Universidade de Brasília

SOFTWARE BASICO
Montador NASM

Aluno: Daniel Moraes da Silva - 18/0112333
Professor: Prof. Dr. Marcelo Ladeira

Fevereiro
2023

Conteúdo

1 Resumo **1**

2 Como debugar um programa NASM Usando o GDB **1**

3 Chamada de um programa em NASM **3**

4 Operações aritiméticas Binary-coded decimal (BCD **4**
4.1 Intrução AAA (Ajuste após a adição) 5
4.2 Instrução DAA 5

5 Variáveis de Ambientes em NASM **7**

6 Passagem de parâmetros **8**

7 Negação Lógica **9**

1 Resumo

Esse trabalho compõem a disciplina de Software Básico ministrada no departamento de Ciência da Computação na Universidade de Brasília.
Tarefas propostas:

- 1 - Exemplifique como debugar um programa em NASM usando um software de debug
- 2 - Construa um programa em C que chama um programa em NASM para exemplificar o uso de bibliotecas nativas do C
- 3 - Construa um programa em NASM que seja chamado por um programa em C para realizar aritmética BCD (soma e subtração)
- 5 - Mostre como lidar com variáveis de ambientes em NASM a partir de um programa C
- 6 - Mostre como lidar com passagem de parâmetros via linha de comando em NASM a partir de um programa C
- 7 - Construa um programa em NASM para ler um arquivo, aplicar um filtro qualquer nele e gravar o arquivo resultante, usando os serviços do S.O..

2 Como debugar um programa NASM Usando o GDB

Primeiro, com seu editor de texto de preferência crie um arquivo .asm , onde terá os comando em NASM que deseja executar.

Compile o programa NASM usando a opção '-g' para incluir informações de depuração no arquivo executável e depois inicie o GDB carregando o programa olaMundo.asm.

