Processadores Superescalares - Avançando na exploração de paralelismo entre instruções

CES-25 – Arquiteturas para Alto Desmpenho
Prof. Paulo André Castro
pauloac@ita.br
Sala 110 – Prédio da Computação
www.comp.ita.br/~pauloac
IEC - ITA

Instruction Level Paralelism (ILP)

- Pipelining cria uma barreira teórica de CPI(Clock por instrução) igual a 1,0.

- Sem pipeline, há muita **ociosidade e ineficiência**:

AR					I1					I2					I3	
EX				I1					I2					I3		
OO			I1					I2					I3			
DI		I1					I2					I3				
RI	I1					I2					I3					
					5					10					15	

- **Com pipeline**, sem instruções de desvio e sem dependências entre operandos, o **ganho** pode ser igual a 5.

AR					I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	
EX				I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11		
OO			I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11			
DI		I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11				
RI	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11					
					5					10					15	

Limites do Pipelining

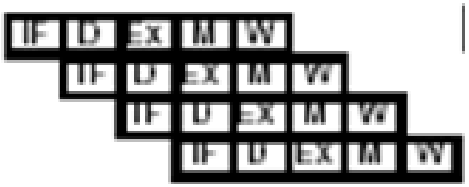
- CPI real = CPI teórico + atrasos estruturais + atrasos por dependência de dados + atrasos por desvios

- CPI teórico em pipeline é aproximadamente 1
  - Logo CPI real >1.

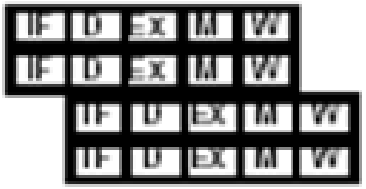
- Como poderia ser alterada a cpu para obtermos CPI < 1 ? Idéias ?

Como avançar na exploração do paralelismo entre instruções

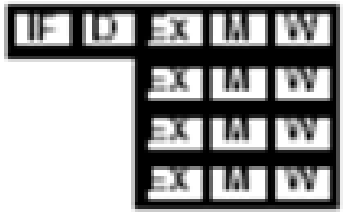
- Pipeline



- Super-pipeline
  - Aprofundar o pipeline, fazendo com que as fases clássicas levem vários ciclos
- Superescalar
  - Linhas de execução em paralelo
  - Novas dependências



