CMP1057 - Arquitetura de Computadores I - Laboratório

Lista de Exercícios

Max Gontijo de Oliveira

Para auxiliar, no fim deste documento existem algumas dicas para programar em NASM.

A documentação detalhada pode ser acessada em http://www.nasm.us/doc/nasmdoc0.html. O Capítulo 3 contém detalhes da linguagem. O Capítulo 11 contém algumas informações adicionais específicas para programação em arquitetura 64-bit, incluindo detalhes de como efetuar a passagem de parâmetros para funções do C, conforme apresentado na Seção 11.3 (para Linux) e na Seção 11.4 (para Windows).

Em todas as questões dessa lista, crie não apenas o programa em NASM mas também o fluxograma que representa o programa. De fato, crie o fluxograma antes de começar a implementação. O fluxograma ajuda muito. No fim deste documento, você encontrará fluxogramas que representam as estruturas de repetição `for`, `while` e `do...while`

1. Crie um programa em NASM que leia dois números inteiros x e y e armazene na memória a soma $x + y$, a diferença $x - y$, o produto $x * y$, o quociente e o resto da divisão x/y. Na sequência, o programa deverá imprimir na tela todos esses resultados. Por exemplo, se o usuário entrar com 30 e 7 para x e y respectivamente, além de armazenar os valores na memória, a saída na tela deverá apresentar o seguinte resultado:

```
Soma: 37
Diferença: 23
Produto: 210
Quociente: 4
Resto: 2
```

2. Crie um programa em NASM que leia do teclado dois números inteiros x e y e armazene-os na memória. Em seguida, o programa deverá trocar os valores de lugar. Ou seja, o valor de x deverá ser aquele armazenado em y e vice-versa.

3. Crie um programa em NASM que declare no segmento de dados inicializados uma string qualquer. O programa deverá alterar o primeiro caractere desse texto para `X`, o segundo para `Y` e o terceiro para `Z`.

4. Crie um programa em NASM que leia do teclado dois números inteiros positivos x e y, calcule x^y (sem usar nenhuma função do C, como a `pow`) e mostre o resultado na tela. Construa uma estrutura de repetição para efetuar o cálculo.

5. Crie um programa em NASM que leia do teclado dois números inteiros positivos x e y, calcule $x * (2^y)$ (sem usar nenhuma função do C, como a `pow` e sem usar a instrução `mul`) e mostre o resultado na tela. Utilize apenas movimentação de bits.

6. Crie um programa em NASM que leia do teclado dois números inteiros positivos x e y, calcule o quociente da divisão $x/(2^y)$ (sem usar nenhuma função do C, como a `pow` e sem usar a instrução `div`) e mostre o resultado na tela. Utilize apenas movimentação de bits.

7. Crie um programa em NASM que leia do teclado uma string de até 100 caracteres. Em seguida, seu programa deverá contar quantas vogais, consoantes e algarismos (números) foram digitados. Por fim, deverá mostrar na tela o resultado dessas três contagens e mais uma quarta categoria que contabiliza o restante de caracteres. Assuma que as letras serão sempre minúsculas, que não haverão sinais de acentuação ou cedilha e que não haverão espaços na string. Assim, por exemplo, se o usuário digitar a frase: "a_resposta_seria_42!"

```
Vogais: 7
Consoantes: 7
Algarismos: 2
Outros: 4
```

8. Crie um programa em NASM que leia do teclado uma string de até 100 caracteres. Em seguida, seu programa deverá converter todos os caracteres que forem letras em maiúsculas. As letras que já estiverem maiúsculas e demais caracteres não deverão sofrer alteração. Por fim, seu programa deverá imprimir a string alterada na tela.

9. Crie um programa em NASM que leia do teclado uma string de até 20 caracteres. Assuma que nessa string hajam apenas algarismos digitados. Ou seja, não haverão quaisquer outros tipos de caracteres a não ser os dez algarismos: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9. Após a leitura do número (na forma de string), seu programa deverá converter o texto digitado em um número de 64 bits e armazená-lo em uma variável na memória. Por fim, o programa deverá exibir na tela o número imediatamente posterior ao digitado. Por exemplo, se o usuário digitar "3571", o programa deverá converter essa string para o número 3571 e exibir na tela o número 3572. Não utilize nenhuma função do C para fazer a conversão.

