PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA E DE ARTES
ENGENHARIA DE COMPUTAÇÃO

LUCAS PEREIRA NUNES
GABRIEL DERREL MARTINS SANTEE

ATIVIDADE N2
ADIÇÃO DE DOIS NÚMEROS

GOIÂNIA
2024

Sumário

Código

Arquivo AtividadeN2_Main.asm

```
section .data
    msg1 db "Insira o primeiro numero: "
    msg1_len equ $ - msg1
    msg2 db "Insira o segundo numero: "
    msg2_len equ $ - msg2
    msg_resultado db "Soma: "
    msg_resultado_len equ $ - msg_resultado
    pular_linha_str db 10,10
    pular_linha_len equ $ - pular_linha_str
    buffer1 db 10 ; String do primeiro número
    buffer2 db 10 ; String do segundo número
    buffer_soma db 10 ; String da soma
    buffer_len equ 10 ; Tamanho máximo do buffer

section .bss
    verificar_erro resb 1 ; Byte que verifica se tem algum erro

section .text
    global _start
    extern print_msg, read_msg, atoi, itoa, pularlinha
    global buffer_len, pular_linha_str, pular_linha_len, verificar_erro
_start:
    ; SYSCALL WRITE 1: "Insira o primeiro numero:"
    leitura1:
        mov byte [verificar_erro], 0 ; Zera o verificador de erro

        push 4                      ; syscall number para sys_write por
        ↪ pilha
        push 1                      ; file descriptor 1 (stdout) por pilha
        push msg1                   ; endereço da mensagem de prompt por
        ↪ pilha
        push msg1_len               ; tamanho da mensagem de prompt por
        ↪ pilha
        call print_msg              ; chamada da função de impressão de
        ↪ mensagem
            ; "Insira o primeiro numero:"

        ; limpando a pilha
```

```
35          pop rax
36          pop rax
37          pop rax
38          pop rax
39
40          ; SYSCALL READ 1
41          mov rcx, buffer1            ; aponta para o buffer do primeiro
            ↪ número
42          call read_msg               ; chamada da função de leitura da
            ↪ entrada
43
44          ; Converte string para número inteiro
45          mov rsi, buffer1            ; apontando para o início da string do
            ↪ primeiro número
46          xor rax, rax                ; zera o valor acumulador(neste caso rax
            ↪ é o acumulador)
47          xor rcx, rcx                ; contador de dígitos lidos
48          call atoi                   ; chamada da função atoi (ascii para
            ↪ inteiro)
49          cmp byte [verificar_erro], 1 ; verifica se tem algum erro
50          je leitura1                  ; se tiver erro, volta para leitura1
51          push rax             ; armazena o resultado em num1
52
53      ; SYSCALL WRITE 2: "Insira o segundo numero:"
54      leitura2:
55          mov byte [verificar_erro], 0 ; Zera o verificador de erro
56
57          push 4                      ; syscall number para sys_write por
            ↪ pilha
58          push 1                      ; file descriptor 1 (stdout) por pilha
59          push msg2                   ; endereço da mensagem de prompt por
            ↪ pilha
60          push msg2_len               ; tamanho da mensagem de prompt por
            ↪ pilha
61          call print_msg              ; chamada da função de impressão de
            ↪ mensagem
                ; "Insira o segundo numero:"
62
63          ; limpando a pilha
64          pop rax
65          pop rax
66          pop rax
```

```
67          pop rax
68          pop rax
69
70          ; SYSCALL READ 2
71          mov rcx, buffer2            ; aponta para o buffer da entrada
72          call read_msg               ; chamada da função de leitura da
            ↪ entrada
73
74          ; Converte string para número inteiro
75          mov rsi, buffer2            ; apontando para o início do buffer
76          xor rax, rax                ; zera o valor acumulador
77          xor rcx, rcx                ; contador de dígitos lidos
78          call atoi                   ; chamada da função atoi (ascii para
            ↪ inteiro)
79          cmp byte [verificar_erro], 1 ; verifica se tem algum erro
80          je leitura2                  ; se tiver erro, volta para leitura2
81
82      ; Soma dos dois números
83      pop rbx                     ; pegar o valor de num1 da pilha
84      add rax, rbx                ; soma os dois valores
85
86      ; Converte o resultado da soma(que agora está em rax) de volta para
        ↪ string
87      mov rbx, rax                ; copia o valor da soma para rbx
88      mov rdi, buffer_soma        ; guarda o inicio da string buffer_soma
89      add rdi, buffer_len         ; faz o ponteiro rdi apontar para o final da
        ↪ string buffer_soma
90      call itoa                   ; chamada da função itoa (inteiro para
        ↪ ascii)
91
92      ; Exibe "Soma: "
93      push 4                  ; syscall number for sys_write
94      push 1                  ; file descriptor 1 (stdout)
95      push msg_resultado      ; endereço da mensagem de resultado
96      push msg_resultado_len  ; tamanho da mensagem de resultado
97      call print_msg                  ; chamada de sistema para escrever
        ↪ "Soma: "
98
99      ; limpando a pilha
100     pop rax
101     pop rax
```

```
    pop rax
    pop rax

    ; Exibe o número convertido para string
    push 4                  ; syscall number for sys_write
    push 1                  ; file descriptor 1 (stdout)
    push rdi                ; aponta para o início do número no buffer
    push buffer_len         ; tamanho máximo (exibe até o '\0')
    call print_msg                  ; chamada de sistema para escrever o
    ↪ número

    ; limpando a pilha
    pop rax
    pop rax
    pop rax
    pop rax

    ; Exibe uma quebra de linha
    call pularlinha

Fim:
    mov rax, 1                  ; Syscall number para sys_exit
    xor rbx, rbx                ; código de saída 0
    int 0x80                    ; chamada de sistema para sair
```

