



UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA - DEPARTAMENTO DE INFORMÁTICA
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Prof. Anderson Faustino da Silva

6895 - Arquitetura e Organização de Computadores II
Segunda Prova

Ricardo Henrique Brunetto

RA: 94182

Maringá
Agosto - 2017

Introdução

O presente documento contém as questões respondidas da segunda prova aplicada pelo Prof. Anderson Faustino da Silva na disciplina de Arquitetura e Organização de Computadores II para a turma de Bacharelado em Ciência da Computação de 2015. Tal documento fora desenvolvido unicamente e explicitamente para o propósito avaliativo.

O conteúdo aqui citado é advindo, além das anotações em aula e materiais disponibilizados pelo professor, da gama de referências bibliográficas por ele recomendadas e encontradas pelo autor. Recomenda-se ?, ? e ? em caso de dúvidas quanto à nomenclatura utilizada.

Questão 01

Para a resolução da questão foi desenvolvido um código em assembly para MIPS32 que realiza a cópia de um subvetor em outro vetor. São argumentados à função `copy` os seguintes dados:

- **\$a0** - Endereço do primeiro vetor
- **\$a1** - Endereço do segundo vetor
- **\$a2** - Índice inicial
- **\$a3** - Índice final

O programa, então, copiará os elementos entre, e inclusive, \$a2 e \$a3 do vetor cujo endereço inicial está em \$a0 para o vetor cujo endereço inicial está em \$a1. O detalhamento do código será omitido aqui, uma vez que o mesmo o faz com comentários.

```

1 .data
2 exit_val: .int 10
3 .text
4 # Copia um subvetor de v1 para v2
5 # Recebe como parametros
6 # a0 — endereco do primeiro vetor (primeiro elemento do vetor)
7 # a1 — endereco do segundo vetor (primeiro elemento do vetor)
8 # a2 — indice inicial
9 # a3 — indice final
10 # Copia o subvetor a0[a2 ... a3] para a1[0 ... a3-a2]
11 copiar_subvetor:
12     sub $v0, $a3, $a2          # $v0 sera o contador
13     li $at, 4                  # carrega 4 em $at
14     mul $a2, $a2, $at          # faz a2 * 4
15     add $a0, $a0, $a2          # coloca $a0 em $a0[$a2]
16     looping:
17         lw $at, 0($a0)          # carrega o elemento do subvetor para $at
18         sw $at, 0($a1)          # armazena o elemento no vetor
19         addiu $a0, $a0, 4        # avanca 4 bytes em a0
20         addiu $a1, $a1, 4        # avanca 4 bytes em a1
21         addi $v0, $v0, -1        # $v0—
22         bgez $v0, looping        # if ($v0 > 0) looping
23 fim:
24     lw $v0, exit_val
25     syscall

```

../copiar_subvetor.s

Questão 02

Para tal questão, serão propostas convenções a respeito de detalhes da execução do algoritmo por um simulador hipotético que aplica o algoritmo de Tomasulo com especulação. Convenciona-se o seguinte:

- O algoritmo de predição de desvios acerta todas as previsões de desvios.
- São emitidas, por ciclo, quantas instruções as Estações de Reserva comportarem, respeitando às seguintes restrições:

Política de emissão *In-Order Issue and Out-of-Order Completion*, o que implica que a emissão respeitará a fila de instruções;

Conflitos de Recursos, onde a emissão cessará;

A execução (despacho para a Unidade Funcional) se iniciará apenas quando não houver mais dependências verdadeiras.

- A quantidade de ciclos de cada instrução do código obedece ao especificado pela arquitetura MIPS32, a saber:

add, sub, addi, addiu e li custam 2 ciclos de clock;

lw e sw custam 4 ciclos de clock;

bgez custa 2 ciclos de clock;

mul custa 5 ciclos de clock.

- A chamada de sistema (syscall) para pôr fim à execução aguarda até que todas as outras Estações de Reserva tenham finalizado o processamento.
- Quanto às Estações de Reserva (cada qual com uma única Unidade Funcional acoplada):
 - 3 Estações de Reserva para operações lógicas e aritméticas;
 - 2 Estações de Reserva para operações de multiplicação e divisão;
 - 5 Estações de Reserva para carregamento de dados (*load*);
 - 5 Estações de Reserva para armazenamento de dados (*store*).

