



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS TRINDADE
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO CIÊNCIAS DA COMPUTAÇÃO

Rita Louro Barbosa – 22203157
Bianca Mazzuco Verzola - 22202621

INE5411 – Organização de Computadores I

Laboratório 06

Introdução

Os códigos e conhecimentos utilizados tomaram como base as informações dos módulos 1 a 12 de conteúdo disponibilizados no Moodle pelo professor.

1. Questão 1

1.1 Apresentação da questão

A questão 1 solicita um programa em Assembly do MIPS, a ser executado no simulador MARS, que calcule o fatorial de um número e atende às seguintes premissas:

- Receba via teclado o valor do número a ser calculado o fatorial;
- Efetue o cálculo do fatorial sem o uso de procedimentos; e
- Mostre o resultado na tela do computador.

1.2 Estrutura do código

Para a implementação em Assembly das operações requisitadas, foram utilizadas instruções aritméticas, de escrita e leitura do console e de desvio para realizar loops.

Inicialmente, no segmento de dados (.data), são colocadas 2 strings que serão, futuramente, escritas no console para facilitar o entendimento do usuário sobre o funcionamento do programa.

Logo depois, no início do segmento de programa (.text), uma dessas strings, "Digite o valor a ser calculado o fatorial: ", é escrita no console indicando ao usuário que ele deve escrever um número. E então, o cálculo do fatorial inicia.

Cálculo do fatorial:

- Número lido é colocado em um registrador temporário que será decrementado ao longo do cálculo;
- Um registrador que guardará o resultado do fatorial é inicializado com 1 para posteriormente as multiplicações serem realizadas;
- Loop inicia:
 - Condição para o loop acontecer: valor que está no registrador do número lido ser maior ou igual a 2;
 - O que acontece no loop:
 - Multiplica o resultado (iniciado com 1) pelo número lido;
 - Número lido é decrementado 1;
 - Volta ao início do loop testando a condição.

Assim, o cálculo acontece de forma que se o número dado pelo usuário for:

- 0 ou 1: o programa não entrará no loop e o resultado será 1 já que o registrador que guarda o resultado é inicializado com 1 antes do loop;

- números maiores ou iguais a 2: o programa entrará no teste de condição do loop n vezes (sendo “n” o número dado), porém entrará no nas operações contidas no loop n-1 vezes (já que no último teste, o registrador do número lido valerá 1), multiplicando n pelos números menores que ele e maiores que 1, ou seja, $n * (n-1) * (n-2) * \dots * 2$. (OBS: o 1 não entra nessa multiplicação pois o programa teria que realizar mais 1 loop sendo que o resultado seria o mesmo)

Após o cálculo ser realizado, a outra string armazenada na memória, “Resultado: “ é escrito no console e então o resultado do cálculo do fatorial é escrito no console.

1.3 Resultados

O correto funcionamento do programa pode ser verificado por meio de alguns testes:

- Fatorial de 0:

The screenshot shows the Mars debugger interface. The **Text Segment** window displays the following assembly code:

Bkpt	Address	Code	Basic	Source
	0x00400000	addiu \$2,\$0,4	7:	li \$v0, 4 # Comando para escrever string no console
	0x00400004	lui \$1,4097	8:	la \$a0, digite # "Digite o valor a ser calculado o fator..."
	0x00400008	ori \$4,\$1,0		
	0x0040000c	syscall	9:	syscall
	0x00400010	addiu \$2,\$0,5	13:	li \$v0, 5 # Comando para ler inteiro do teclado
	0x00400014	syscall	14:	syscall
	0x00400018	addu \$8,\$0,\$2	15:	move \$t0, \$v0 # \$t0 = valor lido
	0x0040001c	addu \$16,\$0,1	17:	li \$s0, 1 # \$s0 será o resultado do fatorial, inici...
	0x00400020	slti \$9,\$8,2	20:	slti \$t1, \$t0, 2 # \$t1 = 1 se \$t0 < 2
	0x00400024	bne \$9,\$0,4	21:	bne \$t1, \$zero, exit # Sai do loop se \$t1 != 0, ou sej...
	0x00400028	mul \$16,\$16,\$8	22:	mul \$s0, \$s0, \$t0 # \$s0 = \$s0 * \$t0

The **Data Segment** window shows memory addresses and values. The **Registers** window shows the state of registers, with \$t0 containing 1. The **Mars Messages** window shows the output: "Digite o valor a ser calculado o fatorial: 0", "Resultado: 1", and "-- program is finished running (dropped off bottom) --".

Enfoque no console:

The screenshot shows the **Mars Messages** window with the following output:

```
Digite o valor a ser calculado o fatorial: 0
Resultado: 1
-- program is finished running (dropped off bottom) --
```

O programa funciona corretamente para o número 0, já que $0! = 1$

- Fatorial de 1:

Text Segment

Bkpt	Address	Code	Basic	Source
7	0x00400000	addiu \$2,\$0,4		li \$v0, 4 # Comando para escrever string no console
8	0x00400004	li \$t0, 1		la \$a0, digite # "Digite o valor a ser calculado o fator..."
9	0x00400008	ori \$4,\$1,0		
13	0x0040000c	syscall		syscall
14	0x00400010	addiu \$2,\$0,5		li \$v0, 5 # Comando para ler inteiro do teclado
15	0x00400014	syscall		syscall
17	0x00400018	addu \$8,\$0,\$2		move \$t0, \$v0 # \$t0 = valor lido
20	0x0040001c	addiu \$16,\$0,1		li \$s0, 1 # \$s0 será o resultado do fatorial, inici...
21	0x00400020	slti \$t1,\$t0,2		slti \$t1, \$t0, 2 # \$t1 = 1 se \$t0 < 2
22	0x00400024	bne \$t1,\$zero,exit		bne \$t1, \$zero, exit # Sai do loop se \$t1 != 0, ou sej...
23	0x00400028	mul \$t6,\$t6,\$t0		mul \$t6, \$t6, \$t0 # \$t6 = \$t6 * \$t0

