INSTITUTO FEDERAL DE MINAS GERAIS - *CÂMPUS* SABARÁ
Rodovia MGC-262 s/n Km 10 – Sobradinho – Sabará – MG
CEP: 34590-390 | Tel: (31) 2102-9370

CURSO: Sistemas de Informação	**DISCIPLINA:** Arquitetura de Computadores	
PROFESSOR: Gabriel Novy	**VALOR:** 15 pontos	**DATA:** 27/07/2025

TRABALHO 2

A arquitetura MIPS é uma arquitetura de computadores do tipo RISC utilizada em alguns eletrônicos, com os consoles PlayStation e PlayStation 2. Essa arquitetura possui a principal característica de ser simples, de fácil entendimento e estudada nos cursos da área de Computação. Esta é a arquitetura que tem sido utilizada para estudo de caso na disciplina de Arquitetura de Computadores.

Em Organização de Computadores estudamos diversas técnicas que tem como objetivo elevar o desempenho do processador, sem a necessidade de aumentar a frequência do clock. Uma delas é o pipeline, que divide a execução da instrução em estágios permitindo a execução de múltiplas instruções em paralelo, cada uma em um diferente estágio. Apesar do ganho trazido pelo pipeline, alguns problemas surgem como efeito colateral. No caso do MIPS, esses problemas são os conflitos de dados do tipo RAW (Read After Write) e conflitos de controle.

Para contornar esse problema são utilizadas algumas técnicas, como a inserção de bolhas ou instruções NOP (No Operation), adiantamentos entre estágios e reordenação de instruções, e previsão de desvio.

Este trabalho tem como objetivo levar a(o) estudante a aperfeiçoar o seu conhecimento acerca do funcionamento de um pipeline e as soluções propostas para os problemas existentes. Para isso, deverá ser feito um programa que leia um arquivo de entrada contendo um conjunto de instruções MIPS e produza um arquivo de saída contendo o número de ciclos de processador que serão gastos para executar as instruções em um cenário com pipeline de 5 estágios com adiantamento de dados e sem reordenamento de instruções. Ou seja, **o seu programa deve identificar as dependências e resolvê-las utilizando BOLHAS/NOPS e os adiantamentos estudados** antes de calcular o total de ciclos que serão gastos. A seguir há um exemplo de arquivo de entrada e saída:

ENTRADA **TESTE-01.txt**	**SAÍDA – APENAS BOLHA** **TESTE-01-RESULTADO.txt**

`lw $t0, 1200($t1)`
`add $t0, $s2, $t0`
`sw $t0, 1200($t1)`
	8

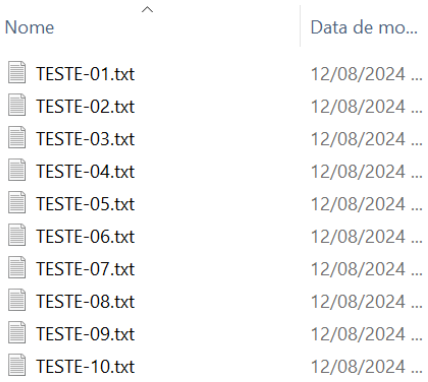
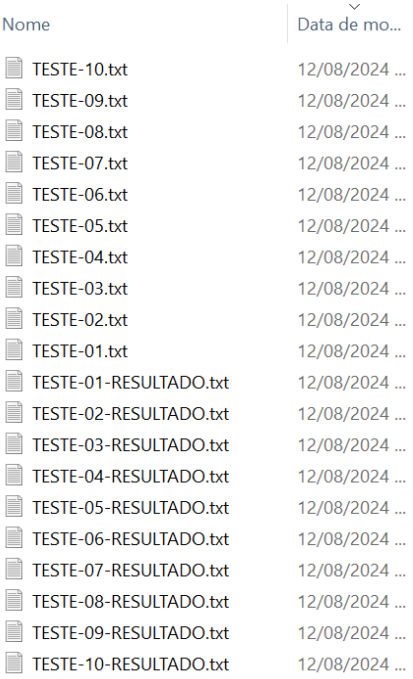
ENTRADA **TESTE-02.txt**	**SAÍDA – APENAS REORDENAMENTO DE INSTRUÇÕES** **TESTE-02-RESULTADO.txt**

