

# Prática 3

Erick Henrique  
Marina Bernardes

08/2021

—  
Laboratório de Arquitetura e  
Organização de Computadores II

2021.1

—  
Daniela Cristina Cascini Kupsch

Centro Federal de Educação Tecnológica de Minas  
Gerais

## OBJETIVOS

Esta prática tem a finalidade de elaborar o algoritmo Tomasulo de despacho simples utilizando Verilog.



## O PROCESSO

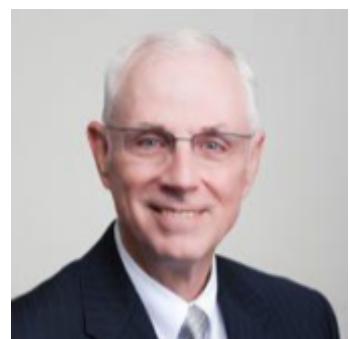
A prática 3 consistiu em projetar o algoritmo Tomasulo de despacho simples utilizando Verilog.

### Introdução

A evolução é uma característica natural de todo ser vivo, bem como todos objetos que estão presentes em sua vida, destacando-se principalmente a tecnologia na vida humana.

No mundo computacional, melhorar e otimizar constantemente o desempenho de um dispositivo, é um desafio desde que essa foi inventada. Tal desafio é semelhante a corrida espacial, onde temos diversas empresas espalhadas pelo mundo buscando sempre estar um passo à frente em suas pesquisas, tendo em vista o desenvolvimento, aprimoramento e melhoramento da performance, seja pelo *software* empregado, e principalmente o *hardware* utilizado.

Sendo assim, em termos de *hardware*, em 1967, Robert Tomasulo desenvolveu um algoritmo que foi implementado em unidades de ponto flutuante de um *mainframe IBM System/360 model 91* um algoritmo capaz de distribuir tarefas dinâmicas, permitindo paralelizar a execução de instruções através do gerenciamento de conflitos de dados. Esse algoritmo permite que a execução de uma determinada instrução comece antes que a outra instrução



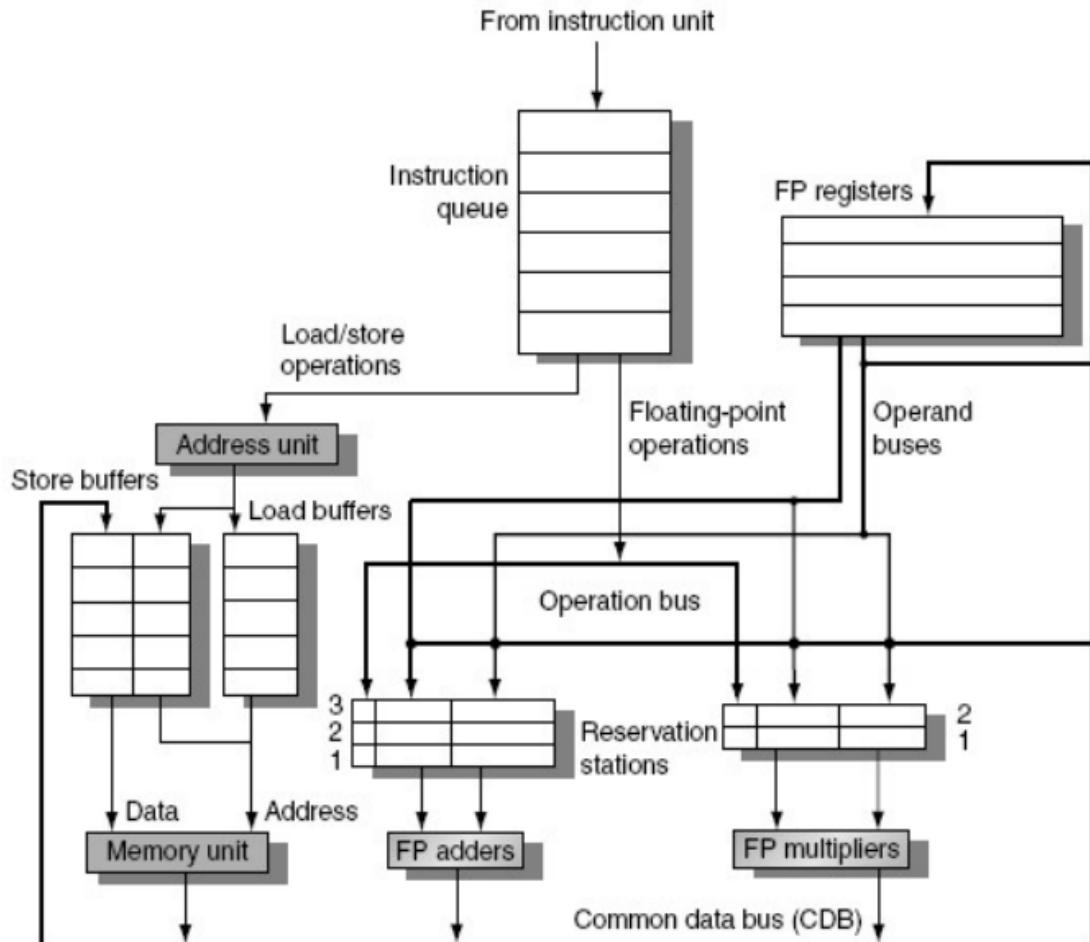
Robert Tomasulo, desenvolvedor do algoritmo de Tomasulo, em 1967 pela IBM.