- VLIW
  - Cada “pacote”, especifica várias instruções

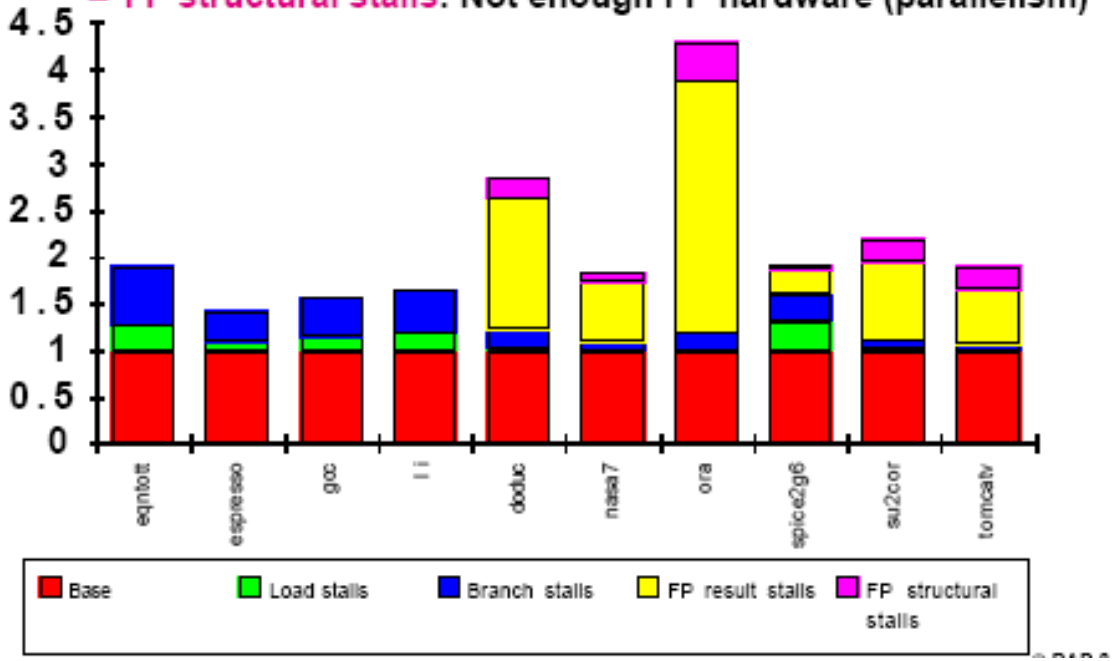


Superpipeline (MIPS R4000)

- 8 stage pipeline:
  - IF–first half of fetching of instruction; PC selection happens here as well as initiation of instruction cache access.
  - IS–second half of access to instruction cache.
  - RF–instruction decode and register fetch, hazard checking and also instruction cache hit detection.
  - EX–execution, which includes effective address calculation, ALU operation, and branch target computation and condition evaluation.
  - DF–data fetch, first half of access to data cache.
  - DS–second half of access to data cache.
  - TC–tag check, determine whether the data cache access hit.
  - WB–write back for loads and register-register operations.
- 8 stages & impact on Load delay? Branch delay? Why?

Avaliação (MIPS R4000)

- Not ideal CPI of 1:
  - Load stalls (1 or 2 clock cycles)
  - Branch stalls (2 cycles + unfilled slots)
  - FP result stalls: RAW data hazard (latency)
  - FP structural stalls: Not enough FP hardware (parallelism)



Em busca de CPI menor que 1

- Superescalar:
  - Instruções ordenadas por compilador ou por hardware (algoritmo de Tomasulo)
    - PowerPC, Sun SPARC, Cell Be (Playstation...), x86, etc.
- Very Long Instruction Word (VLIW):
  - Instruções organizadas em pacotes pelo compilador.
  - Bastante utilizada em sistemas embutidos
  - Chamado de EPIC pela Intel. Ex.: Itanium

Avançando em ILP

- Mesmo em processadores RISC necessariamente há instruções com tempos de execução muito diferentes. Exemplos:
  - soma de inteiros (add) e
  - divisão de ponto flutuante (DIV.D)
- Qual o problema?

- Qual a Solução?

