction status:

ITER	Instruction		j	k	Issue	Exec Comp	Write Result
1	LD	F0	0	R1	1	9	10
1	MULTD	F4	F0	F2	2	14	15
1	SD	F4	0	R1	3	18	19
2	LD	F0	0	R1	6	10	11
2	MULTD	F4	F0	F2	7	15	16
2	SD	F4	0	R1	8	19	20

	Busy	Addr	Fu
Load1	Yes	56	
Load2	No		
Load3	Yes	64	
Store1	No		
Store2	No		
Store3	Yes	64	Mult1

Reservation Stations:

Time	Name	Busy	Op	Vj	Vk (S1)	Qj (S2)	Qk (RS)
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load3	
	Mult2	No					

Code:

```
LD    F0  0   R1
MULTD F4  F0  F2
SD    F4  0   R1
SUBI  R1  R1  #8
BNEZ  R1  Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
20	56	Fu	Load1		Mult1						

- In-order issue, out-of-order execution and out-of-order completion.

Por que Tomasulo sobrepõe iterações de loops?

- Register renaming
  - Múltiplas iterações usam diferentes destinos físicos para os registradores (loop unrolling dinâmico).
- Reservation stations
  - Permite que instruções adiantem em relação as operações inteiras de controle
  - Valores antigos dos regs estão no buffer – evita WAR stall
- Tomasulo monta o data flow dependency graph *on the fly*.

O esquema de Tomasulo tem duas grandes vantagens

(1) Lógica de detecção de hazard distribuída

- **reservation stations e CDB distribuído**
- **Se múltiplas instruções esperam por um único resultado elas podem recebe-lo simultaneamente (broadcast no CDB)**
- **Se é usado um register file centralizado, as unidades só poderão ler seus resultados quando o barramento estiver disponível.**

(2) Eliminação dos hazards WAW e WAR stalls

Interrupções Precisas?

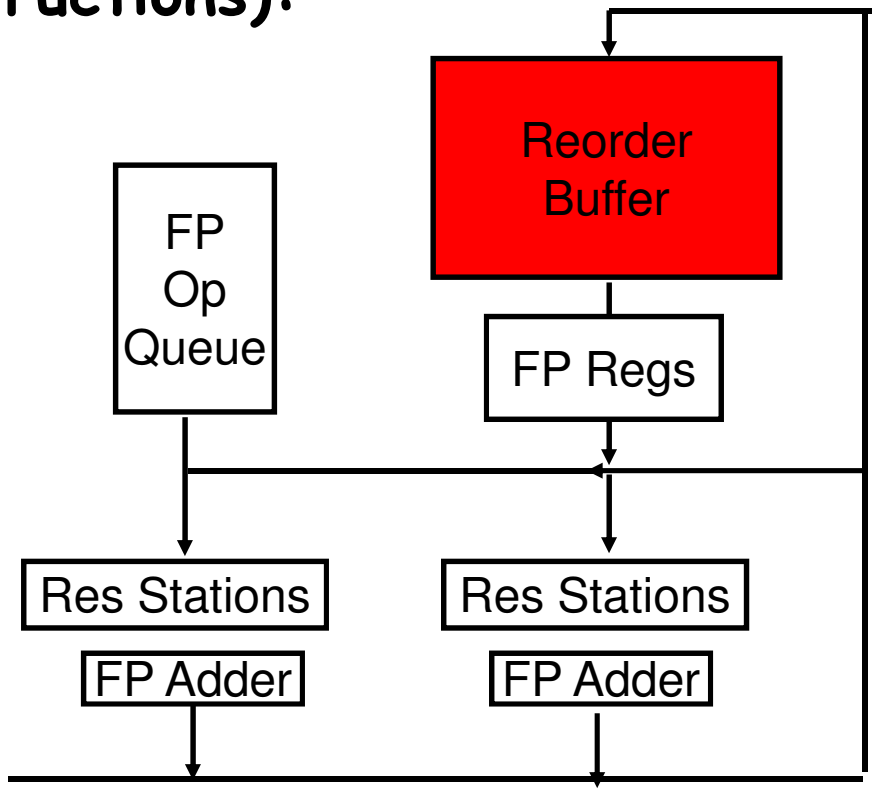
- Tomasulo:
  - In-order issue,
  - out-of-order execution,
  - out-of-order completion
- É necessário cuidados com aspectos do out-of-order completion para que se tenha breakpoints precisos no stream de instruções.

Relação entre Interrupções Precisas e Especulação

- Especulação é uma forma de adivinhação.
- Importante para *branch prediction*:
  - É necessário um bom sistema de adivinhação para prevê a direção do branch.
- Se especularmos e estivermos errado, é necessário voltar e reiniciar a execução a partir do ponto em que foi feita a previsão incorreta :
  - Isto é semelhante ao que ocorre nas exceções precisas!
