Laboratório sobre o Modelo de Computador de Programa Armazenado e Programação em Linguagem de Montagem

Prática: Programação em Linguagem de Montagem do Processador Cleópatra

Recursos: Ambiente de Desenvolvimento de Software para a Arquitetura Cleópatra

Parte I – Introdução e Objetivos

O **Laboratório sobre Circuitos Seqüenciais e Máquinas de Estados Finitas** envolveu o projeto e implementação de sistemas digitais em geral, visando introduzir os alunos ao emprego de sistemas de Projeto Auxiliado por Computador (PAC, ou, em inglês, *CAD*, de *Computer Aided Design*) para o projeto eletrônico. Naquele laboratório, além da introdução ao uso de várias ferramentas de CAD (e.g. editor de esquemáticos, simulador lógico, editor de scripts de simulação, minimizador de conjuntos de funções lógicas Booleanas (Espresso), editor de máquinas de estados finitas, subconjunto de ferramentas de síntese automatizada para FPGAs, e sistema de descarga de arquivos de configuração para uma plataforma de prototipação específica empregando FPGAs), foram revisados alguns conceitos envolvendo o projeto de sistemas digitais **combinacionais** e **seqüenciais**. Além disto, foi exercitado o fluxo básico de implementação a partir de um projeto validado funcionalmente.

O presente Laboratório tem por objetivo estratégico efetivamente iniciar os trabalhos práticos específicos de Organização de Computadores (Unidade 02 da Disciplina, bem como da disciplina teórica companheira). Este primeiro passo consistirá em trabalhar em níveis de abstração mais altos, independentes de hardware, ignorando hardware e lidando com a abstração denominada Arquitetura de Conjunto de Instruções (em inglês, *Instruction Set Architecture*, ou *ISA*), a primeira camada de software acima do hardware de um sistema computacional (caso ignore-se e/ou identifique-se esta camada com o nível de código de máquina). Em resumo, trabalhar-se-á neste Laboratório exclusivamente sobre o tema de programação em Linguagem de Montagem.

Dado o contexto do parágrafo anterior, em acordo com o conteúdo visto de forma concomitante na disciplina teórica, adota-se aqui a arquitetura Cleópatra como alvo do trabalho. O objetivo específico deste laboratório é fixar os conteúdos de programação em linguagem de montagem (em inglês, *assembly language*) para a arquitetura Cleópatra, através do uso do sistema integrado de desenvolvimento, composto de 3 ferramentas acessíveis via **interface gráfica unificada**: um **editor de textos especializado**, o **montador Cleópatra** (**cleoasm.exe**) e o **simulador Cleópatra** (**cleosim.exe**). Todo o sistema é de domínio público, e foi inicialmente desenvolvido por Daniel Carvalho Liedke, quando de sua participação como aluno da disciplina, em 1998/II, e mais tarde como bolsista do Grupo de Apoio ao Projeto de Hardware (GAPH) da FACIN. Atualmente, o trabalho de Daniel vem evoluindo a partir do trabalho de outros bolsistas do grupo de pesquisa GAPH. Os docentes agradecem a Daniel o trabalho dedicado e a qualidade do sistema desenvolvido, que em muito beneficia o trabalho dos alunos de ambas disciplinas: Organização de Computadores e Laboratório de Organização de Computadores.

Obs: O ambiente de desenvolvimento Cleópatra já se encontra devidamente instalado nas máquinas do Laboratório. Caso deseje usá-lo fora do mesmo, faça o seguinte:

