Assembly MIPS

Código verificador de palíndromo

Aluno: William Gabriel Y. A. Braga

Problema:
Verificar palíndromo

A chamada palindromo ('Socorram-me, subi no Onibus em Marrocos!'), por exemplo, deve retornar TRUE.

Matéria: Programação Funcional.
Nível da questão: Difícil

O objetivo do programa é, dada uma string, identificar se a frase é um palíndromo seguindo algumas restrições.

1. Você deve desconsiderar todos os caracteres que não sejam letras.

2. Tanto faz maiúsculas e minúsculas.

Vamos para o código…

Descrição do segmento ***.data***

```
1  .data
2
3  frase:          .space  101                                          # Reserva 101 Bytes de espaço na memória (100 caracteres + '\0')
4  msg0:           .asciiz "========== Verificador de Parindromo ==========\n" # Mensagem de saudação
5  msg1:           .asciiz "Digite uma frase (100 letras): "            # Mensagems de solicitação de entrada
6  TRUE:           .asciiz "\nTRUE"                                     # Texto de saída positiva
7  FALSE:          .asciiz "\nFALSE"                                    # Texto de saída negativa
```

- Espaço reservado para dados usados no programa:
  - *frase*: Aloca 101 bytes para a *string* de entrada.
  - *msg0* e *msg1*: Mensagens exibidas no terminal.
  - *TRUE* e *FALSE*: Resultados possíveis do programa.

Obs.: *.asciiz* significa texto terminado com “*\0*” no final

Fluxo inicial

```
main:

    #   >>>Mensagem de saudação---------

    la      $a0,            msg0                    # Carrega o endereço da primeira mensagem de saudação
    li      $v0,            4                       # Prepara para mostrar uma string no terminal

    syscall                                         # Mostra a mensagem no terminal

    #   ----------------------------


    #   >>>Mensagem de solicitação---------

    la      $a0,            msg1                    # Carrega o endereço da primeira mensagem de saudação
    li      $v0,            4                       # Prepara para mostrar uma string no terminal

    syscall                                         # Mostra a mensagem no terminal

    #   ---------------------------------

    #   Leitura do possível palíndromo ------
    la      $s0,            frase                   # Carrega o endereço da memória no registrador
    move    $a0,            $s0                     # Indica o endereço de entrada do usuário
    li      $a1,            101                     # Indica o tamanho do buffer
    li      $v0,            8                       # Especifica a função de leitura de string

    syscall                                         # Lê uma string do terminal
```

- Exibição de mensagens:
    1. Saudação inicial (*msg0*).
    2. Solicitação de entrada de uma frase (*msg1*).
- Leitura da entrada do usuário:
    1. Usa *syscall* para armazenar a string em frase.

A operação *la* significa “load address”. É responsável por guardar no registrador o endereço de onde na memória principal a *string* armazenada está.

Processamento da *String*

```
    #O endereço da frase está em $s0
    #Precisamos encontrar o caractere \n
    #Além disso, precisamos remover os caracteres indesejados. Após o uppercase, manteremos somente os caracteres entre
    #'A' e 'Z'

    li      $t0,            '\n'                                    # $t0 é o caractere que queremos encontrar
    li      $s1,            0                                       # $s1 é o tamanho da string informada
    li      $t3,            'a'                                     # Char code da letra 'a'
    li      $t4,            'z'                                     # Char code da letra 'z'
    move    $t1,            $s0                                     # Copia o endereço do primeiro caractere da frase
    li      $t5,            'A'                                     # Charcode de 'A'
    li      $t6,            'Z'                                     # Charcode de 'Z'
    move    $t7,            $s0                                     # Salva a posição correta do caractere


str_find_end:

    #$t2 =  $s0[x]
    lb      $t2,            0($t1)                                  # Pega o caractere da string na posição $t1
    beq     $t2,            $t0,    END_str_find_end                # Se encontrar o \n, para o loop. Senão...

    blt     $t2,            $t3,    after_uppercase                 # Se $t2 < 97, ele não precisa ser convertido. Senão, faz a próxima verificação
    bgt     $t2,            $t4,    after_uppercase                 # Se $t2 > 122, ^^^^^^^^^^. Senão, ele converte para caixa alta

    subi    $t2,            $t2,    32                              # Converte usando a tabela ASCII de caracteres

after_uppercase:

    blt     $t2,            $t5,    remover_caractere               # Se for um caractere antes de 'A', não salva o caractere
    bgt     $t2,            $t6,    remover_caractere               # Se for um caractere depois de 'Z', não salva o caractere

    sb      $t2,            0($t7)                                  # Substitui o caractere na memória pelo novo em caixa alta
    addi    $t7,            $t7,    1                               # Atualiza a nova posição da string final
    addi    $s1,            $s1,    1                               # Aumenta o tamanho da string em 1

remover_caractere:

    addi    $t1,            $t1,    1                               # Vai para a próxima posição da string digitada

    j       str_find_end                                            # Reinicia o loop


END_str_find_end:

    li      $t0,            '\0'                                    # Salva o valor do caractere \0 de fim de linha
    sb      $t0,            0($t7)                                  # Posiciona um caractere de fim de texto \0 no final da string
```

