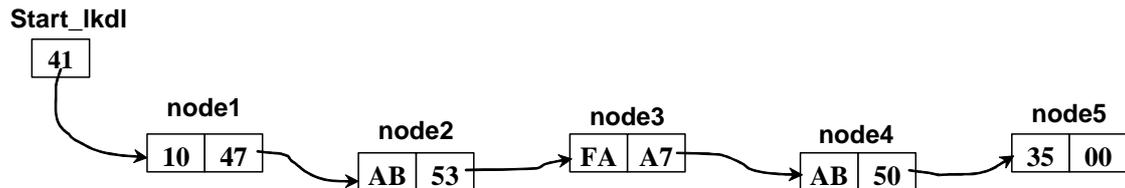


Exercícios Suplementares de Programação Assembly da Cleópatra

1. Listas encadeadas são estruturas de dados importantíssimas em aplicações onde o volume de dados a manipular não está definido antes da execução, e onde a locação dinâmica de memória é possível. A Figura abaixo mostra o exemplo de lista encadeada a ser usado na questão. Abaixo da Figura, encontra-se a área de dados que descreve a lista em questão. O elemento mais à esquerda na Figura é o ponteiro para o início da lista, uma posição de memória que contém o endereço do nodo inicial. Cada nodo da lista corresponde a um par de posições de memória consecutivas da Cleópatra, onde a primeira posição é o dado armazenado, denominado conteúdo e a segunda é o endereço da posição de memória onde inicia o próximo nodo da lista, denominado ponteiro. O fim da lista é indicado por um ponteiro com valor 00h. Todos os valores mostrados estão em hexadecimal. Perceba (1) o uso de diretivas ORG na linguagem de montagem para localizar os nodos em posições de memória específicas; (2) o uso de rótulos para definir o valor dos campos ponteiros.



```

1  .data
2
3  start_lkdl:      org    #40h    ; Abaixo, segue a descrição da lista encadeada exemplo
4                  db      #node1
5
6  node1:          db      #10h    ; primeiro nodo ocupa as posições de memória 41h e 42h
7                  db      #node2  ; campo ponteiro do nodo node1
8
9  node2:          org    #47h    ; segundo nodo ocupa as posições de memória 47h e 48h
10                 db      #0ABh   ; campo de conteúdo do nodo node2
11                 db      #node3  ; campo ponteiro do nodo node2
12
13  node3:          org    #53h    ; terceiro nodo ocupa as posições de memória 53h e 54h
14                 db      #0FAh   ; campo de conteúdo do nodo node3
15                 db      #node4  ; campo ponteiro do nodo node3
16
17  node4:          org    #0A7h   ; quarto nodo ocupa as posições de memória A7h e A8h
18                 db      #0ABh   ; campo de conteúdo do nodo node4
19                 db      #node5  ; campo ponteiro do nodo node4
20
21  node5:          org    #50h    ; quinto nodo ocupa as posições de memória 50h e 51h
22                 db      #35h    ; campo de conteúdo do nodo node5
23                 db      #0h     ; campo ponteiro do nodo node5 - fim da lista encadeada
24 .enddata

```

Pede-se:

- I. Implemente um programa que percorre uma lista encadeada com estrutura similar à do exemplo e **some uma constante** a cada campo de conteúdo de nodos da lista.

```

*** ACRESCENTAR NO INÍCIO DA ÁREA DE DADOS ÁREA DE DADOS:
.data
C:          b      #10h    ; constante a ser somada a cada elemento
pp_element: db      #0h     ; ponteiro para ponteiro para o início da lista

```

- II. Implemente um programa que **conte quantos elementos possui uma lista encadeada** com estrutura similar à do exemplo. Armazene o número de elementos numa posição de memória NUM_EL.

```

*** ACRESCENTAR NO INÍCIO DA ÁREA DE DADOS ÁREA DE DADOS:
.data
NUM_EL:     db      #0h    ; contador para computar num de elementos da lista
pp_element: db      #0h    ; ponteiro para ponteiro para o início da lista

```

- III. Implemente um programa que **conte quantos elementos de uma lista encadeada são pares**. Armazene o número de elementos pares obtidos numa posição de memória NUM_EVEN.

```

*** ACRESCENTAR NO INÍCIO DA ÁREA DE DADOS ÁREA DE DADOS:
.data
NUM_EVEN:   db      #0h    ; contador do número de elementos pares
pp_element: db      #0h    ; ponteiro para ponteiro para o início da lista