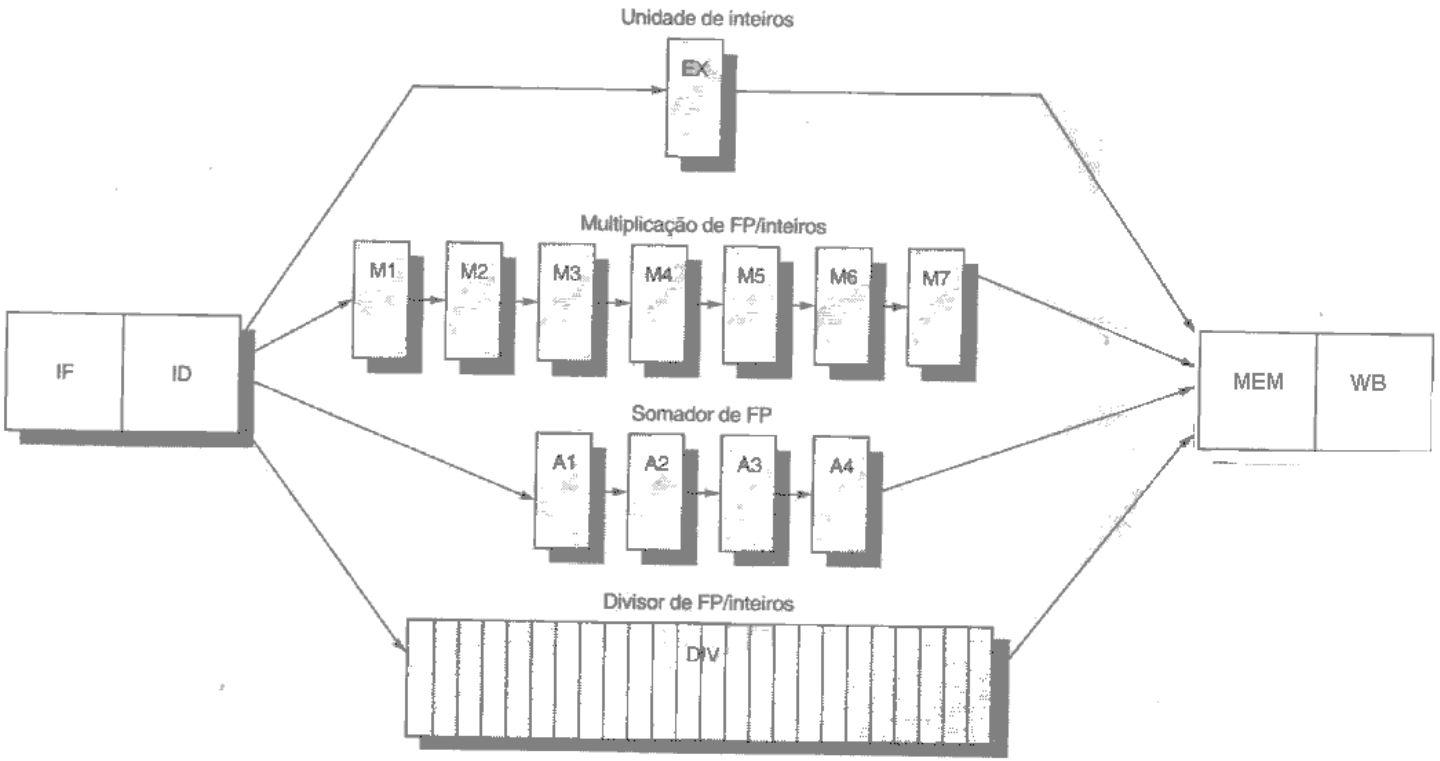
Avançando em ILP

- Permitir mais de uma instrução entrar na fase de execução
  - Emissão em ordem
  - Tempos diferentes de execução
  - Logo, possível conclusão fora de ordem

Execução Fora de Ordem

- Exemplo:
  - DIV.D F0,F2,F4
  - ADD.D F10,F0,F8
  - SUB.D F12,F8,F14
  - Problema: ?
    - SUB é atrasada
- Solução: Executar SUB.D antes de ADD.D

Dependências WAW e WAR e Conclusão de Execução Fora de Ordem



Problemas no Escalonamento Dinâmico

- Exemplo 2:
  - I1: DIV.D F0,F2,F4
  - I2: ADD.D F6,F0,F8
  - I3: SUB.D F8,F10,F14
  - I4: MUL.D F6,F10,F8
- Dependências?
- Executar na ordem: I1,I3,I2,I4 diminui atrasos. Problemas?
- Como resolver?

Solução para Dependências Falsas

- Requisito: Permitir execução fora de ordem, sem mudar o resultado do programa

- **Renomear registradores** para evitar falsas dependências (WAW e WAR)
- Exemplo (WAW):
  - MUL.D R1,R2,R3
  - ADD.D R2, R1,R3
  - SUB.D R1,R4,R5
- Solução:
  - MUL.D R1,R2,R3
  - ADD.D R2, R1,R3
  - SUB.D R5,R4,R5 (Substituir R1 por R5, isto é, outro registrador não utilizado (não lido) até seu próximo ponto de gravação)

Requisitos para Solução

- Exemplo (WAR):
  - MUL.D R1,R2,R3
  - ADD.D R6,R1,R2
  - SUB.D R3,R4,R5
  - ....
  - ADD.D R9,R4,R3
- Solução:
  - MUL.D R1,R2,R3
  - ADD.D R6,R1,R2
  - SUB.D R8,R4,R5 (Substituir R3 por R8, isto é outro registrador não utilizado (não lido) até seu próximo ponto de gravação)
  - ....
  - ADD.D R9,R4,R8

