UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
INE-DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

INE5411 - ORGANIZAÇÃO DE COMPUTADORES I

PEDRO TAGLIALENHA (22203674)
VITOR PRAXEDES CALEGARI (22200379)

RELATÓRIO LABORATÓRIO 6

FLORIANÓPOLIS
2023

1 INTRODUÇÃO....3
2 CÓDIGOS DESENVOLVIDOS....4
2.1 EXERCÍCIO 1....4
2.2 EXERCÍCIO 2....5
3 ANÁLISE DE DESEMPENHO....7
3.1 TESTES DO EXERCÍCIO 1....7
3.2 TESTES DO EXERCÍCIO 2....8
4 CONCLUSÃO....10

1 INTRODUÇÃO

O laboratório 6 de Organização de Computadores envolveu a implementação de programas em linguagem Assembly do MIPS, com o objetivo de calcular o fatorial de um número. Foram propostos dois exercícios distintos para explorar diferentes abordagens no cálculo do fatorial. O primeiro exercício demandou a implementação de um programa que calcula o fatorial sem o uso de procedimentos, enquanto o segundo exercício introduziu o conceito de recursão por meio de uma função fatorial. Esses programas foram desenvolvidos para serem executados no simulador MARS.

Além disso, uma etapa adicional de análise de desempenho foi incluída no laboratório. Foi realizada a simulação de Branch History Table (BHT) com variação nos parâmetros, como a quantidade de entradas (BHT entries), o tamanho da BHT (1 e 2 bits) e o valor inicial. A análise do desempenho dos programas implementados nos exercícios 1 e 2 sob diferentes configurações desses parâmetros é crucial para compreender qual das implementações apresenta um melhor desempenho em diferentes condições. Essa análise permitirá identificar quais abordagens são mais eficientes no contexto do cálculo do fatorial em linguagem Assembly do MIPS.

2 CÓDIGOS DESENVOLVIDOS

Na seção a seguir, serão apresentados e detalhados os códigos referentes aos exercícios 1 e 2 deste laboratório. O exercício 1 demonstra a implementação do cálculo do fatorial sem o uso de procedimentos, enquanto o exercício 2 introduz o conceito de recursão por meio de uma função fatorial

2.1 EXERCÍCIO 1

Figura 1 - Fatorial sem procedimentos

```
.data

.text
    # ============================
    #  Inicializacao do programa
    # ============================

    li    $v0, 5                # Le do terminal o numero que sera utilizado
    syscall                     # para calcular o fatorial

    move     $s0, $v0     # Salva o número lido no registrador $s0

    beq    $s0, 0, zero          # Caso o número lido seja 0 vamos para o tratamento

    move    $s1, $s0      # Inicializa o registrador $s1 com o valor de $s0
                                # para ser utilizado como "variavel de controle" do
                                # laço

loop:
    # ============================
    #   Loop do calculo
    # ============================

    beq $s1, 1, exit      # Verifica se deve sair do laco
    subi $s1, $s1, 1      # Subtrai 1 da "variavel de controle"

    mul $s0, $s0, $s1     # Multiplica o resultado atual($s0) pela variavel
                                # de controle do laco
    j loop                      # retorna para "loop"

zero:
    # ============================
    #   Tratamento do caso 0
    # ============================

    addi     $s0, $zero, 1        # Coloca 1 como o resultado

exit:
    # ============================
    #   Saida do programa
    # ============================
    li     $v0, 1                # Imprime o resultado no terminal
    move    $a0, $s0      #
    syscall                     #
```

Fonte: Elaborado pelos autores(2023)

O código inicia com a leitura de um número a partir do terminal e, em seguida, realiza o cálculo do fatorial desse número. O número lido é armazenado em $s0, e um laço é implementado para calcular o fatorial. O laço é controlado pelo registrador $s1, que é decrementado a cada iteração até que atinja 1. Durante o loop, o valor de $s0 é multiplicado pelo valor de $s1. Caso o número seja 0, o código atribui 1 ao resultado, e no final, o resultado é impresso no terminal.

