

Relatório do Laboratório 05 - Organização de Computadores

Grupo:

- Henrique Mateus Teodoro - 23100472
- Rodrigo Schwartz - 23100471

Exercício 1

O exercício 1 consiste em criar um programa em Assembly MIPS que calcule o fatorial de um número sem o uso de procedimentos. Esse número deve ser digitado pelo usuário e o resultado é impresso na tela. Inicialmente, no segmento .data, são instanciadas as variáveis que serão impressas na tela:

```
.data
    input: .asciiz "insira o valor de x: "
    output: .asciiz "o fatorial de x é: "
```

Começando o segmento .text, faz-se necessário ler o número digitado pelo usuário para se calcular o fatorial. Assim, a string de input é imprimida na tela e é solicitado ao usuário para digitar um número x. Esse número inteiro x é salvo no registrador \$t0. Essas funcionalidades são feitas nas seguintes instruções:

```
.text
    li $v0, 4                # seta a operação de impressão de string
    la $a0, input            # passa um ponteiro que aponta para input para $a0
    syscall

    li $v0, 5                # seta a operação de leitura de inteiro
    syscall                  # salva em $t0
    move $t0, $v0
```

As instruções que calculam efetivamente o fatorial de um número x são as seguintes:

```
    li $t2, 1
    li $t1, 1                # Inicializa o contador do loop (i = 1)
loop:
    bgt $t1, $t0, fim_fatorial

    mul $t2, $t2, $t1        # Multiplica o fatorial acumulado por $t1
    addi $t1, $t1, 1         # Incrementa o contador do loop

    j loop                   # Volta para o início do loop

fim_fatorial:
```

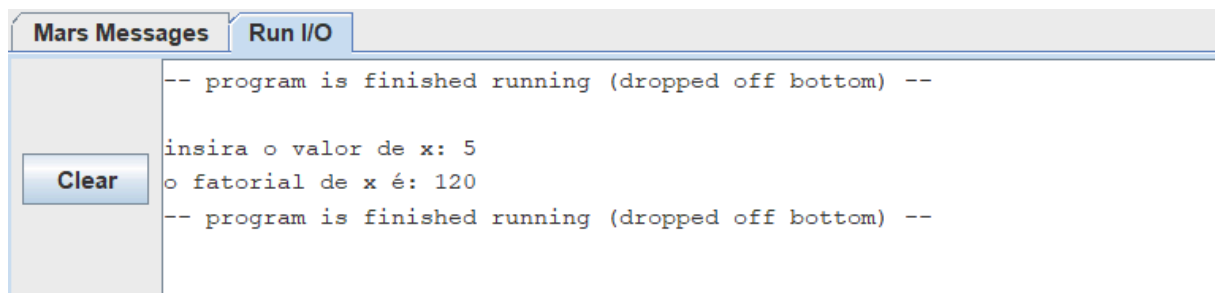
O registrador \$t1, inicializado com o valor 1, vai ser utilizado como um contador no código. Já o registrador \$t2, também inicializado com o valor 1, será o acumulador das multiplicações. Vale lembrar que o registrador \$t0 contém o valor digitado pelo usuário para calcular seu fatorial, ele será utilizado como comparador do loop. O desenvolvimento básico do código funciona de uma maneira que, o contador (\$t1) inicialmente é comparado com o número digitado pelo usuário (\$t0), e caso ele seja maior, há um salto para o label fim_fatorial, encerrando o cálculo. Caso não seja maior, o contador (\$t1) multiplica o valor do acumulador (\$t2) e o resultado é salvo no próprio acumulador (\$t2). Após isso, o contador é incrementado e através da instrução jump voltamos para o início do loop, onde é feita a comparação. No final, o fatorial calculado fica salvo \$t2. Depois dessas operações, basta imprimir a string de output e o valor do fatorial, sendo isso feito pelas seguintes instruções:

```
fim_fatorial:

    li $v0, 4           # seta a operação de impressão de string
    la $a0, output      # passa um ponteiro que aponta para output para $a0
    syscall

    li $v0, 1           # seta a operação de impressão de int
    move $a0, $t2
    syscall
```

Nota-se o bom funcionamento do código com a imagem a seguir:



The screenshot shows the 'Mars Messages' window in the MARS MIPS simulator. The 'Run I/O' tab is selected. The output text is as follows:

```
-- program is finished running (dropped off bottom) --

insira o valor de x: 5
o fatorial de x é: 120
-- program is finished running (dropped off bottom) --
```

On the left side of the window, there is a 'Clear' button.