A cada ciclo passado, informar-se-á quais instruções foram despachadas (inciaram a execução, de fato) e quais estão executando naquele ciclo, através do índice (nome) da Estação de Reserva que a comporta. Além disso, serão omitidos registradores não-utilizados do Banco de Registradores. A nomenclatura dos campos das Estações de Reserva e do Banco de Registradores obedece à estabelecida por ?.

Supõe-se a execução do código da Questão 01 com os seguintes parâmetros:

- **\$a0** = &src (Endereço do primeiro vetor)
- **\$a1** = &dst (Endereço do segundo vetor)

- $\$a2 = 1$ (Índice inicial)
- $\$a3 = 3$ (Índice final)

onde $\text{src} = \{45, 23, 12, 29, 10\}$ e $\text{dst} = \{0, 0, 0, 14, 23\}$.

Dessa forma, a execução deve resultar em $\text{dst} = \{23, 12, 29, 14, 23\}$. A execução passo-a-passo encontra-se nos Apêndices deste documento.

Questão 03

Para tal, utilizar-se-á da prova por meio da demonstração da quantidade de ciclos de clock.

Percebe-se que executar o algoritmo da Questão 01 em um pipeline escalar sem a especulação não garante a prova formal de tal afirmação, uma vez que utilizar-se-ia de um exemplo em particular na tentativa de demonstrar uma propriedade global, de modo que não se pode garantir a generalização, ainda que intuitivamente isso seja evidenciado.

Supõe-se k o número de ciclos necessários para executar um determinado algoritmo com as especificações de simulação da Questão 02 (Tomasulo com especulação). Dessa forma, para o caso específico tratado na Questão 02 (parâmetros dados) e envolvendo o algoritmo da Questão 01, por exemplo, tem-se $k = 23$.

Seja c a quantidade máxima de ciclos necessária para se resolver a condição de qualquer desvio implementado pela arquitetura em questão (sem especificamente tratar da MIPS32, onde $c = 2$). É importante salientar que, em todos os casos, as comparações são dadas entre os desempenhos supondo entradas idênticas para ambas as configurações de simulação (com especulação e sem especulação). Note que, c pode variar muito além, na prática, por limitações da *ILP*, como **dependências verdadeiras**. No entanto, ambas as configurações consideradas estão sujeitas à tal variação e, portanto, ela **não** será considerada na análise a seguir.

Nota-se que, ao seguir a política de emissão *In-Order Issue and Out-of-Order Completion* sem a implementação de algoritmos de predição de desvios ou de execução especulativa, deve-se **cessar a emissão** a cada vez que um desvio condicional for emitido. Isso porque o Processador não tem mecanismos para controlar as emissões enquanto não houver o cômputo da condição.

Seja x o número de desvios condicionais espalhados pelo código que devem ser executa-

dos pelo Processador. Dessa forma, ter-se-á, ao menos $x \times c$ ciclos de clock em que as emissões serão suspendidas. Logicamente, estes $x \times c$ ciclos de clock estão contidos nos k ciclos totais de uma configuração com especulação, visto que, de qualquer forma, as condições dos desvios devem ser computadas. No entanto, durante estes ciclos em que as condições estão sendo computadas **podem** haver outras instruções na Fila de Instruções com condições necessárias atendidas para serem emitidas. Assim, a eficiência da especulação se dá pelas instruções que se emite e computa enquanto a condição do desvio está sendo calculada (quando possível, visto que há casos de limitação de recursos em que não se pode realizar emissões ou execuções).

No melhor caso, o algoritmo de predição terá acertado e as instruções que foram processadas enquanto a condição não havia sido completa deveriam, de fato, ter sido executadas e não perdeu-se c ciclos aguardando para enviar as instruções à execução (evitou-se um *stall* de c ciclos no pipeline). Por outro lado, no pior caso, o algoritmo de predição de desvios terá feito uma suposição errada quanto à tomada do salto e o processamento das outras instruções, realizado durante os c ciclos que a condição levou, terá sido "despedido", visto que elas não serão efetivadas. No entanto, caso não houvesse especulação, estes c ciclos teriam, de fato, servido **apenas** para o cálculo da condição (desconsiderando o que já estava em execução na emissão do desvio).