10. Crie um programa em NASM que leia do teclado dois números inteiros positivos x e y. O programa deverá imprimir na tela um retângulo de asteriscos que tenha x linhas e y colunas. Por exemplo, se o usuário digitar 5 e 15 como valores para, respectivamente, x e y, o resultado na tela deveria ser o seguinte:

```
1 ***************
2 ***************
3 ***************
4 ***************
5 ***************
```

11. Crie um programa em NASM que declare no segmento de dados não inicializados dois vetores A e B de 5 `qwords` cada. Seu programa deverá ler as 5 posições de cada vetor (como `int`). Em seguida, seu programa deverá apresentar na tela o conjunto dos elementos que existem somente em A; o conjunto dos elementos que existem somente em B; o conjunto dos elementos que existem em A e B; e o conjunto dos elementos que existem em A ou B. Vale ressaltar que os valores não deverão ser repetidos em cada conjunto. Assim, como exemplo, se o usuário digitar os valores 3, 7, 15, 23 e 8 para o vetor A e 23, 42, 7, 3 e 19 para o vetor B, o resultado a ser exibido na tela deveria ser:

```
1 Somente em A: 15 8
2 Somente em B: 42 19
3 Em A e B    : 3 7 23
4 Em A ou B   : 3 7 15 23 8 42 19
```

12. Crie um programa em NASM que leia do teclado duas string de até 100 caracteres cada. Seu programa deverá, então, verificar e imprimir na tela se as strings são iguais ou não. Nesse programa, letras maiúsculas deverão ser consideradas diferentes das minúsculas.

13. Crie um programa em NASM que leia do teclado uma string de até 100 caracteres, inverta os caracteres dessa string e apresente na tela a string invertida. Note que a inversão será apenas dos caracteres válidos e não necessariamente dos 100 caracteres de capacidade. Assim, se o usuário digitar **"era uma vez..."**, o resultado exibido na tela deverá ser **"...zev amu are"**.

14. Crie um programa em NASM que declare no segmento de dados não inicializados três matrizes A, B e C de inteiros (`qword`) de dimensão 3×3 cada uma. Seu programa deverá ler cada uma das posições das matrizes A e B. Em seguida, seu programa deverá calcular a soma matricial $A + B$ e armazenar o resultado na matriz C. Por fim, seu programa deverá exibir a matriz resultante C na tela. Dica: cada matriz deve ser declarada como um vetor de 9 posições. Para ter acesso ao valor localizado na matriz A na linha i, coluna j, o endereço pode ser obtido por meio de uma operação simples: $A + i * C + j$, onde C é a quantidade de colunas da matriz.

15. Crie um programa em NASM que declare no segmento de dados não inicializados, uma matriz A de dimensão 3×2 e uma matriz B de dimensão 2×3. Seu programa deverá ler todas as posições da matriz A. Em seguida deverá encontrar a matriz transposta de A e armazenar na matriz B. Por fim, seu programa deverá exibir na tela a matriz B.

Segue modelo contendo os principais segmentos de um programa em NASM.

```
global main          ; Indica que main será visível de fora do arquivo
extern printf, scanf ; Funções escritas em C que serão usadas no programa

section .data     ; Segmento de dados inicializados
var1 db "Texto",0 ; Cria um vetor de 6 bytes (8 bits cada) inicializando
                  ; com os caracteres em questão e o 0 no final
var2 dw 50        ; Cria uma word (16 bits) inicializando o valor com 50
var3 dd 42,77,85  ; Cria um vetor de 3 dwords (32 bits cada) inicializando
                  ; com os valores 42, 77 e 85
var4 dq 37        ; Cria uma qword (64 bits) inicializando o valor com 37

section .bss ; segmento de dados não inicializados
var5 resb 5  ; Reserva um vetor de 5 bytes
var6 resw 10 ; Reserva um vetor de 10 words
var7 resd 1  ; Reserva uma dword
var8 resq 2  ; Reserva um vetor de 2 qwords

section .text ; Segmento de código, onde estarão as instruções
main: ; Função de entrada no programa. Similar ao main do C.
        ; Instruções do seu programa
        ; Instruções do seu programa
        ; Instruções do seu programa
        ; Instruções do seu programa
        ; Instruções do seu programa
ret
```