Exercício 2

O exercício 2 consiste em criar um programa em Assembly MIPS que realiza o cálculo do fatorial de um número de modo recursivo, onde recebe-se o número cujo fatorial será calculado via teclado, e no final, imprime-se o resultado no console.

Inicialmente, é feita a declaração das variáveis que serão utilizadas durante a execução do programa:

```
.data
string_n: .asciiz "Digite o valor de n: "
string_result: .asciiz "O fatorial de n é: "
```

Para este programa, foram declaradas apenas duas variáveis do tipo string, string_n e string_result. A variável string_n será utilizada no início do programa, no input do usuário, enquanto string_result será utilizada no término do programa, para mostrar o resultado final.

No seguimento de instruções, é inicialmente feita a impressão da string_n, e a leitura do inteiro n, para que o fatorial de n possa ser calculado:

```
.text
li $v0, 4                # seta a operação de impressão de string
la $a0, string_n         # passa a string que será impressa para $a0
syscall                 # faz a impressão da string

li $v0, 5                # seta a operação de leitura de inteiro
syscall                 # faz a leitura do inteiro n
move $t0, $v0            # move o valor lido para $t0

move $a0, $t0            # move o valor de n ($t0) para $a0
```

Após a leitura do inteiro n, armazena-se o valor lido no registrador \$t0, e move-se esse valor para o registrador de argumentos \$a0, para que o valor do inteiro n seja passado como um argumento da função fatorial.

É então feita a chamada da função fatorial:

```
jal fatorial             # chama a função fatorial
j exit                  # desvia para a label exit
```

Após a execução da instrução jal, é feito o desvio para a label fatorial, onde será iniciada a computação do cálculo do fatorial de n.

```

fatorial:
    addi $sp, $sp, -8           # aumenta a stack para armazenar dois valores
    sw $ra, 4($sp)             # guarda o reg $ra da chamada atual da função na stack
    sw $a0, 0($sp)             # guarda o valor de n da chamada atual da função na stack

    beq $a0, $zero, return      # desvia para a label return se n == 0

    addi $a0, $a0, -1           # senão, decrementa n
    jal fatorial                # faz a chamada recursiva da função fatorial

```

Dentro da função fatorial, devido ao fato do cálculo ser realizado de forma recursiva, é importante que algumas informações sejam preservadas para que a computação ocorra de forma adequada. Cada chamada da função fatorial possui seu próprio n (devido a operação recursiva), e seu próprio endereço de retorno, e portanto, esses dados devem ser armazenados na stack. Por isso, após cada chamada da função fatorial, decrementa-se o registrador $\$sp$ (stack pointer), para que esses dois dados sejam armazenados na stack, utilizando a instrução `sw` (store word). Feito isso, é realizada uma instrução de `beq` (branch on equal), para verificar se o n da chamada atual da função fatorial chegou ao caso base ($n == 0$). Se a condição for satisfeita, ou seja, n é igual a 0, desvia-se para a label `return`, onde irá-se iniciar o retorno de todas as chamadas recursivas da função fatorial, para que ocorra de fato a computação do fatorial de n . Caso contrário, significa que ainda não se está no caso base, e portanto, n deve ser decrementado e outra chamada da função fatorial deve ser feita.

```

return:
    lw $ra, 4($sp)             # carrega da stack o endereço de retorno da chamada atual da função fatorial
    lw $a0, 0($sp)             # carrega da stack o n da chamada atual da função fatorial
    addi $sp, $sp, 8           # incrementa o $sp em 8, para retirar os valores carregados da stack

    beq $a0, $zero, caso_base   # desvia se n == 0 (caso base de retorno)
    mul $s0, $s0, $a0           # multiplica $s0 por $a0 (n)

    jr $ra                     # retorna para o endereço de retorno armazenado por $ra

```

Como dito anteriormente, quando n for igual a 0, a execução será desviada para a label `return`, onde começará a se ter os retornos de cada chamada da função fatorial. Dentro da label `return` é feita o carregamento da stack dos dados que foram armazenados (valor de n e o endereço de retorno para a chamada atual da função), para que seja realizada a computação do cálculo. Após esse carregamento dos dados usando duas instruções do tipo `lw` (load word), a stack é incrementada em 8 posições, ou seja, é realizada a remoção dos dados carregados da stack. Após isso, faz-se uma verificação utilizando uma instrução do tipo `beq` (branch on equal), para saber se está no caso onde n é igual a 0. Essa verificação é feita visto que fatorial de 0 é 1. Portanto, não se pode multiplicar o resultado do fatorial (armazenado em $\$s0$) por 0, pois se o valor do resultado for iniciado em 0, todas as multiplicações seguintes para cada retorno da fatorial resultará em 0.