Arquivo AtividadeN2_Libs.asm

```
global print_msg, read_msg, atoi, itoa, pularlinha
extern buffer_len, pular_linha_str, pular_linha_len, verificar_erro

print_msg:
    push rbp                    ; coloca rbp no topo da pilha
    mov rbp, rsp                ; rbp e rsp representam o topo da pilha
    mov rdx, [rbp + 16]          ; acessa 8 bytes abaixo da pilha para
    ↪ acessar o valor pressuposto para rdx
    mov rcx, [rbp + 24]         ; valor pressuposto para rcx
    mov rbx, [rbp + 32]         ; valor pressuposto para rbx
    mov rax, [rbp + 40]         ; valor pressuposto para rax
    int 0x80                    ; chamada de sistema para imprimir
    pop rbp                     ; retira rbp da pilha e rsp volta a ser o
    ↪ unico topo da pilha
```

```
13      ret
14
15  read_msg:
16      mov rax, 3                  ; syscall number para sys_read
17      mov rbx, 0                  ; file descriptor 0 (stdin)
18      mov rdx, buffer_len         ; tamanho do buffer
19      int 0x80                    ; chamada de sistema para ler a string
20      ret
21
22  atoi:
23      movzx rbx, byte [rsi]       ; carrega o próximo caractere
24      cmp rbx, 10                 ; verifica se é '\0' (final da string)
25      je finish_conversion
26      ; verifica se o caractere é um dígito válido
27      cmp rbx, '0'
28      jb erro                     ; verifica se é menor que '0'
29      cmp rbx, '9'
30      ja erro                     ; verifica se é maior que '9'
31      sub rbx, '0'                ; converte caractere ASCII para número
32
33      imul rax, rax, 10           ; multiplica o valor atual por 10
34      add rax, rbx                ; adiciona o dígito convertido
35      inc rsi                     ; incrementa para ir para o próximo
        ↪ caractere
36      jmp atoi
37      ; O número convertido está em rax
38      finish_conversion:
39          ret
40      erro:
41          mov byte [verificar_erro], 1
42          ret
43
44  itoa:
45      dec rdi                     ; avança para o próximo caractere (de trás
        ↪ para frente)
46      mov rax, rbx                ; armazena o valor do quociente(rbx) atual
        ↪ em rax
47      mov rdx, 0                  ; zera o registrador rdx, será usado para o
        ↪ resto da divisão
48      mov rcx, 10                 ; seta o divisor, ou seja, qual base(10
        ↪ nesste caso)
```

```
49      div rcx                     ; divide rax por 10
50      add dl, '0'                 ; converte o resto para ASCII
51      mov [rdi], dl               ; armazena na string
52      mov rbx, rax                ; armazena o quociente para a próxima
        ↪ iteração
53      test rbx, rbx               ; verifica se o valor é 0
54      jnz itoa
55      ret
56
57  pularlinha:
58      push 4                      ; syscall number para sys_write
59      push 1                      ; file descriptor 1 (stdout)
60      push pular_linha_str         ; endereço da mensagem de prompt por pilha
61      push pular_linha_len         ; tamanho da mensagem de prompt por pilha
62      call print_msg              ; chamada da função de impressão de mensagem
63      pop rax
64      pop rax
65      pop rax
66      pop rax
67      ret
```