- Técnica para interrupções/exceções precisas e especulação: *in-order completion* ou *commit*

Suporte de HW para Interrupções Precisas

- Buffer para os resultados de instruções que não terminaram (uncommitted instructions): *reorder buffer*
  - 3 campos: instr, destino, valor
  - Usar número do reorder buffer no lugar da reservation station quando a execução completa
  - Suprir operandos entre execution complete & commit
  - (Reorder buffer pode ser operando source => mais regs como RS)
  - Instructions commit
  - Uma vez que a instrução commits, o resultado é colocado no registrador
  - Como resultado é mais fácil desfazer instruções especuladas devido a um "mispredicted branches" ou uma exceção



Algoritmo de Tomasulo Especulativo 4 passos

1. Issue — pega a instrução da FP Op Queue

   Se há reservation station e reorder buffer slot livres: issue instr & envia operandos & reorder buffer no. para o destino (este estágio é comumente chamado de "dispatch")

2. Execution — opera sobre os operandos (EX)

   Quando ambos os operandos estão prontos executa; se não monitora o CDB a espera do resultado; quando ambos estiverem na reservation station, executa; verifica se há RAW

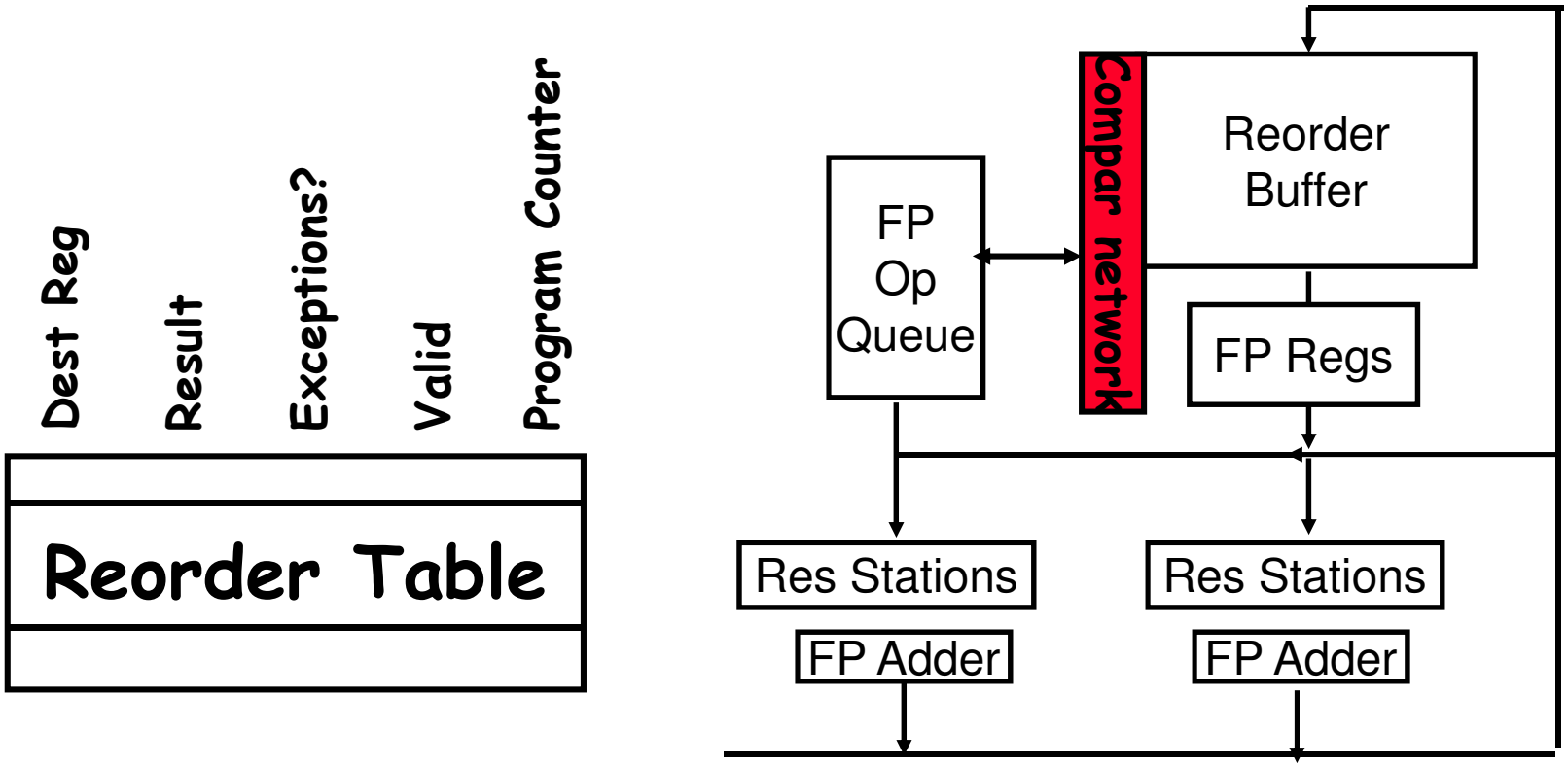
3. Write result — termina a execução (WB)

   Escreve, usando o Common Data Bus, em todas FUs que estão esperando por esse valor & reorder buffer; marca a reservation station como disponível.

4. Commit — atualiza o registrador com o resultado em reorder

   Quando a instr. é a primeira no reorder buffer & o resultado está presente: atualiza o registrador com o resultado (ou store na memória) e remove a instr. do reorder buffer. Se Mispredicted branch então flushes reorder buffer (normalmente chamado de "graduation")

