UNIVERSIDADE FEDERAL DE SANTA CATARINA
EDUARDO BISCHOFF GRASEL - 22200355

# Relatório Organização de computadores

Florianópolis - SC
2024

## Objetivo Geral

Entender como funciona e como implementar a Branch History Table (BHT) para realizar a predição de saltos, ao final do Exercício devemos ser capazes de compreender o funcionamento dessa alternativa e como a implementação deve ser realizada a fim de tornar o programa mais otimizado.

**Exercício 1:**

Começamos declarando o .data, que nesse caso vai ser apenas a mensagem a ser exibida na tela para a recuperação do valor digitado pelo usuário, e a declaração da BHT, como esse é um programa simples, iniciamos ele com apenas uma posição que indicará o status do desvio.

```
.data
        msg: .asciiz "Digite um número: "
        # branch history table
        branch_history: .space 1
```

No .text, iniciamos apenas pedindo o valor do fatorial e inicializando os registradores auxiliares para o cálculo, uma parte padrão e genérica, portanto vamos pular para o loop, o principal objetivo dele é calcular o fatorial sempre multiplicando o resultado pelo resultado * aux e depois adiciona um ao aux e repete o ciclo, até o condicional ser satisfeita. Para a aplicação do BHT foi utilizado uma lógica na qual antes do desvio, carrega-se a BHT e depois do desvio atualiza a tabela com o valor taken ou not taken, como a maioria das vezes o desvio não será tomado, ele executará de forma normal até o último loop, onde o desvio será pego.

```
        loop:  #fatorial
                # verifica branch history table
                lb $t2, branch_history($zero)  # carrega a predição
                beq $t2, 1, branch_taken     # Se o desvio for pego, vai para branch taken


                # Branch not taken
                mul $s0, $s0, $t1
                addi $t1, $t1, 1
                li $t2, 0
                j branch_continue

branch_taken:
        mul $s0, $t1, $s0
        addi $t1, $t1, 1
        li $t2, 1



branch_continue:
        # atualizar branch history table
        sb $t2, branch_history($zero)  # guarda a saida atual (taken ou not taken)

        bgt $t1, $t0, fim

        j loop

fim:
        li $v0, 1
        move $a0, $s0
        syscall
```

```
Digite um número: 5
120
-- program is finished running (dropped off bottom) --


Reset: reset completed.

Digite um número: 6
720
-- program is finished running (dropped off bottom) --


Reset: reset completed.

Digite um número: 7
5040
-- program is finished running (dropped off bottom) --
```

**Exercício 2:**

Inicialmente o programa seguirá o mesmo padrão do anterior, apenas mudando a forma de implementação para uma recursiva. Tendo isso em vista iremos utilizar a predição de salto dentro da chamada da função, segue abaixo o código antes base fora da função fatorial recursiva:

```
.text
        li $v0, 4
        la $a0, msg
        syscall
        li $v0, 5
        syscall
        move $a0, $v0

        jal fatorial
        move $t0, $v0

        li $v0, 4
        la $a0, msg_final
        syscall
        li $v0, 1
        move $a0, $t0
        syscall

        li $v0, 10
        syscall
```

```
data
        msg: .asciiz "Digite um valor: "
        msg_final: .asciiz "O resultado do fatorial é: "

        # Branch History Table com 2 bits para prever o salto em fatorial
        bht: .space 1
```

Dentro da função fatorial, iniciamos a lógica de previsão de saltos, carregando os bits necessários e isolando-os, em seguida decodificamos os estados da BHT e guardamos esse valor em registradores.

```
# Lógica de previsão de saltos
lb $t1, bht # Carrega o valor do histórico de saltos
lb $t2, bht # Carrega novamente para verificar o segundo bit do histórico
andi $t1, $t1, 0x02 # Isola o segundo bit do histórico

# Decodificação dos estados da BHT
li $t3, 1 # Estado 1: 01 - Prever sem salto
li $t4, 2 # Estado 2: 10 - Prever salto
li $t5, 3 # Estado 3: 11 - Prever salto
beq $t1, $zero, skip_predict # Estado 0: 00 - Prever sem salto

beq $t2, $zero, no_jump_predict # Se o segundo bit for 0, prever sem salto
```

Agora que já temos os estados, dividimos as partes para o cálculo recursivo do fatorial de acordo com o estado: Caso tenha salto, caso não tenha salto, caso seja passado.

```
jump_predict:
b Li

no_jump_predict:
        li $v0, 1
        add $sp, $sp, 8
        jr $ra

skip_predict:

slti $t0, $a0, 1
beq $t0, $zero, Li

li $v0, 1
add $sp, $sp, 8
jr $ra
```

```
Li:
        subi $a0, $a0, 1
        jal fatorial

        lw $ra, 0($sp)
        lw $a0, 4($sp)
        add $sp, $sp, 8
        mul $v0, $a0, $v0
```

Enfim falta somente a atualização do histórico de salto, verificando se os saltos foram ou não foram pegos, atualizando os bits de previsão.

```
# Atualização do histórico de saltos
slt $t0, $zero, $v0 # Verifica se o resultado é zero ou não
beq $t0, $zero, update_bht_jump # Se não for zero, atualiza para prever salto
li $t6, 1 # Se for zero, atualiza para prever sem salto
sb $t6, bht # Atualiza o bit de previsão com o resultado atual da condição
jr $ra

update_bht_jump:
        andi $t1, $t1, 0x01 # Isola o primeiro bit do histórico
        sll $t1, $t1, 1 # Desloca o primeiro bit para a esquerda para fazer espaço para o segundo bit
        ori $t1, $t1, 0x01 # Define o primeiro bit como 1
        sb $t1, bht # Atualiza o histórico de saltos para prever salto
        jr $ra
```

```
Digite um valor: 2
O resultado do fatorial é: 2
-- program is finished running --


Reset: reset completed.

Digite um valor: 3
O resultado do fatorial é: 6
-- program is finished running --


Reset: reset completed.

Digite um valor: 4
O resultado do fatorial é: 24
-- program is finished running --
```

```
Digite um valor: 5
O resultado do fatorial é: 120
-- program is finished running --


Reset: reset completed.

Digite um valor: 6
O resultado do fatorial é: 720
-- program is finished running --


Reset: reset completed.

Digite um valor: 7
O resultado do fatorial é: 5040
-- program is finished running --
```

## Conclusão

O maior desafio desse trabalho foi buscar as informações necessárias, pois falta de informações de predição de saltos e pipeline em assembly mips na internet. Precisei ir fazendo testes por testes, e como o mips não tem uma ferramenta específica para verificar predição de salto, foi muito difícil descobrir como a predição deveria funcionar ou mesmo se estava funcionando, no final fiz duas abordagens diferentes para cada exercício que eu acredito estarem corretas, a segunda sendo muito mais complexa do que a primeira.