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	1768384836	1864394100	1818326560	1629516399	1919251232	1818321696	1634497891	1864396644
0x10010020	1952540192	1634300527	2112108	1970496850	1684108396	2112111	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0

Mars Messages

```

Digite o valor a ser calculado o fatorial: 1
Resultado: 1
-- program is finished running (dropped off bottom) --
  
```

Enfoque no console:

```

Digite o valor a ser calculado o fatorial: 1
Resultado: 1
-- program is finished running (dropped off bottom) --
  
```

O programa funciona corretamente para o número 1, já que $1! = 1$

- Fatorial de 2:

Text Segment

Bkpt	Address	Code	Basic	Source
7	0x00400000	addiu \$2,\$0,4		li \$v0, 4 # Comando para escrever string no console
8	0x00400004	li \$t0, 1		la \$a0, digite # "Digite o valor a ser calculado o fator..."
9	0x00400008	ori \$4,\$1,0		
13	0x0040000c	syscall		syscall
14	0x00400010	addiu \$2,\$0,5		li \$v0, 5 # Comando para ler inteiro do teclado
15	0x00400014	syscall		syscall
17	0x00400018	addu \$8,\$0,\$2		move \$t0, \$v0 # \$t0 = valor lido
20	0x0040001c	addiu \$16,\$0,1		li \$s0, 1 # \$s0 será o resultado do fatorial, inici...
21	0x00400020	slti \$t1,\$t0,2		slti \$t1, \$t0, 2 # \$t1 = 1 se \$t0 < 2
22	0x00400024	bne \$t1,\$zero,exit		bne \$t1, \$zero, exit # Sai do loop se \$t1 != 0, ou sej...
23	0x00400028	mul \$t6,\$t6,\$t0		mul \$t6, \$t6, \$t0 # \$t6 = \$t6 * \$t0

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	1768384836	1864394100	1818326560	1629516399	1919251232	1818321696	1634497891	1864396644
0x10010020	1952540192	1634300527	2112108	1970496850	1684108396	2112111	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0

Mars Messages

```

Digite o valor a ser calculado o fatorial: 2
Resultado: 2
-- program is finished running (dropped off bottom) --
  
```

Enfoque no console:

```

Digite o valor a ser calculado o fatorial: 2
Resultado: 2
-- program is finished running (dropped off bottom) --
  
```

O programa funciona corretamente para o número 2, já que $2! = 2 * 1 = 2$

- Fatorial de 3:

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x24020004	addiu \$2,\$0,4	7: li \$v0, 4 # Comando para escrever string no console
	0x00400004	0x3c011001	lui \$1,4097	8: la \$a0, digite # "Digite o valor a ser calculado o fator...
	0x00400008	0x34240000	ori \$4,\$1,0	
	0x0040000c	0x0000000c	syscall	9: syscall
	0x00400010	0x24020005	addiu \$2,\$0,5	13: li \$v0, 5 # Comando para ler inteiro do teclado
	0x00400014	0x0000000c	syscall	14: syscall
	0x00400018	0x00024021	addu \$8,\$0,\$2	15: move \$t0, \$v0 # \$t0 = valor lido
	0x0040001c	0x24100001	addiu \$16,\$0,1	17: li \$s0, 1 # \$s0 será o resultado do fatorial, inici...
	0x00400020	0x29090002	slti \$9,\$8,2	20: slti \$t1, \$t0, 2 # \$t1 = 1 se \$t0 < 2
	0x00400024	0x15200004	bne \$9,\$0,4	21: bne \$t1, \$zero, exit # Sai do loop se \$t1 != 0, ou sej...
	0x00400028	0x72080002	mul \$16,\$16,\$8	23: mul \$s0 = \$s0 * \$t0

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	1768384836	1864394100	1818326560	1629516399	1919251232	1818321696	1634497891	1864396644
0x10010020	1952540192	1634300527	2112108	1970496850	1684108396	2112111	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0

Mars Messages

Run I/O

Clear

Digite o valor a ser calculado o fatorial: 3
Resultado: 6
-- program is finished running (dropped off bottom) --

Registers

Enfoque no console:

Digite o valor a ser calculado o fatorial: 3
Resultado: 6
-- program is finished running (dropped off bottom) --

O programa funciona corretamente para o número 3, já que $3! = 3 * 2 * 1 = 6$

- Fatorial de 7:

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x24020004	addiu \$2,\$0,4	7: li \$v0, 4 # Comando para escrever string no console
	0x00400004	0x3c011001	lui \$1,4097	8: la \$a0, digite # "Digite o valor a ser calculado o fator...
	0x00400008	0x34240000	ori \$4,\$1,0	
	0x0040000c	0x0000000c	syscall	9: syscall
	0x00400010	0x24020005	addiu \$2,\$0,5	13: li \$v0, 5 # Comando para ler inteiro do teclado
	0x00400014	0x0000000c	syscall	14: syscall
	0x00400018	0x00024021	addu \$8,\$0,\$2	15: move \$t0, \$v0 # \$t0 = valor lido
	0x0040001c	0x24100001	addiu \$16,\$0,1	17: li \$s0, 1 # \$s0 será o resultado do fatorial, inici...
	0x00400020	0x29090002	slti \$9,\$8,2	20: slti \$t1, \$t0, 2 # \$t1 = 1 se \$t0 < 2
	0x00400024	0x15200004	bne \$9,\$0,4	21: bne \$t1, \$zero, exit # Sai do loop se \$t1 != 0, ou sej...
	0x00400028	0x72080002	mul \$16,\$16,\$8	23: mul \$s0 = \$s0 * \$t0