Explicação do Código

Arquivo AtividadeN2_Libs.asm

O seguinte trecho é responsável por definir quais funções estão no arquivo atual e quais variáveis estão em um arquivo externo.

```
1   global print_msg, read_msg, atoi, itoa, pularlinha
2   extern buffer_len, pular_linha_str, pular_linha_len, verificar_erro
```

O seguinte trecho define a função responsável por imprimir uma string na tela. Primeiramente colocamos o valor da base da pilha na pilha, isso é necessário pois no passo seguinte o valor do topo da pilha é movido para rsb, isso é feito para termos um ponto fixo onde podemos acessar os valores salvos na pilha.
Para acessar os valores da pilha fazemos uma aritmética de ponteiro, pegamos da partir de 16 pois queremos os valores a partir da terceira posição, já que a primeira posição guarda o valor da base da pilha e a segunda guarda o endereço de retorno. Após passar os valores necessários para os registradores, fazemos a chamada ao sistema, ao chamar o sistema a String será impressa na tela.

Depois disso, resgatamos o valor da base da pilha de volta para o registrador rbp e damos o retorno.

```
print_msg:
    push rbp                    ; coloca rbp no topo da pilha
    mov rbp, rsp                ; rbp e rsp representam o topo da pilha
    mov rdx, [rbp + 16]          ; acessa 8 bytes abaixo da pilha para
    ↪ acessar o valor pressuposto para rdx
    mov rcx, [rbp + 24]         ; valor pressuposto para rcx
    mov rbx, [rbp + 32]         ; valor pressuposto para rbx
    mov rax, [rbp + 40]         ; valor pressuposto para rax
    int 0x80                    ; chamada de sistema para imprimir
    pop rbp                     ; retira rbp da pilha e rsp volta a ser o
    ↪ unico topo da pilha
    ret
```

O seguinte trecho é responsável por ler uma String do teclado, buffer_len é uma macro definida em outro arquivo e a string que receberá o que foi lido está foi passada antes de chamar a função.

```
read_msg:
    mov rax, 3                  ; syscall number para sys_read
    mov rbx, 0                  ; file descriptor 0 (stdin)
    mov rdx, buffer_len         ; tamanho do buffer
    int 0x80                    ; chamada de sistema para ler a string
    ret
```

O seguinte trecho é responsável por converter um número em ASCII para um número inteiro. O registrador rsi guarda o endereço da string a ser convertida, movemos o caractere atual para o registrador rbx utilizando o movzx(nstrução utilizada para mover dados de um registrador para outro registrador, expandindo o valor original com zeros para preenchê-lo totalmente no registrador de destino).
Agora verificamos se o caractere está entre ’0’ e ’9’, se estiver o código continua, senão pula para a label erro, seta a variável de verificação de erro como 1 sai. Após passar pela verificação, subtraimos ’0’ do caractere para obter o seu valor inteiro, após isso multicamos rax por 10(rax é um acumulador), após isso adicionamos ao acumulador o caractere convertido e passamos para o próximo caractere.
Portanto, digamos que queremos converter a string ”123”, os valores de rax serão: rax=0, rax=1, rax=10, rax=12, rax=120, rax=123.

```
atoi:
    movzx rbx, byte [rsi]       ; carrega o próximo caractere
    cmp rbx, 10                 ; verifica se é '\0' (final da string)
    je finish_conversion
    ; verifica se o caractere é um dígito válido
    cmp rbx, '0'
    jb erro                     ; verifica se é menor que '0'
    cmp rbx, '9'
    ja erro                     ; verifica se é maior que '9'
    sub rbx, '0'                ; converte caractere ASCII para número

    imul rax, rax, 10           ; multiplica o valor atual por 10
    add rax, rbx                ; adiciona o dígito convertido
    inc rsi                     ; incrementa para ir para o próximo
    ↪ caractere
    jmp atoi
    ; O número convertido está em rax
    finish_conversion:
        ret
    erro:
        mov byte [verificar_erro], 1
        ret
```