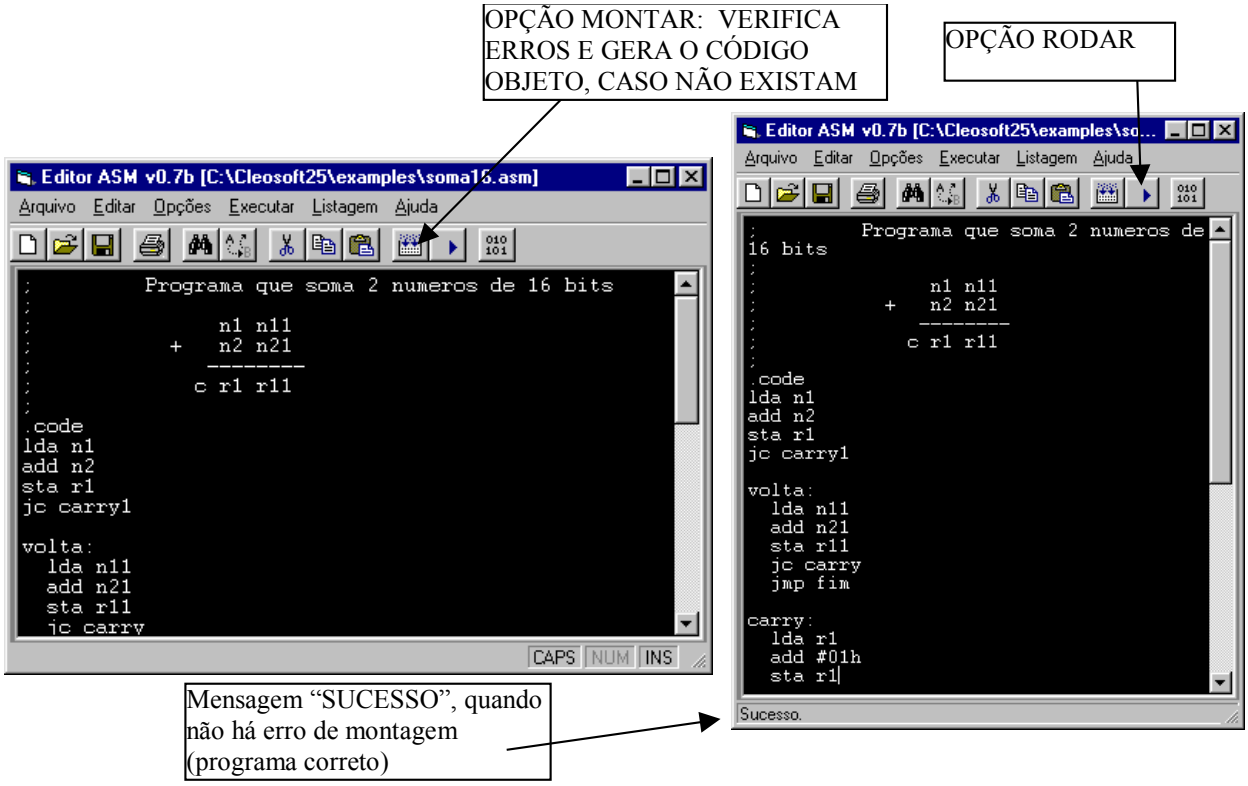
Copiar a distribuição Cleópatra da URL http://www.inf.pucrs.br/~moraes/org/cleosoft.zip. Basta descomprimir **cleosoft.zip** e executar o arquivo setup.exe para iniciar a instalação do simulador. **Obs**.: Após uma pré instalação, o processo de instalação solicita que a máquina seja reinicializada para poder prosseguir. Após reinicializar a máquina, relançar a instalação para concluir.

Abaixo detalhar-se-á a utilização básica dos programas do ambiente Cleópatra. A instalação básica vem com exemplos de programas que funcionam, programas estes de vários graus de

complexidade. Mais exemplos de enunciados e soluções na página de download da disciplina.