---

já enviada, tenha sido concluída, o que é conhecido como execução fora de ordem.

## Desenvolvimento do projeto

Para a implementação do algoritmo foi seguido o projeto presente nos slides da aula teórica.



---

Implementamos sete módulos

- CDB
- Unidade Funcional
- Lista de Instrução
- Estação de Reserva
- MUX
- Banco de registradores
- Principal (que irá chamar o CDB).

O CDB arbitra o funcionamento geral do Tomasulo. Ele chama o buffer que contém as instruções e manda para a estação de reserva.

A estação de reserva recebe essas instruções e envia para a unidade funcional, onde é realizada as operações soma/subtração e multiplicação/divisão. O cálculo é feito e o resultado é enviado de volta para a estação de acordo com a latência necessária.

De volta na estação, as instruções são armazenadas em uma fila. Essa fila de instruções é percorrida e seus registradores são verificados ou não, dependentes de outras funções. Se forem, eles recebem a posição da fila da qual dependem. Caso a unidade funcional retorne que a instrução está pronta, a estação procura por ela na lista, volta os registradores, se dependentes, para zero e envia de volta para o CDB. O CDB faz o broadcast do resultado para o banco de registradores.

Tais instruções seguem o seguinte formato:

Instrução:	16'b0000000000000000					
	16'b	0000	000	000	000	000000
	Representa os 16 bits da instrução	Instrução - Opcode	Ry	Ry	Rz	offset

## Simulação

- Caso Teste
    - Dados
-

Para a simulação do processador implementado, utilizamos dois casos de testes listados a seguir, com sua implementação em binário:

Para a simularmos o algoritmo implementado, temos o seguinte código teste:

Instrução	Emis -são	Executa	Mem	Escreve CDB	Comentário
ADD <b>R3</b> , R1, R2 [1-2]	1	2 [soma]	-	3	Sem atraso
SUB R5, <b>R3</b> , R1 [2-4]	2	4 [soma]	-	5	Mostrar dependência de dados verdadeira
ADD <b>R5</b> , R4, R6 [3-5]	3	5 [soma]	-	6	Não há dependência, mas atrasa por hazard estrutural. FU cheia. Dependência de Saída
MUL R6, <b>R5</b> ,R4 [4-8]	4	7-8 [Mul]	-	9	Mostrar dependência de dados verdadeira
MUL R2, <b>R5</b> , R3 [5-10]	5	9-10 [Mul]	-	11	Unidade funcional cheia
SUB R6, <b>R5</b> , R4 [6-7]	6	7 [Soma]	-	8	Espera R5
ADD R6, <b>R5</b> , <b>R5</b> [7-8]	7	8 [Soma]	-	10	CDB cheio

Esperamos ter como resultado final o seguinte valores apresentados:

