Relatório 06 | Organização de computadores I - INE5411

Alunos: Pedro Henrique Gimenez - 23102766 e Victória Rodrigues Veloso - 23100460

1. Exercício 1

Para o primeiro exercício, foi desenvolvido um programa em Assembly que recebe 2 vetores de tamanho N pelo usuário, calcula a média aritmética de cada um e as imprime no console.

1.1 Implementação

Começamos definindo no .data os vetores utilizando o recurso *.space* para alocar um espaço arbitrário de 4000 bytes (no qual cabem 1000 floats dentro do vetor), também definimos as frases que serão impressas no console posteriormente. Como pode ser visto no seguinte trecho do código:

```
	.data
	resultadoA: .float 0.0
	resultadoB: .float 0.0
	vetorA: .space 4000 #aloca 1000 espaços no array
	vetorB: .space 4000 #aloca 1000 espaços no array
	.align 2                #alinhando para inteiro
	N: .word 0
```

Iniciamos o *.text* salvando os endereços base vetores nos registradores $s0 e $s1 e, logo em seguida, começamos o procedimento de receber o valor de N pelo console por meio do *syscall.* Para preencher cada vetor, criamos uma estrutura *for* que percorre N vezes, em cada iteração é lido o float do console, salva ele no vetor e incrementa 4 do endereço atual do vetor para pular para a próxima posição do vetor. O seguinte código apresenta essa iteração de salvamento dos dados.

```
# ---- recebe vetor A e preenche com os valores recebido do teclado----
	li $t2, 0
	addi $t4, $s0, 0    #copia o endereço base para t4, assim não perdemos ele

	preenche_A:
		li $v0, 6                #comando p receber float
		syscall
		s.s $f0,  0($t4)         #carrega o float lido na posição do vetor
		addi $t4, $t4, 4         #pega a próxima posição do vetor
		addi $t2, $t2, 1         #incrementa registrador auxiliar
		bne $t0, $t2, preenche_A  #repete o loop enquanto os N valores não forem
lidos
```

Após preencher os vetores, chamamos o procedimento *media* duas vezes (uma para cada vetor). O procedimento foi realizado por meio de um *for* que itera N vezes, percorrendo o vetor e somando todas as variáveis no registrador $f2. Com a soma total do vetor, convertemos N para um float (para poder realizar a divisão com o float do somatório), dividimos o resultado da somatória por N e armazenamos em $f2. O procedimento completo pode ser visto no seguinte código:

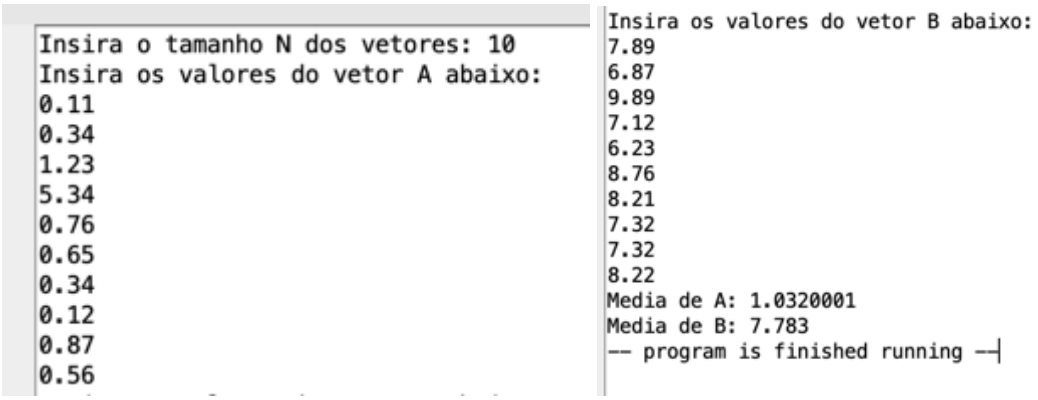
```
media:
      li $t2, 0                     #registrador para controlar o loop
      sub.s $f2, $f2, $f2                 #reseta o resultado final

      loop_soma:
            l.s $f14, 0($a0)
            add.s $f2, $f2, $f14    #somaA +=vetA[i]
            addi $a0, $a0, 4        #pega a proxima posição do vetor A
            addi $t2, $t2,1
            bne $t0, $t2, loop_soma #repete o loop enquanto a soma de todos os
valores dos vetores não for finalizada


      mtc1 $t0, $f6                       #move o valor de N para f6
      cvt.s.w $f6, $f6              #converte para precisão simples, para
efetuar a divisão
      div.s $f2, $f2, $f6                 #divide a soma do vetor A pelos N
elementos

      jr $ra
```

Após calcular a média de cada vetor, imprimimos os resultados no console. Para verificar se o código funciona corretamente, testamos com os vetores fornecidos pelo exercício (Figuras 1 e 2).



Figuras 1 e 2. Console da ferramenta MARS ao rodar o programa de cálculo de médias

1.2 Resultado do uso de instruções do programa

Ao rodar o programa utilizando N igual a 10, utilizamos a ferramenta *Instruction Statistics* do MARS e obtivemos os seguintes dados sobre as instruções usadas durante a execução do programa (Figura 3).

