UNIVERSIDADE FEDERAL DE SANTA CATARINA
EDUARDO BISCHOFF GRASEL - 22200355

Relatório Organização de computadores

Florianópolis - SC
2024

Objetivos

Ao fim do trabalho deveremos ser capazes de ter um conhecimento mais profundo sobre o coprocessador 1 do mips e como utilizar float e double, além de converter um dado .word para .double ou .float, aplicar a lógica por trás dos registradores do coprocessador 1 e treinar os comandos limitados envolvendo os registradores $fx.

Trabalho 1

Iniciamos declarando as necessidades do programa no .data, como as mensagens imprimidas no terminar e os valores .double que serão usados em algumas operações aritméticas ou de movimentação entre registradores $fx.

```
.data
        msg1: .asciiz "Digite o valor de x: "
        msg2: .asciiz "Digite o valor de n: "
        zero: .double 0
        um_double: .double 1
        dois_double: .double 2
        estimativa_double: .double 1
```

O início do .text é extremamente simples, apenas carregando os valores para registradores e pedindo os inputs do usuário e criando um contador para parar o loop. A maior diferença aqui em relação a outros programas foi o comando ldc1 (load double coprocessador 1) que é semelhante ao lw. Vale também lembrar que para ler um dado double, usamos o li $v0, 7 e o resultado é devolvido para $f0, o qual para movimentar precisamos executar uma soma com um 0 double, que já foi carregado previamente, pois não podemos utilizar comandos como move, por exemplo.

```
.text
        # double 0
        la $t0, zero
        ldcl  $f6, 0($t0)

        #  contador
        li $t1, 1

        # estimativa double, um double, dois_double
        la $t0, estimativa_double
        ldcl $f8, 0($t0)
        la $t0, um_double
        ldcl $f4, 0($t0)
        la $t0, dois_double
        ldcl, $fl4, 0($t0)

        # imprimir a mensagem 1 e pedir o valor de x
        la $a0 msg1
        li $v0, 4
        syscall
        li $v0, 7
        syscall
        add.d $f2, $f0, $f6

        # imprimir a mensagem 2 e pedir o valor de n
        la $a0 msg2
        li $v0, 4
        syscall
        li $v0, 5
        syscall
        move $t0, $v0
```

Agora vamos ao loop, no qual a condição de parada será, se o registrador $t1 (contador), foi maior ou igual o $s0 (valor de n), agora apenas fizemos o cálculo que foi dado na folha do trabalho. Segue o loop comentado:

```
        loop:
                bge $t1, $t0, fim_loop
                div.d $f10, $f2, $f8   # divide x pela estimativa_double e coloca em $f10
                add.d $f12, $f10, $f8  # soma x com o resultado anterior
                div.d $f16, $f12, $f14  # divide o resultado anterior por 2 e guarda em $f16
                addi $t1, $t1, 1  # incrementa o contador
                add.d $f8, $f6, $f16 # incrementa a estimativa
                j loop
fim_loop:
        li $v0, 3
        add.d $f12, $f6, $f16 # move para $f12 para ser mostrado na tela
        syscall
```

Com isso, chegamos ao final do programa. O erro no valor absoluto se dá principalmente quando um número não tem raiz exata, que conforme o loop aumenta, aumenta a precisão, ou se é uma raiz exata, o erro se dá apenas em n suficientemente baixo.

```
Reset: reset completed.

Digite o valor de x: 100
Digite o valor de n: 5
10.032578510960604
-- program is finished running


Reset: reset completed.

Digite o valor de x: 100
Digite o valor de n: 50
10.0
-- program is finished running
```

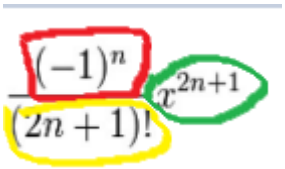
```
Digite o valor de x: 101
Digite o valor de n: 5
10.083685815818733
-- program is finished running (dropped off bottom) --


Reset: reset completed.

Digite o valor de x: 101
Digite o valor de n: 50
10.049875621120089
-- program is finished running (dropped off bottom) --
```

Trabalho 2

Como esse trabalho é muito mais complexo, iniciei com uma análise do que eu tinha e o que tinha que fazer. As rotinas já foram implementadas nas listas de exercícios, porém apenas aceitam valores inteiros, então fiz uma subdivisão na conta, de modo que eu consiga essas partes isoladas utilizando números inteiros, claro que elas foram isoladas pois sempre resultarão em um número inteiro.