Segue ainda um exemplo de código que lê do teclado um número e um texto e, na sequência, imprime tais valores na tela.

```
global main
extern printf, scanf

section .data
msg1   db "Digite um numero: ",0
msg2   db "Digite um texto: ",0
fmtint db "%d",0
fmtstr db "%s",0
msg3   db "Numero digitado: %d",10,0
msg4   db "Texto digitado: %s",10,0

section .bss
v_num resq 1
v_tex  resb 10

section .text
main:
        ;--- Escreve na tela a mensagem msg1 ---------------------
        xor rax,rax
        mov rdi,msg1
        call printf
        ;---------------------------------------------------------

        ;--- Le do teclado um numero e armazena em v_num ----------
        xor rax,rax
        mov rdi,fmtint
        mov rsi,v_num
        call scanf
        ;---------------------------------------------------------

        ;--- Escreve na tela a mensagem msg2 ---------------------
        xor rax,rax
        mov rdi,msg2
        call printf
        ;---------------------------------------------------------

        ;--- Le do teclado um texto e armazena em v_tex -----------
        xor rax,rax
        mov rdi,fmtstr
        mov rsi,v_tex
        call scanf
        ;---------------------------------------------------------

        ;--- Escreve na tela a mensagem msg3 e o valor de v_num ---
        xor rax,rax
        mov rdi,msg3
        mov rsi,[v_num]
        call printf
        ;---------------------------------------------------------

        ;--- Escreve na tela a mensagem msg4 e o valor de v_tex ---
        xor rax,rax
        mov rdi,msg4
        mov rsi,v_tex
        call printf
        ;---------------------------------------------------------
ret
```

Os programas deverão ser escritos em arquivos com extensão *.asm* e compilados da seguinte maneira:

1. `nasm -f elf64 prog.asm`
   - para realizar a montagem (*assembly*) gerando o arquivo prog.o;
2. `gcc prog.o -o prog`
   - para compilar o arquivo de montagem, gerando o executável prog;
3. `./prog`
   - para executar o programa.

Seguem algumas das instruções que podem ser úteis nessa lista de exercício.

Tabela 1: Instruções gerais

Instrução	**Descrição**
`mov ARG1, ARG2`	Copia o valor de `ARG2` para `ARG1`.
`lea ARG1, ARG2`	Calcula a expressão `ARG2` e armazena o resultado em `ARG1`. Em geral, essa expressão é baseada em operações aritméticas com o endereço de alguma unidade de memória. Por exemplo, `lea rax, [vet + 8 * rbx]` calcula a expressão entre colchetes, onde `vet` é a posição inicial de um vetor de `qwords`. Nesse caso, `rbx` poderia ser um contador e o 8 se refere à quantidade de bytes que uma `qword` tem.
`push ARG1`	Empilha o valor `ARG1` na pilha.
`pop ARG1`	Desempilha o topo da pilha e armazena o resultado `ARG1`.
`shl ARG1, ARG2`	Movimenta todos os bits de `ARG1` para a esquerda `ARG2` vezes.
`shr ARG1, ARG2`	Movimenta todos os bits de `ARG1` para a direita `ARG2` vezes.
`call PROC`	Chama o procedimento/função `PROC`.

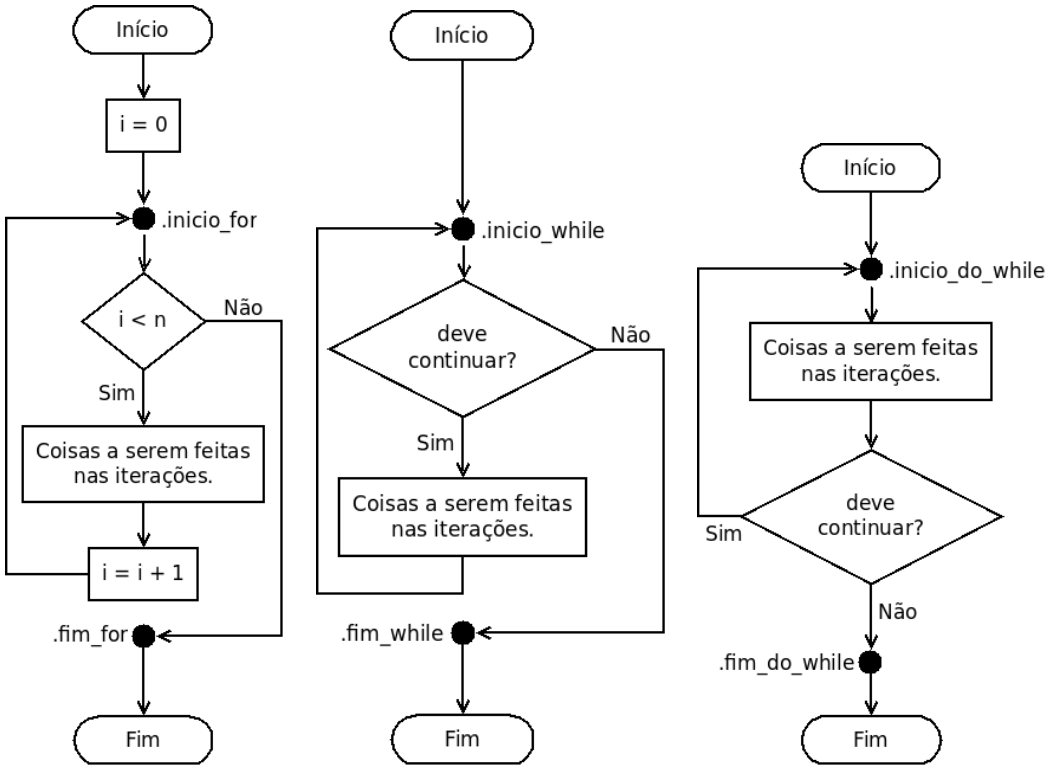
Tabela 2: Instruções lógicas e aritméticas