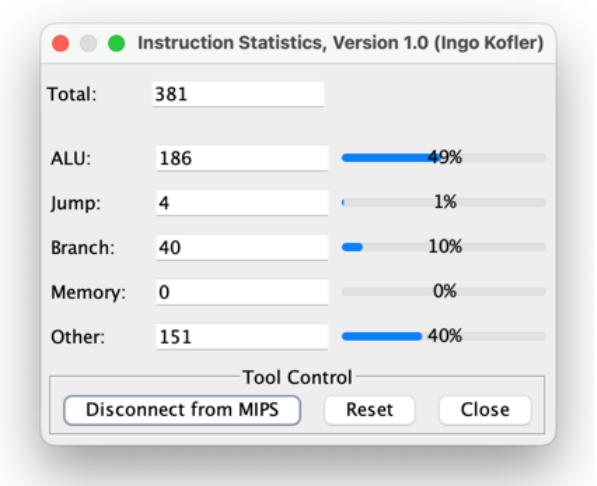


Figura 3. Ferramenta *Instruction Statistics* após a execução da versão 1 do programa

2. Exercício 2

Para o segundo exercício, foi necessária a refatoração do código para que o número total de instruções diminuísse com o intuito de melhorar o desempenho. O que mais consome do desempenho é o número de acessos à memória, como na primeira versão o número de acessos já era baixo, nos preocupamos então em diminuir a quantidade total de instruções.

2.1. Refatoração do código

Para aumentar o desempenho do procedimento *media,* uma medida tomada foi modificá-lo para que ao invés de calcular a média de um vetor por vez, ele calculasse as médias dos dois vetores dentro de um mesmo loop, já que ambos possuem o mesmo tamanho N. Após a modificação, o código do procedimento pode ser visualizado abaixo:

```
        li      $t2, 0                      # registrador para controlar o loop
        loop_soma:
                l.s     $f14, 0($a0)
                l.s     $f16, 0($a1)
                add.s   $f2, $f2, $f14      # somaA += vetA[i]
                add.s   $f4, $f4, $f16      # somaB += vetB[i]
                addi    $a0, $a0, 4         # pega a proxima posição do vetor A
                addi    $a1, $a1, 4         # pega a proxima posição do vetor B
                addi    $t2, $t2, 1
                bne     $t0, $t2, loop_soma   # repete o loop enquanto a soma de todos os valores
dos vetores não for finalizada

        mtc1    $t0, $f6                    # move o valor de N para f6
        cvt.s.w $f6, $f6                    # converte para precisão simples, para efetuar a
divisão
        div.s   $f2, $f2, $f6               # divide a soma do vetor A pelos N elementos
        div.s   $f4, $f4, $f6               # divide a soma do vetor B pelos N elementos
```

```
        jr      $ra
```

Além da modificação do procedimento em si, também removemos algumas alocações de memórias da *.data* que eram desnecessárias, o que contribuiu também para uma melhora do desempenho.

2.2. Análise do desempenho final

Ao executar o novo programa com os mesmos valores da execução anterior, obtivemos os seguintes resultados (Figura 4).

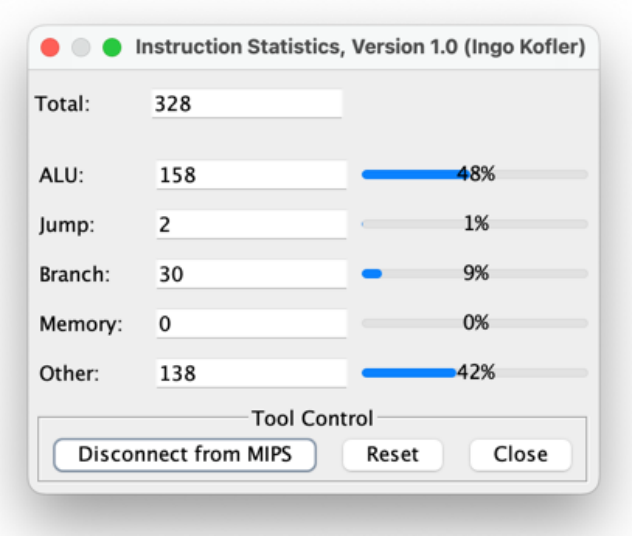


Figura 4. Ferramenta *Instruction Statistics* após a execução da versão refatorada do programa

É notável a diferença entre a primeira versão e a refatoração, na primeira, o número total de instruções era de 381, já na refatorada é de 328. Uma diminuição em 14% da versão anterior. Todos os outros tipos de instruções foram diminuídos também, a quantidade de operações na ALU caiu em 15%, de Jumps em 50%, de Branches em 25% e de outros em 8%.

Analisando os dados, é possível concluir que houve uma melhora no desempenho entre as versões. Isso se justifica pela redução na quantidade de iterações do loop de soma. Agora, não é necessário utilizar dois loops diferentes, um para cada vetor; ambos aproveitam um único loop e uma única chamada de procedimento. Isso economiza significativamente o custo computacional do processamento, já que uma instrução de desvio condicional é bastante custosa.