Implementação de multiplicação e transposição de matrizes com Assembly do MIPS

Carolina Adilino, Ana Julia Botega

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Caixa Postal 5094 – 88035-972 – Florianópolis – SC – Brasil

Resumo. *Esse trabalho visa apresentar o programa desenvolvido pelos autores, cujo objetivo é a resolução de multiplicação de matrizes em Assembly do MIPS, para ser executado no simulador MARS. Tal resultado é obtido através da utilização de laços de repetição sobre os elementos da matriz . Seguindo as diretrizes do projeto, o programa recebe 2 matrizes, "A" e "B" , cuja última deve ser transposta e multiplicada pela primeira. Como saída, são apresentados os resultados em um arquivo .txt cujo nome é inserido pelo terminal*

Palavras-chave: *Assembly, MIPS,matrizes, multiplicação*

1. Objetivo

Escrever um código em assembly que realize a seguinte operação com duas matrizes 3x3: $A.B^T$, escrevendo seu resultado em um arquivo .txt cujo nome é inserido pelo terminal.

2. Materiais e métodos

O código assembly foi desenvolvido para arquitetura MIPS e executado na plataforma MARS (MIPS Assembler and Runtime Simulator).

O primeiro procedimento a ser chamado é o PROC_MUL , o procedimento começa salvando na pilha o endereço de retorno (salvo em $ra) e outros registradores que serão modificados($s0,$s1). O algoritmo para calcular D = A x B , é realizado com três laços, controlados por registros que representam os índices i , j e k. Para cada elemento D[i][j], o laço mais interno (k) calcula o produto escalar entre a linha `i` de `A` e a linha `j` de `C`. Como a linha `j`

de C é, por definição, a coluna j de B, este cálculo resulta no valor correto para D[i][j]. Após o término do laço k, a soma acumulada é armazenada na posição D[i][j].

```
for_k:
    beq  $t8, $t3, fim_k

    # A[i][k]
    mul  $t9, $t4, $t3
    add  $t9, $t9, $t8
    sll  $t9, $t9, 2
    add  $t9, $t1, $t9
    lw   $s2, 0($t9)

    # C[j][k]
    mul  $t9, $t5, $t3
    add  $t9, $t9, $t8
    sll  $t9, $t9, 2
    add  $t9, $t0, $t9
    lw   $s3, 0($t9)

    mul  $s2, $s2, $s3
    add  $t6, $t6, $s2

    addi $t8, $t8, 1
    j for_k
```

Fig1 . Fragmento do código do processo de multiplicação

O segundo processo chamado é o PROC_TRANS , ele é um procedimento folha dentro de PROC_MUL , chamado antes do início do cálculo da multiplicação , essa rotina tem como entrada o endereço da matriz original (B), o endereço da matriz de destino (C) e a dimensão (n). Para isso , ela utiliza dois laços aninhados (para os índices i e j) para percorrer cada elemento da matriz de entrada. Dentro do laço, ele lê o valor de B[i][j] e o escreve na posição C[j][i]. Essa inversão de índices é o que permite a transposição . Por fim , o procedimento retorna ao seu chamador.

```
PROC_TRANS:
    # Entrada:
    # $a0 = B
    # $a1 = C
    # $a2 = n
    li   $t0, 0        # i
loop_i:
    beq  $t0, $a2, end_trans
    li   $t1, 0        # j
loop_j:
    beq  $t1, $a2, end_j
```

Fig2 . parte do processo de transposição

De volta na função main, o próximo processo a ser chamado é o PROC_NOME, cujo objetivo é ler uma string do terminal e concatenar com ".txt". A leitura da string é simples, apenas uma chamada se sistema após alocar ao registrador $v0 o comando 8 e depois armazenar o input em um buffer previamente declarado, "filename".

