PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS

ESCOLA POLITÉCNICA E DE ARTES

ARQUITETURA DE COMPUTADORES

LUCAS PEREIRA NUNES

Este código em Assembly tem como objetivo solicitar, comparar e armazenar uma senha fornecida pelo usuário, realizando uma simples transformação na senha antes de exibi-la como resultado final. A seguir, segue o detalhamento do funcionamento de cada seção do código, explicando as funções essenciais e o fluxo de execução.

Estrutura do Programa

O programa está organizado em três seções principais:

1. **.data**: Contém as mensagens de texto e variáveis fixas.
2. **.bss**: Armazena buffers para a entrada do usuário, onde as senhas serão temporariamente armazenadas.
3. **.text**: Contém o código executável, onde ocorrem as operações de leitura, comparação e processamento das senhas.

Seção `.data`

Nesta seção, temos as variáveis de dados estáticos, como as mensagens a serem exibidas ao usuário:

- **msg_input**: "Digite a senha: " - exibe a solicitação da primeira senha.
- **msg_confirm**: "Confirme a senha: " - exibe a solicitação para a confirmação da senha.
- **msg_error**: "Senhas nao conferem." - mensagem de erro em caso de senhas diferentes.
- **msg_success**: "Senha armazenada: " - mensagem de sucesso ao exibir a senha após a transformação.
- **senha_armazenada**: Buffer de 16 bytes para armazenar a senha processada.

```
section .data

    msg_input db "Digite a senha: ", 0

    msg_confirm db "Confirme a senha: ", 0

    msg_error db "Senhas nao conferem.", 0

    msg_success db "Senha armazenada: ", 0

    senha_armazenada db 16 ; Buffer para armazenar a senha processada
```

Seção `.bss`

Esta seção reserva espaço para variáveis não inicializadas:

- **senha1**: Buffer de 16 bytes para armazenar a primeira senha digitada.
- **senha2**: Buffer de 16 bytes para armazenar a confirmação da senha.

```
section .bss

    senha1 resb 16 ; Armazena a primeira senha

    senha2 resb 16 ; Armazena a segunda senha (confirmação)
```

Seção `.text`

Aqui temos o código executável, com as instruções principais e as funções auxiliares.

1. Início do Programa (`_start`)

- O programa começa exibindo a mensagem `msg_input` e solicita a primeira senha do usuário, que é lida e armazenada no buffer `senha1`.
- Em seguida, exibe `msg_confirm` para solicitar a confirmação da senha, armazenando o resultado em `senha2`.
- Após a entrada das duas senhas, o programa chama a função `compare_senhas`, que verifica se ambas são iguais. Se forem diferentes, o programa exibe a mensagem de erro e termina. Se forem iguais, a senha é processada.

```
section .text

    global _start


_start:

    ; Solicitar a primeira senha

    mov edx, 16 ; Tamanho da mensagem

    mov ecx, msg_input ; Mensagem a ser exibida

    call print_msg
```

- **`print_msg`**: Função para imprimir mensagens no console, usando a syscall `write` (número 4) com `stdout`.

```
; Função para imprimir mensagens
print_msg:
    mov eax, 4 ; syscall write
    mov ebx, 1 ; stdout
    int 0x80
    ret
```

continuação:

```
    mov ecx, senha1 ; Buffer para a senha
    call read_input
```

- **`read_input`**: Lê a entrada do teclado usando a syscall `read` (número 3) com `stdin`.

```
; Função para ler dados do teclado
read_input:
    mov eax, 3 ; syscall read
    mov ebx, 0 ; stdin
    mov edx, 16 ; Número máximo de bytes a ler
    int 0x80
    ret
```

continuação:

```
    call remove_newline ; Remove o caractere de nova linha
```

- **`remove_newline`**: Remove o caractere de nova linha (`\n`) que é adicionado automaticamente ao final da entrada do usuário.

```
; Função para remover o caractere de nova linha
remove_newline:
    mov esi, ecx ; Apontar para o buffer da senha
    mov ecx, 16 ; Verificar até 16 caracteres
remove_loop:
```

```
    cmp byte [esi], 10 ; Verifica se é '\n' (valor ASCII 10)
    je found_newline
    cmp byte [esi], 0 ; Verifica se é o fim da string
    je end_remove_newline
    inc esi
    loop remove_loop
end_remove_newline:
    ret
found_newline:
    mov byte [esi], 0 ; Substitui '\n' por NULL
    ret
```

continuação:

```
    ; Solicitar a confirmação da senha
    mov edx, 17 ; Tamanho da mensagem
    mov ecx, msg_confirm ; Mensagem a ser exibida
    call print_msg
    mov ecx, senha2 ; Buffer para a segunda senha
    call read_input
    call remove_newline ; Remove o caractere de nova linha
```

2. Comparação das Senhas

A função `compare_senhas` compara byte a byte as senhas `senha1` e `senha2`:

