UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
COLEGIADO DE CIÊNCIA DA COMPUTAÇÃO

Alunos: André Henrique dos Santos da Silva e Lucas Ivanov Costa
Docente: Gabriela Stein
Matéria: Linguagem de Montagem

1. Introdução

Este relatório descreve a implementação de uma calculadora em Assembly para ambiente 64 bits, que realiza operações aritméticas básicas (soma, subtração, multiplicação e divisão) utilizando funções externas da linguagem C, como printf, fopen, fprintf, fclose e atof. O código está organizado em seções (data, bss e text) e adota um fluxo modular que facilita a manutenção e o debug.

2. Estrutura e Fluxo do Código

Seção .data:
Declaração das strings de controle.

```
section .data
    modoAberturaArquivo      : db "a", 0
    nomeArquivo              : db "saida.txt", 0
    ctrlArquivo              : db "%.2lf %c %.2lf = %.2lf", 10, 0
    ctrlArquivoNaoDisponivel : db "%.2lf %c %.2lf = funcionalidade não disponível", 10, 0
    ctrlEntradaInvalida      : db "Operando de entrada inválido = %c", 10, 0
    eqMsg                    : db "Uso: ./calculadora operando1 operador operando2", 10, 0
```

Seção .bss:
Reserva de memória para variáveis.

```
section .bss
    op1       : resd 1
    operacao  : resd 1
    op2       : resd 1
    resultado : resd 1
    arq       : resq 1
```

op1, op2: operandos da operação.
operacao: caractere que indica a operação aritmética.
resultado: armazenamento do resultado final.
arq: variável que guarda o endereço do arquivo aberto para escrita.

A função main inicia configurando o ambiente de execução, estabelecendo o stack frame ao salvar o valor do registrador rbp e ajustando o ponteiro da pilha. Em seguida, o código armazena o endereço dos argumentos passados pela linha de comando e verifica se o número de argumentos é suficiente para proceder com a operação (4 argumentos: o nome do programa e os três parâmetros esperados, op1, operacao e op2). Se o número de argumentos for menor que o esperado, uma mensagem de uso, declarada na variável eqMsg, é exibida utilizando a função printf, informando ao usuário a forma correta de invocar a calculadora, e o programa é finalizado com um código de erro.

```
main:
    push rbp
    mov rbp, rsp

    mov rbx, rsi

    cmp rdi, 4
    jge continua

    mov rdi, eqMsg
    call printf
    jmp erro
```

```
_fim:
    mov rdi, qword [arq]
    call fclose

    mov rsp, rbp
    pop rbp

    mov rax, _exit
    mov rdi, 0
    syscall

erro:
    mov rsp, rbp
    pop rbp
    mov rax, _exit
    mov rdi, 1
    syscall
```

Caso os argumentos estejam corretos, na função main utilizamos a função fopen para abrir o arquivo de saída em modo de edição e armazenar o ponteiro do arquivo em uma variável específica. Posteriormente, utilizados a função atof para converter os operandos de entrada, que estão em formato

de string, para valores numéricos do tipo float. Durante esse período, o operador, extraído como um caractere da string de entrada, é capturado e armazenado para definir qual operação aritmética será executada.

```
continua:
    ;Abre o arquivo
    xor rax, rax
    mov rdi, nomeArquivo
    mov rsi, modoAberturaArquivo
    call fopen
    mov [arq], rax

    ;Converte o primeiro operando para float
    mov rax, [rbx+8]
    mov rdi, rax
    call atof
    cvtsd2ss xmm0, xmm0
    movss [op1], xmm0

    ;Obtém o operador
    mov rax, [rbx+16]
    movzx r9, byte [rax]
    mov [operacao], r9b

    ;Converte o segundo operando para float
    mov rax, [rbx+24]
    mov rdi, rax
    call atof
    cvtsd2ss xmm0, xmm0
    movss [op2], xmm0

    movss xmm0, [op1]
    movss xmm1, [op2]

    movzx r9, byte [operacao]
```

Após a conversão dos operandos e a captura do operador, o fluxo de execução segue para uma série de comparações, onde o código determina se a operação solicitada é soma, subtração, multiplicação ou divisão. Dependendo do operador identificado, o programa redireciona a execução para a função correspondente, como callSoma, callSub, callMult ou callDiv. Optamos por não fazer a chamada diretamente naquela parte do código, fizemos labels somente para o CALL de cada função, pois foi assim que funcionou.

```
    cmp r9b, 'a'
    je callSoma

    cmp r9b, 's'
    je callSub

    cmp r9b, 'm'
    je callMult

    cmp r9b, 'd'
    je callDiv

    call entradaInvalida
```

Para a operação de subtração, fizemos uma troca de valores entre os operandos para garantir que a operação ocorra como na especificação, com op2 - op1. Além disso, na função de divisão, fizemos uma verificação extra para evitar a divisão por zero. Se o divisor for zero, o fluxo é direcionado para uma rotina que escreve uma mensagem informando que a funcionalidade não está disponível.

```
callSub:
    call sub
    ;Troca valores de xmm1 e xmm0 para escrever, pois é op2 - op1
    movss xmm4, [op1]
    movss xmm5, [op2]
    movss [op1], xmm5
    movss [op2], xmm4
    call disponivel
    jmp _fim
```

```
div:
    push rbp
    mov rbp, rsp

    mov r9b, "/"
    mov [operacao], r9b

    ; Verifica se o divisor é zero
    cvtss2si r10, xmm1
    mov r11, 0
    cmp r10, r11
    je divPorZero

    vdivss xmm2, xmm0, xmm1
    call disponivel

    mov rsp, rbp
    pop rbp
    ret

divPorZero:
    call naoDisponivel
    mov rsp, rbp
    pop rbp
    ret
```

Por fim, após a realização da operação aritmética, utilizamos a função fprintf para enviar o resultado (ou a mensagem de erro, conforme o caro) para o arquivo de saída, utilizando as strings de controle definidas para formatação. A função main então conclui sua execução fechando o arquivo com fclose, restaurando o stack frame e realizando a saída do programa através de uma syscall.