- Remoção de caracteres inválidos:
  - Somente letras de *'A'* a *'Z'* são mantidas.
- Conversão para maiúsculas:
  - Letras de *'a'* a *'z'* são convertidas usando a tabela *ASCII*.
- Substituição do caractere *\n* por *\0* para indicar o fim da *string*.

Benefício em baixo nível:

Controle direto sobre a manipulação de caracteres permite otimizações específicas, como reduzir *overhead* de funções de biblioteca.

Verificação de Palíndromo

```
#IDEIA A PARTIR DAQUI ---------------------------------------
#Perceba que agora todos os caracteres da palavra estão em maiúsculo. Isto é, o charcode de todos os caracteres devem estar entre 65 (A) e 90 (Z).
#A ideia é então pegar a frase e ir "comendo pelas beiradas". Exemplo: arra -> xrrx -> xxxx -> retorna TRUE no terminal

move    $a0,            $s0                             # Passa uma cópia do endereço do primeiro caractere da string como argumento
subi    $t1,            $s1,    1                       # Pega a última posição da string
add     $a1,            $t1,    $s0                     # Passa o endereço do último caractere da string como argumento

jal     func_palindromo                                 # Chama a função recursiva de palíndromo
```

Preparação

A ideia de "comer pelas beiradas" consiste em comparar simultaneamente os caracteres das extremidades da *string* e avançar em direção ao centro. Esse método é eficiente para verificar se uma string é um palíndromo, pois evita a necessidade de criar uma string invertida ou percorrer toda a cadeia de caracteres.

A função é implementada utilizando recursividade em cauda.

Verificação de Palíndromo

Recursividade em Cauda

A função chama a si mesma passando os ponteiros atualizados (avanço e recuo) sem empilhar estados adicionais, tornando o processo leve.

Benefícios da Recursão em Cauda

1. Otimização de Pilha:
   - Se o ambiente suportar otimização de *tail call*, não há empilhamento excessivo, tornando o processo tão eficiente quanto um laço.

2. Legibilidade:
   - A lógica se mantém limpa e fácil de entender, sem necessidade de controle manual da pilha.

3. Eficiência:
   - A recursão em cauda mantém o desempenho estável, mesmo com *strings* longas.

A abordagem de "comer pelas beiradas" com recursão em cauda é mais eficiente e elegante, especialmente em Assembly, pois utiliza menos recursos de memória e processa apenas o necessário para determinar se a *string* é um palíndromo.

