UNIVERSIDADE FEDERAL DE SANTA CATARINA

CENTRO TECNOLÓGICO - CTC

ORGANIZAÇÃO DE COMPUTADORES I - INE5411-03208B

PROFESSOR: MARCELO DANIEL BEREJUCK

ALUNOS: GUSTAVO BATISTELL (12101228) E LUCAS DE S. MORO (23201136)

RELATÓRIO - LABORATÓRIO 3

Florianópolis, Outubro de 2024.

Introdução

Neste relatório, implementamos o cálculo da raiz quadrada de um número de ponto flutuante e a função seno a partir de chamadas de procedimentos no MIPS. A proposta de ambos os exercícios consistem no cálculo de uma função a partir de implementações de procedimentos iterativos.

Para o primeiro exercício, o desafio era escrever um procedimento que calculasse, de forma aproximada, a raiz quadrada de um número de ponto flutuante a partir da fórmula:

$$Estimativa = \left(\frac{\left(\frac{x}{Estimativa} \right) + Estimativa}{2} \right)$$

Os requisitos para a implementação eram que o programa principal deveriam chamar o procedimento **raiz_quadrada** para o cálculo de raiz, os cálculos deveriam ser feitos com o uso de instruções e registradores de ponto flutuante e o valor de **x** deveria ser informado pelo usuário via teclado.

Para o segundo exercício, foi demandado também um procedimento, que calculasse dessa vez a função seno, dada pela seguinte série:

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

O requisito dessa vez, além de manter os cálculos feitos a partir de instruções e registradores de ponto flutuante (como no exercício 1), era limitar o cálculo aos primeiros 20 termos da série.

Exercício 1

Para a primeira etapa, a dupla optou por armazenar na memória de dados, além dos valores de estimativa e do número de iterações, a constante 2 para o cálculo da fórmula da raiz quadrada devido a dificuldade encontrada para passar de forma imediata uma constante de ponto flutuante. O valor inicial de **estimativa** foi dado igual a **1** como pedido no enunciado, e o valor de **n** foi de escolha da dupla após analisarmos que a partir de **6** iterações o resultado estava devidamente preciso para valores de **x** de até 10.

```
.data
        estimativa: .double 1
        n: .word 6
        dois: .double 2.0
```

Figura 1 - Valores armazenados na memória de dados

No segundo passo, foram iniciados os carregamentos de valores da memória nos registradores e a instrução de leitura de um valor double. Além disso, chamamos o procedimento **raiz_quadrada** com a instrução **jal**.

```
.text

        li  $v0, 7      #leitura do numero flutuante 'x' via teclado
        syscall

        lw  $t0, n          #carrega n (número de iterações) em $t0
        ldc1 $f2, estimativa #carrega da memoria a estimativa inicial em $f2
        ldc1 $f10, dois #carrega da memoria o valor 2 em $f10

        jal raiz_quadrada
        j exit
```

Figura 2 - Leitura de um double via teclado, carregamento de valores nos registradores $f e chamada do procedimento principal

Agora nessa terceira etapa, implementamos o procedimento que calcula a raiz quadrada de **x**, decrementando o registrador $t0 **n** vezes até o fim do loop. Além disso, resolvemos a equação, armazenando o resultado de **x/estimativa** em **$f6**, **(x/estimativa)+estimativa** em **$f8** e o resultado disso dividido por 2 em **$f2**, gerando assim a aproximação da raiz quadrada. Esses cálculos vão ser executados **n** vezes, até que **$t0** tenha o valor 0, comparando com o registrador **$zero** em **bne**, resultando na continuação das instruções (PC = PC + 4), armazenando o valor da raiz em estimativa. Por fim é executada a instrução **jr $ra**, retornando ao endereço chamador do procedimento.

```
raiz_quadrada:
        addi $t0, $t0, -1

        # estimativa = (x/estimativa + estimativa) / 2
        div.d $f6, $f0, $f2  #f6 = x / estimativa
        add.d $f8, $f6, $f2  #f8 = x/estimativa + estimativa
        div.d $f2, $f8, $f10  #f2 = f8 / 2 (nova estimativa)

        bne $t0, $zero, raiz_quadrada  #verifica se $t0 igual a zero, senão volta pro começo do loop

        # armazena a estimativa final em 'estimativa'
        s.d $f2, estimativa

        jr      $ra #retorna para a funcao chamadora (PC = PC + 4)
```