- Se todos os bytes forem iguais, a função retorna `eax = 1`.
- Se houver diferença em qualquer byte, `eax = 0` e o fluxo do programa é desviado para `erro_senhas`.

```
    ; Comparar as senhas
    call compare_senhas
    cmp eax, 0 ; Verifica se são diferentes
    je erro_senhas
```

```
; Função para comparar duas senhas
compare_senhas:
    mov esi, senha1
    mov edi, senha2
    mov ecx, 16
compara_loop:
    lodsb
    scasb
    jne senhas_diferentes
    loop compara_loop
    mov eax, 1 ; Senhas são iguais
    ret
senhas_diferentes:
    mov eax, 0 ; Senhas diferentes
    ret
```

```
erro_senhas:
    ; Exibir mensagem de erro
    mov edx, 20 ; Tamanho da mensagem
    mov ecx, msg_error
    call print_msg
    call exit
```

continuação:

3. Processamento da Senha

Se as senhas forem iguais, o código entra em uma fase de transformação da senha. Ele realiza uma operação de soma no valor de cada byte da senha com o número 5. Essa transformação é feita no bloco `altera_senha` usando o registrador `al` para carregar e modificar cada byte da senha. O resultado processado é armazenado em `senha_armazenada`.

```
; Alterar a senha
    mov esi, senha1
    mov edi, senha_armazenada
    mov ecx, 16



altera_senha:
    lodsb
    add al, 5
    stosb
    loop altera_senha
```

4. Exibição da Senha Processada

Depois de alterar a senha, o programa exibe a mensagem `msg_success`, seguida da senha transformada armazenada no buffer `senha_armazenada`.

```

    ; Exibir a senha armazenada após a transformação
    mov edx, 19 ; Tamanho da mensagem
    mov ecx, msg_success
```

```
    call print_msg

    mov edx, 16 ; Tamanho da senha processada

    mov ecx, senha_armazenada ; Exibe a senha processada

    call print_msg
```

5. Finalização

O programa termina chamando a função `exit`, que invoca a syscall `exit` do sistema operacional, encerrando a execução.

```
    ; Finalizar o programa

    call exit
```

- **`exit`**: Termina o programa usando a syscall `exit`.

```
exit:

    mov eax, 1 ; syscall exit

    xor ebx, ebx

    int 0x80
```

Código Completo:

```
section .data
    msg_input db "Digite a senha: ", 0
    msg_confirm db "Confirme a senha: ", 0
    msg_error db "Senhas nao conferem.", 0
    msg_success db "Senha armazenada: ", 0
    senha_armazenada db 16 ; Buffer para armazenar a senha processada

section .bss
    senha1 resb 16 ; Armazena a primeira senha
    senha2 resb 16 ; Armazena a segunda senha (confirmação)

section .text
    global _start

_start:
    ; Solicitar a primeira senha
    mov edx, 16 ; Tamanho da mensagem
    mov ecx, msg_input ; Mensagem a ser exibida
    call print_msg
    mov ecx, senha1 ; Buffer para a senha
    call read_input
    call remove_newline ; Remove o caractere de nova linha

    ; Solicitar a confirmação da senha
    mov edx, 17 ; Tamanho da mensagem
    mov ecx, msg_confirm ; Mensagem a ser exibida
    call print_msg
    mov ecx, senha2 ; Buffer para a segunda senha
    call read_input
    call remove_newline ; Remove o caractere de nova linha

    ; Comparar as senhas
    call compare_senhas
    cmp eax, 0 ; Verifica se são diferentes
    je erro_senhas

    ; Alterar a senha
    mov esi, senha1
    mov edi, senha_armazenada
```

```
    mov ecx, 16
altera_senha:
    lodsb
    add al, 5
    stosb
    loop altera_senha

    ; Exibir a senha armazenada após a transformação
    mov edx, 19 ; Tamanho da mensagem
    mov ecx, msg_success
    call print_msg
    mov edx, 16 ; Tamanho da senha processada
    mov ecx, senha_armazenada ; Exibe a senha processada
    call print_msg

    ; Finalizar o programa
    call exit

erro_senhas:
    ; Exibir mensagem de erro
    mov edx, 20 ; Tamanho da mensagem
    mov ecx, msg_error
    call print_msg
    call exit

; Função para imprimir mensagens
print_msg:
    mov eax, 4 ; syscall write
    mov ebx, 1 ; stdout
    int 0x80
    ret

; Função para ler dados do teclado
read_input:
    mov eax, 3 ; syscall read
    mov ebx, 0 ; stdin
    mov edx, 16 ; Número máximo de bytes a ler
    int 0x80
    ret

; Função para remover o caractere de nova linha
remove_newline:
    mov esi, ecx ; Apontar para o buffer da senha
```

```
    mov ecx, 16 ; Verificar até 16 caracteres
remove_loop:
    cmp byte [esi], 10 ; Verifica se é '\n' (valor ASCII 10)
    je found_newline
    cmp byte [esi], 0 ; Verifica se é o fim da string
    je end_remove_newline
    inc esi
    loop remove_loop
end_remove_newline:
    ret
found_newline:
    mov byte [esi], 0 ; Substitui '\n' por NULL
    ret

; Função para comparar duas senhas
compare_senhas:
    mov esi, senha1
    mov edi, senha2
    mov ecx, 16
compara_loop:
    lodsb
    scasb
    jne senhas_diferentes
    loop compara_loop
    mov eax, 1 ; Senhas são iguais
    ret

senhas_diferentes:
    mov eax, 0 ; Senhas diferentes
    ret

; Função para encerrar o programa
exit:
    mov eax, 1 ; syscall exit
    xor ebx, ebx
    int 0x80
```