Laboratório 04

Simulação BHT

Alunas: Jéssica Regina dos Santos e Myllena Correa
Disciplina: Organização de Computadores I
Professor: Marcelo Daniel Berejuck
Entrega: 14/05/25

Introdução

Este relatório descreve a implementação de dois programas em linguagem Assembly MIPS para cálculo do fatorial de um número. Ambos os códigos foram desenvolvidos para execução no simulador MARS.

Como parte da análise comparativa, avaliamos o desempenho das soluções utilizando o Simulador BHT (Branch History Table), com variação de parâmetros para investigar seu impacto no comportamento dos algoritmos.

Objetivos:

- Mensurar as diferenças de eficiência entre os métodos iterativo e recursivo;
- Investigar como diferentes configurações da BHT impactam cada implementação;
- Identificar em quais cenários cada abordagem se mostra mais vantajosa.

Questão 1

Este programa calcula o fatorial de um número inteiro não negativo de forma iterativa, utilizando uma abordagem com loop.

1. Segmento de Dados

```
2   .data
3
4   valor_n: .asciiz "Digite o valor de n: " #string prompt 1
5   resultado_fatorial: .asciiz "n! = "  #string prompt 2
```

Este bloco define as duas strings usadas para exibir mensagens ao usuário.

- valor_n: prompt para entrada do número via teclado.
- resultado_fatorial: saída que mostrará o resultado do fatorial.

2. Entrada do Usuário

```
7   .text
8
9   main:
10
11      #recebe input do usuario
12
13      li $v0, 4  # $v0 = comando para imprimir string
14      la $a0, valor_n  # $a0 = endereço da string
15      syscall  #impirme string
16
17      li $v0, 5  #$v0 = comando para ler inteiro
18      syscall #v0 = n (input do usuário)
```

Essa parte do código imprime o texto "Digite o valor de n: " na tela, solicitando ao usuário que forneça um número inteiro. Em seguida, usa syscall com $v0 = 5 para ler o inteiro digitado. O valor lido é retornado em $v0.

3. Cálculo do Fatorial (Loop Principal)

```
20      #calculo do fatorial (n!)
21
22      move $a0, $v0  #$a0 = n
23      li $v1, 1 # $v1 = 1 (elemento neutro multiplicaco)
24
25  loop:  #nesse loop, $v1 = n!
26      beqz $a0, fim_loop  #se $a0 = 0, sai do loop
27      mul $v1, $v1, $a0  #$v1 = vi * n
28      addi $a0, $a0, -1 #$a0 = $a0 - 1
29      j loop #volta pro loop com $a0 - 1
```

Primeiro:

- $a0 ← n: guarda o valor de entrada para iteração.
- $v1 ← 1: inicializa o resultado do fatorial com o elemento neutro da multiplicação.

Loop:

O loop realiza o seguinte:

Enquanto $a0 ≠ 0:

Multiplica $v1 ← $v1 × $a0.

Decrementa $a0 ← $a0 - 1.

Repete.

Quando $a0 = 0, sai do loop.

O resultado final de n! está em $v1.

4. Saída dos Resultados

```
33      #output
34
35      li $v0, 4  #$v0 = comando para imprimir string
36      la $a0, resultado_fatorial  #$a0 = endereço da string que precede output
37      syscall  #imprime string
38
39      li $v0, 1  #$v0 = comando para imprimir inteiro
40      move $a0, $v1  #$a0 = n!
41      syscall  #imprime n!
```

Só imprime a string "n! = " e depois imprime o valor do fatorial ($v1), que foi calculado no loop.

Questão 2

No segundo exercício, o programa também recebe um número via teclado para calcular seu fatorial. A principal diferença é que o cálculo é realizado por meio de uma função recursiva chamada **fatorial**. Essa função é chamada repetidamente até que o resultado final seja obtido.

1. Segmento de Dados

```
2   .data
3
4   valor_n: .asciiz "Digite o valor de n: " #string prompt 1
5   resultado_fatorial: .asciiz "n! = "  #string prompt 2
```

Este bloco define as duas strings usadas para exibir mensagens ao usuário.

- valor_n: prompt para entrada do número via teclado.
- resultado_fatorial: saída que mostrará o resultado do fatorial.

2. Entrada do Usuário

```
8   .text
9
10
11  main:
12
13      #recebe input do usuario
14
15      li $v0, 4  #$v0 = comando para imprimir string
16      la $a0, valor_n  #$a0 = endereço da string
17      syscall  #impirme string
18
19      li $v0, 5  #$v0 = comando para ler inteiro
20      syscall #v0 = n (input do usuário)
```

Imprime o prompt "Digite o valor de n:", depois lê um número inteiro do teclado e o armazena em $v0.

3. Preparação para o Cálculo Recursivo

```
24      move $a0, $v0  #$a0 = n
25      li $v1, 1 #$v1 = 1 (elemento neutro multiplicaco)
26
27      jal fatorial
```

- $a0: parâmetro da função (n).
- $v1: acumulador do fatorial, inicializado como 1.
- jal fatorial: chama a função fatorial, usando recursão.

