Programação em Linguagem de Montagem do Computador Cleópatra – Versão 5

Para realizar este trabalho, os alunos devem compor grupos de até ***3 (três) componentes***. Cada grupo irá receber do professor um conjunto de exercícios descritos na Tabela 1. Para cada exercício implementado deve ser entregue:

1. Uma **descrição de alto nível** (fluxograma, fonte em C, Java ou português estruturado) – **(1,5 pontos)**;
2. O **código fonte em linguagem de montagem**, contendo programa e dados sobre os quais este executa de maneira correta – **(5 pontos)**;
3. A **tela do montador da Cleópatra** com o conteúdo de memória e registradores que mostrem o resultado do programa ao encerrar a sua execução – **(0,5 pontos)**;
4. A **função de cálculo do número de ciclos** que o programa leva para executar – **(2,5 pontos)**; e
5. Uma **estimativa de tempo de execução** presumindo que a Cleópatra opera a 1 GHz. Caso o programa dependa de alguma variável, deve ser informado o valor da mesma que levou ao resultado estimado – **(0,5 pontos)**.

Tabela 1 - Especificação de programas em lingaugem de montagem.

	Grupos									
	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
Exercícios	1	3	5	7	9	2	4	6	8	11
	12	13	14	15	16	20	19	18	17	10
	21	22	22	23	22	21	20	19	18	17
	7	8	9	10	11	12	13	14	15	16

1. Descobrir se um número ***n*** *positivo* é múltiplo de número do seu grupo + 5 (se por acaso, o valor de grupo + 5 ultrapassar o número do maior grupo, deve ser subtraído o número do maior grupo. Ex.: ***grupo = 7*** ➔ 7 + 5 = 12 ➔ 12 > 10 ➔ 12 - 10 = 2 ➔ ***n = 2***).
2. Dados dois inteiros armazenados, respectivamente, nas posições de memória cujos rótulos são ***n1*** e ***n2***, armazenar o valor máximo entre estes na posição de memória com rótulo ***max***, o valor mínimo na posição de memória com rótulo ***min***, e a média aritmética na posição de memória com o rótulo ***med***. OBS.: (i) Devem ser considerados valores *positivos e negativos*; (ii) os cálculos de mínimo, máximo e média devem ser feitos *com chamadas de função*.
3. Fazer um programa que calcula o *máximo divisor comum* entre dois valores inteiros positivos, que devem estar nos rótulos ***v1*** e ***v2*** e o resultado deve ser colocado em uma variável com o rótulo **max**. Este programa deve chamar a função ***mdc***, que tem como parâmetros de entrada variáveis globais com rótulos ***a*** e ***b*** e cujo retorno da função é colocado no registrador ***AC*** (acumulador).
4. Multiplicar, por somas sucessivas, 2 inteiros positivos (*1 byte*), armazenando o resultado em um inteiro longo (*2 bytes*). Considere que estes números podem ser tanto positivos quanto negativos, ou seja, é uma multiplicação com sinal. Multiplicando e multiplicador devem estar armazenados nas posições de memória com rótulos ***n1*** e ***n2***, respectivamente. O resultado deverá ser armazenado nas posições de memória com rótulos ***mh*** (a parte mais significativa do resultado) e ***ml*** (a parte menos significativa do resultado). O código deve ser construído de tal forma que o número de iteração, necessários para realizar as somas sucessivas, seja o menor possível.
5. Descobrir se um número ***n*** é múltiplo de um número ***m*** qualquer. Sendo que ***n*** e ***m*** são números inteiros positivos. Faça a chamada da função ***EhMultiplo***, que deve ler as posições de memória referentes a ***n*** e ***m*** e verificar a multiplicidade.
6. Faça um algoritmo que calcula a divisão de dois números de *1 byte* de tamanho (armazenados nas posições de memória com rótulos ***v1*** e ***v2***) através de subtrações sucessivas. Ao final do algoritmo a parte inteira da divisão deve estar na posição de memória com rótulo ***int*** e o resto na posição de memória com rótulo ***resto***. Considere que estes números podem ser tanto positivos quanto negativos, ou seja, é uma

divisão com sinal.

7. Faça um programa que contenha a função ***Potencia***. Esta função deve ser capaz de receber dois valores ***V*** e ***EXP*** e ter como saída $\boldsymbol{V^{EXP}}$. Esta saída deve estar associada ao acumulador.