Instrução	**Descrição**
`add ARG1, ARG2`	Soma `ARG1` e `ARG2` e armazena o resultado em `ARG1`.
`sub ARG1, ARG2`	Subtrai de `ARG1` o valor `ARG2` e armazena o resultado em `ARG1`.
`mul ARG1`	Multiplica o valor do registrador `RAX` com o valor de `ARG1`. Os operandos possuem 64 bits e o resultado é dado em 128 bits, de modo que os 64 bits menos significativos serão armazenados em `RAX` e os mais significativos em `RDX`.
`div ARG1`	Divide o valor de 128 bits formado por `RDX` e `RAX` (`RDX` com os 64 bits mais significativos e `RAX` com os 64 bits menos significativos) pelo valor `ARG1`. O quociente é armazenado em `RAX` enquanto o resto é armazenado em `RDX`.
`and ARG1, ARG2`	Executa a operação lógica *and* bit a bit entre os valores `ARG1` e `ARG2`, armazenando o resultado em `ARG1`.
`or ARG1, ARG2`	Executa a operação lógica *or* bit a bit entre os valores `ARG1` e `ARG2`, armazenando o resultado em `ARG1`.
`xor ARG1, ARG2`	Executa a operação lógica *xor* bit a bit entre os valores `ARG1` e `ARG2`, armazenando o resultado em `ARG1`.
`inc ARG1`	Efetua o incremento de `ARG1` armazenando o resultado no próprio `ARG1`.

Tabela 3: Instruções de controle

Instrução	**Descrição**
`jmp LABEL`	Efetua um desvio incondicional para LABEL.
`cmp ARG1, ARG2`	Efetua uma comparação entre os dois argumentos, setando algumas flags que informam coisas como, se são iguais, qual é maior, etc. Normalmente, essa instrução é executada imediatamente antes de um desvio incondicional.
`ja LABEL`	Desvia para `LABEL` se `ARG1` $>$ `ARG2` (números **sem** sinal).
`jae LABEL`	Desvia para `LABEL` se `ARG1` $\geqslant$ `ARG2` (números **sem** sinal).
`jb LABEL`	Desvia para `LABEL` se `ARG1` $<$ `ARG2` (números **sem** sinal).
`jbe LABEL`	Desvia para `LABEL` se `ARG1` $\leqslant$ `ARG2` (números **sem** sinal).
`jg LABEL`	Desvia para `LABEL` se `ARG1` $>$ `ARG2` (números **com** sinal).
`jge LABEL`	Desvia para `LABEL` se `ARG1` $\geqslant$ `ARG2` (números **com** sinal).
`jl LABEL`	Desvia para `LABEL` se `ARG1` $<$ `ARG2` (números **com** sinal).
`jle LABEL`	Desvia para `LABEL` se `ARG1` $\leqslant$ `ARG2` (números **com** sinal).
`je LABEL`	Desvia para `LABEL` se `ARG1` $=$ `ARG2`.
`jne LABEL`	Desvia para `LABEL` se `ARG1` $\neq$ `ARG2`.

Seguem fluxogramas que representam as estruturas de repetição. Certamente serão bastante úteis para resolver essa lista.



(a) `for` (b) `while` (c) `do...while`

Figura 1: Estruturas de repetição