MAC 0216 - EP 1

Técnicas de Programação 1 - Linguagem de Montagem

Data de entrega: 18/09/2021 (sem extensões!)

1 Introdução

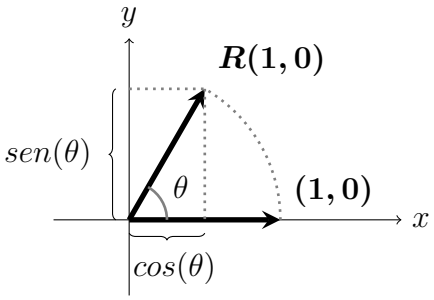
Neste exercício, você deve implementar um algoritmo, em MIPS, que faça a rotação de um vetor em um ângulo θ, por meio de matrizes de rotação. Para isso, também será necessário obter uma aproximação do seno de um dado ângulo. Esse tipo de operação de rotação é muito utilizada em programas que lidam com computação gráfica ou jogos eletrônicos.

2 Conceitos importantes

Para a implementação deste EP, além dos conteúdos abordados em sala de aula, será necessária a aplicação dos seguintes itens:

2.1 Matrizes de rotação

Uma matriz de rotação é uma matriz quadrada que, quando multiplicada por um vetor (representado por uma matriz coluna), rotaciona esse vetor no espaço em que ele está contido. O resultado dessa multiplicação é uma matriz coluna que contém as coordenadas do vetor rotacionado. Neste EP, utilizaremos apenas matrizes de rotação 2×2 e nos limitaremos a rotacionar um ângulo θ no $\mathbb{R}^2$. Para chegar à matriz que faz a rotação no sentido trigonométrico (anti-horário), podemos começar visualizando o que acontece quando rodamos esses vetores:



Temos que:

$$R(1,0) = (cos(\theta), sen(\theta))$$
$$R(0,1) = (-sen(\theta), cos(\theta))$$

Logo,

$$R = \begin{bmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{bmatrix}$$

A multiplicação da matriz de rotação pelo vetor dará como resultado o vetor rotacionado:

$$Rv = \begin{bmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \cdot cos(\theta) - y \cdot sen(\theta) \\ x \cdot sen(\theta) + y \cdot cos(\theta) \end{bmatrix}$$

2.2 Matrizes em MIPS

Para a representação de matrizes, disporemos os elementos da matriz linearmente em um vetor, de forma a simplificar o gerenciamento de memória. Dessa forma, uma matriz 2×2,

$$M = \begin{bmatrix} w & x \\ y & z \end{bmatrix}$$

que seria representada normalmente em código como $M = [[w, x], [y, z]]$, será representada como $M_{vet} = [w, x, y, z]$. Ou seja, a posição $M[i][j]$ da matriz será acessada com $M_{vet}[i \cdot \text{nCol} + j]$, onde nCol é o número de colunas da matriz.

Para a alocação de memória para a matriz temos duas opções. Disponibilizamos uma implementação para alocar uma matriz 2×2 de inteiros com cada uma delas. Você pode modifica-las e usá-las em seu EP.

- Alocação dinâmica:

  Para alocar dinamicamente, usaremos a chamada ao sistema `sbrk`, que recebe um inteiro x, aloca x bytes de memória e devolve um ponteiro para a memória alocada. Segue a implementação para uma matriz 2×2:

```
.text
  li $v0, 9 # identificador da chamada ao sistema operacional
  li $a0, 16 # número de bytes: 2(linhas) * 2(colunas) * 4(tamanho de um int)
  syscall # chama o sistema operacional
  move $s0, $v0 # guarda o endereço devolvido em s0
```

- Alocação estática:

  Para esse tipo de alocação, usaremos o identificador *.space* dentro da seção *.data*. Segue a alocação para uma matriz 2×2:

```
.data
  mat: .space 16 # número de bytes

.text
  la $s0, mat
```

Em seguida, como já temos o endereço de memória da matriz em $s0, podemos executar $M[i][j] = x$ usando:

```
li      $t1, i # i é o indice da linha que queremos acessar
sll     $t1, $t1, 1 # i * nCols(2)
addi    $t1, $t1, j # j é o indice da coluna que queremos acessar
sll     $t1, $t1, 2   # offset = indice do vetor * tamanho de um int (4)
add     $t1, $t1, $s0 # posicão da célula = offset + posição da matriz
li      $t2, x # x é o número que queremos inserir na matriz
sw      $t2, ($t1) # guarda na posição de memória
```