Qual a complexidade do hardware com Reorder Buffer (ROB)?



- Como encontrar a última versão do registrador?
  - (como definido no artigo Smith) é necessário uma rede de comparação associativa
  - Uso de register result status buffer para descobrir qual reorder buffer recebeu o valor
- É necessário que o ROB tenha várias portas (como o banco de registradores)

Tomasulo: Resumo

- Reservations stations: *register renaming* implícito aumentando o conjunto de regs + buffering source operands
  - Evita que os registradores sejam o gargalo
  - Evita WAR e WAW hazards
  - Implementa loop unrolling em HW
- Não se limita a blocos básicos

- Hoje, ajuda nos cache misses
  - Não stall para L1 Data cache miss

- Contribuições
  - Dynamic scheduling
  - Register renaming
  - Load/store disambiguation
- "Descendentes" do 360/91: Pentium III; PowerPC 604; MIPS R10000; HP-PA 8000; Alpha 21264

Algoritmo de Tomasulo e Branch Prediction

- 360/91 prevê branches mas não especula: o pipeline é parado até que o branch seja resolvido
  - Não há especulação; somente instruções podem ser completadas

- Especulação com Reorder Buffer permite que a execução ultapasse o branch, e que haja descarte se o branch falha
  - Apenas é necessário manter as instruções no buffer até que haja o commit do branch

Branch Prediction com N instruções Issue por ciclo de clock

1. Branches irão chegar *n* vezes mais rápidos em um processador *n*-issue

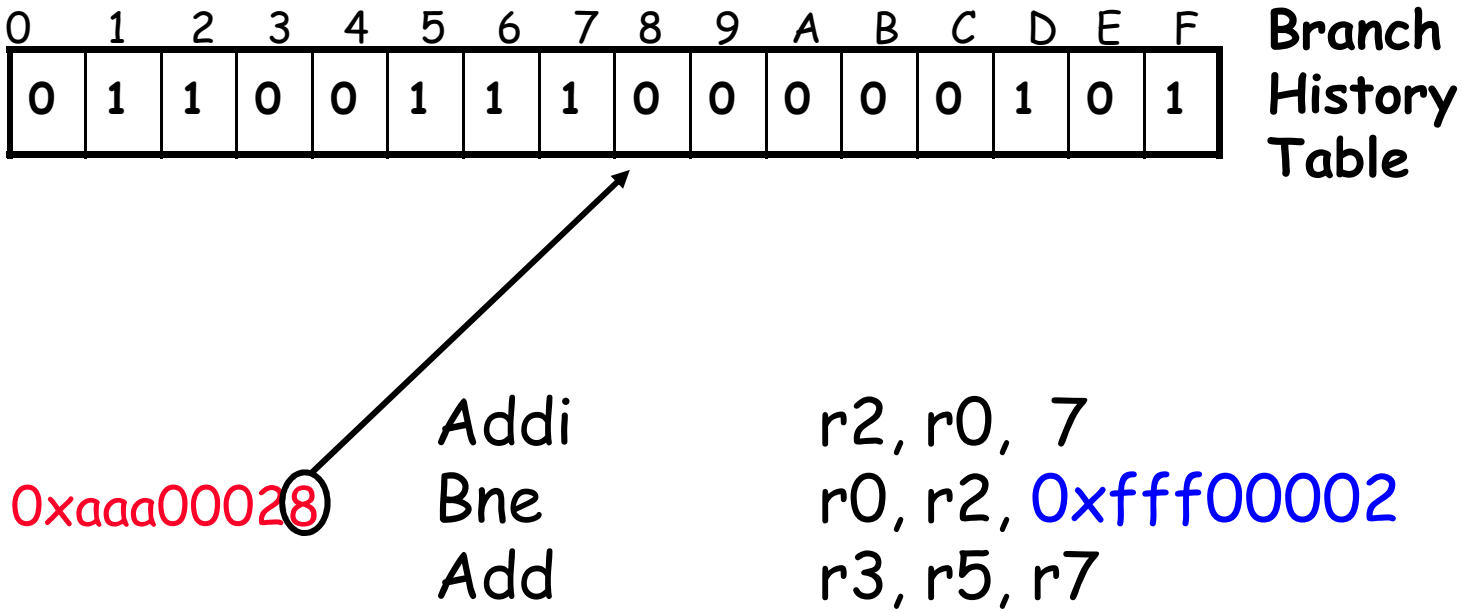
2. Lei de Amdahl => o impacto relativo do controle de stalls será maior que a potencial diminuição do CPI em um processador *n*-issue

7 Esquemas de Branch Prediction