Assim, no pior dos casos da execução especulativa, tem-se o mesmo caso da execução sem especulação (com *stall* no pipeline). Isso implica que:

$$k \leq k + (x \times c) \quad \forall x, c \geq 0$$

Para fins de exemplo, assume-se que deseja-se executar a mesma sequência de código da Questão 01 (`copiar_subvetor`) em uma máquina onde não haja implementação de especulação, bem como nenhum algoritmo de predição de desvios. Nesse caso, para a máquina sem predição, como foi suposto, ter-se-ia de cessar a emissão a cada emissão da instrução `bgez` do código `copiar_subvetor`, na arquitetura MIPS32. Como `bgez` é emitido 3 vezes e o processamento de sua condição se dá em 2 ciclos, haveria, ao menos 6 ciclos onde nenhuma instrução é emitida. Isso implica que o total de ciclos para a execução do algoritmo seria **em torno de**¹ 29 ciclos. No

¹ Isso se deve ao fato de que uma emissão tardia poderia evitar dependências verdadeiras com instruções que já estão em execução, ou, ainda causá-las em consequência de não terem ainda saído do pipeline (efeito cascata).

entanto, o mínimo seria de 26 ciclos.

Questão 04

Para tal questão, propõe-se dois códigos a serem executados em paralelo. Tais códigos manipulam as variáveis globais inteiras a, b e c.

```
1 void foo () {  
2     write (b);  
3     read (a);  
4     write (c);  
5     write (a);  
6     read (b);  
7     read (a);  
8 }
```

../paralelo_1.c

```
1 void bar () {  
2     read (a);  
3     write (c);  
4     read (c);  
5     read (b);  
6     write (a);  
7     write (b);  
8 }
```

../paralelo_2.c

As funções `read` e `write` implicam em uma leitura e uma escrita, respectivamente, da variável passada como parâmetro. Tal função exemplifica, de maneira abstrata, uma leitura de um valor da memória e sua correspondente escrita.

Dessa forma, aplicar-se-á os Protocolos de Coerência de Cache de Invalidação de Dados **MSI**, **MESI** e **MOESI**. Para todos os casos, supõe-se que as três variáveis encontram-se em **linhas diferentes** da Cache. No decorrer da execução, supõe-se que cada um dos fluxos é executado por um processador diferente (P_0 executará `paralelo_1` enquanto P_1 executará `paralelo_2`). Serão desprezados efeitos externos de outras questões de projeto no que tange ao Paralelismo à Nível de Fluxo (*Thread Level Parallelism - TLP*), bem como efeitos internos de questões relativas ao Paralelismo à Nível de Instrução (*Instruction Level Parallelism - ILP*), uma vez que há interesse apenas no comportamento dos Controladores de Cache em relação ao protocolo (e às requisições).

Para todos os protocolos, a execução é realizada em **passos** (ou momentos), de modo que não se considera a unidade de tempo em ciclos de clock. Isso serve para simplificar a execução e abstrair detalhes não relevantes à aplicação do protocolo em questão. Cada passo possui um

indicador para explicitar a ordem em que tais requisições ocorreram (ex: 2.1 significa que tal requisição foi a primeira a ocorrer no passo 2), visto que uma requisição interna do Processador pode gerar uma requisição externa para outro Controlador de Cache (que, no caso, seria denotado por 2.2).

Quanto à notação, utiliza-se a seguinte:

$$[\text{Passo.Indicador}]: \text{requisição} \mid \text{Estado_Antigo} \rightarrow \text{Estado_Novo} \mid L_{\text{variavel}}$$

onde L_{variavel} indica a Linha da Cache em que a *variavel* está armazenada.

