

Relatório 4 – Organização de Computadores I – INE5411

Alunas: Ana Luiza Sales Gobbi (24105453), Fernanda Bertotti Guedes (24100601) e

Laísa Ágathe Bridi Dacroce (24100598)

Data: 14/05/25

1 INTRODUÇÃO

O presente relatório visa discorrer sobre os exercícios propostos no Laboratório 4 da disciplina de Organização de Computadores I. São abordadas as soluções desenvolvidas para os exercícios 1 e 2, assim como a análise do desempenho desses algoritmos com base na ferramenta BHT Simulator do MARS. Adicionalmente, tratamos dos principais desafios encontrados ao longo das atividades.

2 EXERCÍCIOS PROPOSTOS

2.1 EXERCÍCIO 1

O código do exercício 1 calcula o fatorial de um inteiro lido do teclado pela chamada de sistema 5 e armazenado no registrador \$t0. Não foi identificada a necessidade de armazenar valores na memória, então, a seção .data está vazia.

Inicialmente, o registrador \$t1, que irá armazenar o resultado da operação, recebe o valor 1, uma vez que esse é o menor valor que se pode obter de um cálculo de fatorial. Por definição, o fatorial de 1 e 0 é 1, então, é feita a comparação do valor lido no teclado com o valor de \$t1, por meio da instrução ble (branch if less or equal). Caso \$t0 seja menor ou igual a 1, o código avança diretamente para a label que imprime o resultado, a qual realiza uma chamada de sistema para mostrar o valor 1 no console. Caso contrário, o desvio não é efetuado e o código da label fatorial é executado.

A base da lógica do fatorial é realizar seguidas multiplicações do valor de \$t1 pelo valor de \$t0, este que é decrementado a cada novo desvio para a label fatorial. Quando as subtrações fazem com que \$t0 chegue em 1, deixa-se de desviar à label fatorial e o resultado armazenado em \$t1 é mostrado no console por meio da syscall 1. Então, o programa chega ao fim. A imagem abaixo comprova seu correto funcionamento para uma entrada igual a 10, pois, ao final do programa, o valor de \$t1 contém o resultado $10! = 3628800$, e o valor de \$t0, usado para sinalizar o fim do loop do fatorial, armazena 1.

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0
\$at	1	1
\$v0	2	10
\$v1	3	0
\$a0	4	3628800
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	1
\$t1	9	3628800
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	0
\$s1	17	0
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$t8	24	0
\$t9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	0
pc		4194364
hi		0
lo		3628800

Figura 1 – Registradores ao final da execução do programa 1 para calcular 10!

2.2 EXERCÍCIO 2

O algoritmo do exercício 2 utiliza um procedimento não-folha que realiza chamadas recursivas para o cálculo do fatorial de um número lido no teclado. Também neste código não foi identificada a necessidade de armazenar valores no segmento de dados, motivo pelo qual a seção .data está vazia.

Na seção main, é recebido o número cujo fatorial será calculado, através de uma chamada de sistema. Então, é feita a chamada da função fatorial, e seu resultado retornado em \$v0 é armazenado no registrador \$a0, para que possa ser mostrado no console após outra chamada de sistema.

A função fatorial, por sua vez, calcula o fatorial do valor armazenado em \$a0. Antes da chamada recursiva, o procedimento salva o endereço de retorno e o argumento na pilha, utilizando os comandos sw \$ra, 4(\$sp) e sw \$a0, 0(\$sp). Esse armazenamento é necessário

para garantir que, ao retornar de uma chamada recursiva, o programa continue de onde parou e com os valores corretos.