Requisitos para Solução

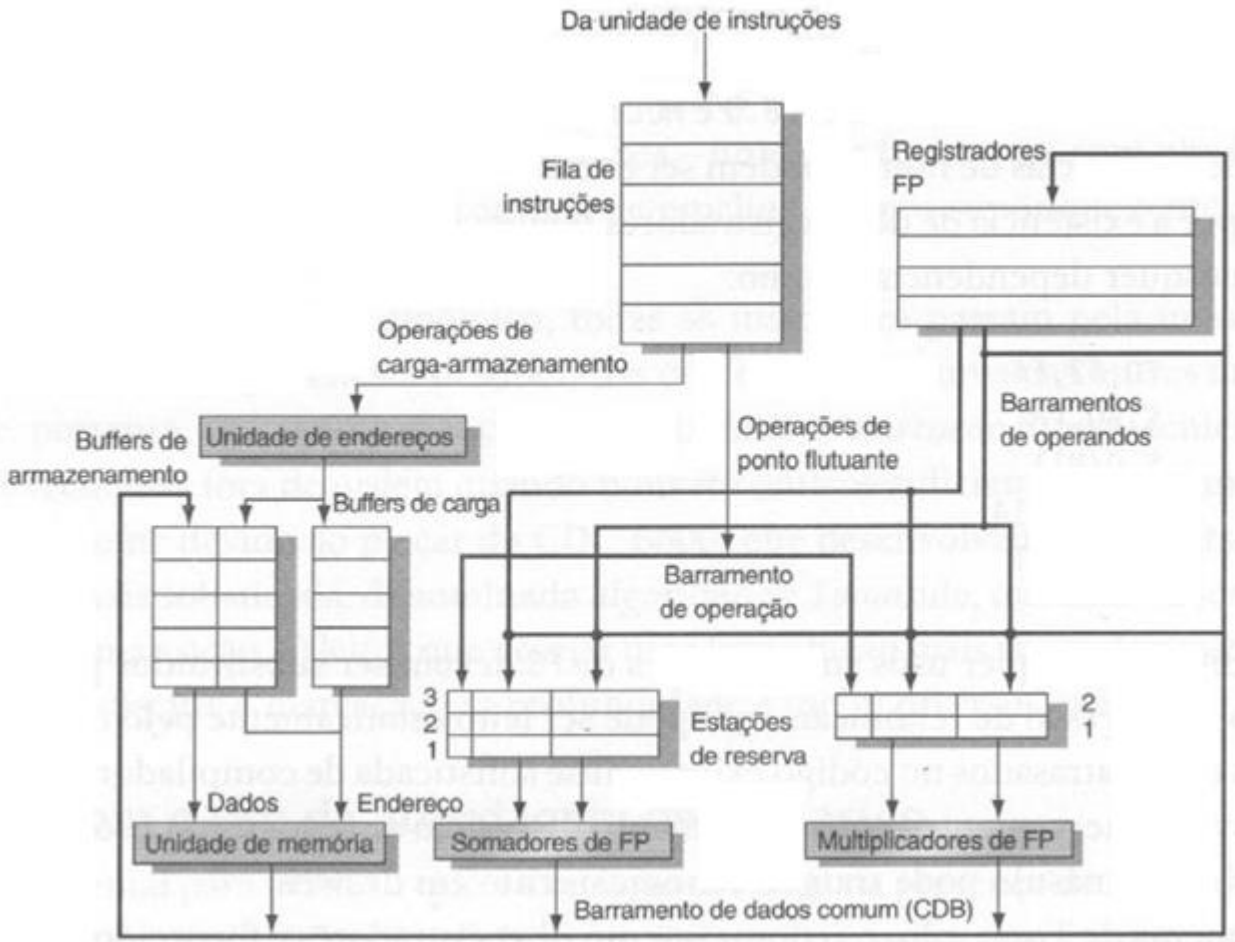
- Identificar instruções sem dependências e permitir que ultrapassem instruções com dependências
- Identificar e Bloquear instruções com dependências de dados ou dependências estruturais
- Manter o pipeline o mais eficiente possível

Métodos de Solução

- Software
  - Com grandes conjuntos de registradores, o compilador pode eliminar perigos WAR e WAW através de renomeação
  - Eventualmente pode usar “moves” entre registradores
  - Erros (ou omissão) no compilador podem levar a baixa eficiência do processador!!!