Protocolo MSI

P_0 (paralelo_1)	P_1 (paralelo_2)
[1.1]: write_miss I \rightarrow M L_b	
	[2.1]: read_miss I \rightarrow S L_a
[3.1]: read_miss I \rightarrow S L_a	[3.2]: REQ_read S \rightarrow S L_a
	[4.1]: write_miss I \rightarrow M L_c
[5.1]: write_miss I \rightarrow M L_c	[5.2]: REQ_write M \rightarrow I L_c
[6.2]: REQ_read M \rightarrow S L_c	[6.1]: read_miss I \rightarrow S L_c
[7.1]: write_hit S \rightarrow M L_a	[7.2]: REQ_write S \rightarrow I L_a
[8.2]: REQ_read M \rightarrow S L_b	[8.1]: read_miss I \rightarrow S L_b
[9.1]: read_hit S \rightarrow S L_b	
[10.2]: REQ_write M \rightarrow I L_a	[10.1]: write_miss I \rightarrow M L_a
[11.1]: read_miss I \rightarrow S L_a	[11.2]: REQ_read M \rightarrow S L_a
[12.2]: REQ_write S \rightarrow I L_b	[12.1]: write_hit S \rightarrow M L_b

Protocolo MESI

P_0 (paralelo_1)	P_1 (paralelo_2)
[1.1]: write_miss $I \rightarrow M$ L_b	
	[2.1]: read_miss $I \rightarrow E$ L_a
[3.1]: read_miss $I \rightarrow S$ L_a	[3.2]: REQ_read $E \rightarrow S$ L_a
	[4.1]: write_miss $I \rightarrow M$ L_c
[5.1]: write_miss $I \rightarrow M$ L_c	[5.2]: REQ_write $M \rightarrow I$ L_c
[6.2]: REQ_read $M \rightarrow S$ L_c	[6.1]: read_miss $I \rightarrow S$ L_c
[7.1]: write_hit $S \rightarrow M$ L_a	[7.2]: REQ_write $S \rightarrow I$ L_a
[8.2]: REQ_read $M \rightarrow S$ L_b	[8.1]: read_miss $I \rightarrow S$ L_b
[9.1]: read_hit $S \rightarrow S$ L_b	
[10.2]: REQ_write $M \rightarrow I$ L_a	[10.1]: write_miss $I \rightarrow M$ L_a
[11.1]: read_miss $I \rightarrow S$ L_a	[11.2]: REQ_read $M \rightarrow S$ L_a
[12.2]: REQ_write $S \rightarrow I$ L_b	[12.1]: write_hit $S \rightarrow M$ L_b

Protocolo MOESI

P_0 (paralelo_1)	P_1 (paralelo_2)
[1.1]: write_miss $I \rightarrow M$ L_b	
	[2.1]: read_miss $I \rightarrow E$ L_a
[3.1]: read_miss $I \rightarrow S$ L_a	[3.2]: REQ_read $E \rightarrow S$ L_a
	[4.1]: write_miss $I \rightarrow M$ L_c
[5.1]: write_miss $I \rightarrow M$ L_c	[5.2]: REQ_write $M \rightarrow I$ L_c
[6.2]: REQ_read $M \rightarrow O$ L_c	[6.1]: read_miss $I \rightarrow S$ L_c
[7.1]: write_hit $S \rightarrow M$ L_a	[7.2]: REQ_write $S \rightarrow I$ L_a
[8.2]: REQ_read $M \rightarrow O$ L_b	[8.1]: read_miss $I \rightarrow S$ L_b
[9.1]: read_hit $O \rightarrow O$ L_b	
[10.2]: REQ_write $M \rightarrow I$ L_a	[10.1]: write_miss $I \rightarrow M$ L_a
[11.1]: read_miss $I \rightarrow S$ L_a	[11.2]: REQ_read $M \rightarrow O$ L_a
[12.2]: REQ_write $O \rightarrow I$ L_b	[12.1]: write_hit $S \rightarrow M$ L_b

Questão 05

Para tal questão toma-se como base a ideia de que a Barreira deve impor dependência (sequência ordenada obrigatória) entre trechos de código (fases do programa) paralelos.