Para lidar com esse problema, dentro da label `caso_base`, é feito o carregamento do valor 1 para o registrador `$s0` (armazena o resultado do fatorial de `n`), sentando caso base ($0! == 1$).

```
caso_base:
    li $s0, 1                # faz $s0 iniciar em 1 (0! == 1)

    jr $ra                  # retorna para o endereço de retorno
```

Após isso, é feito o retorno da execução da chamada de caso base utilizando a instrução `jr` (jump and return). Após isso, a execução irá retornar para a instrução que segue a chamada da função `jal` fatorial, ou seja, irá executar o retorno das outras chamadas da função fatorial. Com isso, cada chamada que precede o caso base irá retornar, multiplicando o `n` de sua chamada com `$s0`, computando o cálculo do fatorial de forma correta. Ao término de todas as chamadas, será realizada uma instrução de `jr $ra` da primeira chamada da função fatorial, que fará com que a execução volte para a instrução que segue a primeira chamada da função fatorial, ou seja, a instrução `j exit`:

```
jal fatorial                # chama a função fatorial
j exit                     # desvia para a label exit
```

Com isso, a execução do programa será desviada para a label `exit`:

```
exit:
    li $v0, 4                # seta a operação de impressão de string
    la $a0, string_result    # passa a string que será impressa para $a0
    syscall                  # faz a impressão da string

    li $v0, 1                # seta a operação de impressão de int
    move $a0, $s0            # move o int a ser impresso para $a0
    syscall                  # faz a impressão do int
```

Nessa label basicamente, será setada a operação de impressão de string, para realizar a impressão da `string_result`, e a impressão de um inteiro, que consiste no resultado do cálculo do fatorial.

Vamos realizar alguns testes, afim de verificar se o programa ocorre como esperado:

Para `n` igual a 0, ou seja, o caso base, tem-se:

```
Digite o valor de n: 0
O fatorial de n é: 1
-- program is finished running (dropped off bottom) --
```

Para n igual a 5, tem-se:

```
Digite o valor de n: 5
O fatorial de n é: 120
-- program is finished running (dropped off bottom) --
```

Para n igual a 10, tem-se:

```
Digite o valor de n: 10
O fatorial de n é: 3628800
-- program is finished running (dropped off bottom) --
```

Portanto, é possível perceber o correto funcionamento do programa.

Exercício 3

Análise do exercício 1

O código do exercício 1 possui um único branch, que é o comparador básico do loop. O branch utilizado é o bgt, uma pseudoinstrução, que desvia se o valor do primeiro registrador foi maior que o segundo registrador. Essa pseudoinstrução é transformada nas instruções slt e bne, de fato:

4194340	0x0109082a	slt \$1,\$8,\$9	16:	bgt \$t1, \$t0, fim fatorial
4194344	0x14200003	bne \$1,\$0,3		

Com base nessa informação, ao executar o código do exercício com o BHT Simulator conectado, com 8 entradas, 1 bit de tamanho da BHT e valor inicial como Not take, obtivemos o seguinte resultado:

```
insira o valor de x: 10
o fatorial de x é: 3628800
-- program is finished running (dropped off bottom) --
```

BHT Simulator, Version 1.0 (Ingo Kofler) X

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	T	TAKE	10	1	90,91
	3	NT	NOT TAKE	0	0	0,00
	4	NT	NOT TAKE	0	0	0,00
	5	NT	NOT TAKE	0	0	0,00
	6	NT	NOT TAKE	0	0	0,00
	7	NT	NOT TAKE	0	0	0,00

Instruction

@ Address

-> Index

Log

```
instruction bne $1,$0,3 at address 0x400028, maps to index 2
branches to address 0x4194360
prediction is: do not take...
branch taken, prediction was incorrect
```