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	1768384836	1864394100	1818326560	1629516399	1919251232	1818321696	1634497891	1864396644
0x10010020	1952540192	1634300527	2112108	1970496850	1684108396	2112111	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0

Mars Messages

Run I/O

Clear

Digite o valor a ser calculado o fatorial: 7
Resultado: 5040
-- program is finished running (dropped off bottom) --

Registers

Enfoque no console:

Digite o valor a ser calculado o fatorial: 7
Resultado: 5040
-- program is finished running (dropped off bottom) --

O programa funciona corretamente para o número 7, já que $7! = 7 * 6 * 5 * 4 * 3 * 2 * 1 = 5040$

- Fatorial de 12:

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x24020004	addiu \$2,\$0,4	7: li \$v0, 4 # Comando para escrever string no console
	0x00400004	0x3c011001	lui \$1,4097	8: la \$a0, digite # "Digite o valor a ser calculado o fator..."
	0x00400008	0x34240000	ori \$4,\$1,0	
	0x0040000c	0x0000000c	syscall	9: syscall
	0x00400010	0x24020005	addiu \$2,\$0,5	13: li \$v0, 5 # Comando para ler inteiro do teclado
	0x00400014	0x0000000c	syscall	14: syscall
	0x00400018	0x00024021	addiu \$8,\$0,\$2	15: move \$t0, \$v0 # \$t0 = valor lido
	0x0040001c	0x24100001	addiu \$16,\$0,1	17: li \$s0, 1 # \$s0 será o resultado do fatorial, inici...
	0x00400020	0x29090002	slti \$9,\$8,2	20: slti \$t1, \$t0, 2 # \$t1 = 1 se \$t0 < 2
	0x00400024	0x15200004	bne \$9,\$0,4	21: bne \$t1, \$zero, exit # Sai do loop se \$t1 != 0, ou sej...
	0x00400028	0x72080002	mul \$16,\$16,\$8	22: mul \$a0, \$a0, \$t0 # \$a0 = \$a0 * \$t0

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	1768384836	1864394100	1818326560	1629516399	1919251232	1818321696	1634497891	1864396644
0x10010020	1952540192	1634300527	2112108	1970496850	1684108396	2112111	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0

Registers

Name	Number	Value
\$v0	2	1
\$v1	3	0
\$a0	4	479001600
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	1
\$t1	9	1
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	479001600
\$s1	17	0
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$s8	24	0
\$s9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	0
pc		4194388
hi		0
lo		479001600

Mars Messages

Run I/O

Clear

Digite o valor a ser calculado o fatorial: 12
Resultado: 479001600
-- program is finished running (dropped off bottom) --

Enfoque no console:

```
Digite o valor a ser calculado o fatorial: 12
Resultado: 479001600
-- program is finished running (dropped off bottom) --
```

O programa funciona corretamente para o número 12, já que $12! = 12 * 11 * 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 = 479001600$

- Para números iguais ou maiores que 13 o programa não gera o resultado correto pois os registradores usados no programa tem apenas 32 bits cada e eles não são suficientes para armazenar o resultado. Exemplo: fatorial de 13:

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x24020004	addiu \$2,\$0,0x00000004	7: li \$v0, 4 # Comando para escrever string no console
	0x00400004	0x3c011001	lui \$1,0x00010001	8: la \$a0, digite # "Digite o valor a ser calculado o fator..."
	0x00400008	0x34240000	ori \$4,\$1,0x00000000	
	0x0040000c	0x0000000c	syscall	9: syscall
	0x00400010	0x24020005	addiu \$2,\$0,0x00000005	13: li \$v0, 5 # Comando para ler inteiro do teclado
	0x00400014	0x0000000c	syscall	14: syscall
	0x00400018	0x00024021	addiu \$8,\$0,\$2	15: move \$t0, \$v0 # \$t0 = valor lido
	0x0040001c	0x24100001	addiu \$16,\$0,0x0000...	17: li \$s0, 1 # \$s0 será o resultado do fatorial, inici...
	0x00400020	0x29090002	slti \$9,\$8,0x00000002	20: slti \$t1, \$t0, 2 # \$t1 = 1 se \$t0 < 2
	0x00400024	0x15200004	bne \$9,\$0,0x00000004	21: bne \$t1, \$zero, exit # Sai do loop se \$t1 != 0, ou sej...
	0x00400028	0x72080002	mul \$16,\$16,\$8	22: mul \$a0, \$a0, \$t0 # \$a0 = \$a0 * \$t0

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x69676944	0x6f206574	0x6c617620	0x6120726f	0x72657320	0x6c616320	0x616c7563	0x6f206f64
0x10010020	0x74616620	0x6169726f	0x00203a6c	0x75736552	0x6461746c	0x00203a6f	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Registers

Name	Number	Value
\$v0	2	0x00000001
\$v1	3	0x00000000
\$a0	4	0x7328cc00
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000001
\$t1	9	0x00000001
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x7328cc00
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10000000
\$sp	29	0x7ffffc00
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400054
hi		0xffffffff
lo		0x7328cc00

Mars Messages

Run I/O

Clear

Digite o valor a ser calculado o fatorial: 13
Resultado: 1932053504
-- program is finished running (dropped off bottom) --

Enfoque no console:

```
Digite o valor a ser calculado o fatorial: 13
Resultado: 1932053504
-- program is finished running (dropped off bottom) --
```


O programa não funciona para esse valor já que $13! = 6227020800$

Se o programa usasse os registradores de ponto flutuante de dupla precisão, ele funcionaria corretamente para 13 e alguns números maiores que 13, porém o programa ainda teria um limite de funcionamento pois não existe registrador com uma quantidade infinita de bits.