A implementação de uma Barreira ocorre via um **Contador**. No entanto, incrementa-se uma variável compartilhada e, portanto, tal incremento deve ser **atômico**. É importante considerar que, em MIPS32, os parâmetros são enviados pelos registradores (inciando de \$a0). O código desenvolvido é o exposto abaixo:

```
1 .data
2 shared_count: .int 0
3 .text
4
5 inicializar_barreira:
6
7 jr $ra
8
9 executar_barreira:
10     fi shared_count    # fetch_and_increment shared_count (atômico)
11     looping:
12         lw $v0, shared_count
13         bne $v0, $a0, looping # segura enquanto shared_count for diferente do
            numero de threads
14     jr $ra
```

../barreira.s

Referências

Appendices

Execução do Algoritmo - Tomasulo com Especulação

Ciclo 1											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	1	sub	0	3	0	1		a0	add3	&src	addiu addiu addi bgez
add2	1	li					4	a1	0	&dst	
add3	1	add	0	&src	mul1			a2	mul1	1	
mul1	1	mul	0	1	add2			a3	0	3	
mul2	0							v0	add1	0	
load1	1	lw	add3				0	at	load1	0	
load2	0							Despachadas			
load3	0							add1	add2		
load4	0							Executando			
load5	0							add1(1/2)	add2(1/2)		
store1	1	sw	0	&dst	load1		0				
store2	0										
store3	0										
store4	0										
store5	0										
dst = {0, 0, 0, 14, 23}											

Ciclo 2											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	1	sub	0	3	0	1		a0	add3	&src	addiu
add2	1	li					4	a1	0	&dst	addiu
add3	1	add	0	&src	mul1			a2	mul1	1	addi
mul1	1	mul	0	1	add2			a3	0	3	bgez
mul2	0							v0	add1	0	
load1	1	lw	add3				0	at	load1	0	
load2	0										
load3	0							Despachadas			
load4	0										
load5	0										
store1	1	sw	0	&dst	load1		0				
store2	0							Executando			
store3	0							add1(2/2)	add2(2/2)		
store4	0										
store5	0										
dst = {0, 0, 0, 14, 23}											

Ciclo 3											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	1	addiu	add3				4	a0	add1	&src	addi bgez
add2	1	addiu	0	&dst			4	a1	add2	&dst	
add3	1	add	0	&src	mul1			a2	mul1	1	
mul1	1	mul	0	1	0	4		a3	0	3	
mul2	0							v0	0	2	
load1	1	lw	add3				0	at	load1	0	
load2	0							Despachadas			
load3	0							mul1	add2		
load4	0							Executando			
load5	0							mul1(1/5)	add2(1/2)		
store1	1	sw	0	&dst	load1		0				
store2	0										
store3	0										
store4	0										
store5	0										
dst = {0, 0, 0, 14, 23}											

Ciclo 4											Fila de Instruções
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	
add1	1	addiu	add3				4	a0	add1	&src	addi
add2	1	addiu	0	&dst			4	a1	add2	&dst	bgez
add3	1	add	0	&src	mul1			a2	mul1	1	
mul1	1	mul	0	1	0	4		a3	0	3	
mul2	0							v0	0	2	
load1	1	lw	add3				0	at	load1	0	
load2	0							Despachadas			
load3	0										
load4	0										
load5	0										
store1	1	sw	0	&dst	load1		0	Executando			
store2	0							mul1(2/5)	add2(2/2)		
store3	0										

Execução do Algoritmo - Tomasulo com Especulação

store4	0							
store5	0							
dst = {0, 0, 0, 14, 23}								

Ciclo 5													
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções		
add1	1	addiu	add3				4	a0	add1	&src	bgez		
add2	1	addi	0	2			-1	a1	0	&dst+4			
add3	1	add	0	&src	mul1			a2	mul1	1			
mul1	1	mul	0	1	0	4		a3	0	3			
mul2	0							v0	add2	2			
load1	1	lw	add3				0	at	load1	0			
load2	0							Despachadas					
load3	0												
load4	0												
load5	0												
store1	1	sw	0	&dst	load1		0	Executando					
store2	0												
store3	0												
store4	0												
store5	0												
dst = {0, 0, 0, 14, 23}								mul1(3/5) add2(1/2)					

Ciclo 6											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	1	addiu	add3				4	a0	add1	&src	bgez
add2	1	addi	0	2			-1	a1	0	&dst+4	
add3	1	add	0	&src	mul1			a2	mul1	1	
mul1	1	mul	0	1	0	4		a3	0	3	
mul2	0							v0	add2	2	
load1	1	lw	add3				0	at	load1	0	
load2	0							Despachadas			
load3	0										
load4	0										
load5	0										
store1	1	sw	0	&dst	load1		0				
store2	0							Executando			
store3	0							mul1(4/5)	add2(2/2)		
store4	0										
store5	0										
dst = {0, 0, 0, 14, 23}											