```

- IV. Implemente uma subrotina para **inserir um elemento na lista encadeada**, cujo valor seja #0F6h, após o elemento cujo conteúdo seja #0FFh. (ou no final da lista caso este não exista).

```

*** INÍCIO DO PROGRAMA:
lkd_lst_ex:    lda    #0F6h        ; campo de conteúdos do nodo a inserir
              sta    new_node
              lda    #0ffh        ; campo de conteúdos para teste de inserção
              sta    tst_value
              jsr    ch_lst_ins    ; chama rotina de inserção
              hlt

*** ACRESCENTAR NA ÁREA DE DADOS:
.data
new_node:     db    #0h          ; campo de conteúdos do nodo a inserir
new_node_l:   db    #0          ; campo de ligação do nodo a inserir, vazio no início

tst_value:    db    #0h          ; campo de conteúdos para teste de inserção
pp_element:   db    #0h          ; ponteiro para ponteiro para o início da lista

```

Atenção: os programas **NÃO DEVEM** usar os rótulos *node1*, *node2*, etc. A única informação que se deve assumir disponível é *start_lkd*, o ponteiro para o início da lista encadeada. Se for necessário, criar campos adicionais na memória de dados, e use-os no seu programa.

2. Considere o seguinte problema: dado um vetor de 'n' elementos, criar um novo vetor de 'm' elementos obtido pela eliminação de elementos repetidos do vetor original. Por exemplo, dado $n=10$ e $vet=\{ 9, 3, 1, 2, 1, 6, 3, 4, 2, 6 \}$, o programa deve retornar: $m=6$ e $novo=\{9, 3, 1, 2, 6, 4\}$. Os elementos em negrito foram suprimidos por estarem repetidos. Uma solução parcial é dada abaixo, onde é fornecido o laço principal do programa e a área de dados. Fazer:

- a) **rotina seta_rep**. Esta rotina deve percorrer o vetor *novo* verificando se neste vetor há elemento igual ao apontado pela posição atual de vet (vetor original). Não havendo valor igual gravar 0 em *repetido* e havendo valor repetido gravar 1 em *repetido*.
- b) **Rotina novo_valor**: acrescentar no vetor *novo* o elemento apontado pela posição atual de vet.

```

.CODE

LACO:         JMP     SETA_REP,R        ; verifica se em 'novo' tem elemento igual ao
                                              ; apontado por vet (vet,I)

VOLTA:        LDA     REPETIDO
              JZ      NOVO_VALOR,R    ; não havendo, acrescente em 'novo'

V2:           LDA     VET
              ADD     #01
              STA     VET              ; avança um elemento de 'vet'
              LDA     N
              ADD     #0FFH
              JZ      FIM,R
              STA     N
              JMP     LACO,R          ; novo elemento de 'vet', até o final do vetor

FIM:          HLT

SETA_REP:     a fazer
NOVO_VALOR:   a fazer

.ENDCODE

.DATA
TMP:         DB #00                    ; ARMAZENA TEMPORARIAMENTE O ENDEREÇO DO NOVO VETOR
K:           DB #00                    ; VARIÁVEL TEMPORÁRIA
REPETIDO:    DB #00                    ; FLAG QUE INDICA REPETIÇÃO DE VALOR
N:           DB #0AH                    ; TAMANHO DO VETOR
M:           DB #00                    ; INDICE DO NOVO VETOR
VET:         DB V1                      ; VETOR CONTENDO OS DADOS
NOVO:        DB VN                      ; VETOR QUE CONTERÁ OS DADOS QUE NÃO SÃO REPETIDOS
V1:          DB #09H,#03H,#01H,#02H,#01H,#06H,#03H,#04H,#02H,#06H ; VETOR
VN:          DB #0

.ENDDATA

```

I - SOMA UMA CONSTANTE A CADA ELEMENTO:

```
.code
ch_lst_cte:  lda    start_chl    ; get start pointer pointer
             sta    pp_element ; store in pointer to pointer element

add_c_loop:  jz     fim        ; see if list is at the end. If yes, stop
             lda    pp_element,I ; else, get next node contents field
             add    C          ; add constant
             sta    pp_element,I ; store new value back
             lda    pp_element  ;
             add    #1         ; generate address of link field for current node
             sta    pp_element  ; and update in memory
             lda    pp_element,I ; get link field
             sta    pp_element  ; update pointer to next node
             jmp   add_c_loop   ; return to look for the insertion place

fim:        hlt
.endcode
```

II - NÚMERO DE ELEMENTOS DA LISTA:

```
.code
ch_lst_cte:  lda    #0          ;
             sta    NUM_EL     ; initialize counter
             lda    start_chl  ; get start pointer pointer
             sta    pp_element ; store in pointer to pointer element

count_loop:  jz     fim        ; see if list is at the end. If yes, stop
             lda    NUM_EL     ;
             add    #1         ; else, increment counter
             sta    NUM_EL     ; and store it back
             lda    pp_element ;
             add    #1         ; generate address of link field for current node
             sta    pp_element ; and update in memory
             lda    pp_element,I ; get link field
             sta    pp_element ; update pointer to next node
             jmp   count_loop  ; return to look for the insertion place

fim:        hlt
.endcode
```