Figura 3 - Estrutura do procedimento que calcula a raiz quadrada

Comparando o resultado do programa com a instrução sqrt.d, podemos analisar que, com **6** iterações, o valor da raiz quadrada de **x** tem seu resultado dado de forma precisa. Porém, na medida que aumentamos o valor, é demandado um aumento na quantidade de iterações para um resultado preciso. No primeiro exemplo, podemos demonstrar a diferença dos dois resultados com um mesmo valor de **x**, mas dessa vez, com diferentes quantidades de iterações (valores de **n**).

```
17424
293.25149463105623
-- program is finished running (dropped off bottom) --
```

Resultado de √x com **n** = 6

```
17424
132.0
-- program is finished running (dropped off bottom) --
```

Resultado de √x com **n** = 20

Figuras 4 - Diferenças de resultado no cálculo de √17424

Visto isso, partiremos para a diferença do resultado do procedimento e do resultado via instrução **sqrt.d**. Iremos comparar a instrução com o procedimento, testando diferentes valores de **n** (números de loop) para uma comparação mais precisa. Para o teste, utilizaremos o seguinte código:

```
.text
        li  $v0, 7      #leitura do numero flutuante 'x' via teclado
        syscall
        sqrt.d $f12, $f0 #instrução que calcula a raiz quadrada
        li $v0, 3       #executa o print do resultado na tela
        syscall
```

Figura 5 - exercicio1_teste.asm

Alternamos o número de iterações para 5, 10, 15 e 20, a partir disso, consideramos o valor de **x = 793.881,** organizamos o resultado de cada código e calculamos o valor do erro absoluto com a seguinte fórmula:

Erro Absoluto = |estimativa_procedimento - estimativa_sqrt.d|

Assim, chegamos aos seguintes resultados:

n	Estimativa (raiz_quadrada)	Estimativa (sqrt.d)	Erro Absoluto
5	24819.436587348766	891.0	23928.43658
10	1089.8916978007444	891.0	198.89169
15	891.0	891.0	0.0
20	891.0	891.0	0.0

Podemos observar que a partir de 15 iterações o resultado de **raiz_quadrada** (para o valor específico de **x**) se manteve o mesmo, porém, com grande discrepância para valores de **n <= 10**. Já com o uso da instrução **sqrt.d**, como não é utilizado iterações com **n**, o valor se manteve o mesmo, mas de forma precisa.

Exercício 2

No segundo exercício, a proposta foi implementar a série de Taylor para sen(x). O desafio de utilizar ponto flutuante com dupla precisão ficou em entender como utilizar convenção de chamada para os registradores $f, colocar em pilha dados de diferentes formatos, converter inteiro para double, etc

O código solicita a entrada do valor de x ao usuário com syscall específico para double, inicia a rotina principal do seno e chama as sub-rotinas folha para calcular a potenciação e o fatorial. Isso produz um termo que é somado ao próximo, até n==20.

Resultados obtidos

ângulo em °	angulo em RAD	sin(x) esperado	sin(x) obtido	Erro absoluto
0	0.00000000	0.00000000	0.0	0.00E+00
30	0.52359878	0.50000000	0.50000000	3.81E-09
45	0.78539816	0.70710678	0.70710678	2.40E-09
60	1.04719755	0.86602540	0.86602540	5.98E-10
90	1.57079633	1.00000000	1.0	0.00E+00
180	3.14159265	0.00000000	0.00000000	3.59E-09
270	-1.57079633	-1.00000000	-1.0	0.00E+00

Conclusão

A execução de procedimentos folha diminui consideravelmente a preocupação com preservação de dados/endereços em pilha. Como os trechos de potenciação e fatorial da série de Taylor para sen(x) são rotinas folhas, que não chamam subrotina, elas não precisam preservar conteúdos de registradores. Já a rotina Seno, se estiver sendo invocada por outra, precisa preservar em pilha os registradores conforme convenção de chamada. Como é uma convenção, o uso desses registradores $f pode variar entre programadores e um resumo tomado como coerente para convenção de chamada dos Registradores de Ponto Flutuante foi:

Tipo de Uso	**Registradores para ponto flutuante**
Argumentos (float)	$f12, $f14
Argumentos (double)	$f12/$f13, $f14/$f15
Retorno (float)	$f0
Retorno (double)	$f0/$f1
Temporários	$f0 a $f10, $f16 a $f19
Preservados	$f20 a $f31

No nosso código para o exercício 2, pode-se notar que seguindo a convenção apresentada acima, o conteúdo de diversos registradores deveria ser preservado e restaurado em pilha (observe os trechos comentados com PUSH e POP, respectivamente). Para minimizar uso dessas instruções, caso seno fosse uma rotina invocada, convinha empregar mais dos registradores convencionados como temporários ao invés dos “saved” de $f20 a $f31. Dessa maneira o Push e o Pop poderia se resumir a $ra mais ou outro “saved”.