2.3 Números em Ponto flutuante em MIPS

Ao usar inteiros em MIPS, utilizamos a instrução li para carregar números imediatos. Porém, já que não há uma instrução correspondente para números imediatos em ponto flutuante, devemos usar a instrução analoga à lw (usada para carregar inteiros da memória): a instrução lwc1, que carrega dados da seção .data para os registradores. As instruções aritméticas em ponto flutuante

são semelhantes às operações em inteiros, mas com o sufixo .s no final, para precisão simples. O exemplo a seguir mostra a soma e a divisão entre dois valores de ponto flutuante, utilizando precisão simples.

```
.data
  # para inteiros
  int1: .word 5
  int2: .word 10
  int3: .word 20
  # correspondente para números em ponto flutuante de precisão simples
  float1: .float 5.0
  float2: .float 10.0
  float3: .float 20.5

.text
  # para inteiros
  lw $t0, int1
  lw $t1, int2
  lw $t2, int3
  # correspondente para números em ponto flutuante de precisão simples
  lwc1 $f0, float1
  lwc1 $f1, float2
  lwc1 $f2, float3

  #soma
  add $t3, $t1, $t2 # para inteiros
  add.s $f3, $f1, $f2 # para float

  # divisão
  div $t4, $t0, $t2 # para inteiros
  div.s $f4, $f0, $f2 # para float

  # conversão e tranferência entre registradores normais e flutuantes
  mtc1 $t1, $f1 # transferir bytes brutos de t1 para f1
  cvt.s.w $f1, $f1 # converte de inteiro para flutuante de precisão simples
  cvt.w.s $f1, $f1 # converte de volta para inteiro
  mfc1 $t1, $f1 # tranfere bytes brutos de volta
```

3 Descrição do exercício

A implementação do exercício deve ser dividida em funções. O objetivo final é construir uma função `rotaciona (v, θ)` que recebe um vetor e um ângulo e devolve o vetor rotacionado nesse ângulo. Para facilitar a organização, construa as seguintes funções auxiliares a essa função principal:

(a) Função que recebe um ângulo θ e devolve a matriz de rotação para aquele ângulo:

$$R = \begin{bmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{bmatrix}$$

(b) Função que recebe um ângulo θ e devolve uma estimativa de $sen(\theta)$ usando a série de Taylor:

$$sen(x) = \sum_{k=0}^{\infty} \frac{(-1)^k \cdot x^{2k+1}}{(2k+1)!}$$

Os termos maiores que 10^{-4} devem ser incluídos na soma. O cosseno pode ser obtido a partir do seno ou ter sua própria função auxiliar.

(c) Função que recebe uma matriz 2×2 e um vetor e devolve a multiplicação entre eles (esse será o vetor a ser rotacionado):

$$Rv = \begin{bmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \cdot cos(\theta) - y \cdot sen(\theta) \\ x \cdot sen(\theta) + y \cdot cos(\theta) \end{bmatrix}$$

Se você quiser criar outras funções auxiliares, não há problema nenhum.
Escreva, logo no início do seu código, um exemplo de chamada da função `rotaciona (v, θ)` para verificarmos que tudo está funcionando.

4 Recomendações Finais

Seu código deve ser **elegante, claro e eficiente**. Os nomes das variáveis, funções e rótulos devem ser o mais claros possíveis. Não misture línguas (faça tudo em português ou tudo em inglês, sem misturar as duas).

O trabalho deve ser realizado em grupos de 1 ou 2 alunos. A entrega será feita por meio do eDisciplinas. Caso seja feito em dupla, apenas um dos membros da dupla deve submeter o exercício no edisciplinas, mas o arquivo deve conter claramente o nome e NUSP de ambos os alunos. No caso de dupla, ambos os alunos devem obrigatoriamente participar ativamente da escrita do programa; vocês podem optar por fazer programação em par ou dividir o código de forma que cada um escreva uma parte e revise a outra parte.

Obviamente, *o plágio é terminantemente proibido*. Caso seja detectado plágio, todos os membros do grupo plagiador e todos os membros do grupo plagiado receberão nota -10 no exercício. Portanto, compartilhe seu código apenas com seu colega de grupo.

OPCIONAL: Se você quiser ganhar 1 ponto extra de bônus no seu EP, pode propor alguma melhoria no seu projeto e explicar bem nos comentários o que você fez para pleitear o bônus. Três coisas que me vêem à cabeça agora: (1) manipular vetores 3D com matrizes 3x3, (2) tocar uma musiquinha baseada de alguma forma nas coordenadas dos vetores de entrada e saída :-) e (3) imprimir uma representação gráfica do vetor antes e depois da rotação usando *ASCII art* (putz, viajei nessa, estou me superando nas ideias malucas!!! :-)