4. Função Recursiva “fatorial”

```
41  fatorial:  #procedimento que realiza n!
42
43      #salvar dados na pilha
44      addi $sp, $sp, -4  #aumenta tamanho da pilha
45      sw $ra, 0($sp)  #salva endereço de retorno na pilha
46
47      beqz $a0, fim_fatorial  #se $a0 = 0, para as recursões
48      mul $v1, $v1, $a0  #$v1 = v1 * n
49      addi $a0, $a0, -1  #$a0 = n -1
50      jal fatorial #chama função para n - 1 (recursão)
51
52  fim_fatorial:  #fim das recursões
53      #resgata dados da pilha
54      lw $ra, 0($sp)  #resgata endereço de retorno da pilha
55      addi $sp, $sp, 4 #diminui tamanho da pilha
56
57      jr $ra  #pula para o endereço de retorno
```

Seguindo passo a passo:

1. Salvar endereço de retorno: manipula-se a pilha para guardar o registrador $ra (endereço de retorno da chamada de função).
2. Condição de parada: se $a0 = 0, não há mais recursão. O fatorial de 0 é 1 (condição base).
3. Passo recursivo:
   Multiplica $v1 ← $v1 × $a0.
   Decrementa $a0 ← $a0 - 1.
   Chama recursivamente a própria função fatorial.
4. Retorno: primeiro recupera o endereço de retorno da pilha e volta para a instrução seguinte à chamada da função (jr $ra).

5. Saída do Resultado

```
29      #output
30
31      li $v0, 4  #$v0 = comando para imprimir string
32      la $a0, resultado_fatorial  #$a0 = endereço da string que precede output
33      syscall  #imprime string
34
35      li $v0, 1  #$v0 = comando para imprimir inteiro
36      move $a0, $v1  #$a0 = n!
37      syscall  #imprime n!
38
39      j exit  #fim do programa
```

- Exibe a string "n! = " com o resultado do fatorial contido em $v1.
- Leva para o final do programa (como o rótulo “exit” está vazio, essa linha marca o fim da execução).

Questão 3

Resposta:

Utilizamos o simulador de Branch History Table (BHT) do ambiente MARS para analisar o desempenho dos dois programas de cálculo de fatorial. As simulações foram realizadas variando os seguintes parâmetros: quantidade de entradas na BHT (8, 16 e 32), tamanho da BHT (1 e 2 bits) e valor inicial da predição (0 para NOT TAKEN e 1 para TAKEN).

Devido à simplicidade dos nossos códigos, apenas uma entrada da tabela é efetivamente utilizada. Essa única entrada reflete o comportamento do único desvio condicional presente, que, em ambas as implementações, corresponde ao ramo responsável por encerrar o loop ou a recursão. Por esse motivo, o comportamento da BHT nas duas versões do programa é muito semelhante.

Observamos que a principal diferença entre os dois tamanhos de BHT (1 e 2 bits) está relacionada ao impacto de um chute inicial incorreto. Quando a predição inicial é incorreta, o BHT com 2 bits tende a apresentar mais erros de predição do que o de 1 bit. Isso ocorre porque, no esquema de 2 bits, são necessárias duas predições erradas consecutivas para que a direção do desvio seja corrigida. Já no modelo de 1 bit, um único erro já é suficiente para inverter a predição. Como o desvio no nosso programa é tomado apenas uma vez (no encerramento da recursão ou do loop), um valor inicial incorreto em uma BHT de 2 bits leva a mais erros antes que a predição se ajuste corretamente.

Além disso, é importante notar que, independentemente do tamanho da BHT ou da predição inicial, o último chute sempre será incorreto. Isso se deve à transição final do desvio condicional. Com base nas observações realizadas, temos os seguintes cenários: com BHT de 1 bit, há apenas um erro se a predição inicial for correta e dois erros se for incorreta. Já com BHT de 2 bits, há um erro se a predição inicial estiver correta e até três erros se estiver incorreta.

Dessa forma, quanto maior o número de bits utilizados na BHT, maior tende a ser a penalização em caso de uma predição inicial equivocada. No contexto do nosso programa, cujo comportamento de desvio é altamente previsível e ocorre raramente, a BHT de 1 bit se mostrou mais eficiente quando o valor inicial é incorreto. Por outro lado, se o valor inicial estiver correto, não há diferença significativa entre as duas configurações. Em resumo, devido à previsibilidade do único branch presente nos programas, o uso de uma BHT de 1 bit oferece desempenho equivalente — ou até superior — ao de uma BHT de 2 bits, dependendo da predição inicial.

Saídas dos Programas

- Questão 1

```
Digite o valor de n: 0
n! = 1
-- program is finished running (dropped off bottom) --


Reset: reset completed.

Digite o valor de n: 3
n! = 6
-- program is finished running (dropped off bottom) --


Reset: reset completed.

Digite o valor de n: 5
n! = 120
-- program is finished running (dropped off bottom) --


Reset: reset completed.

Digite o valor de n: 7
n! = 5040
-- program is finished running (dropped off bottom) --
```

- Questão 2

```
Digite o valor de n: 4
n! = 24
-- program is finished running (dropped off bottom) --


Reset: reset completed.

Digite o valor de n: 9
n! = 362880
-- program is finished running (dropped off bottom) --


Reset: reset completed.

Digite o valor de n: 8
n! = 40320
-- program is finished running (dropped off bottom) --
```