Em seguida, a função verifica se o valor de \$a0 é menor que 1, o que caracteriza o caso base da recursão. Se essa condição for verdadeira, o programa define o valor de retorno como 1, restaura o endereço de retorno da pilha e encerra a execução do procedimento. Caso contrário, o valor de \$a0 é decrementado em 1 e uma nova chamada recursiva de fatorial é feita. Quando a execução retorna, o valor original de \$a0 é restaurado da pilha, e o resultado da multiplicação $n * \text{fatorial}(n-1)$ é armazenado no registrador \$v0. Finalmente, o endereço de retorno de \$ra também é restaurado e a pilha é ajustada para remover os dados armazenados temporariamente, encerrando a execução da função com jr \$ra.

Esse procedimento não-folha demonstra o uso da pilha para suportar chamadas recursivas em Assembly MIPS, garantindo a preservação do estado de cada chamada. A imagem abaixo comprova o funcionamento do exercício 2.

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0
\$at	1	0
\$v0	2	10
\$v1	3	0
\$a0	4	3628800
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	1
\$t1	9	0
\$t2	10	0
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	0
\$s1	17	0
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$t8	24	0
\$t9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$sp	29	2147479548
\$fp	30	0
\$ra	31	4194320
pc		4194340
hi		0
lo		3628800

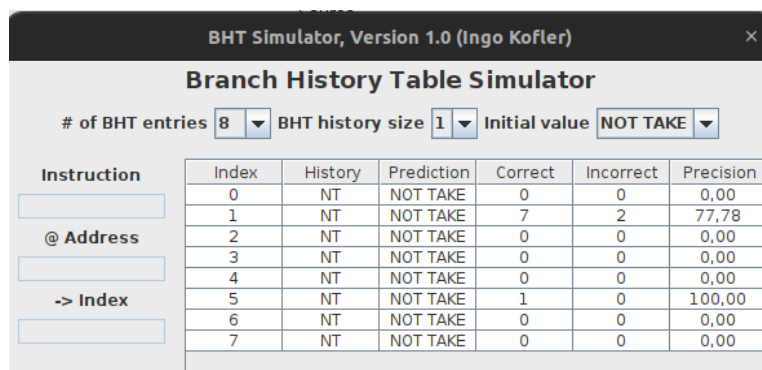
Figura 2 - Registradores ao final da execução do programa 2 para calcular 10!

2.3 EXERCÍCIO 3

Tivemos dificuldade para comparar os resultados da BHT Simulator entre os dois códigos de fatorial, porque a primeira implementação propõe dois usos de desvio condicional, enquanto a segunda demanda apenas um. Diante disso, tentamos alterar o primeiro código para que ele tivesse somente um desvio, possivelmente usando instruções do tipo set ou alterando a lógica de desvio da label fatorial, porém o caso de entrada igual a 0 se mostrou desafiador. Então, optamos por realizar a comparação entre os códigos, mesmo com a diferença no número de desvios condicionais.

Para a análise dos algoritmos, assumimos que aquele que tem melhor desempenho apresenta a maior taxa de acertos de predições na Branch History Table do MARS, pois isso significa uma menor quantidade de atrasos do processador. Isso porque, a cada erro de previsão de instruções de desvio, instruções já processadas precisam ser descartadas, ocasionando penalidades de desempenho.

Primeiramente, chegamos à conclusão de que não há mudança no desempenho, para ambos os códigos, ao mudarmos o número de entradas da tabela, mantendo tudo o mais constante (valor de entrada, número de bits da tabela e valor da primeira predição). Isso se deve ao fato de que, no pior caso, temos dois branches no exercício 1, o que ainda assim é um número muito pequeno para que haja a possibilidade de colisão quando o número de entradas é maior ou igual a 8. As figuras 3, 4 e 5 abaixo exemplificam a obtenção do mesmo valor para predições que têm como diferença apenas o número de entradas BHT.



Index	History	Prediction	Correct	Incorrect	Precision
0	NT	NOT TAKE	0	0	0,00
1	NT	NOT TAKE	7	2	77,78
2	NT	NOT TAKE	0	0	0,00
3	NT	NOT TAKE	0	0	0,00
4	NT	NOT TAKE	0	0	0,00
5	NT	NOT TAKE	1	0	100,00
6	NT	NOT TAKE	0	0	0,00
7	NT	NOT TAKE	0	0	0,00

Figura 3 – BHT de 8 entradas para o cálculo de 10!