- Hardware
  - Não tem como observar instruções à frente, mas consegue eliminar dependência em relação a registradores
  - Scoreboard
  - Algoritmo de Tomasulo

Tomasulo


Da unidade de instruções
Fila de instruções
Registradores FP
Operações de carga-armazenamento
Buffers de armazenamento
Unidade de endereços
Buffers de carga
Operações de ponto flutuante
Barramentos de operandos
Barramento de operação
3
2
1
Estações de reserva
2
1
Dados
Endereço
Unidade de memória
Somadores de FP
Multiplicadores de FP
Barramento de dados comum (CDB)

Informações de Controle

- Estações de Reserva
  - Op: código da operação
  - Qj,Qk : Estações de reserva que produzirão o operando
  - Vj,Vk : O valor dos operandos de origem, apenas um valor entre V e Q é valido. Em instruções de carga, guarda o endereço
  - A : Usado para guardar informações sobre cálculo de endereços
  - Busy: Indica se a estação de reserva está sendo usada ou não
- Registradores
  - Qi: O número da estação de reserva que irá gerar o resultado, se zero o valor do registrador é o valor correto

Exemplo de aplicação com Tomasulo

- Instruções
  - L.D F6,34(R2)
  - L.D F2,45(R3)
  - MUL.D F0,F2,F4
  - SUB.D F8,F2,F6
  - DIV.D F10,F0,F6
  - ADD.D F6,F8,F2

Exemplo Tomasulo – Instante 1

	Status de instrução		
Instrução	Emitir	Executar	Gravar resultado
L.D F6,34(R2)	√		
L.D F2,45(R3)	√		
MUL.D F0,F2,F4			
SUB.D F8,F2,F6			
DIV.D F10,F0,F6			
ADD.D F6,F8,F2			

Estações de reserva

Nome	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	sim	Load					34 + Regs[R2]
Load2	sim	Load					45 + Regs[R3]
Add1	não						
Add2	não						
Add3	não						
Mult1	não						
Mult2	não						

Status de registrador

Campo	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi		Load2		Load2					

Exemplo Tomasulo - Instante 2

	Status de instrução		
Instrução	Emitir	Executar	Gravar resultado
L.D F6,34(R2)	√	√	√
L.D F2,45(R3)	√	√	
MUL.D F0,F2,F4	√		
SUB.D F8,F2,F6	√		
DIV.D F10,F0,F6	√		
ADD.D F6,F8,F2	√		

	Estações de reserva						
Nome	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	não						
Load2	sim	Load					45 + Regs[R3]
Add1	sim	SUB		Mem[34 + Regs[R2]]	Load2		
Add2	sim	ADD			Add1	Load2	
Add3	não						
Mult1	sim	MUL		Regs[F4]	Load2		
Mult2	sim	DIV		Mem[34 + Regs[R2]]	Mult1		

	Status de registrador								
Campo	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Mult1	Load2		Add2	Add1	Mult2			