Parte II – Editor Cleópatra e Montador Cleópatra

- Função do programa montador: gerar o código objeto (arquivos **.cle*) a partir de arquivos texto contendo descrições de programas válidos em linguagem de montagem (assembly language) da Arquitetura Cleópatra. Se o programa não for sintaticamente correto, **cleoasm.exe** indica os erros (para uma descrição detalhada da sintaxe e da semântica da linguagem de montagem, ver o documento o_cleo2.01.pdf).
  - Entrada do montador: descrição em linguagem de montagem (assembly language) em texto ASCII, correspondendo a arquivos com terminação .asm.
  - Saídas do montador:
    1. código objeto (binário), terminação .cle;
    2. arquivos de listagem, terminação .txt;
    3. .hex para execução na placa de prototipação.
- Chamar o simulador a partir de "Start" → "Programs" → "Cleo Simulator 2.0" → "Cleo Simulator 2.0", ou caminho similar (depende da instalação e versão). A janela mostrada na Figura 1.a é exibida. Se não houver caminho, procurar diretório de nome **cleosoft20** ou similar no drive C. O nome do executável inicial do ambiente é **cleosim.exe**.



(a) janela inicial do montador. (b) montador com o programa soma16 carregado.

Figura 1 - Janela do programa Montador.

- A idéia aqui é abrir um programa fornecido com o software, a título de exemplo. Ir no menu "Arquivo", opção "Abrir". Escolher o arquivo **soma16.asm**, localizado no subdiretório **examples** da instalação do ambiente. Observar a estrutura do programa, com as *diretivas de montagem*[1] para designar início e fim da área de código (*.code* e *.endcode*), início e fim da área de dados (*.data* e *.enddata*), e definição de nome de área de dados e sua inicialização (*db*). Tarefa 1: **Estude, Pense e Responda** - São estas as únicas diretivas aceitas pelo montador

[1] Uma *diretiva de montagem* contém informações para o programa montador, orientando-o na geração do código objeto. Não se deve confundir diretivas de montagem com *mnemônicos da linguagem de montagem*, que são textos que fazem com que o programa montador gere código objeto executável pelo processador.

Cleópatra? Se não for este o caso, qual(is) a(s) diretiva(s) restante(s), qual sua utilidade e como funciona(m)?

- Utilizar a opção de menu "Executar" → "Montar" (Tecla de atalho F5 ou antepenúltimo botão mais à esquerda) para gerar o código objeto. Caso o programa esteja corretamente escrito, a mensagem ***sucesso*** é exibida na parte inferior esquerda da janela do montador. Senão, uma mensagem de erro aparece no mesmo lugar. Alterar o programa para que isto ocorra, re-executando o montador e analisando a mensagem gerada. Por exemplo, na primeira linha após o rótulo *volta*, trocar o mnemônico *lda* por *lad*. Corrigir o erro e montar o programa outra vez.
- O programa em código objeto, com terminação *.cle*, será utilizado pelo simulador. Ele se encontra no mesmo diretório do programa em linguagem de montagem.

Parte III - Simulador Cleópatra

- Após montar com sucesso o programa, clicar na opção de menu "Executar" → "Rodar" (Tecla de atalho *shift* F5 ou penúltimo botão à esquerda) na janela do editor. A seguinte janela aparece:

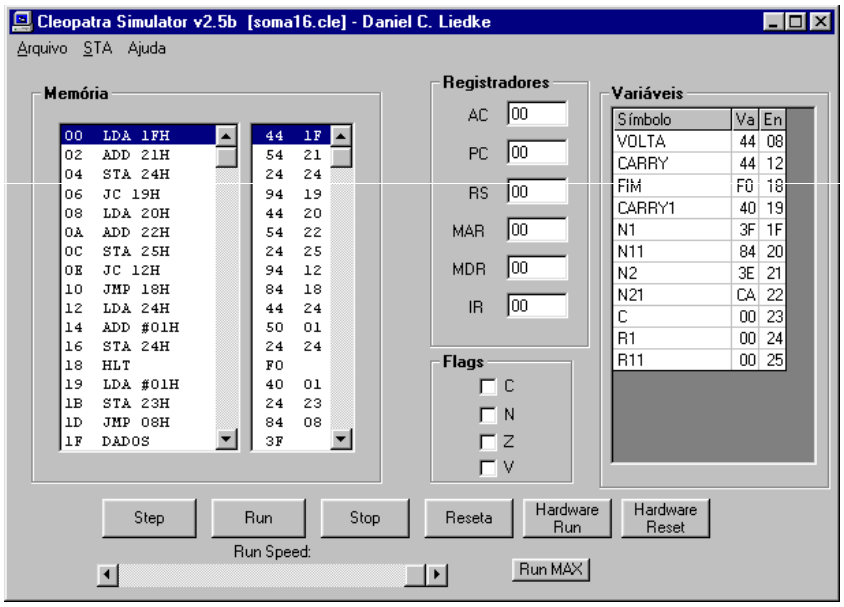


Figura 2 - Simulador da Cleópatra, com o arquivo soma16.cle carregado.