	R1	R2	R3	R4	R5	R6
Inicialização	1	1	1	2	0	1
ADD <b>R3</b> , R1, R2 [1-2]	1	1	<b>2</b>	2	0	1
SUB R5, <b>R3</b> , R1 [2-4]	1	1	2	2	<b>1</b>	1
ADD <b>R5</b> , R4, R6 [3-5]	1	1	2	2	<b>3</b>	1
MUL R6, <b>R5</b> ,R4 [4-8]	1	1	2	2	3	<b>6</b>

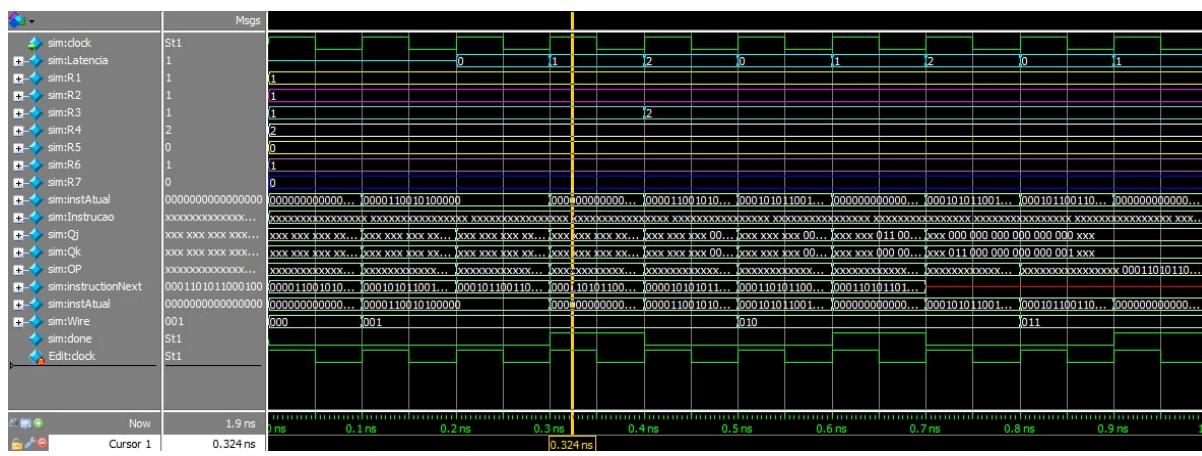
MUL R2, <b>R5</b> , R3 [5-10]	1	<b>6</b>	2	2	3	6
SUB R6, <b>R5</b> , R4 [6-7]	1	6	2	2	3	<b>1</b>
ADD R6, <b>R5</b> , <b>R5</b> [7-8]	1	6	2	2	3	<b>6</b>