`add $s1, $s3, $s3`
`sw $s1, 1240($zero)`
`lw $t0, 1200($t1)`
`beq $t1, $s2, 200`
`sw $t0, 1500($t1)`
	12

Os arquivos terão uma quantidade entre 1 e 1000 instruções. Cada instrução estará em uma linha. Os componentes da instrução poderão estar separados por **espaços**, **tabulação**, **vírgula** e **parênteses**. Os arquivos podem conter qualquer instrução das descritas no anexo I. Para maiores informações sobre as instruções, procure na internet ou o

professor em seu horário de atendimento (quintas-feiras, das 13:00 às 14:00). É necessário agendar com antecedência.

Para os testes a serem realizados no dia da apresentação, o programa deverá ler, em sequência, um total de 10 arquivos de entrada que **estarão na raiz de um pendrive** e que serão nomeados seguindo o formato "TESTE-XX.txt", sendo XX o número do arquivo de teste que irá de 01 a 10. Para cada arquivo de entrada, o programa deverá produzir um arquivo de saída contendo a resposta e salvar esses arquivos no mesmo diretório dos arquivos de entrada. Além disso, o arquivo de saída deverá ter o mesmo nome do arquivo de entrada acrescido de "-RESULTADO.txt". Ou seja, se o arquivo de entrada tiver o nome "TESTE-01.txt", o arquivo de saída deverá conter o nome "TESTE-01-RESULTADO.txt". Abaixo segue uma imagem de como estará a pasta de arquivos de teste antes da execução do seu programa e como é esperado que ela esteja após a execução do seu programa.

ANTES DA EXECUÇÃO DO PROGRAMA	**APÓS A EXECUÇÃO DO PROGRAMA**
	

O trabalho será desenvolvido em dupla e o software poderá ser implementado utilizando a linguagem que a dupla se sentir mais confortável. No entanto, **certifique-se de que os laboratórios possuem instalados os recursos necessários para a execução do programa. Se a dupla preferir, poderá utilizar notebook pessoal para a apresentação.**

A entrega ocorrerá exclusivamente pelo SUAP até o dia **27 de julho de 2025, às 23:59**. O trabalho, contendo **APENAS O CÓDIGO FONTE**, deverá ser entregue em um arquivo compactado (ZIP ou RAR) contendo apenas o nome e sobrenome dos dois integrantes da dupla. Exemplo: "Gabriel Novy e Cristiane Targa.zip". **ATENÇÃO:** Não é "Trabalho II - Fulano e Ciclano.zip" e nem "Trabalho Arquitetura.rar". Se atente às instruções.

O trabalho tem valor de 15 pontos e a avaliação seguirá a seguintes distribuição:

- 1 ponto para cada um dos 10 arquivos de teste: se o arquivo de saída corresponder ao esperado é acerto e ganha 1 ponto. Se o arquivo de saída não corresponder ao esperado, por qualquer motivo que seja, é 0 (zero). Então fique atento e siga as regras, inclusive no nome do arquivo de saída.
- 2,5 pontos para perguntas que irei fazer durante a execução dos testes.
- 2,5 pontos para o código.

A apresentação será no dia 28 de julho de 2025 no laboratório 101 e os dois integrantes da dupla deverão estar presentes. Cada dupla terá, no máximo, 5 minutos para a sua apresentação, que seguirá uma escala com o horário definido de apresentação. As duplas deverão enviar e-mail para gabriel.novy@ifmg.edu.br informando o nome dos integrantes. A escala será definida pela ordem inversa de envio dos e-mails. Quem enviar primeiro ficará por último na escala de apresentação A escala será disponibilizada neste mesmo arquivo posteriormente. Apenas serão aceitas apresentações posteriores nos casos previstos na legislação.