1. 1-bit Branch-Prediction Buffer
2. 2-bit Branch-Prediction Buffer
3. Correlating Branch Prediction Buffer
4. Tournament Branch Predictor
5. Branch Target Buffer
6. Return Address Predictors

Dynamic Branch Prediction

- Desempenho = *f*(precisão, custo do misprediction)
- Branch History Table: Bits menos significativos do PC usados como índice de uma tabela de valores de 1 bit
  - Informa se o branch foi tomado ou não na última vez
  - Não há comparação do endereço (menos HW, mas pode não ser o branch correto)



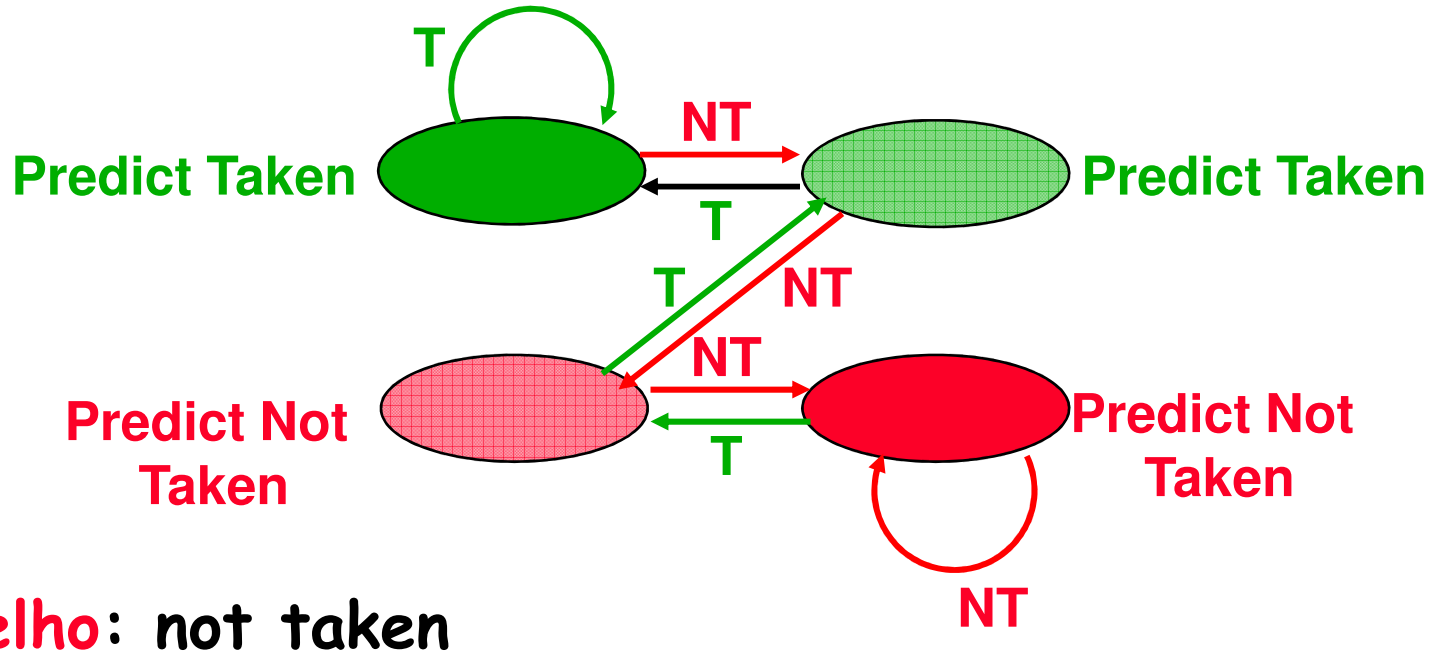
Dynamic Branch Prediction

- Quando descobre que errou, atualiza a entrada correta, elimina as instruções erradas do pipeline e recomeça o fetch de 0xfff00002

- Problema: em um loop, 1-bit BHT irá causar 2 mispredictions (em média nos loops – na entrada e na saída):
  - No fim do loop quando ele termina
  - Na entrada do loop quando ele preve *exit* no lugar de looping
  - Em um loop com 10 iterações
    - » somente 80% de precisão
    - » mesmo que os Taken sejam 90% do tempo

Dynamic Branch Prediction
(Jim Smith, 1981)

- Solução: esquema com 2-bit onde só há troca na previsão se houver duas misprediction:



- Vermelho: not taken
- verde: taken
- *Adicionado uma Histerese (inércia)* para tomar a decisão

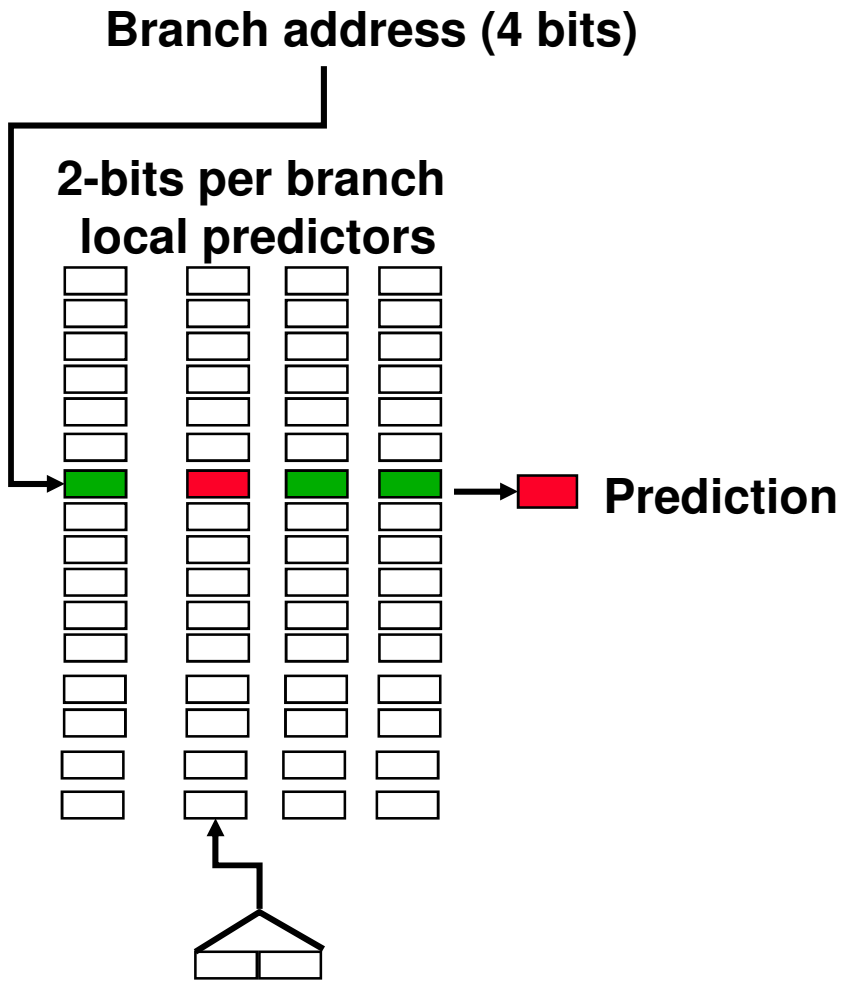
Correlating Branches

Idéia: taken/not taken dos branches recentes são relacionados com o comportamento dos próximos branches (como um histórico do comportamento dos branches)

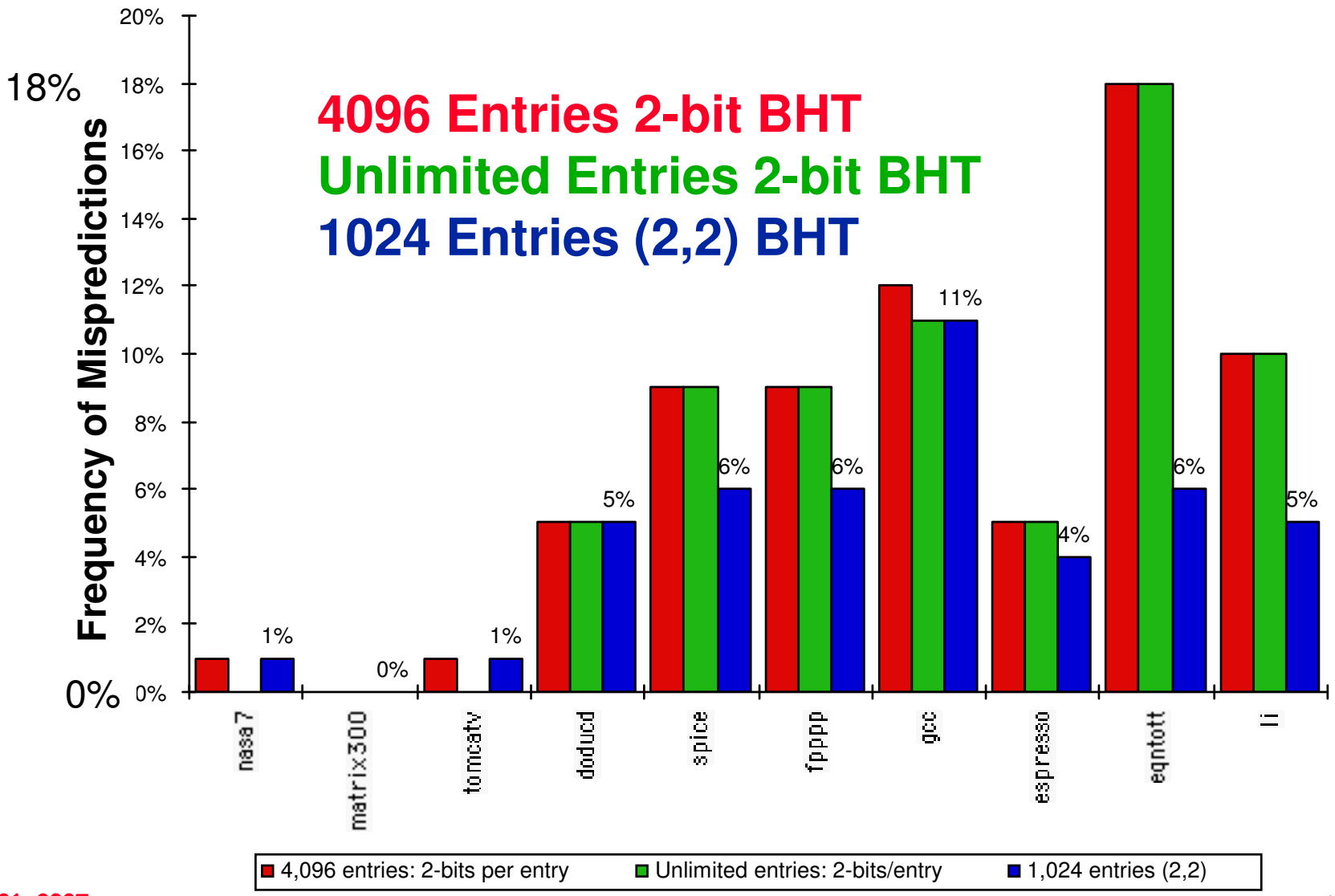
- O comportamento recente dos branches seleciona entre 4 predições para o próximo branch, atualizando a predição

  (2,2) predictor: 2-bit globais e 2-bit locais

```
If (d==0) d=1;
.....