2. Questão 2

2.1 Apresentação da questão

A questão 2 solicita um programa em Assembly do MIPS, a ser executado no simulador MARS, que calcule o fatorial de um número e atende às seguintes premissas:

- Receba via teclado o valor do número a ser calculado o fatorial;
- Chame uma função fatorial() - procedimento - para calcular, de modo recursivo, o fatorial do número; e
- Mostre o resultado na tela do computador.

2.2 Estrutura do código

Para a implementação em Assembly das operações requisitadas, foram utilizadas instruções aritméticas, de escrita e leitura do console, de desvio e de chamada e retorno de procedimentos.

Inicialmente, assim como na questão 1, no segmento de dados (.data), são colocadas 2 strings e, logo depois, no início do segmento de programa (.text), uma dessas strings, "Digite o valor a ser calculado o fatorial: ", é escrita no console indicando ao usuário que ele deve escrever um número. E então, o cálculo do fatorial inicia.

Cálculo do fatorial:

- Número lido é colocado como argumento \$a0 do procedimento *fatorial*;
- *fatorial* é chamado;
- Se o número dado pelo usuário for menor que 2 (0 ou 1):
 - Registrador de retorno \$v0 passa a ser 1;
 - O programa volta ao chamador do procedimento (código principal, conforme indicado no código).
- Caso contrário:
 - Salva na pilha o endereço atual de retorno e o valor do argumento \$a0 do procedimento atual;
 - O procedimento *fatorial* é chamado recursivamente com o argumento \$a0 decrementado 1 em relação ao \$a0 do procedimento que estava anteriormente.

- O novo procedimento verificará se \$a0 é maior ou igual a 2:
 - Se sim, \$a0 e \$sp serão salvos na pilha e *fatorial* será chamada novamente com $\$a0 = \$a0 - 1$ (isso continuará acontecendo até que \$a0 for igual a 1);
 - Se não:
 - O retorno \$v0 será inicializado com 1;
 - O programa volta ao chamador do procedimento atual (esse chamador é o *fatorial* anterior);
 - A pilha é restaurada, recuperando o endereço de retorno e o \$a0 do procedimento;
 - O valor do retorno \$v0 do *fatorial* anterior é multiplicado pelo \$a0 do *fatorial* atual, e o resultado é colocado no \$v0 (retorno do *fatorial* atual);
 - O programa volta ao chamador do procedimento atual, que pode estar novamente em outro *fatorial* e fará os 2 passos anteriores de novo até que o chamador seja parte do código principal, ou pode estar no código principal.

De volta ao código principal, a outra string armazenada na memória, “Resultado: “ é escrita no console e então o resultado do cálculo do fatorial é escrito no console. Após isso, o programa é encerrado.

Dessa forma, o cálculo acontece de forma que se o número dado pelo usuário for:

- 0 ou 1: o procedimento *fatorial* será chamado apenas 1 vez e o retorno dele será 1 (resultado do fatorial de 0 ou de 1);
- números maiores ou iguais a 2: o procedimento será chamado n vezes (sendo “n” o número dado), entrando os números de n a 1 como argumento \$a0 desses procedimentos e multiplicando todos esses argumentos ($1 * 2 * 3 * 4 * \dots * n$).

2.3 Resultados

O correto funcionamento do programa pode ser verificado por meio de alguns testes:

- Fatorial de 0:

The screenshot displays the Mars VM simulator interface. The **Text Segment** shows assembly code for a factorial program. The **Data Segment** shows memory values. The **Mars Messages** window shows the user input and output.

Bkpt	Address	Code	Basic	Source
7:	0x00400000	0x24020004	addiu \$2,\$0,0x00000004	li \$v0, 4 # Comando para escrever string no console
8:	0x00400004	0x3c011001	lui \$1,0x00001001	la \$a0, digite # "Digite o valor a ser calculado o fator...
9:	0x00400008	0x34240000	ori \$4,\$1,0x00000000	
12:	0x00400010	0x24020005	addiu \$2,\$0,0x00000005	li \$v0, 5 # Comando para ler inteiro do teclado
13:	0x00400014	0x0000000c	syscall	
15:	0x00400018	0x00022021	addu \$4,\$0,\$2	move \$a0, \$v0 # Passa o valor lido para o argumento \$a0...
16:	0x0040001c	0x0c100012	jal 0x00400048	jal fatorial # Chama o procedimento "fatorial"
17:	0x00400020	0x0002a821	addu \$21,\$0,\$2	move \$s5, \$v0
19:	0x00400024	0x24020004	addiu \$2,\$0,0x00000004	li \$v0, 4 # Comando para escrever string no console
20:	0x00400028	0x3c011001	lui \$1,0x00001001	la \$a0, resultado # "Resultado: " será escrito

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x69676944	0x6f206574	0x6c617620	0x6120726f	0x72657320	0x6c616320	0x616c7563	0x6f206f64
0x10010020	0x74616620	0x6169726f	0x00203a6c	0x75736552	0x6461746c	0x00203a6f	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Mars Messages

Run I/O

Clear

Digite o valor a ser calculado o fatorial: 0
Resultado: 1
-- program is finished running --

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x0000000a
\$v1	3	0x00000000
\$a0	4	0x00000001
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000001
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000001
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00400020
pc		0x00400048

Enfoque no console:

```
Digite o valor a ser calculado o fatorial: 0
Resultado: 1
-- program is finished running --
```

O programa funciona corretamente para o número 0, já que $0! = 1$

- Fatorial de 1:

The screenshot displays the Mars VM simulator interface. The **Text Segment** shows assembly code for a factorial program. The **Data Segment** shows memory values. The **Mars Messages** window shows the user input and output.