Estados das Instruções

Estado de Instrução		Esperar até	Ação ou contabilidade
Emitir	Operação de FP	Estação r vazia	`if (RegisterStat[rs].Qi≠0) {RS[r].Qj ← RegisterStat[rs].Qi} else {RS[r].Vj ← Regs[rs]; RS[r].Qj ← 0}; if (RegisterStat[rt].Qi≠0) {RS[r].Qk ← RegisterStat[rt].Qi else {RS[r].Vk ← Regs[rt]; RS[r].Qk ← 0}; RS[r].Busy ← yes; RegisterStat[rd].Qi=r;`
	Load ou Store	Buffer r vazio	`if (RegisterStat[rs].Qi≠0) {RS[r].Qj ← RegisterStat[rs].Qi} else {RS[r].Vj ← Regs[rs]; RS[r].Qj ← 0}; RS[r].A ← imm; RS[r].Busy ← yes;`
	Load somente		`RegisterStat[rt].Qi=r;`
	Store somente		`if (RegisterStat[rt].Qi≠0) {RS[r].Qk ← RegisterStat[rs].Qi} else {RS[r].Vk ← Regs[rt]; RS[r].Qk ← 0};`
Executar	Operação de FP	(RS[r].Qj = 0) e (RS[r].Qk = 0)	Calcular resultado: operandos estão em Vj e Vk
	Load-Store etapa 1	RS[r].Qj = 0 & r é o início da fila de carga-armazenamento	`RS[r].A ← RS[r].Vj + RS[r].A;`
	Load etapa 2	Load etapa 1 concluída	Ler de Mem[RS[r].A]
Gravar Resultado	Operação de FP ou Load	Execução concluída em r e CDB disponível	`∀x(if (RegisterStat[x].Qi=r) {Regs[x] ← result; RegisterStat[x].Qi ← 0}); ∀x(if (RS[x].Qj=r) {RS[x].Vj ← result;RS[x].Qj ← 0}); ∀x(if (RS[x].Qk=r) {RS[x].Vk ← result;RS[x].Qk ← 0}); RS[r].Busy ← no;`
	Store	Execução concluída em r e RS[r].Qk = 0	`Mem[RS[r].A] ← RS[r].Vk; RS[r].Busy ← no;`

Instante 2 - Tomasulo

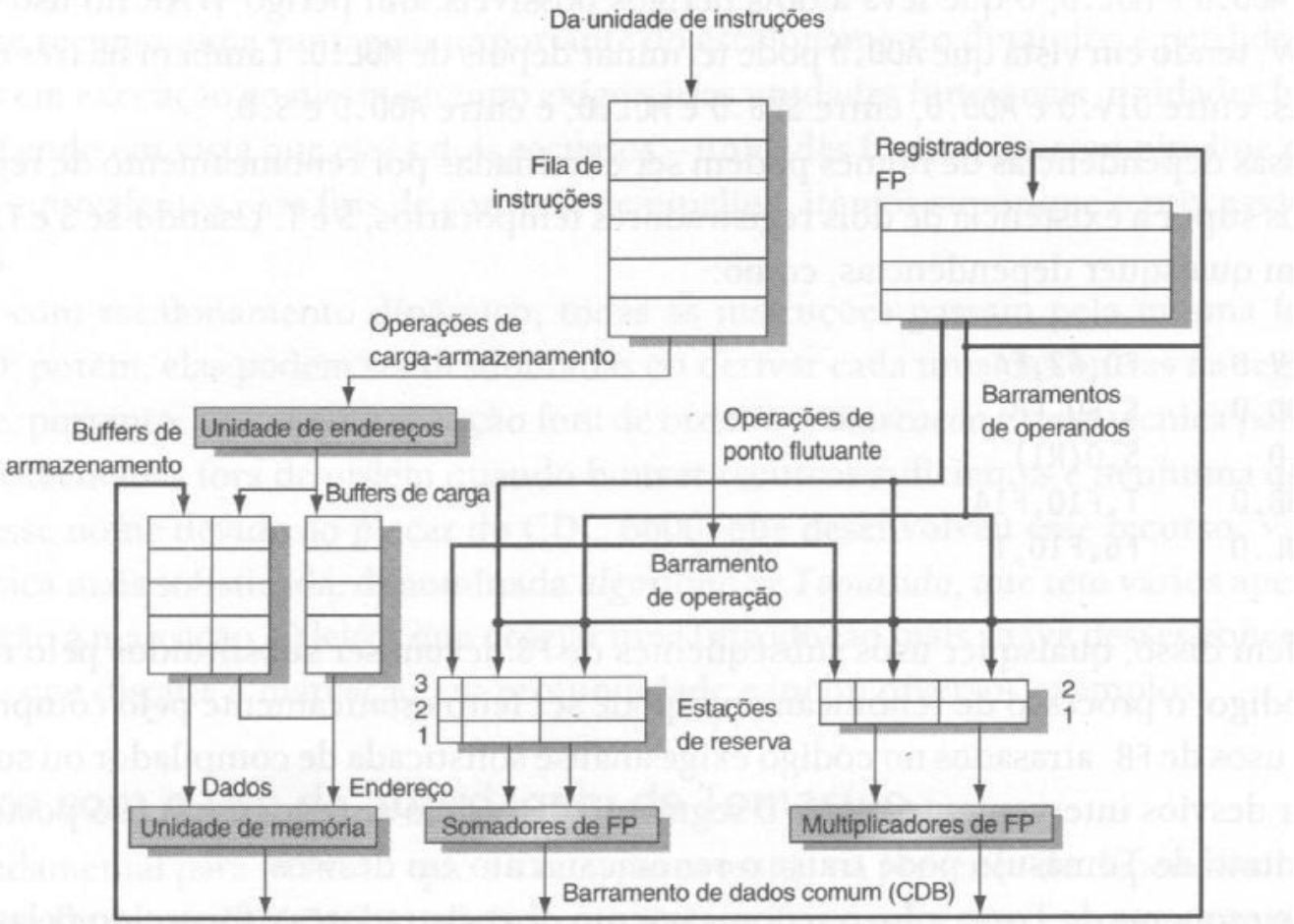
	Status de instrução		
Instrução	Emitir	Executar	Gravar resultado
L.D F6,34(R2)	√	√	√
L.D F2,45(R3)	√	√	√
MUL.D F0,F2,F4	√	√	
SUB.D F8,F2,F6	√	√	√
DIV.D F10,F0,F6	√		
ADD.D F6,F8,F2	√	√	√