If (d==1) ....;
```



Precisão dos Diferentes Esquemas

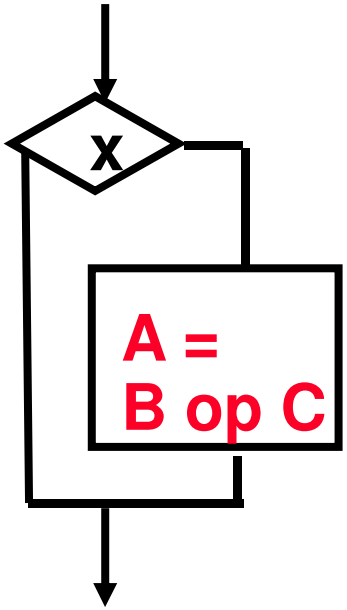


Execução Predicada

- Evita-se branch prediction "colocando-se" os branches em instruções com execução condicional:

  if (x) then A = B op C else NOP

  - Se falso, então nem há store result nem causa exceção
  - ISA expandido do Alpha, MIPS, PowerPC, SPARC possue move condicional; PA-RISC pode anular (annul) qualquer instrução seguinte.
  - IA-64: 64 1-bit condition fields seleciona a execução condicional de qq instrução
  - Esta transformação é chamada de "if-conversion"



- Desvantagens de conditional instructions
  - Gasta um clock se "annulled"
  - Se a condição é avaliada atrasada -> Stall
  - Condições complexas reduzem a eficácia; a condição é conhecida tardiamente no pipeline

Precisão BHT

- Mispredict motivado por:
  - Adivinhação errada para o branch
  - Acesso à história do branch errado (indice)
- Uma tabela com 4096 entradas varia de 1% (nasa7, tomcatv) a 18% (eqntott) de misprediction, com 9% para o spice e 12% para o gcc
- Para o SPEC92,
  4096 entradas é tão bom quanto infinitas entradas

Tournament Predictors

- Motivação: correlating branch predictors falha na previsão de tipos de branches importantes; pode-se melhorar o desempenho provendo informações globais e não somente locais