Ciclo 7											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	1	addiu	add3				4	a0	add1	&src	addiu addiu addi bgez
add2	1	bgez	0	1				a1	0	&dst+4	
add3	1	add	0	&src	mul1			a2	mul1	1	
mul1	1	mul	0	1	0	4		a3	0	3	
mul2	0							v0	0	1	
load1	1	lw	add3				0	at	load2	0	
load2	1	lw	add1				0	Despachadas			
load3	0										
load4	0										
load5	0										
store1	1	sw	0	&dst	load1		0				
store2	1	sw	0	&dst+4	load2		0	Executando			
store3	0										
store4	0										
store5	0										
dst = {0, 0, 0, 14, 23}											

Ciclo 8														
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções			
add1	1	addiu	add3				4	a0	add1	&src	addiu			
add2	1	bgez	0	1				a1	0	&dst+4	addiu			
add3	1	add	0	&src	0	4		a2	0	4	addi			
mul1	0							a3	0	3	bgez			
mul2	0							v0	0	1				
load1	1	lw	add3				0	at	load2	0				
load2	1	lw	add1				0	Despachadas						
load3	0													
load4	0													
								add3						

Execução do Algoritmo - Tomasulo com Especulação

load5	0									
store1	1	sw	0	&dst	load1		0			
store2	1	sw	0	&dst+4	load2		0			
store3	0									
store4	0									
store5	0									
dst = {0, 0, 0, 14, 23}										

Ciclo 9											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	1	addiu	add3				4	a0	add2	&src	addi bgez
add2	1	addiu	add1				4	a1	0	&dst+4	
add3	1	add	0	&src	0	4		a2	0	4	
mul1	0							a3	0	3	
mul2	0							v0	0	1	
load1	1	lw	add3				0	at	load2	0	
load2	1	lw	add1				0				
load3	0							Despachadas			
load4	0							Executando add3(2/2)			
load5	0										
store1	1	sw	0	&dst	load1		0				
store2	1	sw	0	&dst+4	load2		0				
store3	0										
store4	0										
store5	0										
dst = {0, 0, 0, 14, 23}											

Ciclo 10											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	1	addiu	0	&src+4			4	a0	add2	&src	addi bgez
add2	1	addiu	add1				4	a1	add3	&dst+4	
add3	1	addiu	0	&dst+4			4	a2	0	4	
mul1	0							a3	0	3	
mul2	0							v0	0	1	
load1	1	lw	0	&src+4			0	at	load2	0	
load2	1	lw	add1				0				
load3	0							Despachadas			
load4	0							add1	load1	add3	
load5	0										
store1	1	sw	0	&dst	load1		0				
store2	1	sw	0	&dst+4	load2		0	Executando			
store3	0							add1(1/2)	load1(1/4)	add3(1/2)	
store4	0										
store5	0										
dst = {0, 0, 0, 14, 23}											

Ciclo 11											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	1	addiu	0	&src+4			4	a0	add2	&src	addi bgez
add2	1	addiu	add1				4	a1	add3	&dst+4	
add3	1	addiu	0	&dst+4			4	a2	0	4	
mul1	0							a3	0	3	
mul2	0							v0	0	1	
load1	1	lw	0	&src+4			0	at	load2	0	
load2	1	lw	add1				0				
load3	0							Despachadas			
load4	0										
load5	0										
store1	1	sw	0	&dst	load1		0	Executando			
store2	1	sw	0	&dst+4	load2		0				
store3	0							add1(2/2)	load1(2/4)	add3(2/2)	
store4	0										
store5	0										
dst = {0, 0, 0, 14, 23}											

Ciclo 12											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	1	addi	0	1			-1	a0	add2	&src	addiu addiu addi bgez
add2	1	addiu	0	&src+8			4	a1	0	&dst+8	
add3	1	bgez	add1					a2	0	4	
mul1	0							a3	0	3	
mul2	0							v0	add1	1	