Tool Control

Disconnect from MIPS

O único branch do código foi mapeado para o index 2. Com as definições estabelecidas anteriormente, nota-se um bom desempenho, com muitos acertos na predição. Houve somente 1 erro e 10 acertos, com uma precisão de 90,91%. Há um alto número de predições corretas, pois ao executar o loop principal para calcular o fatorial, o desvio na maioria das vezes não é tomado. O único erro que houve foi na última iteração, onde o desvio foi tomado para sair do loop. Ao aumentar o número de entradas, poucas coisas mudam, afinal há somente um branch. A única mudança é no index que o branch é mapeado. Ao mudar o valor inicial para Take, nota-se uma mudança na predição (o número testado é sempre para calcular o fatorial de 10):

BHT Simulator, Version 1.0 (Ingo Kofler) X

Branch History Table Simulator

of BHT entries BHT history size Initial value

Instruction	Index	History	Prediction	Correct	Incorrect	Precision
	0	T	TAKE	0	0	0,00
	1	T	TAKE	0	0	0,00
	2	T	TAKE	9	2	81,82
	3	T	TAKE	0	0	0,00
	4	T	TAKE	0	0	0,00
	5	T	TAKE	0	0	0,00
	6	T	TAKE	0	0	0,00
	7	T	TAKE	0	0	0,00

Instruction

@ Address

-> Index

Log

```
instruction bne $1,$0,3 at address 0x400028, maps to index 2
branches to address 0x4194360
prediction is: do not take...
branch taken, prediction was incorrect
```

Tool Control

Disconnect from MIPS

Como nas primeiras iterações o desvio não é tomado, ao começar com o valor inicial Taken, contabiliza-se um erro a mais do que na vez passada, totalizando 2 erros e 9 acertos. Logo, o primeiro erro acontece na primeira iteração e o segundo na última iteração, onde é necessário tomar o desvio para sair do laço. Isso gera uma precisão de 81,82%. Ao aumentar o tamanho da BHT para 2 bits, e estabelecer o valor inicial com Not taken, nota-se que o resultado não divergiu dos mostrados anteriormente:

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 TA...	0	0	0,00
	1	NT, NT	NOT TA...	0	0	0,00
@ Address	2	NT, T	NOT TA...	10	1	90,91
	3	NT, NT	NOT TA...	0	0	0,00
	4	NT, NT	NOT TA...	0	0	0,00
	5	NT, NT	NOT TA...	0	0	0,00
	6	NT, NT	NOT TA...	0	0	0,00
-> Index	7	NT, NT	NOT TA...	0	0	0,00

Log

```
instruction bne $1,$0,3 at address 0x400028, maps to index 2
branches to address 0x4194360
prediction is: do not take...
branch taken, prediction was incorrect
```

Tool Control

Disconnect from MIPS Reset Close

Entretanto, ao utilizar 2 bits como tamanho da BHT, e estabelecer o valor inicial com Taken, nota-se uma diferença na precisão, se comparada com os valores anteriores:

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, T	TAKE	0	0	0,00
@ Address	2	NT, T	NOT TAKE	8	3	72,73
	3	T, T	TAKE	0	0	0,00
	4	T, T	TAKE	0	0	0,00
	5	T, T	TAKE	0	0	0,00
	6	T, T	TAKE	0	0	0,00
-> Index	7	T, T	TAKE	0	0	0,00

Log

```
instruction bne $1,$0,3 at address 0x400028, maps to index 2
branches to address 0x4194360
prediction is: do not take...
branch taken, prediction was incorrect
```

Tool Control

Disconnect from MIPS Reset Close

Agora, ocorreu um erro a mais do que quando estava na mesma situação, mas com tamanho da BHT de 1 bit. Houveram 8 acertos, 3 erros e uma precisão de 72,73%. Isso ocorre pois a predição foi feita de maneira errada durante, além da última iteração, nas DUAS primeiras iterações.

BHT Simulator, Version 1.0 (Ingo Kofler)

Branch History Table Simulator

of BHT entries: 8 BHT history size: 2 Initial value: TAKE

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

Instruction: bne \$1,\$0,3
@ Address: 0x400028
-> Index: 2

Log
instruction bne \$1,\$0,3 at address 0x400028, maps to index 2
branches to address 0x4194360
prediction is: take...
branch not taken, prediction was incorrect

Tool Control
Disconnect from MIPS Reset Close

A imagem anterior representa a execução da simulação após a primeira iteração. Como o valor inicial era Take e o desvio não foi tomado, a predição foi incorreta. Assim, o histórico ficou como T, NT, logo, a próxima predição vai ser um Taken. Como o desvio não vai ser tomado de novo, a predição vai estar novamente incorreta:

BHT Simulator, Version 1.0 (Ingo Kofler)

Branch History Table Simulator

of BHT entries: 8 BHT history size: 2 Initial value: TAKE

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

Instruction: bne \$1,\$0,3
@ Address: 0x400028
-> Index: 2

Log
instruction bne \$1,\$0,3 at address 0x400028, maps to index 2
branches to address 0x4194360
prediction is: take...
branch not taken, prediction was incorrect

Tool Control
Disconnect from MIPS Reset Close

Agora, como o histórico ficou como NT, NT, a próxima previsão será um Not taken e começará a ser acertada nas próximas iterações. Somente na última iteração, onde é necessário tomar o desvio para sair do laço, a previsão fica incorreta novamente:

Branch History Table Simulator

of BHT entries: 8 BHT history size: 2 Initial value: TAKE

Instruction	Index	History	Prediction	Correct	Incorrect	Precision
bne \$1,\$0,3	0	T, T	TAKE	0	0	0,00
	1	T, T	TAKE	0	0	0,00
@ Address	2	NT, T	NOT TAKE	0	3	72,73
0x400028	3	T, T	TAKE	0	0	0,00
-> Index	4	T, T	TAKE	0	0	0,00
2	5	T, T	TAKE	0	0	0,00
	6	T, T	TAKE	0	0	0,00
	7	T, T	TAKE	0	0	0,00

Log

instruction bne \$1,\$0,3 at address 0x400028, maps to index 2
 branches to address 0x4194360
 prediction is: do not take...
 branch taken, prediction was incorrect

Tool Control

Disconnect from MIPS Reset Close

Com base nesses fatos pode-se afirmar que a solução se mostrou com melhor desempenho quando o valor inicial é Not taken, independentemente do tamanho da BHT, pois há um alto índice de acerto nas previsões (90,91%). Alterando o valor inicial para Taken, a precisão diminui um pouco (81,82%), e alterando o tamanho da BHT para 2 bits, ela diminuiu mais ainda (72,73%).

Análise do exercício 2

O código do exercício 2 utiliza duas instruções beq (branch on equal), uma utilizada dentro da função fatorial, que verifica se já se chegou no caso base da função fatorial ($n == 0$), e outra é utilizada dentro da label return, para verificar também se está no caso base ($n == 0$).

```

fatorial:
    addi $sp, $sp, -8
    sw $ra, 4($sp)
    sw $a0, 0($sp)

    beq $a0, $zero, return

    addi $a0, $a0, -1
    jal fatorial

return:
    lw $ra, 4($sp)
    lw $a0, 0($sp)
    addi $sp, $sp, 8

    beq $a0, $zero, caso_base
    mul $s0, $s0, $a0

    jr $ra

```

Com isso, ao executar o código do exercício 2 com o BHT Simulator conectado, com 8 entradas, 1 bit de tamanho da BHT e valor inicial como Not take, obtivemos o seguinte resultado:

```

insira o valor de x: 10
o fatorial de x é: 3628800
-- program is finished running (dropped off bottom) --

```

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	9	2	81,82
	4	NT	NOT TAKE	0	0	0,00
	5	T	TAKE	10	1	90,91
	6	NT	NOT TAKE	0	0	0,00
	7	NT	NOT TAKE	0	0	0,00

Log

```

instruction beq $4,$0,2 at address 0x40004c, maps to index 3
branches to address 0x4194392
prediction is: do not take...
branch not taken, prediction was correct

```

Tool Control

Disconnect from MIPS Reset Close

O primeiro branch do código foi mapeado para o index 5 e o segundo branch do código foi mapeado para o index 3. Usando as configurações anteriores, para o primeiro beq, houve uma predição muito boa, visto que houveram 10 acertos e apenas 1 erro. Ou seja, a precisão da predição usando o valor inicial como Not take para uma BHT com 1 bit foi de 90,91%. Isso ocorre porque, para o primeiro branch, deseja-se que ele desvie apenas quando o x for igual a 0. Como x inicia em 10, usando Not take como valor inicial, ele seguirá acertando até quando x for igual a 1. Quando x for 0, ele irá contabilizar um erro, porque ele irá prever um Not take, quando na verdade deveria prever um take.