Em outras palavras, a estratégia adotada foi conseguir esses valores, em seguida convertê-los para double e então aplicar a conta final com as 3 partes que foram separadas.

No .data declaramos x e n, juntamente com os doubles 0 e 1 para realizar operações nos registradores $fx.

```
.data
        x: .word 1
        n: .word 0
        zero: .double 0
        um: .double 1
```

No .text inicialmente apenas fazemos a rotina de inicializar os registradores, com a adição do comando mtc1 e cvt.d.w, que move um valor para o coprocessador 1 e o converte para double.

```
.text

        li $s5, 3 # controle loop

        la $t0, x
        lw $s3, 0($t0) # x

        la $t0, n
        lw $s0, 0($t0) # n

        li $s1, -1 # -1

        li $s2, 1 # 1
        mtc1.d $s2, $f4
        cvt.d.w $f4, $f4

        la $t0, zero
        ldc1 $f16, 0($t0) # zero para double
        la $t0, um
        ldc1 $f18, 0($t0) # um para double
```

Agora entrando no loop, seguiremos o mesmo esquema de cores na equação acima para o entendimento.

```
loop:
        bge $s0, $s5, fim_loop
        # caluclo potencia
        move $a0, $s1
        move $a1, $s0
        jal pow # calculando -1^n
        mtc1 $v0, $f8 # EM F8 TEM O VALOR DA POTENCIA
        cvt.d.w $f8, $f8
        # calculando fatorial
        li $t7, 2
        mul $t0, $s0, $t7
        addi $t1, $t0, 1
        move $a0, $t1
        jal fatorial
        mtc1.d $v0, $f10 # EM F10 TEM O RESULTADO DO FATORIAL
        cvt.d.w $f10, $f10
        # calculado a potencia restante
        li $t7, 2
        mul $t0, $s0, $t7
        addi $t1, $t0, 1
        move $a0, $s3
        move $a1, $s0
        jal pow
        mtc1 $v0, $f12 # EM F12 TEM O RESULTADO DA SEGUNDA POTENCIA
        cvt.d.w $f12, $f12
```

Vale lembrar que cada um destes pedaços individuais estão sendo convertidos para float antes do cálculo final com a instrução cvt.d.w $fx, $fx. Por fim, precisamos apenas multiplicar a parte vermelha pela amarela, e em seguida dividir pela verde, então somar o resultado com ele mesmo em cada loop.

```
        # AGORA PEGAMOS A CONVERSÃO PARA DOUBLE DESSES VALORES E REALIZAMOS A CONTA FINAL
        div.d $f14, $f8, $f10
        mul.d $f16, $f14, $f12

        add.d $f20, $f16, $f20

        # controle do loop
        addi $s0, $s0, 1

        j loop
fim_loop:
        li $v0, 10
        syscall
```

O Resultado final estará no registrador $f20, Com x sendo 1 e crescendo o valor de n até o ponto que o programa não quebre por seu um número muito grande por se tratar de fatorial, teremos uma expectativa que o valor em $f20 seja

0.8414709848, e abaixo encontramos a progressão de valor conforme n aumenta de 1 para 10

$f20	0.0	1.0
$f21	1.875	

$f20	-9.553225E-39	0.8251984126984128
$f21	1.8312995	

$f20	8.9735724E-4	0.8252011433595676
$f21	1.8313003	

$f20	7.88617E-8	0.8252011307730152
$f21	1.8313003	

Ou seja, ele está cada vez mais perto do resultado desejado, ou seja, ao tender ao infinito teremos o valor que esperamos.

Conclusão

Ao final do trabalho ficou claro que a parte mais difícil foi referente a parte 2, na qual houve muita conversão de .word para .double, pois não consegui passar os valores double para os parâmetros de modo convencional, além disso a falta de comandos para manipular registradores $fx e endereços da memória se provou algo muito trabalhoso.