Bkpt	Address	Code	Basic	Source
7:	0x00400000	0x24020004	addiu \$2,\$0,0x00000004	li \$v0, 4 # Comando para escrever string no console
8:	0x00400004	0x3c011001	lui \$1,0x00001001	la \$a0, digite # "Digite o valor a ser calculado o fator...
9:	0x00400008	0x34240000	ori \$4,\$1,0x00000000	
12:	0x00400010	0x24020005	addiu \$2,\$0,0x00000005	li \$v0, 5 # Comando para ler inteiro do teclado
13:	0x00400014	0x0000000c	syscall	
15:	0x00400018	0x00022021	addu \$4,\$0,\$2	move \$a0, \$v0 # Passa o valor lido para o argumento \$a0...
16:	0x0040001c	0x0c100012	jal 0x00400048	jal fatorial # Chama o procedimento "fatorial"
17:	0x00400020	0x0002a821	addu \$21,\$0,\$2	move \$s5, \$v0
19:	0x00400024	0x24020004	addiu \$2,\$0,0x00000004	li \$v0, 4 # Comando para escrever string no console
20:	0x00400028	0x3c011001	lui \$1,0x00001001	la \$a0, resultado # "Resultado: " será escrito

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x69676944	0x6f206574	0x6c617620	0x6120726f	0x72657320	0x6c616320	0x616c7563	0x6f206f64
0x10010020	0x74616620	0x6169726f	0x00203a6c	0x75736552	0x6461746c	0x00203a6f	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Mars Messages

Run I/O

Clear

Digite o valor a ser calculado o fatorial: 1
Resultado: 1
-- program is finished running --

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x0000000a
\$v1	3	0x00000000
\$a0	4	0x00000001
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000001
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000001
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00400020
pc		0x00400048

Enfoque no console:

```
Digite o valor a ser calculado o fatorial: 1
Resultado: 1
-- program is finished running --
```

O programa funciona corretamente para o número 1, já que $1! = 1$

- Fatorial de 2:

Text Segment

Bkpt	Address	Code	Basic	Source
0x00400000	0x24020004	addiu \$2,\$0,0x00000004	7:	li \$v0, 4 # Comando para escrever string no console
0x00400004	0x3c011001	lui \$1,0x00001001	8:	la \$a0, digite # "Digite o valor a ser calculado o fator...
0x00400008	0x34240000	ori \$4,\$1,0x00000000		
0x0040000c	0x0000000c	syscall	9:	syscall
0x00400010	0x24020005	addiu \$2,\$0,0x00000005	12:	li \$v0, 5 # Comando para ler inteiro do teclado
0x00400014	0x0000000c	syscall	13:	syscall
0x00400018	0x00022021	addu \$4,\$0,\$2	15:	move \$a0, \$v0 # Passa o valor lido para o argumento \$a0...
0x0040001c	0x0c100012	jal 0x00400048	16:	jal fatorial # Chama o procedimento "fatorial"
0x00400020	0x0002a821	addu \$21,\$0,\$2	17:	move \$s5, \$v0
0x00400024	0x24020004	addiu \$2,\$0,0x00000004	19:	li \$v0, 4 # Comando para escrever string no console
0x00400028	0x3c011001	lui \$1,0x00001001	20:	la \$a0, resultado # "Resultado: " será escrita

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x69676944	0x6f206574	0x6c617620	0x6120726f	0x72657320	0x6c616320	0x616c7563	0x6f206f64
0x10010020	0x74616620	0x6169726f	0x00203a6c	0x75736552	0x6461746c	0x00203a6f	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Mars Messages

Run I/O

Clear

Digite o valor a ser calculado o fatorial: 2
Resultado: 2
-- program is finished running --

Registers

Coproc 1

Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x0000000a
\$v1	3	0x00000000
\$a0	4	0x00000002
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000001
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000002
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffeffc
\$fp	30	0x00000000
\$ra	31	0x00400020
pc		0x00400048

Enfoque no console:

Digite o valor a ser calculado o fatorial: 2
Resultado: 2
-- program is finished running --

O programa funciona corretamente para o número 2, já que $2! = 2 * 1 = 2$

- Fatorial de 3:

Text Segment

Bkpt	Address	Code	Basic	Source
0x00400000	0x24020004	addiu \$2,\$0,0x00000004	7:	li \$v0, 4 # Comando para escrever string no console
0x00400004	0x3c011001	lui \$1,0x00001001	8:	la \$a0, digite # "Digite o valor a ser calculado o fator...
0x00400008	0x34240000	ori \$4,\$1,0x00000000		
0x0040000c	0x0000000c	syscall	9:	syscall
0x00400010	0x24020005	addiu \$2,\$0,0x00000005	12:	li \$v0, 5 # Comando para ler inteiro do teclado
0x00400014	0x0000000c	syscall	13:	syscall
0x00400018	0x00022021	addu \$4,\$0,\$2	15:	move \$a0, \$v0 # Passa o valor lido para o argumento \$a0...
0x0040001c	0x0c100012	jal 0x00400048	16:	jal fatorial # Chama o procedimento "fatorial"
0x00400020	0x0002a821	addu \$21,\$0,\$2	17:	move \$s5, \$v0
0x00400024	0x24020004	addiu \$2,\$0,0x00000004	19:	li \$v0, 4 # Comando para escrever string no console
0x00400028	0x3c011001	lui \$1,0x00001001	20:	la \$a0, resultado # "Resultado: " será escrita