Já para o segundo beq que foi mapeado para o index 3, houve uma boa precisão, mas inferior a precisão do primeiro beq, visto que no segundo beq houveram 9 acertos e 2 erros, ou seja, 1 erro a mais que no primeiro beq. Assim, usando o valor inicial como Not take para uma BHT com 1 bit foi de 81,82%. Isso ocorre porque o segundo beq é usado para verificar se está no caso base onde x é igual a 0. Como quando houver um desvio para a label return, onde o segundo beq está, x será igual a 0, e o beq deverá ser tomado. No entanto, como o valor inicial será not take, ele irá errar a primeira vez. Na próxima iteração, x será igual a 1, e o beq não deve ser tomado. No entanto, como na iteração anterior ele devia ter desviado (take), nessa iteração a predição será de take, e então, ele contabiliza o segundo erro. Nas próximas iterações até o término do programa ele não deve desviar, visto que o beq é só para o caso base ($x == 0$), e como na iteração anterior ele errou a predição, dizendo que era um take ao invés de um not take, a predição será de um not take, que se repetirá até o término do programa. Assim, serão contabilizados 9 acertos consecutivos.

Com isso, percebe-se que a utilização do BHT de 1 bit com valor inicial not take mostrou uma precisão média de 86,37%, que é uma boa precisão.

Ao mudar o valor inicial para take, percebe-se uma diferença na BHT de 1 bit:

Branch History Table Simulator

of BHT entries: 8 BHT history size: 1 Initial value: TAKE

Instruction	Index	History	Prediction	Correct	Incorrect	Precision
	0	T	TAKE	0	0	0,00
	1	T	TAKE	0	0	0,00
	2	T	TAKE	0	0	0,00
	3	NT	NOT TAKE	10	1	90,91
	4	T	TAKE	0	0	0,00
	5	T	TAKE	9	2	81,82
	6	T	TAKE	0	0	0,00
	7	T	TAKE	0	0	0,00

Log

```

instruction beq $4,$0,2 at address 0x40004c, maps to index 3
branches to address 0x4194392
prediction is: do not take...
branch not taken, prediction was correct
  
```

Tool Control

Disconnect from MIPS Reset Close

Houve uma inversão na precisão do primeiro beq (index 5) com o segundo beq (index 3) se comparado ao caso anterior onde o valor inicial era not take. E isso já era previsível.

Para o primeiro beq, houveram 9 acertos e 2 erros, contabilizando uma precisão de 81,82%. Isso ocorre porque, como o valor inicial agora é take, no primeiro beq só se deve tomar o desvio no caso base ($x == 0$), e então ele irá na primeira iteração, predizer que o beq deve ser tomado, quando na verdade não deve, e portanto, irá contabilizar o primeiro erro. Depois, seguirá acertando, visto que as predições serão de not take, até chegar no caso base, onde deve tomar o desvio. No entanto, como a predição será not take, irá contabilizar um erro na última iteração, visto que a predição correta seria um take.

Já para o segundo beq, houveram 10 acertos e 1 erro, contabilizando uma precisão de 90,91%. Isso ocorre porque, na primeira iteração da label return, onde o segundo beq está, ele deve desviar apenas no caso base. Como na primeira iteração do return se está no caso base, e a predição inicial é de take, ele irá acertar a primeira iteração. Entretanto, a partir da segunda iteração ele não deve desviar (not take). Porém, como houve um acerto na primeira iteração usando take, na segunda iteração ele terá uma predição de take, mas não irá desviar, contabilizando assim 1 erro. Na terceira iteração, como houve um erro de predição, para essa iteração ele irá predizer not take, e acertará, predizendo e acertando também as próximas iterações até o término do programa. Assim, contabiliza-se 10 acertos em sequência.

Com isso, percebe-se que a utilização do BHT de 1 bit com valor inicial take mostrou uma precisão média de 86,37%, que é uma boa precisão.