- Observar as 5 áreas desta janela:
  1. Conteúdo da memória principal do processador Cleópatra (áreas de dados e programas): a coluna da esquerda contém o código fonte produzido pela interpretação do código objeto em suas áreas de instruções e dados. A primeira área é delimitada pelas diretivas *.code*/*.endcode*, enquanto que a segunda área o é pelas diretivas *.data*/*.enddata*. A memória possui posições numeradas de 00H até 0FFH. Note que as instruções podem ocupar uma ou duas palavras de memória, enquanto que os dados são sempre mostrados palavra a palavra (identificados na coluna da esquerda pelo texto *DADOS* e na da direita pelo valor destes. A coluna da direita mostra o conteúdo total da memória em hexadecimal. A primeira coluna dá informação sobretudo sobre memória de programa (instruções), e a segunda sobretudo sobre a memória de dados. Tarefa 2: **Estude, Pense e Responda -** Onde começam e terminam as memórias de instruções e de dados, respectivamente? O que acontece com todas as posições de memória não referidas explicitamente como memória de dados ou de programa no simulador?
  2. Registradores internos da arquitetura: acumulador (AC), contador de programa (PC), registrador de endereço de retorno de subrotina (RS), registrador de endereço de memória (MAR), registrador de dados da memória (MDR) e registrador de instrução corrente (IR).

3. Qualificadores (flags): carry (C), negativo (N), zero (Z) e transbordo, ou overflow (V).
4. Rótulos definidos pelo usuário (os rótulos de posições do programa e também os identificadores na memória de dados, criados usando a diretiva *db*). Tarefa 3: **Estude, Pense e Responda** - Como se dá a geração dos endereços associados aos rótulos? Dê pelo menos dois exemplos de geração, pelo menos um de geração de rótulo para posição do programa e pelo menos um para posição da memória de dados.
5. Controle de execução: do passo-a-passo até execução direta (até que se encontre a primeira instrução *hlt*). A opção "RESETA" permite reinicializar o estado do programa.

- Execute as três primeiras linhas do programa exemplo, utilizando a opção "STEP".
  1. *LDA n1*. Foi traduzida para *LDA 1FH*. Observe se o conteúdo do acumulador após a execução desta primeira linha confere, ou seja, se AC está com 3FH.
  2. *ADD n2*. Instrução traduzida como *ADD 21H*. Após este passo executado, observe o conteúdo do acumulador, 7DH, que corresponde à soma de 3F + 3E.
  3. *STA r1*. Use o mouse para visualizar a posição de memória 24H (valor de r1) e observe que esta posição de memória recebeu o valor 7DH após a execução deste passo, e está marcada com um "*", indicando que o endereço foi alterado pelo programa.

Tarefa 4: Serão indicados, no momento da aula, quais exercícios cabem a cada equipe de alunos. Cada equipe deverá realizar 3 exercícios, 1 de cada grupo da lista contida no Apêndice abaixo. Os grupos referem-se a exercícios com grau de dificuldade similar. O grau de dificuldade da lista abaixo é crescente (Grupo 1 < Grupo 2 < Grupo 3). Logo, aconselha-se que seja seguida a ordem da lista para a resolução dos exercícios. Há também exercícios avançados, todos opcionais.

Tabela 1 – Especificação de programas para a Tarefa 4 (1 coluna por equipe, o professor atribui).