2.2 EXERCÍCIO 2

Figura 2 - Fatorial recursivo

```
.data

.text

main:
    # ====================================
    #           Funcao principal
    # ====================================
    # Recebe numero a ser calculado fatorial,
    # chama procedimento que realiza o cal-
    # culo e então imprime no terminal.
    #

    li    $v0, 5                # Receba via teclado o valor do número a
    syscall                     # ser calculado o fatorial
    move     $a0, $v0     #

    jal     fatorial      # Chama fatorial

    move    $s0, $v0      # Salva valor retornado pelo fatorial

    li     $v0, 1                # Imprime resultado no terminal
    move    $a0, $s0      #
    syscall                     #

    li    $v0, 10               # Encerra programa
    syscall                     #

fatorial:

    # ====================================
    #           Procedimento fatorial
    # ====================================
    # Calcula o fatorial de um numero forne-
    # cido em $a0.
    #
    # argumentos:
    #   $a0 = Valor para calcular o fatorial
    #
    # retorno:
    #   $v0 = Valor do fatorial calculado

    addi     $sp, $sp, -8         # Salva $s0 e $ra na pilha
    sw    $s0, 4($sp)     #
    sw    $ra, 8($sp)     #
```

```
    move    $s0, $a0

    beq    $s0, 0, igualZero    # Se é igual a zero retorna 1

    bne    $s0, 1, diferenteDeUm    # Se é igual a 1, fatorial retorna 1
                                # e volta executando as mults
igualZero:
    li    $v0, 1                # Carrega o registrador de retorno($v0) com 1

    lw    $s0, 4($sp)     # Retorna $s0 e $ra da pilha
    lw    $ra, 8($sp)     #
    addi    $sp, $sp, 8           #

    jr    $ra             # Retorna ao procedimento chamador

diferenteDeUm:

    subi    $a0, $s0, 1

    jal     fatorial      # Chama fatorial

    mul    $v0, $s0, $v0          # Multiplica X * fatorial(X-1)

    lw    $s0, 4($sp)     # Retorna $s0 e $ra da pilha
    lw    $ra, 8($sp)     #
    addi    $sp, $sp, 8           #

    jr    $ra             # Retorna ao procedimento chamador
```

Fonte: Elaborado pelos autores(2023)

O código é dividido em duas partes: a função principal "main" e o procedimento fatorial. A função principal inicia lendo um número do terminal, chama o procedimento fatorial para calcular o fatorial desse número e, em seguida, imprime o resultado no terminal. A função fatorial é um procedimento que recebe um argumento em $a0 e calcula o fatorial desse número.

Dentro do procedimento fatorial, há uma verificação para tratar os casos em que o número é igual a 0 ou igual a 1. Se o número for igual a 0 ou 1, o procedimento retorna 1. Caso contrário, ele chama a si mesmo recursivamente para calcular o fatorial do número anterior e multiplica o resultado pelo valor atual, até que o fatorial completo seja calculado.

3 ANÁLISE DE DESEMPENHO

A seguinte seção compara os códigos dos exercícios 1 e 2, que calculam o fatorial. A análise investiga como diferentes implementações afetam a eficiência do programa, variando parâmetros como BHT, tamanho da BHT e valor inicial. Isso ajuda a identificar qual abordagem é mais eficaz sob diferentes condições, aprofundando nosso entendimento do código Assembly do MIPS.

Vale ressaltar que os testes foram todos realizados calculando o fatorial do número 10, essa informação foi relevante para calcularmos a precisão que os programas apresentaram. Em adicional, para os programas desenvolvidos, maiores entradas resultaram em um número de precisão ainda maior, visto que a quantidade de erros se manteria igual e a quantidade de acertos cresceria devido a natureza de “loop” dos programas.

3.1 TESTES DO EXERCÍCIO 1

A análise dos testes no Exercício 1 revelou que a variação na quantidade de entradas da Branch History Table (BHT) não impactou o desempenho do programa, uma vez que o código contém apenas dois branches. Sob a técnica "NotTaken," a precisão de acerto foi de 90,9% em todas as configurações da BHT, uma vez que o programa cometeu um único erro ao sair do laço.

No caso da técnica "Taken," o programa apresentou um mínimo de três erros, um ao verificar o valor zero, outro ao entrar no laço e um terceiro ao sair do laço no final das iterações. Quando utilizando histórico de 2 bits, um erro adicional foi esperado, pois o programa precisou errar duas vezes antes de entrar no padrão NT,NT, e somente então "entendeu" que estava dentro do laço. A precisão continuou em torno de 72,72% para configurações de 1 bit e diminuiu para cerca de 63,63% com 2 bits.