8. Deseja-se implementar em assembly a função ***RotateLeft***. Esta função deve rotacionar o registrador **AC** para a esquerda o número de bits que estiver informado na variável de rótulo ***rot***. Faça um programa que utilize esta função para rotacionar o byte 0x34, seis bits para a esquerda.

9. Somar a cada posição do vetor de rótulo ***vetDst*** o conteúdo da próxima posição, com exceção da última posição que é somada com a primeira. Supondo um vetor de 4 posições, tem-se: ***vetDst[0] <= vetDst[0] + vetDst[1]; vetDst[1] <= vetDst[1] + vetDst[2]; vetDst[2] <= vetDst[2] + vetDst[3]; vetDst[3] <= vetDst[3] + vetDst[0]***. O número de elementos do vetor está armazenado na posição de memória cujo rótulo é ***tam***.

10. Fazer um algoritmo que lê um vetor de números inteiros com sinal, calcula o total de números inteiros pares (colocar a resposta na posição de memória referenciada pelo rótulo ***totPar***), o total de números inteiros ímpares (colocar a resposta na posição de memória referenciada pelo rótulo ***totImpar***) e informa qual é o maior par positivo (colocar a resposta na posição de memória referenciada pelo rótulo ***maiorParPos***) e qual é o menor ímpar negativo (colocar a resposta na posição de memória referenciada pelo rótulo ***menorImparNeg***). O vetor inicia na posição de memória referenciada por ***vet*** e tem o número de elementos armazenado na posição de memória referenciada por ***tam***.

11. Escrever um programa que copia o conteúdo de um vetor (***vetOrig*****),** definido entre os endereços de memória com rótulos ***inicio1*** e ***fim1*****,** para outro vetor (***vetDest***), definido entre os endereços rótulos são ***inicio2*** e ***fim2***. Esta cópia, porém deve respeitar um deslocamento definido por uma constante ***cte***. O tamanho dos vetores ***tam*** origem e destino são sempre os mesmos e a constante deve ser 0 <= ***cte*** < ***tam***. Assim, ***vetDest***[índice + ***cte***] <= ***vetOrig***[indice] e sempre que a posição do vetor destino, definido pela soma índice + ***cte***, ultrapassar a última posição deste vetor, deve-se começar do início do vetor. Por exemplo, assumindo o deslocamento 2 e um vetor de tamanho 4, teríamos: ***vetDest[2] <= vetOrig[0]; vetDest[3] <= vetOrig[1]; vetDest[0] <= vetOrig[2]; vetDest[1] <= vetOrig[3];***

12. Fazer um programa que gere os ***n*** primeiros números da sequência de Fibonnacci, e armazene a sequências em endereços consecutivos a partir da posição de memória com rótulo ***idx***. Esta operação deve ser feita com uma chamada de função para o rótulo ***SeqFibonnacci***.

13. A transmissão de dados binários é o que viabiliza a existência de tecnologias como a Internet. A transmissão de dados a longas distâncias é uma tarefa muito propensa a erros, devido a efeitos ambientais externos (raios, interferências eletromagnéticas nas linhas de transmissão devidas a equipamentos elétricos, raios cósmicos ou manchas solares que interferem nas transmissões de satélites, etc.). Uma maneira de detectar erros de transmissão é acrescentar bits de controle ao dado transmitido, de forma que o receptor possa verificar a validade dos dados transmitidos. Um esquema simples de detecção de erros são as técnicas de *bit de paridade*. A ideia consiste em contar o número de bits de um determinado valor e acrescentar um bit na mensagem que diz se a contagem destes valores na mensagem é par ou ímpar. Assim, podem-se imaginar alguns tipos básicos de cálculo de paridade: paridade de 0s ou paridade de 1s, paridade par ou paridade ímpar, paridade ativa em 0 ou paridade ativa em 1. Implemente uma função que recebe ***n***, o número de bits de uma mensagem; ***msg***, a mensagem de tamanho ***n***; e produz uma nova mensagem ***msgp*** com ***n*** + 1 bits, onde o bit mais significativo é a paridade da mensagem. Faça um programa que teste esta função.