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x69676944	0x6f206574	0x6c617620	0x6120726f	0x72657320	0x6c616320	0x616c7563	0x6f206f64
0x10010020	0x74616620	0x6169726f	0x00203a6c	0x75736552	0x6461746c	0x00203a6f	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Mars Messages

Run I/O

Clear

Digite o valor a ser calculado o fatorial: 3
Resultado: 6
-- program is finished running --

Registers

Coproc 1

Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x0000000a
\$v1	3	0x00000000
\$a0	4	0x00000006
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000001
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000006
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffeffc
\$fp	30	0x00000000
\$ra	31	0x00400020
pc		0x00400048

Enfoque no console:

Digite o valor a ser calculado o fatorial: 3
Resultado: 6
-- program is finished running --

O programa funciona corretamente para o número 3, já que $3! = 3 * 2 * 1 = 6$

- Fatorial de 7:

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x24020004	addiu \$2,\$0,0x00000004	7: li \$v0, 4 # Comando para escrever string no console
	0x00400004	0x3c011001	lui \$1,0x00001001	8: la \$a0, digite # "Digite o valor a ser calculado o fator...
	0x00400008	0x34240000	ori \$4,\$1,0x00000000	
	0x0040000c	0x0000000c	syscall	9: syscall
	0x00400010	0x24020005	addiu \$2,\$0,0x00000005	12: li \$v0, 5 # Comando para ler inteiro do teclado
	0x00400014	0x0000000c	syscall	13: syscall
	0x00400018	0x00022021	addu \$4,\$0,\$2	15: move \$a0,\$v0 # Passa o valor lido para o argumento \$a0...
	0x0040001c	0x0c100012	jal 0x00400048	16: jal fatorial # Chama o procedimento "fatorial"
	0x00400020	0x0002a821	addu \$21,\$0,\$2	17: move \$s5,\$v0
	0x00400024	0x24020004	addiu \$2,\$0,0x00000004	19: li \$v0, 4 # Comando para escrever string no console
	0x00400028	0x3c011001	lui \$1,0x00001001	20: la \$a0, resultado # "Resultado: " será escrito

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x69676944	0x6f206574	0x6c617620	0x6120726f	0x72657320	0x6c616320	0x616c7563	0x6f206f64
0x10010020	0x74616620	0x6169726f	0x00203a6c	0x75736552	0x6461746c	0x00203a6f	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Mars Messages

Run I/O

Clear

Digite o valor a ser calculado o fatorial: 7
Resultado: 5040
-- program is finished running --

Registers

Enfoque no console:

```
Digite o valor a ser calculado o fatorial: 7
Resultado: 5040
-- program is finished running --
```

O programa funciona corretamente para o número 7, já que $7! = 7 * 6 * 5 * 4 * 3 * 2 * 1 = 5040$

- Fatorial de 12:

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x24020004	addiu \$2,\$0,0x00000004	7: li \$v0, 4 # Comando para escrever string no console
	0x00400004	0x3c011001	lui \$1,0x00001001	8: la \$a0, digite # "Digite o valor a ser calculado o fator...
	0x00400008	0x34240000	ori \$4,\$1,0x00000000	
	0x0040000c	0x0000000c	syscall	9: syscall
	0x00400010	0x24020005	addiu \$2,\$0,0x00000005	12: li \$v0, 5 # Comando para ler inteiro do teclado
	0x00400014	0x0000000c	syscall	13: syscall
	0x00400018	0x00022021	addu \$4,\$0,\$2	15: move \$a0,\$v0 # Passa o valor lido para o argumento \$a0...
	0x0040001c	0x0c100012	jal 0x00400048	16: jal fatorial # Chama o procedimento "fatorial"
	0x00400020	0x0002a821	addu \$21,\$0,\$2	17: move \$s5,\$v0
	0x00400024	0x24020004	addiu \$2,\$0,0x00000004	19: li \$v0, 4 # Comando para escrever string no console
	0x00400028	0x3c011001	lui \$1,0x00001001	20: la \$a0, resultado # "Resultado: " será escrito

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x69676944	0x6f206574	0x6c617620	0x6120726f	0x72657320	0x6c616320	0x616c7563	0x6f206f64
0x10010020	0x74616620	0x6169726f	0x00203a6c	0x75736552	0x6461746c	0x00203a6f	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Mars Messages

Run I/O

Clear

Digite o valor a ser calculado o fatorial: 12
Resultado: 479001600
-- program is finished running --

Registers

Enfoque no console:

```
Digite o valor a ser calculado o fatorial: 12
Resultado: 479001600
-- program is finished running --
```

O programa funciona corretamente para o número 12,

já que $12! = 12 * 11 * 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 = 479001600$

- Para números iguais ou maiores que 13 o programa não gera o resultado correto pois, assim como na questão 1, os registradores usados no programa tem apenas 32 bits cada e eles não são suficientes para armazenar o resultado. Exemplo: fatorial de 13:

The screenshot shows a MIPS simulator interface. The 'Text Segment' window displays assembly code for calculating the factorial of 13. The 'Data Segment' window shows memory addresses and values. The 'Registers' window shows the state of registers, with \$s0 containing the value 0x7328cc00 (1932053504) instead of the correct result 6227020800.