INTEGRANTES DA DUPLA	HORÁRIO DE APRESENTAÇÃO
	7:10
	7:15
	7:20
	7:25
	7:30
	7:35
	7:40
	7:45
INTERVALO	7:50 - 8:00
	8:00
	8:05
	8:10
	8:15
	8:20
	8:25
	8:30
	8:35

ANEXO I

FORMATO DE INSTRUÇÕES

Type	-31- format (bits) -0-					
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

INSTRUÇÕES LOAD E STORE

Instruction name	Mnemonic	Format	Encoding				EXEMPLO
Load Byte	LB	I	32_{10}	rs	rt	offset	lb $s1, 200($s2)
Load Halfword	LH	I	33_{10}	rs	rt	offset	lh $s1, 200($s2)
Load Word	LW	I	35_{10}	rs	rt	offset	lw $s1, 200($s2)
Store Byte	SB	I	40_{10}	rs	rt	offset	sb $s1, 200($s2)
Store Halfword	SH	I	41_{10}	rs	rt	offset	sh $s1, 200($s2)
Store Word	SW	I	43_{10}	rs	rt	offset	sw $s1, 200($s2)

INSTRUÇÕES LÓGICAS E ARITMÉTICAS

Instruction name	Mnemonic	Format	Encoding						EXEMPLO
Add	ADD	R	0_{10}	rs	rt	rd	0_{10}	32_{10}	add $s1, $s2, $s3
Subtract	SUB	R	0_{10}	rs	rt	rd	0_{10}	34_{10}	sub $s1, $s2, $s3
And	AND	R	0_{10}	rs	rt	rd	0_{10}	36_{10}	and $s1, $s2, $s3
Or	OR	R	0_{10}	rs	rt	rd	0_{10}	37_{10}	or $s1, $s2, $s3
Exclusive Or	XOR	R	0_{10}	rs	rt	rd	0_{10}	38_{10}	xor $s1, $s2, $s3
Add Immediate	ADDI	I	8_{10}	rs	rd	immediate			addi $s1, $s2, 245
And Immediate	ANDI	I	12_{10}	$s	$d	immediate			andi $s1, $s2, 245
Or Immediate	ORI	I	13_{10}	$s	$d	immediate			ori $s1, $s2, 245
Exclusive Or Immediate	XORI	I	14_{10}	$s	$d	immediate			xori $s1, $s2, 245
Load Upper Immediate	LUI	I	15_{10}	0_{10}	$d	immediate			liu $s1, $s2, 245

INSTRUÇÕES DE DESLOCAMENTO DE BITS

Instruction name	Mnemonic	Format	Encoding						EXEMPLO
Shift Left Logical	SLL	R	0_{10}	0_{10}	rt	rd	ra	0_{10}	sll $s1, $s2, 12
Shift Right Logical	SRL	R	0_{10}	0_{10}	rt	rd	sa	2_{10}	srl $s1, $s2, 12

INSTRUÇÕES DE DESVIO

Instruction name	Mnemonic	Format	Encoding						EXEMPLO
Jump Register	JR	R	0_{10}	rs	0_{10}	0_{10}	0_{10}	8_{10}	jr $s1
Jump	J	J	2_{10}	instr_index					j 1024
Branch on Equal	BEQ	I	4_{10}	rs	rt	offset			beq $s1, $s2, 300
Branch on Not Equal	BNE	I	5_{10}	rs	rt	offset			bne $s1, $s2, 300
Branch on Less Than or Equal to Zero	BLEZ	I	6_{10}	rs	0_{10}	offset			blez $s1, 300
Branch on Greater Than Zero	BGTZ	I	7_{10}	rs	0_{10}	offset			bgtz $s1, 300