Grupo/Equipe	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
Grupo 1	5	2	3	4	6	7	8	9	9	8
Grupo 2	16	15	14	10	10	14	15	16	10	14
Grupo 3	17	18	19	20	21	21	20	19	18	17

Grupo/Equipe	E11	E12	E13	E14	E15	E16	E17	E18	E19	E20
Grupo 1	7	6	4	3	2	5	8	9	6	7
Grupo 2	15	16	16	15	14	10	10	14	15	16
Grupo 3	21	20	19	18	17	17	18	19	20	21

Parte IV - A fazer:

Tarefa 5: Elabore um relatório contendo todas as suas medidas, bem como a resposta a todas as questões que aparecem no texto. Entregue este relatório no prazo especificado na homepage da disciplina. Execute todas as 5 tarefas propostas acima.

- ***A Entregar:***
  - O relatório do Laboratório;
  - Para cada programa implementado – o **fluxograma**, o **código fonte** e **objeto** em versão magnética, contendo o programa e dados sobre os quais este executa de maneira correta;
  - Resposta às questões precedidas da menção **Estude, Pense e Responda** no texto acima.

Percentuais de nota atribuídos às Tarefas:

Item Avaliado	Percentual
Fluxogramas dos programas	**30%**
Implementação correta dos programas	**50%**
Respostas às questões	**20%**

Apêndice - Lista de Exercícios (* antes do enunciado indica problema com solução)

IMPORTANTE: palavras em itálico na lista são rótulos associados a endereços específicos de memória.

GRUPO 1:

1) *Somar uma constante a um vetor armazenado à partir da posição de memória cujo rótulo é *end1*. O número de elementos está armazenado na posição de memória cujo rótulo é *end2*.
2) Fazer um algoritmo que lê um array, calcula o número de elementos pares e ímpares e informa qual é o maior par e o maior ímpar.
3) Escrever um programa para mover um vetor armazenado entre os endereços de memória com rótulos *inicio1* e *fim1* para os endereços cujos rótulos são *inicio2* e *fim2*. Assumir que as seguintes condições são verdadeiras: valor(*inicio1*) < valor (*fim1*), valor (*inicio2*) < valor (*fim2*), (*fim1*-*inicio1*) = (*fim2*-*inicio2*) = (tamanho do vetor –1).
4) Contar o número de posições de memória com conteúdo igual a *0AAH* no vetor armazenado entre os endereços 080H e 0F5H.
5) Dados dois inteiros, A e B, armazenados, respectivamente, nas posições de memória cujos rótulos são *n1* e *n2*, armazenar na posição de memória com rótulo *max* o max(A,B) (valor máximo entre A e B) e na posição de memória com rótulo *min* o min(A,B) (definido de forma similar ao max), de acordo com a ordem total de inteiros.
6) Faça um algoritmo que mostre o processo do cálculo do valor de um byte representado em complemento de dois, decompondo o mesmo em uma série de potências de base 2. Ou seja, um byte será representado por 8 bytes.
   Exemplo: 47 = –0*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0
7) Descobrir se um número **n** é múltiplo de 4
8) Descobrir se um número **n** é múltiplo de 3.
9) Descobrir se um número **n** é múltiplo de um número **m** qualquer. Pressupor que **n** possa ser um número positivo ou negativo.

GRUPO 2:

10) 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 idéia 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, pode-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 um programa que recebe *n*, o número de bits de uma mensagem (assuma, para facilitar, que 0<n<8), *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. Escolha o tipo de paridade e documente-o. Dica: A função paridade de n bits tem relação estreita com a função ou-exclusivo de n bits.
11) *Multiplicar por somas sucessivas 2 inteiros positivos de 8 bits, armazenando o resultado em 16 bits. O multiplicando e o 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).
12) *Fazer um programa que gere o *n* primeiros números da seqüência de Fibonnacci, e armazenar a seqüências em endereços consecutivos à partir da posição de memória com rótulo *idx*. Assuma que *n* está entre 0 e 14 e explique como se resolveria o problema surgido a partir de valores de *n* acima de 14. Primeiro defina qual é este problema.
13) *Escreva um programa para criar um vetor *novo*, à 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*.