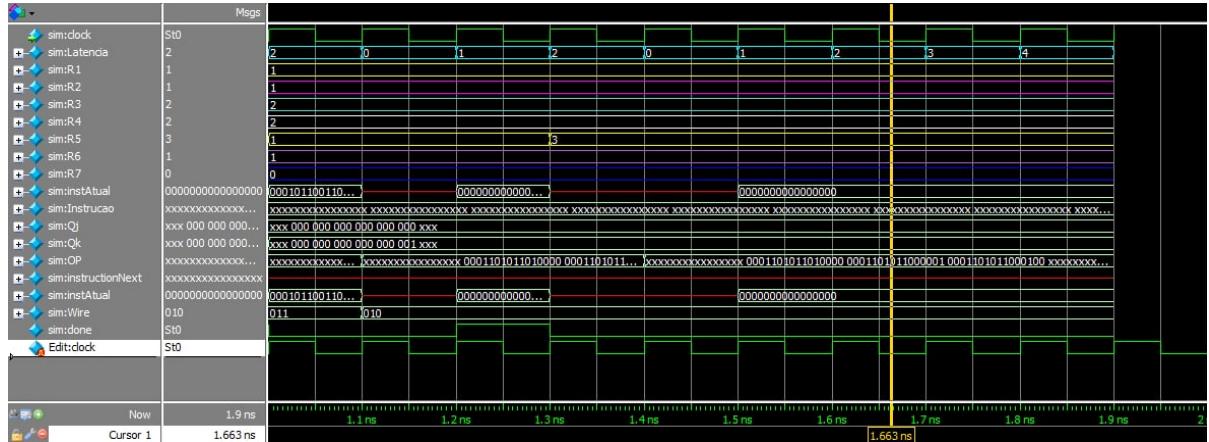
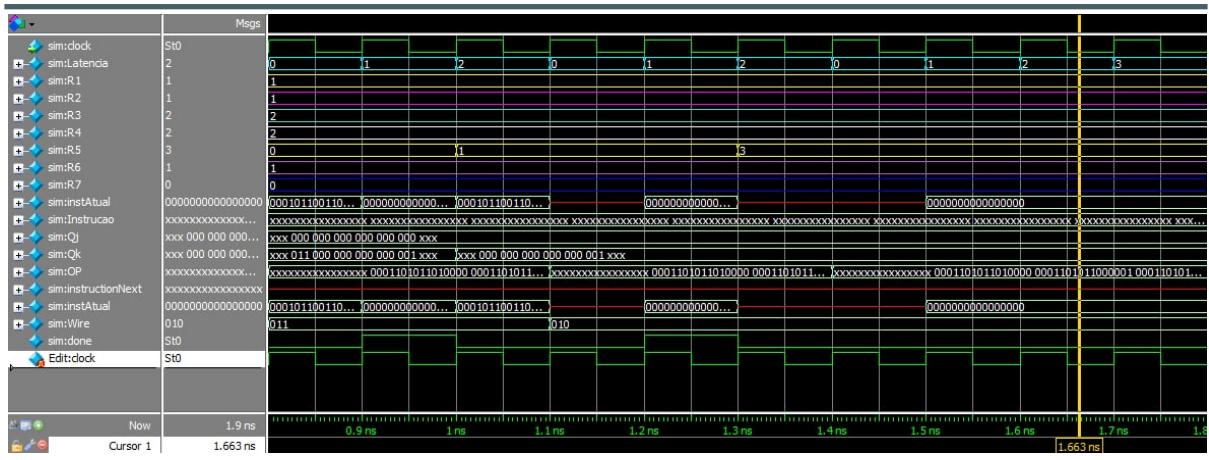
Sendo assim, temos para o nosso algoritmo de Tomasulo implementado, que as operações em binário representam:

Instrução	Instrução em binário
ADD <b>R3</b> , R1, R2 [1-2]	16'b0000110010100000
SUB R5, <b>R3</b> , R1 [2-4]	16'b0001010110010001
ADD <b>R5</b> , R4, R6 [3-5]	16'b0001011001100000
MUL R6, <b>R5</b> , R4 [4-8]	16'b0001101011000100
MUL R2, <b>R5</b> , R3 [5-10]	16'b0000101010110100
SUB R6, <b>R5</b> , R4 [6-7]	16'b0001101011000001
ADD R6, <b>R5</b> , <b>R5</b> [7-8]	16'b0001101011010000

## - Simulação

A partir da simulação do processador implementado em Verilog HDL, utilizamos o ModelSim para sua simulação, e o resultado é demonstrado da seguinte forma:





Pode-se perceber a quebra do código quando ele tenta armazenar o resultado da multiplicação no R6 (done está 1, logo a unidade funcional conseguiu calcular) e não consegue, uma vez que Wire não assume a posição 4, mas sim a posição 2. Isso faz com que o CDB entenda que a instrução número 4 não está pronta, então ele espera pela instrução 2, que já foi processada.

## Dificuldades

A própria linguagem Verilog se mostrou uma dificuldade na implementação. A falta de recursos básicos como um ‘break’ ocasionou a quebra do projeto a partir da terceira instrução. Além disso, apenas conseguimos ver de fato o que acontece no código através das simulações no ModelSim, tornando cansativo e difícil encontrar onde os erros ocorrem.

## Sugestões

---

O algoritmo Tomasulo é extremamente complexo, pois qualquer detalhe que foi mal implementado, ou um simples descuido pode ocasionar em diversos erros em formato

exponencial durante a simulação do algoritmo. Sendo assim, acreditamos que ao disponibilizar um pseudocódigo, ou até mesmo um esqueleto inicial de implementação, pode garantir um melhor entendimento do algoritmo, guiando-se para uma implementação segura e correta.

## Comentários

Não foi possível entregar o Tomasulo funcionando em pleno sucesso, muito devido às dificuldades apresentadas acima e devido também à não separação das unidades funcionais e estações de reserva para as operações de soma/subtração e de multiplicação/divisão.