BHT Simulator, Version 1.0 (Ingo Kofler)

Branch History Table Simulator

of BHT entries BHT history size Initial value

Instruction	Index	History	Prediction	Correct	Incorrect	Precision
	0	NT	NOT TAKE	0	0	0,00
	1	NT	NOT TAKE	0	0	0,00
	2	NT	NOT TAKE	0	0	0,00
	3	NT	NOT TAKE	0	0	0,00
	4	NT	NOT TAKE	0	0	0,00
	5	NT	NOT TAKE	1	0	100,00
	6	NT	NOT TAKE	0	0	0,00
	7	NT	NOT TAKE	0	0	0,00
	8	NT	NOT TAKE	0	0	0,00
	9	NT	NOT TAKE	7	2	77,78
	10	NT	NOT TAKE	0	0	0,00
	11	NT	NOT TAKE	0	0	0,00
	12	NT	NOT TAKE	0	0	0,00
	13	NT	NOT TAKE	0	0	0,00
	14	NT	NOT TAKE	0	0	0,00
	15	NT	NOT TAKE	0	0	0,00

Figura 4 – BHT de 16 entradas para o cálculo de 10!

BHT Simulator, Version 1.0 (Ingo Kofler)

Branch History Table Simulator

of BHT entries BHT history size Initial value

Instruction	Index	History	Prediction	Correct	Incorrect	Precision
	0	NT	NOT TAKE	0	0	0,00
	1	NT	NOT TAKE	0	0	0,00
	2	NT	NOT TAKE	0	0	0,00
	3	NT	NOT TAKE	0	0	0,00
	4	NT	NOT TAKE	0	0	0,00
	5	NT	NOT TAKE	1	0	100,00
	6	NT	NOT TAKE	0	0	0,00
	7	NT	NOT TAKE	0	0	0,00
	8	NT	NOT TAKE	0	0	0,00
	9	NT	NOT TAKE	7	2	77,78
	10	NT	NOT TAKE	0	0	0,00
	11	NT	NOT TAKE	0	0	0,00
	12	NT	NOT TAKE	0	0	0,00
	13	NT	NOT TAKE	0	0	0,00
	14	NT	NOT TAKE	0	0	0,00
	15	NT	NOT TAKE	0	0	0,00
	16	NT	NOT TAKE	0	0	0,00
	17	NT	NOT TAKE	0	0	0,00
	18	NT	NOT TAKE	0	0	0,00
	19	NT	NOT TAKE	0	0	0,00
	20	NT	NOT TAKE	0	0	0,00
	21	NT	NOT TAKE	0	0	0,00
	22	NT	NOT TAKE	0	0	0,00
	23	NT	NOT TAKE	0	0	0,00
	24	NT	NOT TAKE	0	0	0,00

Figura 5 – BHT de 32 entradas para o cálculo de 10!

Adicionalmente, ao analisar vários resultados do código do exercício 1, foi possível perceber que a BHT de 1 bit se mostrou mais precisa que a de 2 bits quando NOT take é a primeira predição. Nesse caso, o padrão de execução é quase sempre “tomar”, com exceção do primeiro e último ciclos. Como o preditor de 1 bit muda mais rápido, ele se adapta mais rapidamente ao comportamento do loop, o que é vantajoso para esse código, em que o padrão é fortemente dominante em uma direção.

BHT Simulator, Version 1.0 (Ingo Kofler)

Branch History Table Simulator

of BHT entries BHT history size Initial value

Instruction	Index	History	Prediction	Correct	Incorrect	Precision
	0	NT	NOT TAKE	0	0	0,00
	1	NT	NOT TAKE	4	2	66,67
	2	NT	NOT TAKE	0	0	0,00
	3	NT	NOT TAKE	0	0	0,00
	4	NT	NOT TAKE	0	0	0,00
	5	NT	NOT TAKE	1	0	100,00
	6	NT	NOT TAKE	0	0	0,00
	7	NT	NOT TAKE	0	0	0,00

Figura 6 – BHT de 1 bit, cálculo de 7!, precisão total = 71,4%.