O próximo passo é adicionar ao final do buffer a extensão desejada ".txt". Para isso, é utilizado um loop que foi nomeado "remover_enter", que percorre nosso buffer em busca do caractere "\n". Ao ser encontrado, uma instrução beq encaminha a execução do programa para "substituir_zero", que irá substituir esse caractere por 0, indicando o final do buffer. Isso mostra-se necessário pois, após digitar a string no terminal, o usuário irá clicar em enter, que também será adicionado ao buffer. Sem essa correção, o nome do arquivo ficaria "input \n .txt", com esse \n significando uma quebra de linha.

Imediatamente após terminar de substituir \n por 0, o programa pula para "fim_remover", que irá percorrer todo o buffer em busca do final da string, indicado pelo 0 anteriormente mencionado. Com o ponteiro apontando para o último caractere, uma instrução de comparação encaminha o programa para "pronto_para_anexar", que irá adicionar "." depois "t", "x" e "t", cada um por vez através do loop "copiar_extensao". Com todo o processo realizado e nosso input do teclado salvo no buffer "filename" junto à extensão .txt, voltamos ao main através da última instrução jr $ra.

De volta ao corpo principal do programa, os últimos passos para sua conclusão são converter todos os números da matriz para ASCII e salvá-los em um buffer. Primeiramente, alguns registradores recebem alguns valores importantes, com destaque à um ponteiro para o início da matriz D, um contador de iterações, um ponteiro para o buffer principal "ascii_buffer", um ponteiro para o buffer auxiliar "buffer_temp" e dois contadores de bytes, um para cada um

desses buffers.

```
la  $s2, D              #carregando o endereço de inicio de D em s2
li  $t5, 0              #contador de iterações
li  $t6, 9              #carregando 9 em t9
la  $s0, ascii_buffer   #salvar o endereço do buffer em s0
la  $s1, buffer_temp    #salva o endereco do buffer temporario/auxiliar
li  $t8, 0              #contador de bytes do buffer principal
li  $t7, 0              #contador de bytes do buffer temporario
```

Fig 3. Alocando valores importantes para registradores

Agora, o próximo passo é iniciar o “matriz_loop” checando se o contador de iterações é igual ao número de elementos da matriz. Não sendo, o que significa que não terminamos de iterar a matriz, é checado se o número é negativo. Caso seja, o caractere “-” é adicionado ao buffer e o número é multiplicado por -1, para que seu valor absoluto seja passado para a próxima parte.

O próximo passo é a conversão do número para ASCII, que é realizada por cada dígito do número, começando pelas unidades. Isso nos daria o número invertido, por exemplo, 123 viraria 321. Para contornar esse problema, é utilizado um buffer auxiliar, que guardará o número com os dígitos invertidos temporariamente.

Vale ressaltar que todas as vezes que um buffer recebe um novo caractere, seu respectivo ponteiro e contador de bytes são incrementados.

A conversão é realizada dividindo o número por dez e guardando o resto (número atual) e o quociente (próximo número) em registradores. Ao número atual, somamos 48. Agora temos o número em sua representação ASCII. Como mencionado anteriormente, ele é salvo no buffer auxiliar e uma checagem é realizada: se o próximo número for diferente de zero, o loop é repetido. Caso o número seja zero, vamos para a próxima etapa.

O número está salvo com a ordem de seus dígitos invertida no buffer auxiliar, vamos então subtrair um ao ponteiro que aponta para o final dele. Agora, o ponteiro aponta para o dígito que foi o último a ser inserido no buffer auxiliar e registra ele no buffer principal. Esse loop é repetido para todos os dígitos do número, subtraindo 1 ao ponteiro em cada iteração.