O seguinte trecho é responsável por converter um inteiro para uma string. O algorítmo funciona da seguinte forma, rdi guarda a ultima posição da string de destino, isso é necessário pois a inclusão dos caracteres ocorre na ordem inversa, após irmos para a posição anterior, guardamos o quociente anterior(em rbx) em rax, caso seja a primeira iteração o quociente será o inteiro a ser convertido, zeramos o registrador que guardará o resto(rdx) e salvamos a base de converção(nesse caso base 10) em rcx.
Dividimos rax por rcx e o resultado será guardado em dois registradores, rax(quociente) e rdx(resto), adicionamos ’0’ ao resto para converte-lo para ASCII e o salvamos o caractere convertido para a posição atual da string; salvamos o quociente em rbx para a próxima iteração. Agora testamos se o quociente é 0, se for a converção terminou, senão a converção continua.

```
itoa:
    dec rdi                     ; avança para o próximo caractere (de trás
    ↪ para frente)
    mov rax, rbx                ; armazena o valor do quociente(rbx) atual
    ↪ em rax
    mov rdx, 0                  ; zera o registrador rdx, será usado para o
    ↪ resto da divisão
```

```
    mov rcx, 10                 ; seta o divisor, ou seja, qual base(10
    ↪ nesste caso)
    div rcx                     ; divide rax por 10
    add dl, '0'                 ; converte o resto para ASCII
    mov [rdi], dl               ; armazena na string
    mov rbx, rax                ; armazena o quociente para a próxima
    ↪ iteração
    test rbx, rbx               ; verifica se o valor é 0
    jnz itoa
    ret
```

O seguinte trecho é responsável pela quebra de linha, para isso imprimimos a string pular_linha_str passando os valores pela pilha e chamando a função se imprimir uma mensagem na tela.

```
pularlinha:
    push 4                      ; syscall number para sys_write
    push 1                      ; file descriptor 1 (stdout)
    push pular_linha_str         ; endereço da mensagem de prompt por pilha
    push pular_linha_len         ; tamanho da mensagem de prompt por pilha
    call print_msg              ; chamada da função de impressão de mensagem
    pop rax
    pop rax
    pop rax
    pop rax
    ret
```

Arquivo AtividadeN2_Main.asm

O Seguinte trecho é responsável por definir as variáveis e definir quais são as variáveis globais e quais são as funções externas.

```
section .data
    msg1 db "Insira o primeiro numero: "
    msg1_len equ $ - msg1
    msg2 db "Insira o segundo numero: "
    msg2_len equ $ - msg2
    msg_resultado db "Soma: "
    msg_resultado_len equ $ - msg_resultado
    pular_linha_str db 10,10
```

```
    pular_linha_len equ $ - pular_linha_str
    buffer1 db 10 ; String do primeiro número
    buffer2 db 10 ; String do segundo número
    buffer_soma db 10 ; String da soma
    buffer_len equ 10 ; Tamanho máximo do buffer

section .bss
    verificar_erro resb 1 ; Byte que verifica se tem algum erro

section .text
    global _start
    extern print_msg, read_msg, atoi, itoa, pularlinha
    global buffer_len, pular_linha_str, pular_linha_len, verificar_erro
```