14. Escreva um programa para criar um vetor ***novo***, a partir de dois vetores ***v1*** e ***v2*** de mesma dimensão ***n***, segundo a relação estabelecida na equação abaixo. A interpretação da equação é: cada elemento de ***novo***, na posição ***k***, receberá o somatório dos máximos dos vetores ***v1*** e ***v2***, entre *0* e *k*.

$$novo_k = \sum_{i=0}^{k} \max(v1_i, v2_i), \quad 0 \le k < n$$

15. Escrever um algoritmo que calcule os primeiros ***n*** números primos e os armazene sequencialmente, a partir da posição de memória cujo rótulo é ***nprimos***. Este programa deve chamar a função ***CalculaPrimos***.

16. Implemente um algoritmo de ordenação **crescente** ***que apenas ordene os números pares***. O vetor de origem deve estar armazenado entre as posições de memória de rótulos ***ini1*** e ***fim1***, respectivamente, e o

vetor ordenado deve estar armazenado entre as posições de memória ***ini2*** e ***fim2***, respectivamente.

17. Contar o número de vezes que um determinado padrão, definido por um vetor armazenado entre dois endereços marcados com rótulo ***padIni*** e ***padFim***, aparece em um vetor armazenado entre dois endereços marcados com os rótulos ***vetIni*** e ***vetFim***. Por exemplo, definido o vetor {0,1,2,1,3,1,4,1,2,1} e o padrão {1,2,1}, pode-se observar duas ocorrências naquele primeiro vetor, conforme destacado.

18. Seja dado um ponto inicial (x0, y0) e uma sequência de *n* valores inteiros (o valor de *n* deve estar armazenado em uma posição de memória com rótulo ***n***, e a sequência de valores deve estar armazenada como um vetor iniciando na posição de memória com rótulo ***seq***). Implemente um algoritmo que desloque o ponto inicial alternadamente na horizontal e na vertical (iniciando com um deslocamento horizontal). Suponha que cada valor da sequência dá a magnitude do deslocamento seguinte. Considere que os deslocamentos são tais que o deslocamento atual e o imediatamente anterior circunscrevem um arco que avança no sentido anti-horário se o deslocamento atual for negativo, e no sentido horário se o deslocamento atual for positivo. Note que o deslocamento inicial pode ser em qualquer sentido horizontal, pois não há deslocamento anterior. Documente sua solução para indicar a escolha feita pelo seu programa neste caso.

    Exemplo: dados de entrada [-6, -6, -4, 2, 3, 4] (ver Figura 1). Resultado -5 para x e -4 para y. Decisão: primeiro deslocamento para a esquerda se negativo, para a direita se positivo.

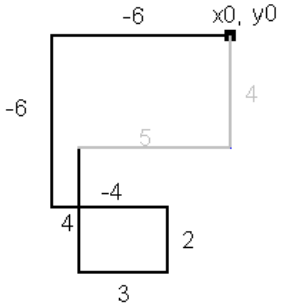

Figura 1 – Representação gráfica da entrada de dados [-6, -6, -4, 2, 3, 4].

19. Faça um programa para ordenar 10 valores hexadecimais que estão armazenados em 5 bytes. Sendo que dentro de cada byte, o *nibble* (conjunto de 4 bits) menos significativo deve conter o menor valor.

    Exemplo: Vetor de entrada: 0x34, 0xFA, 0xBC, 0x77, 0x1C
    Vetor ordenado: 0x13, 0x47, 0x7A, 0xBC, 0xCF

20. Faça um programa que descubra o maior e o menor valor de todos os elementos da diagonal principal de uma matriz 5x5. O menor e o maior valor devem estar ao final do programa em endereços referenciados pelos rótulos ***menor*** e ***maior***, respectivamente. Obs.: o algoritmo deve ser genérico para qualquer matriz **n**x**n**.

21. Faça um programa que calcule a multiplicação de todos os elementos da diagonal principal de uma matriz 5x5. Obs.: o algoritmo deve ser genérico para qualquer matriz **n**x**n**. Este programa deve chamar uma função para fazer a multiplicação.

22. Faça um programa que calcule a soma de todos os elementos da diagonal secundária de uma matriz 5x5. Obs.: o algoritmo deve ser genérico para qualquer matriz **n**x**n**.

23. Faça um programa que copie a matriz **IN** 4x4 para a matriz **OUT** 4x4. Sendo que a cópia deve ser feita transpondo cada linha da matriz **IN** para uma coluna da matriz **OUT**.