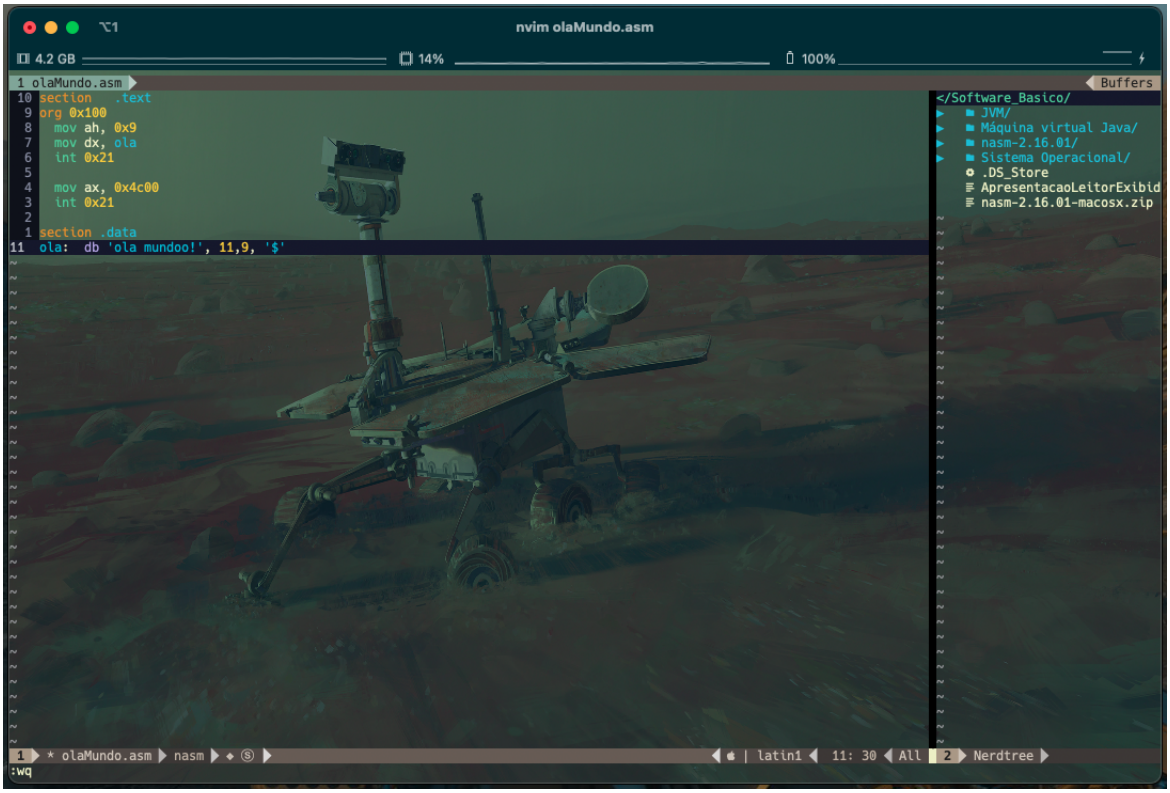

Figura 1: olaMundo.asm

Inicie o GDB e carregue o programa:

```
; ——————————————————————————;
    >> dgb olaMundo
    >> break int 0x21
    >> run
```

Adicione breakpoints no seu programa, para sempre verificar em cada execução o estado da memória e dos registradores;
Quando o programa chegar ao seu primeiro Breakpoint, examine o estado da memória e dos registradores:

```
; ——————————————————————————;
    >> info reg
```

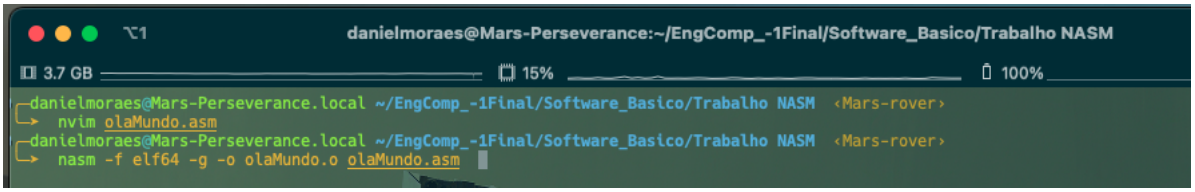

Figura 2:

```
    >> x/21xw $rsp
; ------------------------------;
```

Avance a execução do programa, um passo de cada vez com o comando 'next' ou 'step'

3 Chamada de um programa em NASM

2 - Construa um programa em C que chama um programa em NASM para exemplificar o uso de bibliotecas nativas do C
chamada.c :

```
#include <stdio.h>

extern int soma(int a, int b);

int main(int argc, char *argv[]) {
    int result = soma(5, 7);
    printf("Soma = %d\n", result);
    return 0;
}
```

E o código NASM correspondente:

```
section .text
    global soma

soma:
```

```
    push rbp
    mov rbp, rsp
    mov eax, [rbp + 8]
    add eax, [rbp + 12]
    pop rbp
    ret
```

O programa C inlcui a função de soma do programa NASM como uma função externa, permitindo que ela seja chamada diretamente pelo programa em C. O programa NASM define a função Soma e realiza uma soma aritimética de dois números inteiros passados como argumentos.
O Assembly não possui bibliotecas de acesso, sendo assim esse acesso é feito direto ao harware via instruções 'in' e 'out'. Porém ao interfacear rotinas assembly com o programa em C, o Assembly pode usar rotinas da biblioteca padrão "stdio.h".
Nas sessões (6 - Passagem de parâmetros e 7 - Aplica Filtros) apresentam códigos em que utiliza-se a biblioteca padrão "stdio.h"e"stdlib.h"para a utilizadação de várias função.

4 Operações aritiméticas Binary-coded decimal (BCD

3 - Construa um programa em NASM que seja chamado por um programa em C para realizar aritmética BCD (soma e subtração).
Conforme, mencionado na sessão anterior, é possível interfacear rotinas em assembly para fazer chamadas por um programa em C. A seguir temos um exemplo de um programa em NASM que será chamado por um programa em C, veja:

```
section .text

    global soma_BINARIA ;
```

```
    global sub_BINARIA   ;

soma_BINARIA:
    push rbp
    mov rbp, rsp
    mov al, [rbp + 8]
    add al, [rbp + 12]
    aaa ;AJUSTE DE ADICAO ASCII
    mov [rbp -8], al
    pop rbp
    ret
sub_BINARIA:
    push rbp
    mov rbp, rsp
    mov al, [rbp + 8]
    sub al, [rbp + 12]
    daa ;ACOMULADOR DE AJUSTE DE SUBTRACAO ASCII
    mov [rbp -8], al
    pop rbp
    ret
```

Note que nesse programa são utilizados os comando 'aaa' e 'daa'.