No seguinte trecho, o comportamento de leitura1 e de leitura2 é identico, zerando o verificador de erro, imprimindo msg1 pela função externa print_msg, lendo uma String do usuário pela função externa read_msg passando a string que guardará o que foi digitado para a função pelo registrador rcx, e convertendo a string lida chamando a função externa atoi.
Caso haja um erro, a lógica de leitura será recomaçada, ou seja, caso em leitura1 a string seja inválida, o primeiro número será pedido de novo.

```
; SYSCALL WRITE 1: "Insira o primeiro numero:"
leitura1:
    mov byte [verificar_erro], 0 ; Zera o verificador de erro

    push 4                      ; syscall number para sys_write por pilha
    push 1                      ; file descriptor 1 (stdout) por pilha
    push msg1                   ; endereço da mensagem de prompt por pilha
    push msg1_len               ; tamanho da mensagem de prompt por pilha
    call print_msg              ; chamada da função de impressão de mensagem
        ; "Insira o primeiro numero:"

    ; limpando a pilha
    pop rax
    pop rax
    pop rax
    pop rax

    ; SYSCALL READ 1
    mov rcx, buffer1            ; aponta para o buffer do primeiro número
    call read_msg               ; chamada da função de leitura da entrada
```

```
21
22      ; Converte string para número inteiro
23      mov rsi, buffer1            ; apontando para o início da string do
        ↪  primeiro número
24      xor rax, rax                ; zera o valor acumulador(neste caso rax é o
        ↪  acumulador)
25      xor rcx, rcx                ; contador de dígitos lidos
26      call atoi                   ; chamada da função atoi (ascii para
        ↪  inteiro)
27      cmp byte [verificar_erro], 1 ; verifica se tem algum erro
28      je leitura1                  ; se tiver erro, volta para leitura1
29      push rax            ; armazena o resultado em num1
30
31  ; SYSCALL WRITE 2: "Insira o segundo numero:"
32  leitura2:
33      mov byte [verificar_erro], 0 ; Zera o verificador de erro
34
35      push 4                      ; syscall number para sys_write por pilha
36      push 1                      ; file descriptor 1 (stdout) por pilha
37      push msg2                   ; endereço da mensagem de prompt por pilha
38      push msg2_len               ; tamanho da mensagem de prompt por pilha
39      call print_msg              ; chamada da função de impressão de mensagem
40          ; "Insira o segundo numero:"
41
42      ; limpando a pilha
43      pop rax
44      pop rax
45      pop rax
46      pop rax
47
48      ; SYSCALL READ 2
49      mov rcx, buffer2            ; aponta para o buffer da entrada
50      call read_msg               ; chamada da função de leitura da entrada
51
52      ; Converte string para número inteiro
53      mov rsi, buffer2            ; apontando para o início do buffer
54      xor rax, rax                ; zera o valor acumulador
55      xor rcx, rcx                ; contador de dígitos lidos
56      call atoi                   ; chamada da função atoi (ascii para
        ↪  inteiro)
57      cmp byte [verificar_erro], 1 ; verifica se tem algum erro
```

```
58      je leitura2                     ; se tiver erro, volta para leitura2
```

O seguinte trecho é responsável por somar os 2 números e converter a soma para string chamando a função externa itoa.

```
1   ; Soma dos dois números
2   pop rbx                         ; pegar o valor de num1 da pilha
3   add rax, rbx                    ; soma os dois valores
4
5   ; Converte o resultado da soma(que agora está em rax) de volta para string
6   mov rbx, rax                    ; copia o valor da soma para rbx
7   mov rdi, buffer_soma            ; guarda o inicio da string buffer_soma
8   add rdi, buffer_len             ; faz o ponteiro rdi apontar para o final da
    ↪  string buffer_soma
9   call itoa                       ; chamada da função itoa (inteiro para ascii)
```

O seguinte trecho imprime a string convertida e encerra o programa. O registrador rdi é passado ao invés de buffer_soma pois não necessáriamente o inicio da variável é o inicío da string so número convertido, visto que a inclusão caracteres em buffer_soma ocorre de trás pra frente.

```
1   ; Exibe "Soma: "
2   push 4                      ; syscall number for sys_write
3   push 1                      ; file descriptor 1 (stdout)
4   push msg_resultado          ; endereço da mensagem de resultado
5   push msg_resultado_len      ; tamanho da mensagem de resultado
6   call print_msg                      ; chamada de sistema para escrever "Soma:
    ↪  "
7
8   ; limpando a pilha
9   pop rax
10  pop rax
11  pop rax
12  pop rax
13
14  ; Exibe o número convertido para string
15  push 4                      ; syscall number for sys_write
16  push 1                      ; file descriptor 1 (stdout)
17  push rdi                    ; aponta para o início do número no buffer
18  push buffer_len             ; tamanho máximo (exibe até o '\0')
19  call print_msg                      ; chamada de sistema para escrever o
    ↪  número
```

```
20
21  ; limpando a pilha
22  pop rax
23  pop rax
24  pop rax
25  pop rax
26
27  ; Exibe uma quebra de linha
28  call pularlinha
29
30  Fim:
31      mov rax, 1                  ; Syscall number para sys_exit
32      xor rbx, rbx                ; código de saída 0
33      int 0x80                    ; chamada de sistema para sair
```