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

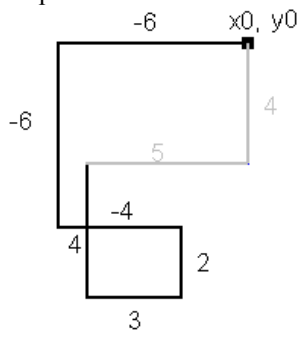
14) Escreva um programa para criar um vetor *novo*, à partir de dois vetores *v1* e *v2* de mesma dimensão *n*, segundo a relação estabelecida na equação abaixo. A interpretação da equação é: cada elemento de *novo*, na posição *k*, receberá o somatório dos mínimos dos vetores *v1* e *v2*, entre *0*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

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

15) Dados dois vetores, *a* e *vet*, de dimensão *n*, implemente um algoritmo para preencher *vet* a partir de *a* , segundo a equação abaixo. A interpretação da equação é: cada elemento de *vet*, na posição *k*, receberá o somatório de todos os elementos do vetor *a*, entre os índices *0* e *k*.

$$vet_k = \sum_{i=0}^{k} a_i \ , \quad 0 \le k < n$$

16) Seja dado um ponto inicial (x0, y0) e uma seqüência de **n** valores inteiros (o valor de *n* deve estar armazenado em uma posição de memória com rótulo *n*, e a seqüência 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 seqüência 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]. Resultado -5 para x e -4 para y. Decisão: primeiro deslocamento para a esquerda se negativo, para a direita se positivo.



GRUPO 3:

17) Implemente um algoritmo simples de ordenação **crescente**. 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.
18) Implemente um algoritmo simples de ordenação **decrescente**. O vetor de origem deve estar armazenado entre as posições de memória de rótulos *ini1* e *fim1*, e o vetor ordenado deve estar armazenado entre as posições de memória *ini2* e *fim2*.
19) Escrever um algoritmo que calcule os primeiros *n* números primos e os armazene seqüencialmente, a partir da posição de memória cujo rótulo é *nprimos*.
20) Faça um algoritmo que calcula a divisão de dois números de 8 bits (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*.
21) Faça um programa para ordenar 20 valores hexadecimais, que estão armazenados em 10 bytes. Sendo que dentro de cada byte, o nibble (conjunto de 4 bits) menos significativo deve conter o menor valor. Exemplo: Vetor de entrada: 034h, 0FAh, 0BCh, 077h, 01Ch (Vetor ordenado: 013h, 047h, 07Ah, 0BCh, 0CFh)

GRUPO AVANÇADO (PARA QUEM DESEJA IR MAIS LONGE):

22) Fazer um programa para automatizar uma agência de casamentos. Considere que a agência dispõe do cadastro de n homens e n mulheres. Neste cadastro existe uma ordenação de preferencia que os homens e as mulheres tem uns pelos outros. Selecione os n casais de forma a favorecer a melhor escolha do casal.
Exemplo: Os homens 1 e 2 têm como preferência maior a mulher 1, que tem como preferência o homem 3

Número	Homens	Mulheres
1	1, 2, 3	3, 2, 1
2	1, 3, 2	3, 1, 2
3	3, 2, 1	1, 3, 2

23) Dadas **n** caixas de 3 dimensões, fazer um programa para descobrir qual é o maior número de caixas que podem ser colocadas umas dentro das outras. Cada caixa pode ser representada, neste problema, como um vetor de três dimensões (**dx**, **dy**, **dz**) e as **n** caixas formam uma matriz **n** x 3.
24) Seja dada uma máquina que tem somente duas operações **inteiras**: multiplicar por 2 e dividir por 3, e que conhece inicialmente o número 1. Fazer um programa que gere qualquer número de 1 a 18 como uma expressão: número = (2^X)/(3^Y).
Exemplo: 2 = 2^1/3^0 3 = 2^5/3^2 4 = 2^2/4^0 5 = 2^4/3^1