- Tournament predictors: usa 2 sistemas de predição, 1 baseado em informações globais e 1 baseado em informações locais, combinando-os com o uso de um seletor

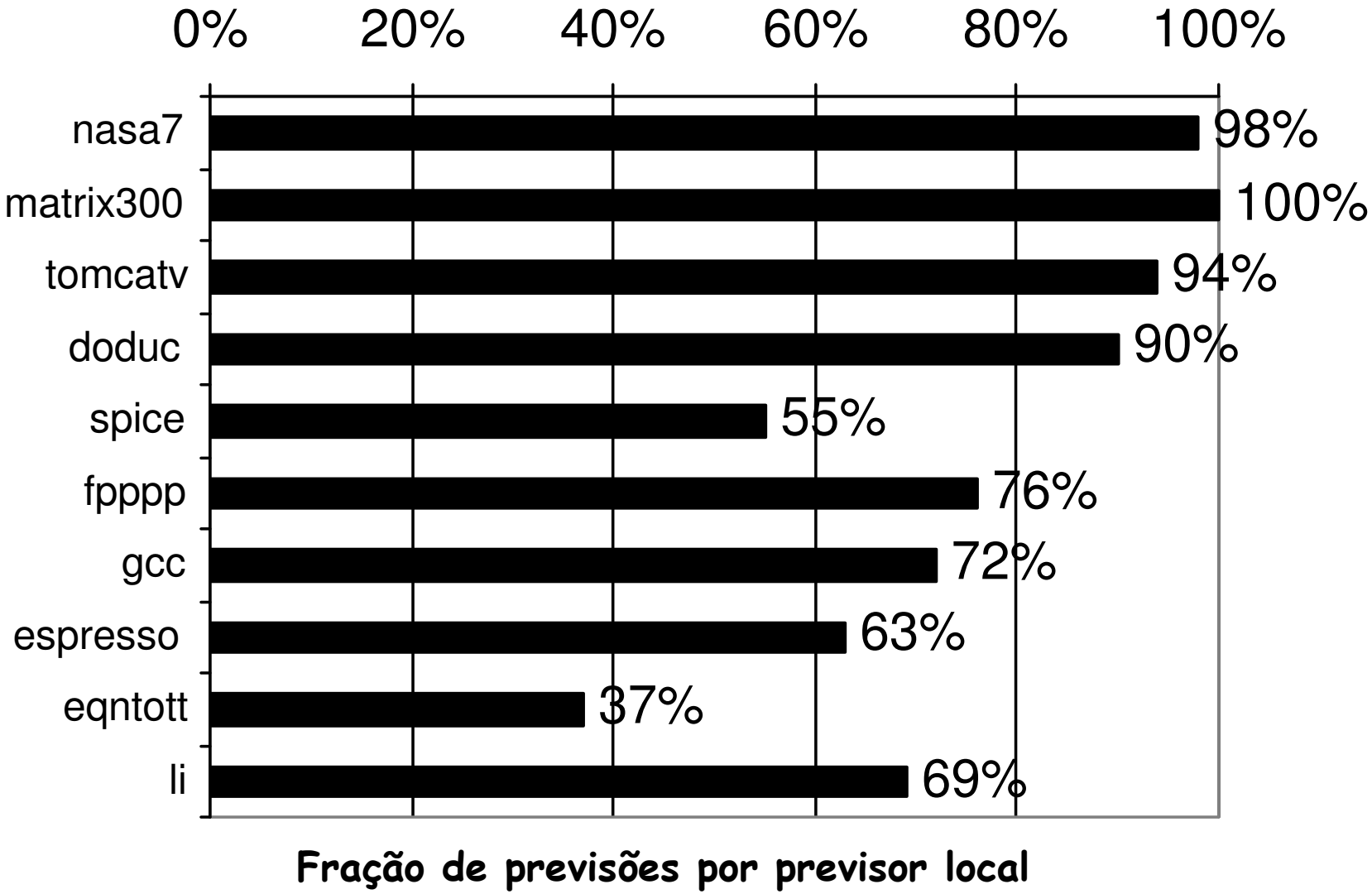
- Ajuda a selecionar a previsão correta para o branch certo

Tournament Predictor no Alpha 21264

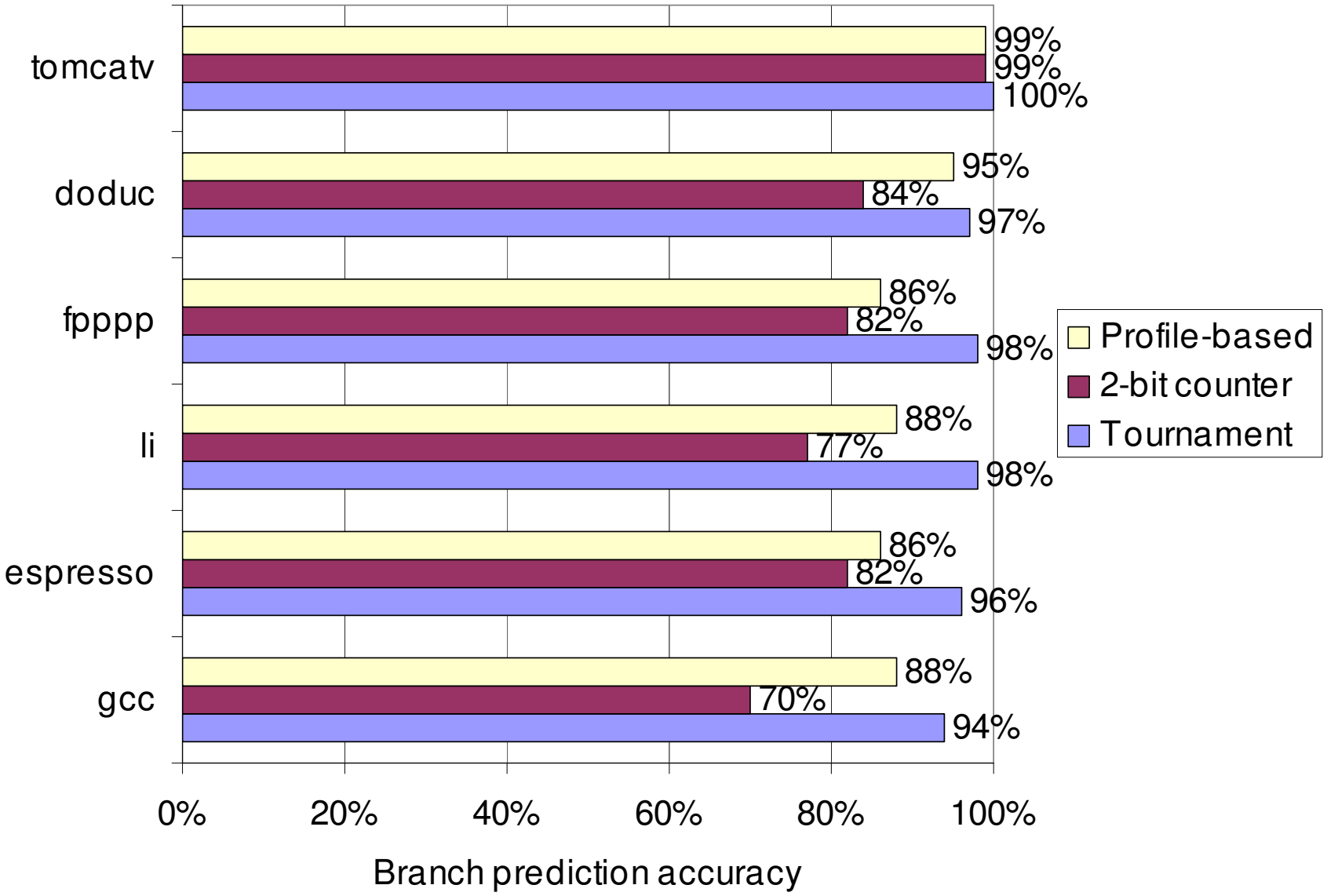
- 4K 2-bit counters para escolha a partir da predição global ou da predição local
- Global predictor: também tem 4K entradas e é indexado por uma história dos últimos 12 branches; cada entrada no global predictor é uma predição standard de 2-bit
  - 12-bit pattern: bit i 0 => i-ésimo branch (anterior) not taken;
    bit i 1 => i-ésimo branch (anterior) taken;
- Local predictor: consiste de uma predição de 2 níveis:
  - Top level: local history table consistindo de 1024 entradas de 10-bit; cada entrada de 10-bit corresponde aos 10 branches mais recentes para a entrada. História de 10-bit permite casar 10 branches.
  - Next level: A entrada selecionada a partir da local history table é usada como indice da tabela de 1K entradas formando um contador de 3-bit, que fornece a predição local
- Tamanho total: 4K*2 + 4K*2 + 1K*10 + 1K*3 = 29K bits!