Execução do Algoritmo - Tomasulo com Especulação

load1	1	lw	0	&src+4			&src+4	at	load3	0
load2	1	lw	0	&src+8			0	<div>Despachadas</div> <div>add2load2add1</div>		
load3	1	lw	add2				0			
load4	0									
load5	0							<div>Executando</div> <div>load1(3/4)add2(1/2)load2(1/4) add1(1/2)</div>		
store1	1	sw	0	&dst	load1		0			
store2	1	sw	0	&dst+4	load2		0			
store3	1	sw	0	&dst+8	load3		0			
store4	0									
store5	0									
dst = {0, 0, 0, 14, 23}										

Ciclo 13											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	1	addi	0	1			-1	a0	add2	&src	addiu addiu addi bgez
add2	1	addiu	0	&src+8			4	a1	0	&dst+8	
add3	1	bgez	add1					a2	0	4	
mul1	0							a3	0	3	
mul2	0							v0	add1	1	
load1	1	lw	0	&src+4			&src+4	at	load3	0	
load2	1	lw	0	&src+8			0				
load3	1	lw	add2				0	Despachadas			
load4	0										
load5	0										
store1	1	sw	0	&dst	load1		0				
store2	1	sw	0	&dst+4	load2		0				
store3	1	sw	0	&dst+8	load3		0	Executando			
store4	0							load1(4/4) add1(2/2)	add2(2/2)	load2(2/4)	
store5	0										
dst = {0, 0, 0, 14, 23}											

Ciclo 14											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	1	addiu	0	&src+12			4	a0	add1	&src+12	addi bgez
add2	1	addiu	0	&dst+8			4	a1	add2	&dst+8	
add3	1	bgez	0	0				a2	0	4	
mul1	0							a3	0	3	
mul2	0							v0	0	0	
load1	0							at	load3	0	
load2	1	lw	0	&src+8			&src+8	Despachadas			
load3	1	lw	0	&src+12			0	add3 add2	load3 store1	add1	
load4	0							Executando			
load5	0							load2(3/4) add1(1/2)	load3(1/4) add2(1/2)	add3(1/2) store1(1/4)	
store1	1	sw	0	&dst	0	23	0				
store2	1	sw	0	&dst+4	load2		0				
store3	1	sw	0	&dst+8	load3		0				
store4	0										
store5	0										
dst = {0, 0, 0, 14, 23}											

Ciclo 15															
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções				
add1	1	addiu	0	&src+12			4	a0	add1	&src+12	addi				
add2	1	addiu	0	&dst+8			4	a1	add2	&dst+8	bgez				
add3	1	bgez	0	0				a2	0	4					
mul1	0							a3	0	3					
mul2	0							v0	0	0					
load1	0							at	load3	0					
load2	1	lw	0	&src+8			&src+8	Despachadas							
load3	1	lw	0	&src+12			0								
load4	0														
load5	0														
store1	1	sw	0	&dst	0	23	0	Executando							
store2	1	sw	0	&dst+4	load2		0								
store3	1	sw	0	&dst+8	load3		0								
store4	0											load2(4/4) add1(2/2)	load3(2/4) add2(2/2)	add3(2/2) store1(2/4)	
store5	0														
dst = {0, 0, 0, 14, 23}															

Ciclo 16											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	1	addi	0	0			-1	a0	0	&src+16	

Execução do Algoritmo - Tomasulo com Especulação

add2	1	bgez	add1					a1	0	&dst+12
add3	1	syscall	load1					a2	0	4
mul1	0							a3	0	3
mul2	0							v0	load1	0
load1	1	lw					exit_val	at	load3	0
load2	0							Despachadas		
load3	1	lw	0	&src+12			&src+12	store2	add1	load1
load4	0							Executando		
load5	0							load3(3/4)	store1(3/4)	store2(1/4)
store1	1	sw	0	&dst	0	23	&dst	add1(1/2)	load1(1/4)	
store2	1	sw	0	&dst+4	0	12	0			
store3	1	sw	0	&dst+8	load3		0			
store4	0									
store5	0									

dst = {0, 0, 0, 14, 23}

Ciclo 17											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	1	addi	0	0			-1	a0	0	&src+16	
add2	1	bgez	add1					a1	0	&dst+12	
add3	1	syscall	load1					a2	0	4	
mul1	0							a3	0	3	
mul2	0							v0	load1	0	
load1	1	lw					exit_val	at	load3	0	
load2	0										
load3	1	lw	0	&src+12			&src+12	Despachadas			
load4	0										
load5	0										
store1	1	sw	0	&dst	0	23	&dst				
store2	1	sw	0	&dst+4	0	12	0	Executando			
store3	1	sw	0	&dst+8	load3		0	load3(4/4)	store1(4/4)	store2(2/4)	
store4	0							add1(2/2)	load1(2/4)		
store5	0										
dst = {0, 0, 0, 14, 23}											