4.1 Intrução AAA (Ajuste após a adição)

Para operações BCD que, para esse exemplo, utilza-se de digitos representados pela tabela ASCII. Assim, a intrução AAA permite a operação de adição sem mascarar o nibble da rotina. Desta forma ela garante que o resultado final seja armazenado no formato BCD e dentro do registrador AL.

4.2 Instrução DAA

A intrução DAA (Decimal Adjust Accumulator) também é armazenada no registrador AL e ela permite a conversão de dois números BCD compac-

tados em um número BCD válido.

A seguir temos o código em C correspondente:

```
#include <stdio.h>

extern unsigned char soma_BINARIA(unsigned char a, unsigned char b);
extern unsigned char sub_BINARIA(unsigned char a, unsigned char b);

int main(int argc, char *argv[]){
    unsigned char a = 0x12;
    unsigned char b = 0x34;
    unsigned char result;

    result = soma_BINARIA(a,b);
    printf("SOMA = %x\n",result);

    result = sub_BINARIA(a,b);
    print("SUB = %x\n",result);


    return 0;
}
```

O programa NASM define as duas funções soma_BINARIA e sub_BINARIA, que realizam as operações BCD. Essas funções são declaradas como funções externas no programa em C, permitindo assim que elas sejam chamadas diretamentes no programa em c.
Para compilar esse programa, primeiro execute o programa NASM separadamente e crie uma biblioteca como ilustrado a seguir:

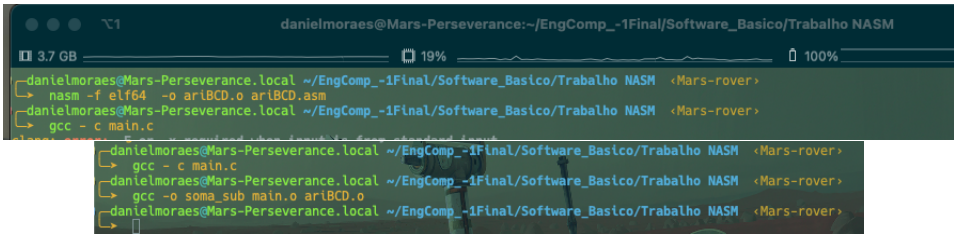

Figura 3:

5 Variáveis de Ambientes em NASM

5 - Mostre como lidar com variáveis de ambientes em NASM a partir de um programa C
Em um programa NASM:

```
section .text
    global get_variavel_ambiente

get_variavel_ambiente:
    push rbp
    mov rbp, rsp
    mov rdi, [rbp + 8]
    call getAMBIENTE
    mov [rbp - 8], rax
    pop rbp
    ret
```

O código C correspondente:

```
#include <stdio.h>
#include <stdlib.h>

extern char *get_variavel_ambiente(const char *name);

int main(int argc, char *argv[], char *getAMBIENTE[]) {
    char *path = get_variavel_ambiente("PATH");
```

```
    if (path == NULL) {
        printf("PATH Variavel de ambiente n o declarada\n");
    } else {
        printf("PATH: %s\n", path);
    }
    return 0;
}
```

O programa NASM define a função 'get-Variavel-ambiente', que usa a função 'getAMBIENTE' da biblioteca padrão C para recuperar o valor de uma variavel ambiente. A função 'getAMBIENTE' recebe um ponteiro para uma string como argumento, que representa o nome da variável de ambiente que deseja recuperar.
Nesse exemplo, temos a variável PATH a qual deseja-se recuperar o seu valor e imprimi-lo. Uma pratica bastante comum em operações de desenvolvimento em Sistemas Operacionais.