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x0000000a
\$v1	3	0x00000000
\$a0	4	0x7328cc00
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000001
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x7328cc00
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x00400020
pc		0x00400048

Enfoque no console:

```
Digite o valor a ser calculado o fatorial: 13
Resultado: 1932053504
-- program is finished running --
```

O programa não funciona para esse valor já que $13! = 6227020800$

3. Questão 3

3.1 Apresentação da questão

A questão solicita que, para os programas desenvolvidos na questão 1 e na questão 2, sejam feitas simulações sucessivas, com a variação dos seguintes parâmetros:

- Quantidade de entradas (BHT entries)
- Tamanho da BHT (1 e 2 bits)
- Valor inicial

Tais simulações deveriam ser feitas no simulador de BHT (Branch History Table) disponível no MARs.

Sobre estas simulações, é solicitada uma análise interpretativa dos resultados obtidos com a variação dos parâmetros, dando enfoque na comparação entre os resultados para os dois diferentes programas. O aluno deveria opinar sobre as diferenças de desempenho, dada determinada condição de teste.

3.2 Resultado das simulações

Conforme requisitado na descrição da questão, foram simulados casos teste de funcionamento e acurácia da BHT modificando os seguintes parâmetros:

- Número de entradas (BHT entries): Variando entre 8, 16 e 32 entradas.
- Número de bits da BHT (n bits BHT): Variando entre 1 e 2 bits
- Valor inicial da branch: TAKE ou NOT TAKE.

Para possibilitar a visualização das implicações da mudança de cada parâmetro em determinado cenário, foram simulados todos os cenários possíveis, com a variação de 1 parâmetro por vez. Para as simulações, foi utilizado o cálculo do fatorial de 12, por ser o maior fatorial possível de ser representado sem overflow em registradores de 32 bits.

O resultados obtidos estão expostos na tabela a seguir:

Parâmetros			Corretos		Incorretos		Precisão	
BHT entries	n bits BHT	Valor inicial	Código 1	Código 2	Código 1	Código 2	Código 1	Código 2
8	1	NOT TAKE	11	10	1	2	91,67	83,33
16	1	NOT TAKE	11	10	1	2	91,67	83,33
32	1	NOT TAKE	11	10	1	2	91,67	83,33
8	2	NOT TAKE	11	9	1	3	91,67	75
16	2	NOT TAKE	11	9	1	3	91,67	75
32	2	NOT TAKE	11	9	1	3	91,67	75
8	1	TAKE	10	11	2	1	83,33	91,67
16	1	TAKE	10	11	2	1	83,33	91,67
32	1	TAKE	10	11	2	1	83,33	91,67
8	2	TAKE	9	11	3	1	75	91,67
16	2	TAKE	9	11	3	1	75	91,67
32	2	TAKE	9	11	3	1	75	91,67

Um resultado notório apontado pelas simulações é a diferença na assertividade entre as predições iniciadas com NOT TAKE e TAKE, para os dois códigos. A predição que tem como valor inicial NOT TAKE tem maior taxa de acertos no código 1 em comparação com o código dois, levando a apenas 1 erro de predição para o primeiro código. Enquanto isso, As predições iniciadas com TAKE tem melhor resultado no código 2 que no código 1, com também apenas 1 erro de predição.

Essa diferença pode ser explicada pelo fato de que a eficiência da predição está diretamente relacionada à lógica de salto adotada no código.

O código 1, o salto a ser analisado está na seguinte linha:

```
21      bne      $t1, $zero, exit      # Sai do loop se $t1 != 0, ou seja, se $t1 < 2
```

Neste caso, se o salto for tomado, o fatorial para de ser calculado. Ou seja, o salto não deve ser tomado até que o cálculo do fatorial esteja pronto.

Já no código 2, o salto a ser analisado está na seguinte linha:

```
32      beq      $t1, $zero, l1      # Vai para o label "l1" se $a0 >= 2
```

Neste caso, o salto encaminha o programa para o cálculo do fatorial. Ou seja, o salto deve ser tomado enquanto o fatorial não estiver pronto.

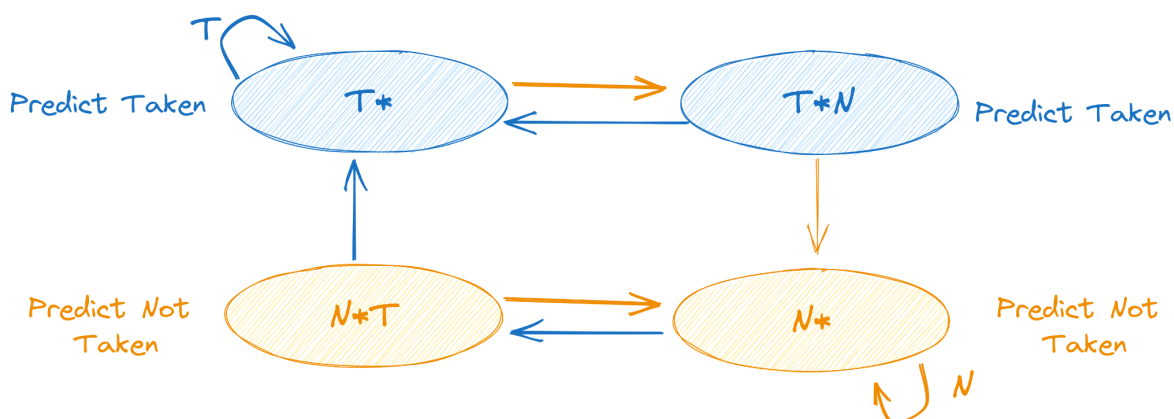
Sendo assim, quando iniciamos a predição com NOT TAKE, a predição do salto no código 1 já se inicia corretamente, permanecendo correta até o último salto, que precisa mudar de NOT TAKE para TAKE, gerando o único erro da sequência. Semelhantemente, quando iniciamos a predição do salto do código 2 com TAKE, já a iniciamos com o valor correto, mantendo-se assim até o fim do cálculo, com a mudança de TAKE para NOT TAKE, gerando também o único erro da sequência.