Ciclo 18											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	0							a0	0	&src+16	
add2	1	bgez	0	-1				a1	0	&dst+12	
add3	1	syscall	load1					a2	0	4	
mul1	0							a3	0	3	
mul2	0							v0	load1	0	
load1	1	lw					exit_val	at	0	29	
load2	0							Despachadas			
load3	0							store3	add2		
load4	0										
load5	0										
store1	0										
store2	1	sw	0	&dst+4	0	12	&dst+4	Executando			
store3	1	sw	0	&dst+8	0	29	0	store3(1/4)	load1(3/4)	store2(3/4)	
store4	0							add2(1/2)			
store5	0										
dst = {23, 0, 0, 14, 23}											

Ciclo 19											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	0							a0	0	&src+16	
add2	1	bgez	0	-1				a1	0	&dst+12	
add3	1	syscall	load1					a2	0	4	
mul1	0							a3	0	3	
mul2	0							v0	load1	0	
load1	1	lw					exit_val	at	0	29	
load2	0										
load3	0							Despachadas			
load4	0										
load5	0										
store1	0										
store2	1	sw	0	&dst+4	0	12	&dst+4	Executando			
store3	1	sw	0	&dst+8	0	29	0	store3(2/4)	load1(4/4)	store2(4/4)	
store4	0							add2(2/2)			
store5	0										
dst = {23, 0, 0, 14, 23}											

Execução do Algoritmo - Tomasulo com Especulação

Ciclo 20											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	0							a0	0	&src+16	
add2	0							a1	0	&dst+12	
add3	1	syscall	0	10				a2	0	4	
mul1	0							a3	0	3	
mul2	0							v0	0	10	
load1	0							at	0	29	
load2	0							Despachadas			
load3	0										
load4	0										
load5	0										
store1	0							Executando			
store2	0										
store3	1	sw	0	&dst+8	0	29	0				
store4	0										
store5	0										
dst = {23, 12, 0, 14, 23}											

Ciclo 21											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	0							a0	0	&src+16	
add2	0							a1	0	&dst+12	
add3	1	syscall	0	10				a2	0	4	
mul1	0							a3	0	3	
mul2	0							v0	0	10	
load1	0							at	0	29	
load2	0							Despachadas			
load3	0										
load4	0										
load5	0										
store1	0							Executando store3(3/4) add3(2/2)			
store2	0										
store3	1	sw	0	&dst+8	0	29	&dst+8				
store4	0										
store5	0										
dst = {23, 12, 0, 14, 23}											

Ciclo 22											
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções
add1	0							a0	0	&src+16	
add2	0							a1	0	&dst+12	
add3	1	syscall	0	10				a2	0	4	
mul1	0							a3	0	3	
mul2	0							v0	0	10	
load1	0							at	0	29	
load2	0							Despachadas			
load3	0										
load4	0										
load5	0										
store1	0							Executando store3(4/4) add3(2/2)			
store2	0										
store3	1	sw	0	&dst+8	0	29	&dst+8				
store4	0										
store5	0										
dst = {23, 12, 0, 14, 23}											

Ciclo 23												
ER	Busy	Op	Qj	Vj	Qk	Vk	A	Reg	Qi	Valor	Fila de Instruções	
add1	0							a0	0	&src+16		
add2	0							a1	0	&dst+12		
add3	0							a2	0	4		
mul1	0							a3	0	3		
mul2	0							v0	0	10		
load1	0							at	0	29		
load2	0							Despachadas				
load3	0											
load4	0											
load5	0											
store1	0							Executando				
store2	0											
store3	0											

Execução do Algoritmo - Tomasulo com Especulação

[illegible]