Agora, é checado se estamos no final de uma linha. Caso esse seja o caso, “\n” é adicionado ao buffer. Caso não seja, é adicionado um espaço. Garantindo, assim, que a matriz vai

estar no formato correto.

```
48  matriz_loop:
49          beq     $t5, $t6, buffer_done   #checando se já percorremos toda a matriz
50
51          lw   $t0, 0($s2)        #valor numerico armazenado no endereço
52          move $t9, $t0           #passando esse valor pro t9
53          li   $t2, 10            #botando 10 em t2
54
55          bgez    $t9, convert_loop
56                  addi    $t3, $zero, 45  # printando o "-"
57                  sb      $t3, 0($s0)     # store ASCII char in buffer
58                  addi    $s0, $s0, 1     # ir pro proximo espaço do buffer
59                  addi    $t8, $t8, 1     # contador de bytes no buffer
60
61                  neg     $t9, $t9        #transformando o numero em positivo
62
63          convert_loop:
64                  divu    $t9, $t2        # dividindo o item atual da matriz por 10
65                  mfhi    $t3             # resto = digito atual
66                  mflo    $t4             # quociente = prox numero
67
68                  #Convert digit to ASCII
69
70                  addi    $t3, $t3, 48    # somar o ultimo digito com 48 do ASCII
71                  move    $t9, $t4        #carregando em t9 o proximo digito para repetir o loop com ele
72
73                  sb      $t3, 0($s1)     #salvando o caracter em ascii no buffer auxiliar
74                  addi    $s1, $s1, 1
75                  addi    $t7, $t7, 1     # contador de bytes no buffer auxiliar
76
77                  bne  $t9, $zero, convert_loop #se o numero tiver mais de um digito, roda dnv
78
79
80                  saving_loop:
81
82                  addi    $s1, $s1, -1    #voltando uma casa no buffer auxiliar
83                  addi    $t7, $t7, -1    # contador de bytes no buffer auxiliar
84                  lb      $t1, 0($s1)     #armazenando em t1 o valor do digito
85
86                  sb      $t1, 0($s0)    # store ASCII char in buffer
87                  addi    $s0, $s0, 1    # ir pro proximo espaço do buffer
88                  addi    $t8, $t8, 1        # contador de bytes no buffer
89
90                  bne     $t7, $zero, saving_loop #se ainda tiver digitos no buffer auxiliar, roda dnv
91
92
93                  beq $t5, 2, prox_linha
94                  beq $t5, 5, prox_linha
95                  beq $t5, 8, prox_linha
96
97
98                  li $t9, 32      # ASCII do espaço
99                  sb $t9, ($s0)
100                 addi $s0, $s0, 1        #endereço do buffer principal
```

Fig 4. Parte da lógica de matriz_loop

Quando todos os elementos da matriz forem convertidos em ASCII e salvos no buffer principal, a instrução da linha 49 encaminha o programa para buffer_done, que vai escrever o conteúdo do buffer no arquivo.

Primeiramente o arquivo é aberto através do comando 13 e uma chamada de sistema. O nome do arquivo vai ser o conteúdo do buffer “filename”, previamente escolhido pelo usuário. No arquivo é escrito os conteúdos de ascii_buffer, e seu tamanho é definido pelo registrador $t8,

que contou a quantidade de caracteres. Feita a chamada de sistema, apenas resta fechar o arquivo através do comando 16.

3. Resultados e discussões

A execução do programa foi bem-sucedida, com todas as etapas ocorrendo de maneira esperada. Durante a fase de desenvolvimento, o valor de todos os registradores foi acompanhado e os resultados obtidos pelo código foram conferidos.

5. Conclusões

Esse trabalho foi desenvolvido com o objetivo de realizar operações aritméticas simples, a fim de calcular as raízes de uma equação quadrática usando a arquitetura de MIPS no programa de desenvolvimento MARS. Tal meta foi alcançada com êxito, de modo que os dois programas foram escritos e executados sem erros.

O desenvolvimento desse código foi um ótimo auxílio para o melhor entendimento de como a arquitetura do MIPS é organizada. Além de uma melhora nas habilidades de uso da plataforma MARS, também facilitou a fluência em assembly.

As habilidades adquiridas com esse simples projeto são uma forte evidência da importância do aprendizado de linguagens de baixo-nível para o desenvolvimento de um programador. Os conhecimentos aqui desenvolvidos serão utilizados em qualquer futuro projeto e desafio que será futuramente enfrentado no curso.