Verificação de Palíndromo

```
1  func_palindromo:
2      subi    $sp,            $sp,    4                   # Abre espaço na pilha
3      sw      $ra,            0($sp)                      # Guarda o endereço de retorno na pilha
4
5      bgt     $a1,            $a0,    recursivo_palindromo    # Caso base : $a1 <= $a0. Recursividade: em recursivo_palindromo
6      li      $v0,            1                           # Se for um palíndromo, a função retorna 1
7
8      lw      $ra,            0($sp)                      # Recupera o endereço de retorno da função
9      addi    $sp,            $sp,    4                   # Libera o espaço alocado na pilha
10     jr      $ra                                         # Encerra a execução da recursividade
11
12 recursivo_palindromo:                                   #>>>Recursividade da função
13
14     lb      $t1,            0($a0)                      # Carrega o caractere na posição $a0
15     lb      $t2,            0($a1)                      # Carrega o caractere na posição $a1
16     bne     $t1,            $t2,    diferente           #Verifica se os caracteres são diferentes. Se não forem...
17
18     addi    $a0,            $a0,    1                   # Atualiza o argumento acumulador 1 #2147479548 2147479544
19     subi    $a1,            $a1,    1                   # Atualiza o argumento acumulador 2
20
21     jal     func_palindromo                             #  Chamada recursiva da função
22
23     lw      $ra,            0($sp)                      # Recupera o endereço da chamada anterior da função
24     addi    $sp,            $sp,    4                   # Libera o espaço na pilha
25
26     jr      $ra                                         # Retorna para a chamada anterior
27
28
29 diferente:
30     li      $v0,            0                           # Se não for um palíndromo, a função retorna 0
31
32     addi    $sp,            $sp,    4                   # Libera o espaço na pilha
33
34     jr      $ra                                         # Encerra a execução da recursividade
```

Resultados

```
    move    $s2,            $v0                         # Pegando o retorno da função

    bne     $s2,            $zero,  eh_palindromo       # Se $s2 != 0 (ou seja, é palindromo) faz um salto para mostrar a mensagem "TRUE". Senão...
    la      $a0,            FALSE                       # Carrega a palavra "FALSE"
    j       mostrar_resultado                           # Pula para a parte de mostrar o resultado

eh_palindromo:

    la      $a0,            TRUE                        # Carrega a palavra "TRUE"

mostrar_resultado:

    li      $v0,            4                           # Configura a operação de mostrar uma string no terminal
    syscall                                             # Mostra o resultado final



end:

    li      $v0,            10                          # Prepara o programa para ser finalizado
    syscall                                             # Finaliza o programa
```

Exibição do resultado final:

TRUE: A frase é um palíndromo *FALSE*: A frase não é um palíndromo.

Comparação com a linguagem C

Implementação em baixo nível

1. **Benefícios**:
    - Controle granular sobre a memória e o fluxo do programa.
    - Eliminação de dependências de biblioteca, reduzindo o tamanho do programa.

Exemplo:

No C, funções como *toupper()* e *isalpha()* introduzem *overhead* que foi eliminado em Assembly.

2. **Desvantagens em Assembly**:
    - Maior complexidade de implementação.
    - Requereu mais tempo para escrever e depurar.

Conclusão

Resumo

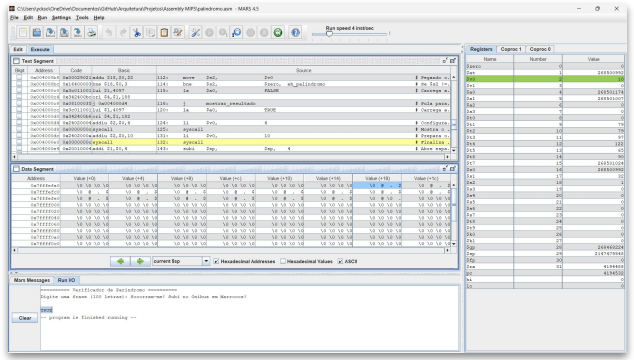
O programa demonstra como Assembly permite manipular strings e implementar algoritmos complexos com controle detalhado.

A verificação de palíndromos ilustra o poder da recursão em cauda em baixo nível.

Pontos de aprendizado

- Manipulação direta de memória.
- Conversão e limpeza de strings sem funções auxiliares.
- Implementação eficiente de algoritmos recursivos.

Vídeo demonstrativo:
https://youtu.be/XSHyXKEACXQ



Repositório do GitHub da disciplina:

https://github.com/Yckson/Arquitetura

Obrigado!👋