Para o caso com o BHT com 8 entradas, 2 bits de tamanho e valor inicial como Not take, obtivemos o seguinte resultado:

BHT Simulator, Version 1.0 (Ingo Kofler)

Branch History Table Simulator

of BHT entries: 8 BHT history size: 2 Initial value: NOT TAKE

Instruction	Index	History	Prediction	Correct	Incorrect	Precision
	0	NT, NT	NOT TAKE	0	0	0,00
	1	NT, NT	NOT TAKE	0	0	0,00
	2	NT, NT	NOT TAKE	0	0	0,00
	3	NT, NT	NOT TAKE	10	1	90,91
	4	NT, NT	NOT TAKE	0	0	0,00
	5	NT, T	NOT TAKE	10	1	90,91
	6	NT, NT	NOT TAKE	0	0	0,00
	7	NT, NT	NOT TAKE	0	0	0,00

Log

Instruction beq \$4,\$0.2 at address 0x40004c, maps to index 3
 branches to address 0x4194392
 prediction is: do not take...
 branch not taken, prediction was correct

Tool Control

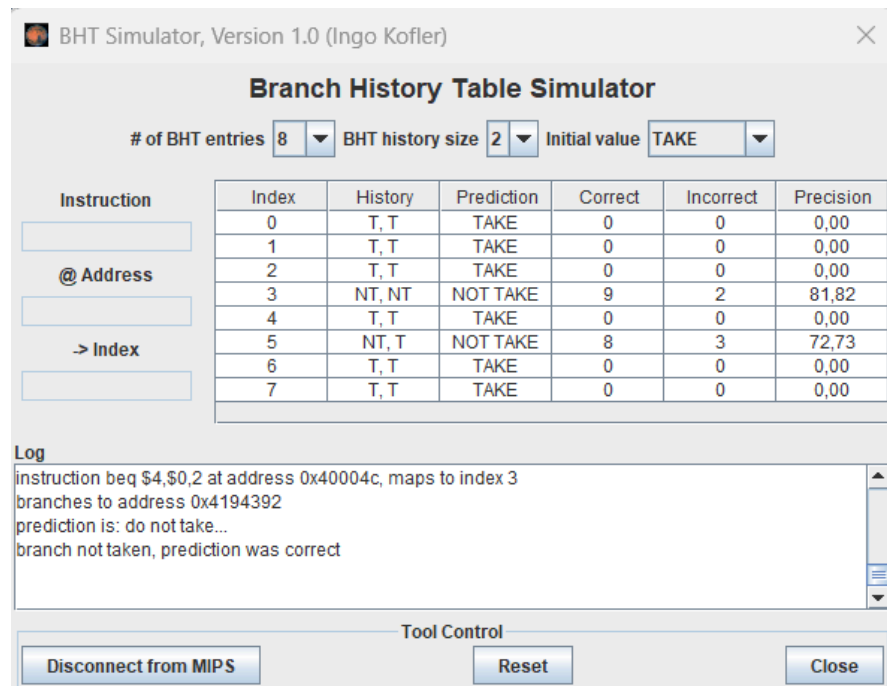
Disconnect from MIPS Reset Close

Tanto para o primeiro beq (index 5) quanto o segundo beq (index 3), tiveram 10 acertos e 1 erro, contabilizando uma precisão de 90,91%, usando o valor inicial not take em uma BHT de 2 bits. Para o primeiro beq, só deve ser tomado o desvio no caso base. Com isso, todas as predições serão de not take, que serão acertadas 10 vezes em sequência, até que x seja igual a 0. Quando x for zero (caso base), o desvio será tomado, mas como as predições anteriores eram de not take, a da iteração do caso base também será, e portanto, será contabilizado 1 erro.

Já no segundo beq, só se deve tomar o desvio também no caso base. Com isso, na primeira iteração o desvio deve ser tomado. Como se está utilizando o valor inicial como not take, na primeira iteração será contabilizado um erro. Já nas próximas iterações, não se deve mais tomar o desvio. Como na primeira iteração houve um erro visto que a predição correta seria de take, mas houve a predição de not take, como se está utilizando uma BHT de 2 bits, a predição para a próxima iteração será de not take, que será acertada, e se manterá como not take até o término do programa, pois não se deve mais desviar. Com isso, serão contabilizados 10 acertos em sequência no segundo beq.

Com isso, percebe-se que a utilização do BHT de 2 bits com valor inicial not take mostrou uma precisão média de 90,91%, que é uma ótima precisão.