Tabela 1 - Dados dos testes do exercício 1

Técnica	Histórico	Configuração BHT	Acertos	Erros	Precisão
NotTaken	1 bit	BHT 8	10	1	90,9%

NotTaken	1 bit	BHT 16	10	1	90,9%
NotTaken	1 bit	BHT 32	10	1	90,9%
NotTaken	2 bits	BHT 8	10	1	90,9%
NotTaken	2 bits	BHT 16	10	1	90,9%
NotTaken	2 bits	BHT 32	10	1	90,9%
Taken	1 bit	BHT 8	8	3	72,72%
Taken	1 bit	BHT 16	8	3	72,72%
Taken	1 bit	BHT 32	8	3	72,72%
Taken	2 bits	BHT 8	7	4	63,63%
Taken	2 bits	BHT 16	7	4	63,63%
Taken	2 bits	BHT 32	7	4	63,63%

Fonte: Elaborado pelos autores(2023)

Esses resultados apontam para a estabilidade do desempenho do programa em relação à variação de parâmetros da BHT, uma vez que as mudanças na quantidade de entradas tiveram impacto limitado devido à simplicidade do código. Nesse caso, nota-se que a escolha de parâmetros mais adequada para obter o maior desempenho era a combinação da técnica NotTaken, com 8 entradas na BHT e apenas um bit de tamanho, visto que essa opção foi a que resultou em menos erros de predição.

3.2 TESTES DO EXERCÍCIO 2

O teste no Exercício 2 revelou algumas características interessantes em relação à técnica "NotTaken" e "Taken" ao calcular o fatorial em Assembly do MIPS. No teste "teste 0", utilizando a técnica “NotTaken”, o programa sempre acertou a previsão, a menos que o input fosse igual a 0, o que resultaria em um erro. Para o teste diferente de um, utilizando a técnica “NotTaken”, o programa sempre acertou a previsão quando o input era igual a 1. No entanto, caso o input fosse maior que 1, o programa cometeria dois erros.

Tabela 2 - Dados dos testes do exercício 2

Técnica	Histórico	Configuração BHT	Acertos teste 0	Acertos no diferente de 1	Erros no teste 0	Erros no diferente de 1	Precisão
NotTaken	1 bit	BHT 8	10	8	0	2	90%
NotTaken	1 bit	BHT 16	10	8	0	2	90%
NotTaken	1 bit	BHT 32	10	8	0	2	90%
NotTaken	2 bits	BHT 8	10	7	0	3	85%
NotTaken	2 bits	BHT 16	10	7	0	3	85%
NotTaken	2 bits	BHT 32	10	7	0	3	85%
Taken	1 bit	BHT 8	9	9	1	1	90%
Taken	1 bit	BHT 16	9	9	1	1	90%
Taken	1 bit	BHT 32	9	9	1	1	90%
Taken	2 bits	BHT 8	8	9	2	1	85%
Taken	2 bits	BHT 16	8	9	2	1	85%
Taken	2 bits	BHT 32	8	9	2	1	85%

Fonte: Elaborado pelos autores(2023)

Novamente, a variação no número de entradas da Branch History Table (BHT) mostrou-se irrelevante, uma vez que o programa faz uso de apenas 2 branches. A precisão cresceu à medida que aumentamos o valor fornecido ao programa, pois sempre houve apenas dois erros (para inputs maiores que 1). Nesse caso, nota-se que a escolha de parâmetros mais adequada para obter o maior desempenho era a combinação de 8 entradas na BHT, apenas um bit de histórico e ambas as técnicas apresentaram o mesmo números de erros previstos, o que significa que ambas apresentam desempenho muito semelhantes.

4 CONCLUSÃO

Em conclusão, os testes realizados evidenciaram que a variação no número de entradas da Branch History Table (BHT) não teve impacto significativo no desempenho de nossos programas. Isso se deve ao fato de que ambos os programas tinham apenas duas instruções de branch, o que limitou a influência da diferença no número de entradas da BHT em suas previsões.

Ao alterar a BHT de 1 para 2 bits, quando houve diferença, ela se resumiu ao fato de que o preditor errou uma vez a mais para "entender" o comportamento que deveria adotar. Nosso programa de cálculo de fatorial era baseado em loops, onde a condição para sair do laço determinava qual técnica (Taken ou NotTaken) seria mais adequada. No primeiro programa, usamos uma instrução beq para a saída, o que implicava que ela só deveria ser tomada uma única vez, tornando o "not taken" a melhor opção.

No exercício 2, seguimos um raciocínio semelhante, mas devido à recursão, testamos sempre o caso 0, o que tornou as técnicas "taken" e "not taken" equivalentes, mesmo com números maiores como entrada para o programa. Portanto, os resultados reforçaram a ideia de que a escolha da técnica de predição de branch depende fortemente da estrutura do código e das condições de saída dos loops, destacando a importância da análise do contexto de uso em decisões de otimização de desempenho.