Nestes dois casos, a previsão se mantém constante e ótima (apenas 1 erro de predição) independentemente da mudança dos demais parâmetros da BHT.

Outro ponto a ser observado a respeito dos resultados da simulação, é que a modificação do número de entradas da BHT não gera nenhuma mudança na qualidade da predição. Isso porque, em ambos os códigos, há apenas 1 linha de código com salto condicional a ser previsto. Logo, independentemente do número de entradas disponíveis para uso, apenas 1 entrada será utilizada.

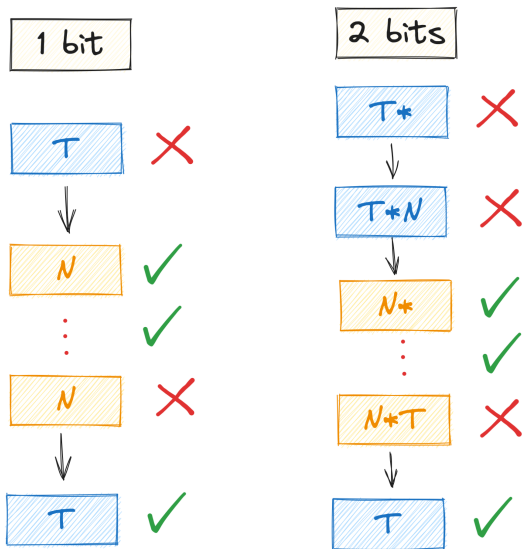
Ademais, nos cenários em que a predição não é favorecida pelo valor correto inicial (já abordado anteriormente), a mudança do parâmetro no número de bits da BHT de 1 para 2 bits tem como consequência uma piora nos resultados.

Para compreender o motivo da modificação do percentual de erro com a mudança no número de bits, realizou-se um estudo das sequências de previsões possíveis. Para explicar isso com exatidão, no caso do preditor com 2 bits, seria necessário saber o tipo de esquema de estados que o simulador MARs utiliza. Como não foi encontrada tal informação no simulador, adotou-se o esquema de estados dos contadores disponível no slide 25 do módulo 12 presente no Moodle. Sendo ele:



Para o código 1, o número de erros das previsões com valor inicial TAKE passa de 2 erros para 3 com a mudança de 1 bit para 2 bits na BHT. O esquema a seguir ilustra a interpretação encontrada para essa mudança:

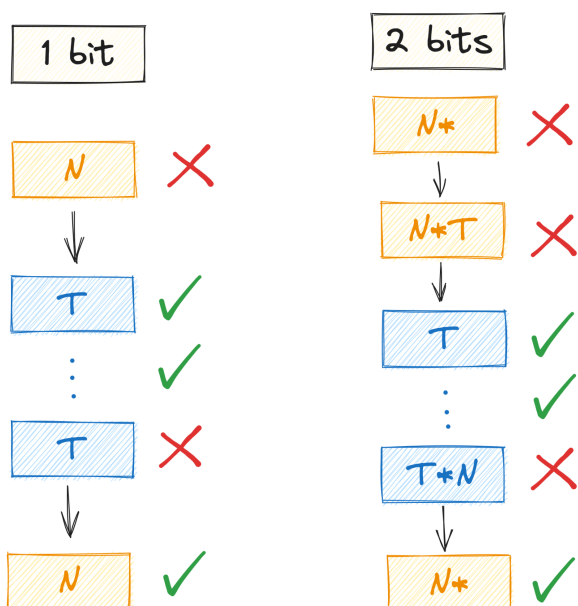
Código 1



Como mostrado na imagem, o erro é incrementado por conta do estágio intermediário necessário para a mudança de valor TAKE para NOT TAKE inicialmente.

Já no código 2, a precisão das previsões com valor inicial NOT TAKE passa de 2 erros para 3 com a mudança de 1 bit para 2 bits na BHT. O motivo desse aumento pode ser visualizado na seguinte imagem:

Código 2



Pelo mesmo motivo apresentado para o caso do código 1, o erro é incrementado por conta do estágio intermediário necessário para a mudança de valor NOT TAKE para TAKE inicialmente na sequência de predições.

Com isso, conclui-se que o aumento do número de bits não corresponde a uma melhoria na predição em qualquer caso, já que em ambos os cenários experimentados, o que ocorre é, na verdade, um decréscimo da eficiência da predição.

Por fim, respondendo a pergunta proposta no enunciado do trabalho “Qual das duas implementações você julga ter melhor desempenho? Em que condições?”, a resposta encontrada é: Em relação ao desempenho das predições, este varia, em primeiro lugar, com a determinação apropriada do valor inicial. Em segundo lugar, com a escolha de 1 bit apenas na BHT. Com isso posto, ambos os códigos possuem os mesmos resultados para as condições relativas analisadas, obtendo os mesmos valores nos melhores e piores cenários (pior caso: 3 erros, melhor caso: 1 erro).