Para o caso com o BHT com 8 entradas, 2 bits de tamanho e valor inicial como take, obtivemos o seguinte resultado:



Branch History Table Simulator

of BHT entries: 8 BHT history size: 2 Initial value: TAKE

Instruction	Index	History	Prediction	Correct	Incorrect	Precision
	0	T, T	TAKE	0	0	0,00
	1	T, T	TAKE	0	0	0,00
	2	T, T	TAKE	0	0	0,00
	3	NT, NT	NOT TAKE	9	2	81,82
	4	T, T	TAKE	0	0	0,00
	5	NT, T	NOT TAKE	8	3	72,73
	6	T, T	TAKE	0	0	0,00
	7	T, T	TAKE	0	0	0,00

Log

```
instruction beq $4,$0,2 at address 0x40004c, maps to index 3
branches to address 0x4194392
prediction is: do not take...
branch not taken, prediction was correct
```

Tool Control

Disconnect from MIPS Reset Close

Para o primeiro beq (index 5) houveram 8 acertos e 3 erros, que contabilizaram uma precisão de 72,73%. Isso ocorre porque, para o primeiro beq, o desvio só deve ser tomado no caso base ($x == 0$), que ocorre na última iteração da label fatorial. No entanto, como o valor inicial é take, na primeira iteração ele irá

predizer take, quando a predição correta seria not take, e portanto, irá contabilizar 1 erro. Como se está usando uma BHT de 2 bits, como houve erro na predição da primeira iteração, ainda sim a predição da segunda iteração será de take, e portanto será contabilizado mais 1 erro. A partir da terceira, será feito a predição de not take, que será correta até a penúltima iteração, contabilizando assim 8 acertos consecutivos. Já na última iteração (caso base), será feito a predição de not take, quando na verdade será feito o desvio. Com isso, será contabilizado mais um erro, totalizando 3 erros no total para o primeiro beq.

Para o segundo beq (index 3) houveram 9 acertos e 2 erros, que contabilizaram uma precisão de 81,82%. Isso ocorre porque, para o segundo beq, o desvio só deve ser tomado no caso base ($x == 0$), que ocorre na primeira iteração da label return. Assim, como o valor inicial é take, na primeira iteração ele irá predizer take, e irá contabilizar o primeiro acerto. A partir da segunda iteração, não se deve mais desviar no segundo beq. No entanto, como na primeira iteração a predição foi de take, para a segunda iteração ele irá predizer take novamente, contabilizando assim 1 erro. Na terceira, o mesmo irá ocorrer, visto que se está em uma BHT de 2 bits. Assim, será contabilizado mais 1 erro. A partir da 4 iteração, a predição será de not take, que é a predição correta, e então serão contabilizados 8 acertos consecutivos até o término do programa.

Com isso, percebe-se que a utilização do BHT de 2 bits com valor inicial take mostrou uma precisão média de 77,28%, que é uma precisão ok.

Conclusões Finais

Através das análises realizadas dos dois programas que calculam o fatorial de um número com implementações diferentes (sem recursão e com recursão), pode-se perceber que ambas as implementações possuem bons desempenhos, conforme condições específicas. Pelo fato da primeira implementação possuir apenas um beq, ela tende a ter uma média de predições melhores que a implementação recursiva, que possui 2 beq. Entretanto, pode-se perceber os seguintes resultados:

- Implementação do fatorial não recursivo:
 - BHT de 1 bit com valor inicial not take: Precisão de 90,91%.
 - BHT de 1 bit com valor inicial take: Precisão de 81,82%.
 - BHT de 2 bits com valor inicial not take: Precisão de 90,91%.
 - BHT de 2 bits com valor inicial take: Precisão de 72,73%.
 - Média geral de precisão: 84,09%.
- Implementação do fatorial recursivo:
 - BHT de 1 bit com valor inicial not take: Precisão média de 86,37%.
 - BHT de 1 bit com valor inicial take: Precisão média de 86,37%.

- BHT de 2 bits com valor inicial not take: Precisão média de 90,91%.
- BHT de 2 bits com valor inicial take: Precisão média de 77,28%.
- Média geral de precisão: 85,23%.

Olhando as médias gerais das duas implementações, pode-se perceber que elas possuem uma precisão média muito próxima e boa, com a implementação não recursiva sendo um pouco melhor de um modo geral. Porém, com base em casos mais específicos, as melhores implementações foram:

- Implementação não recursiva: BHT de 1 bit com valor inicial not take: Precisão de 90,91%.
- Implementação não recursiva: BHT de 2 bits com valor inicial not take: Precisão de 90,91%.
- Implementação recursiva: BHT de 2 bits com valor inicial not take: Precisão média de 90,91%.

Portanto, a fim de se obter o melhor desempenho, o ideal seria utilizar uma dessas 3 implementações, visto que cada uma delas possui precisão de 90,91%.