6 Passagem de parâmetros

6 - Mostre como lidar com passagem de parâmetros via linha de comando em NASM a partir de um programa C
Programa NASM:

```
    section .data
    format db 'Argumentos: %s', 0
section .text
    global print_argumentos
print_argumentos:
    push rbp
    mov rbp, rsp
    mov rdi, [rbp + 8]
    mov rsi, format
    xor rax, rax
```

```
    call printf
    pop rbp
    ret
```

Codigo em C correspondente:

```
#include <stdio.h>
#include <stdlib.h>

extern void print_argumentos(const char *arg);

int main(int argc, char *argv[], char *getAMBIENTE[]) {
    if (argc < 2) {
        printf("Utilize o formato: busca <argumento>\n");
        return 1;
    }
    print_argumentos(argv[1]);
    return 0;
}
```

O programa NASM define a função 'print-argumentos', que usa a função ' printf' da biblioteca padrão do C para imprimir o argumento passado via linha de comando. A função ' printf' aceita um ponteiro para uma string como priemiro argumento.
Assim, a função print-argumentos é chamada pelo programa em C após verificar a passagem de argumento via linha de comando. O primeiro argumento argv[0] é o nome do programa, enquanto o segundo argumento argv[1] é o argumento passado na linha de comando.

7 Negação Lógica

7 - Construa um programa em NASM para ler um arquivo, aplicar um filtro qualquer nele e gravar o arquivo resultante, usando os serviços do S.O

Para essa solução utilizei 4 funções da biblioteca padrão do C para realizar as manipulações de arquivos. A função *fopen* abre um arquivo, *fread* lê dados de um arquivo, *fwrite* escreve dados em um arquivo e *fclose* fecha um arquivo. O funcionamento do programa segue conforme o fluxograma ilustrado abaixo.

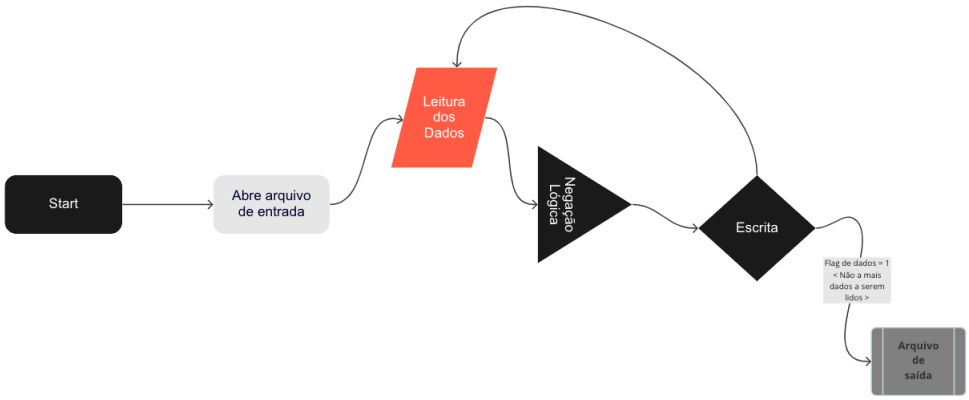

Figura 4:

Programa NASM:

```
section .data
    input_file db 'input.bin', 0
    output_file db 'output.bin', 0
    mode db 'wb', 0
    buf db 8192 dup (0)

section .bss
    file_handle resb 8

section .text
    global negate_file

negate_file:
    push rbp
    mov rbp, rsp

    ; abre o arquivo de entrada
```

```
mov rdi, input_file
mov rsi, 0 ; read-only mode
call fopen
mov [file_handle], rax

; verifica se o arquivo foi aberto com sucesso
cmp rax, 0
jz end

; l   e nega os dados do arquivo
.read:
    ; l   os dados do arquivo
mov rdi, [file_handle]
mov rsi, buf
mov rdx, 8192
call fread
test rax, rax
jz end

; nega os dados
mov rcx, rax
.negate:
     xor byte [rsi + rcx - 1], 1
    dec rcx
    jnz negate

; escreve os dados negados no arquivo de sa da
mov rdi, output_file
mov rsi, mode
call fopen
cmp rax, 0
jz end
mov rdi, rax
```

```
    mov rsi, buf
    mov rdx, rax
    call fwrite
    jmp .read

    ; fecha o arquivo de entrada e de sa da
    end:
    mov rdi, [file_handle]
    call fclose

    mov rdi, rax
    call fclose

    pop rbp
ret
```