  (~180,000 transistores)

% de predições a partir da predição local no esquema Tournament Prediction

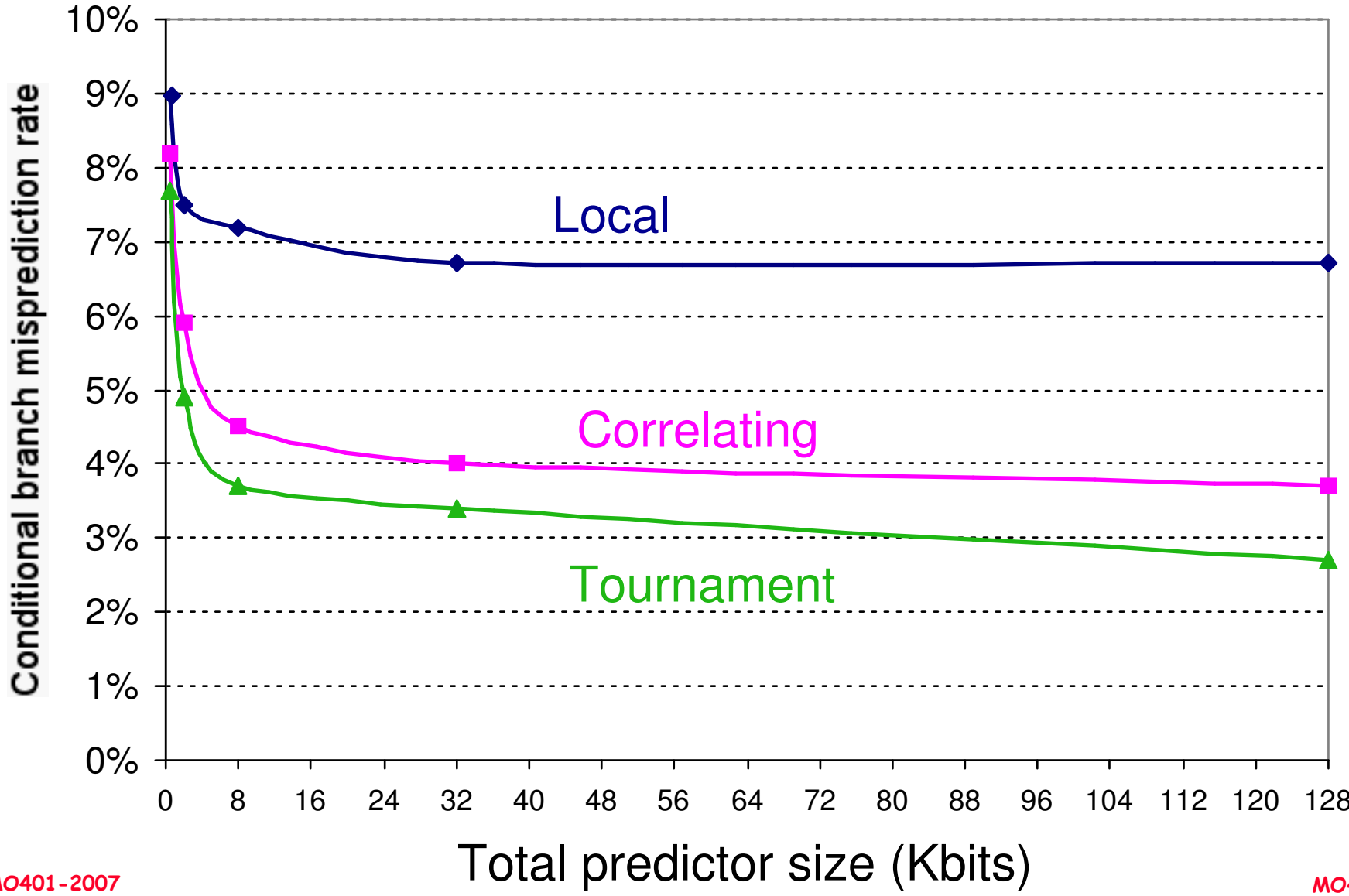


Precisão de Branch Prediction



- **Profile: branch profile feito na última execução**

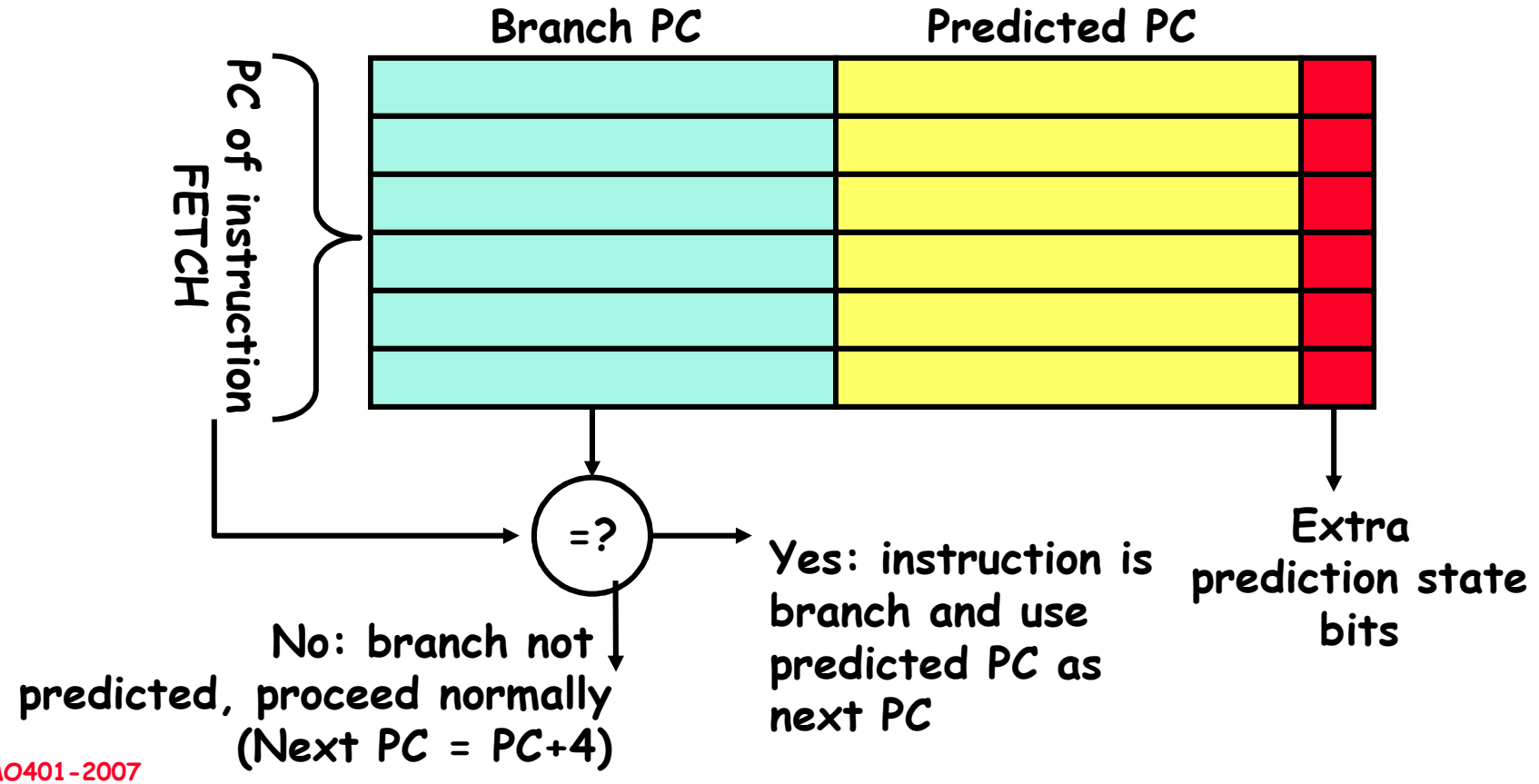
Precisão v. Tamanho (SPEC89)



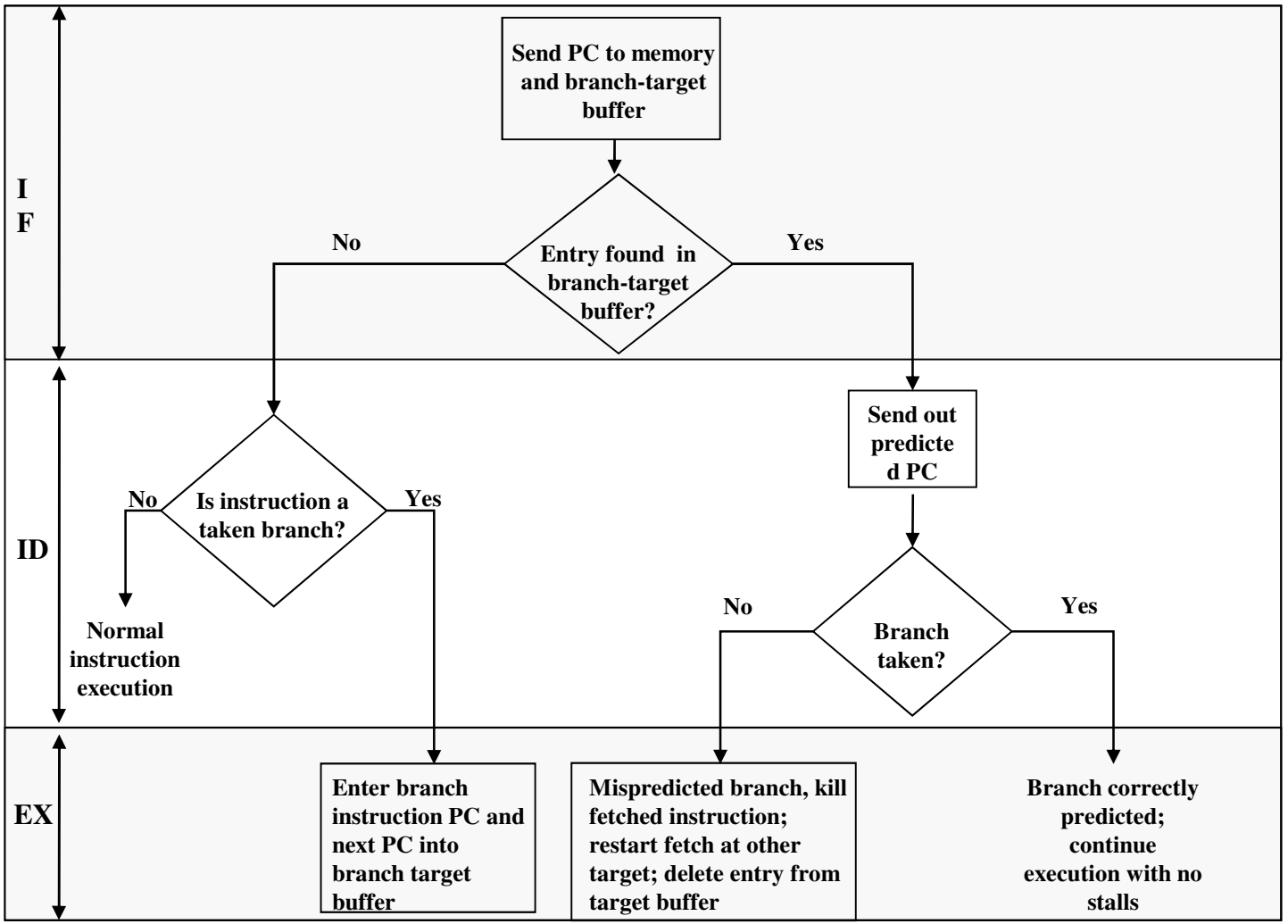
O Endereço é necessário junto a Predição

- Branch Target Buffer (BTB): Endereço do branch é indice para acesso à predição e ao endereço alvo do branch (se taken)

  (Figure 3.19)



Algoritmo para branch-target buffer



Caso Especial: Return Addresses

- Register Indirect Branch: difícil de ser predito o endereço
- SPEC89: 85% dos register indirect branches são retornos de procedimentos
- Já que os procedimentos usam uma pilha para seu controle, salvar os return address em pequenos buffers do tipo pilha : 8 a 16 entradas tem baixa taxa de miss

Quanto maior (sofisticado) melhor?

- 21264 usa tournament predictor (29 Kbits)
- 21164 usa um simple 2-bit predictor com 2K entradas (ou um total de 4 Kbits)
- SPEC95 benchmarks: 21264 é melhor
  - 21264 média de 11.5 mispredictions por 1000 instruções
  - 21164 média de 16.5 mispredictions por 1000 instruções

- Transaction processing (TP) !
  - 21264 média de 17 mispredictions por 1000 instruções
  - 21164 média de 15 mispredictions por 1000 instruções

Resumo: Dynamic Branch Prediction

- Predição é um recurso importante na execução escalar
- Branch History Table: 2 bits para melhorar a precisão em loops
- Correlation: Explora a corelação entre loops executados recentementes e o próximo loop.
  - Mesmo que sejam branches diferentes
  - Ou execuções do memo branch
- Tournament Predictor: mais recursos para soluções que competem entre si, escolha de uma solução
- Branch Target Buffer: inclui o branch address & a predição
- Predicated Execution: pode reduzir o número de branches e o númerro de mispredicted branches
- Return address stack: usado em predição de jumps indiretos