Arquitetura e Organização de Computadores I

Trabalho Prático 1

Heitor Lourenço Werneck
heitorwerneck@hotmail.com

25 de Junho de 2019

1 Introdução

Esse trabalho consiste na solução do problema de calcular o valor do número π e o máximo divisor comum de dois números recursivamente em assembly MIPS.

2 Número π

O número π pode ser calculado pela seguinte série: $\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \frac{4}{13} - \frac{4}{15} \cdots$

O algoritmo a seguir exemplifica o cálculo do π com esta série.

Algorithm 1 Calcula π.

Input: termos

1: **procedure** PI
2: $\pi = 0$
3: $sinal = 1$
4: $divisor = 1$
5: **while** $termos \neq 0$ **do**
6: $\pi = \pi + sinal \cdot 4/divisor$
7: $sinal = sinal \cdot (-1)$
8: $divisor = divisor + 2$
9: $termos = termos - 1$

Basta transformar esse algoritmo para assembly MIPS.

Para guardar o valor do número π foi utilizado um registrador do tipo *double* para ter uma alta precisão. Logo foi utilizado por simplicidade todos registradores nos cálculos como *double*. Para inicializar cada registrador com seu valor foi criado os valores na seção *.data*.

```
1         .data
2 dn1:    .double     -1.0
3 d0:     .double     0.0
4 d1:     .double     1.0
5 d2:     .double     2.0
6 d4:     .double     4.0
```

Na função principal a primeira coisa a ser feita é a chamada a função que lé a quantidade de termos, que é um inteiro, depois chama o algoritmo que calcula o pi e posteriormente imprime na tela o valor retornado.

```
main:
        addi        $v0,$zero,5         # $v0 = 5, função le inteiro
        syscall                         # a função receberá o número de termos a
            serem somados
        add         $a0,$v0,$zero       # número de termos
        jal         pi
        mov.d       $f12,$f0
        addi        $v0,$zero,3         # imprime double
        syscall                         # imprime o registrador $f12
```

A função *pi* irá começar preservando os valores dos registradores a serem utilizados. Primeiro é guardado o valor do *stack pointer* antes de ser alinhado e então depois de alinhado é feito o salvamento dos registradores.

```
pi:
        move        $t1,$sp             # guarda stack pointer antigo
        andi        $sp,$sp, 0xfffffff8 # alinha o sp
        addi        $sp,$sp,-40         # abre espaço no sp
        sdc1        $f2,0($sp)          # salva valores
        sdc1        $f4,8($sp)
        sdc1        $f6,16($sp)
        sdc1        $f8,24($sp)
        sdc1        $f12,32($sp)
```

O proximo passo é inicializar os registradores para o seu uso posterior.

```
        l.d         $f0,d0              # f0 = 0, variavel que guarda o resultado
        l.d         $f2,d1              # f2 = 1, divisor
        l.d         $f4,d1              # f4 = 1, sinal
        l.d         $f6,dn1             # f6 = -1, mudador de sinal
        l.d         $f8,d2              # f8 = 2, taxa de crescimento divisor
        l.d         $f12,d4             # f12 = 4, dividendo sempre 4
        move        $t0,$a0             # numero de termos
```

No *loop* primeiro é checado se o número de termos é igual a zero, se não for então faz todo o processo de cálculo do termo a ser somado e soma-o no resultado final, depois muda o sinal, incrementa o divisor com 2, diminui o número de termos e então faz um salto incondicional para o *loop*.

```
loop:
        beq         $t0,$zero,end       # checa se acabou os termos
        div.d       $f10,$f12,$f2       # 4/divisor
        mul.d       $f10,$f10,$f4       # sinal*(4/divisor)
        add.d       $f0,$f0,$f10        # resultado = resultado + sinal*(4/divisor)
        mul.d       $f4,$f4,$f6         # muda o sinal, $f4*-1
        add.d       $f2,$f2,$f8         # divisor + 2
        addi        $t0,$t0,-1          # diminui numero de termos
        j           loop
```

No fim basta voltar com os valores originais dos registradores utilizados e retornar para o lugar que a função foi chamada.

```
        ldc1        $f2,0($sp)
        ldc1        $f4,8($sp)
        ldc1        $f6,16($sp)
        ldc1        $f8,24($sp)
        ldc1        $f12,32($sp)        # volta com os valores antigos
        move        $sp,$t1             # libera espaço sp, valor inicial
        jr          $ra                 # retorna
```

Execução do programa:

```
50
3.1215946525910l1
-- program is finished running (dropped off bottom) --
```

3 Máximo divisor comum

Esse algoritmo necessita de uma função que recebe dois valores que são os que se quer achar o máximo divisor comum e retorna o máximo divisor comum deles.

O algoritmo do máximo divisor comum é o seguinte:

Algorithm 2 Máximo divisor comum.

Input: a, b

```
1: procedure MDC
2:     if b == 0 then
3:         return a
4:     else
5:         return mdc(b, a%b)
```

Basta transformar o algoritmo no seu respectivo código assembly MIPS.

O primeiro passo do algoritmo é guardar os valores de entrada e retorno da função e depois de chamado o algoritmo que calcula o *mdc* retornar com esses valores preservados.

```
fmdc:
        addi        $sp,$sp,-12 # guarda valores na pilha
        sw          $a0,0($sp)
        sw          $a1,4($sp)
        sw          $ra,8($sp)
        jal         mdc
        lw          $a0,0($sp) # desempilha
        lw          $a1,4($sp)
        lw          $ra,8($sp)
        addi        $sp,$sp,12
        jr          $ra
```

A primeira parte do algoritmo que irá realmente calcular o *mdc* é intuitiva e não difere muito do que se espera. Se o segundo argumento for igual a 0 retorna o primeiro argumento, se não vai para a *label else*.

```
mdc:                                    # mdc(a,b)
        bne         $a1,$zero,else      # if (b == 0)
        move        $v0,$a0             #
        jr          $ra                 # return a
```

Já no se não do algoritmo é preciso guardar o valor do registrador de endereço na memória para se chamar recursivamente a função e preservar o contexto. Após salvar é feito a chamada recursiva e depois volta com o valor da memória e retorna o valor retornado pela função.

```
else:
        addi        $sp,$sp,-4          # salva $ra
        sw          $ra,0($sp)
        move        $t0,$a1             # temp = b
        rem         $a1,$a0,$a1         # b = a % b
        move        $a0,$t0             # a = temp
        jal         mdc                 # mdc(b,a%b) chama a função recursivamente
        lw          $ra,0($sp)          # desempilha valor salvo
        addi        $sp,$sp,4
        jr          $ra
```

Na função principal o primeiro passo é obter os dois valores que se quer achar o máximo divisor comum. Depois disso é só chamar a função *mdc* e imprimir o inteiro obtido.

```
main:
        addi        $v0,$zero,5         # $v0 = 5, função le inteiro
        syscall                         # a função receberá o número de termos a
            serem somados
        add         $a0,$v0,$zero
        addi        $v0,$zero,5         # $v0 = 5, função le inteiro
        syscall                         # a função receberá o número de termos a
            serem somados
        add         $a1,$v0,$zero
        jal         mdc
        move        $a0,$v0
        addi        $v0,$zero,1         # $v0 = 1, função imprime inteiro
        syscall
```

Execução do programa:

```
2000
256
16
-- program is finished running (dropped off bottom) --
```