	Estações de reserva						
Nome	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	não						
Load2	não						
Add1	não						
Add2	não						
Add3	não						
Mult1	sim	MUL	Mem[45 + Regs[R3]]	Regs[F4]			
Mult2	sim	DIV		Mem[34 + Regs[R2]]	Mult1		

	Status de registradores								
Campo	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Mult1					Mult2			

Fonte: Hennessy & Patterson, pg. 140. Erro no Livro: F0 (e não F2) será gravada por Mult1

Tomasulo e Desvios



Linearização de Loops no Algoritmo de Tomasulo

- Considerando o código abaixo para multiplicar elementos de um vetor por um escalar em F2, como ele seria executado considerando desvios seguidos
- Loop:
  - L.D F0,0(R1)
  - MUL.D F4,F0,F2
  - S.D F4, 0 (R1)
  - ADDI R1,R1,-8 ; R1= R1 -8
  - BNE R1,R2,Loop ; desvia se R1≠ R2

Linearização de Loops

	Status de instrução			
Instrução	Da iteração	Emitir	Executar	Gravar resultado
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	√		

	Estações de reserva						
Nome	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	sim	Load					Regs[R1] + 0
Load2	sim	Load					Regs[R1] – 8
Add1	não						
Add2	não						
Add3	não						
Mult1	sim	MUL		Regs[F2]	Load1		
Mult2	sim	MUL		Regs[F2]	Load2		
Store1	sim	Store	Regs[R1]			Mult1	
Store2	sim	Store	Regs[R1] – 8		Mult2		

	Status de registrador								
Campo	F0	F2	F4	F6	F8	F10	F12	...	F30
Qi	Load2		Mult2						

Sumário Tomasulo

- Permite conclusão fora de ordem, mas com emissão em ordem
  - Permite reduzir atrasos causados por diferenças de tempo de execução entre instruções
- Elimina perigos WAR e WAW através da Renomeação de registradores (estações de reserva)
- Bloqueia instruções devido a perigos RAW
- Permite a linearização de Loops, **mesmo sem execução especulativa!!!**