BHT Simulator, Version 1.0 (Ingo Kofler)

Branch History Table Simulator

of BHT entries BHT history size Initial value

Instruction	Index	History	Prediction	Correct	Incorrect	Precision
	0	NT, NT	NOT TAKE	0	0	0,00
	1	T, NT	TAKE	3	3	50,00
	2	NT, NT	NOT TAKE	0	0	0,00
	3	NT, NT	NOT TAKE	0	0	0,00
	4	NT, NT	NOT TAKE	0	0	0,00
	5	NT, NT	NOT TAKE	1	0	100,00
	6	NT, NT	NOT TAKE	0	0	0,00
	7	NT, NT	NOT TAKE	0	0	0,00

Figura 7 – BHT de 2 bits, cálculo de 7!, precisão total = 57,1%.

Entretanto, quando a predição inicial é take, as BHTs de 1 e 2 bits têm o mesmo desempenho para o mesmo valor de entrada, conforme mostram as figuras 8 e 9 abaixo:

BHT Simulator, Version 1.0 (Ingo Kofler)

Branch History Table Simulator

of BHT entries BHT history size Initial value

Instruction	Index	History	Prediction	Correct	Incorrect	Precision
	0	T, T	TAKE	0	0	0,00
	1	T, NT	TAKE	8	1	88,89
	2	T, T	TAKE	0	0	0,00
	3	T, T	TAKE	0	0	0,00
	4	T, T	TAKE	0	0	0,00
	5	T, NT	TAKE	0	1	0,00
	6	T, T	TAKE	0	0	0,00
	7	T, T	TAKE	0	0	0,00

Figura 8 - BHT de 2 bits, TAKE, precisão total = 80%

BHT Simulator, Version 1.0 (Ingo Kofler)

Branch History Table Simulator

of BHT entries BHT history size Initial value

Instruction	Index	History	Prediction	Correct	Incorrect	Precision
<input type="text"/>	0	T	TAKE	0	0	0,00
@ Address	1	NT	NOT TAKE	8	1	88,89
<input type="text"/>	2	T	TAKE	0	0	0,00
-> Index	3	T	TAKE	0	0	0,00
<input type="text"/>	4	T	TAKE	0	0	0,00
	5	NT	NOT TAKE	0	1	0,00
	6	T	TAKE	0	0	0,00
	7	T	TAKE	0	0	0,00

Figura 9 - BHT de 1 bit, TAKE, precisão total = 80%

A partir disso, foi possível identificar, por exemplo, que para o fatorial de 0, os exercícios 1 e 2 apresentam 100% de precisão, seja para BHT de 1 ou 2 bits. Contudo, o caso do fatorial de 1 representou uma quebra desse padrão e o desempenho do código 1 foi favorecido em detrimento do código 2. Este apresentou 0% de precisão, com dois desvios incorretos, na BHT de 1 bit, e 50% de precisão na BHT de 2 bits, enquanto que o código 1 manteve 100% de precisão. Nesse caso específico, o código 1 apresentou maior previsibilidade de desvios, a despeito de tê-los em maior quantidade.

Em geral, foi possível perceber que a solução do exercício 2 apresentou melhor desempenho em relação ao exercício 1, em todas BHTs de 1 bit, independentemente dos outros parâmetros, como valor de entrada e predição inicial. Esse comportamento pode ser explicado, uma vez que a solução 2 possui menos branches no total. Porém, na BHT de 2 bits, o código do exercício 2 apresenta maior precisão somente quando NOT take é considerada a primeira predição. Portanto, adotando BHT de 1 bit, o código 2 apresentou melhor desempenho.