"Ejercicios codificacion"

JOSE LUIS SANDOVAL PEREZ

Ingeniería en Computacion inteligente

7-A

04/11/2024

Programa 1:

Implemente un programa en ensamblador que ordene de forma ascendente 100 localidades de memoria, comenzando en la localidad 200h.

Codigo

```
; Programa para ordenar 100 localidades de memoria en forma ascendente, comenzando en la localidad 200h

ORG 100h

MOV SI, 200h          ; Inicio de la lista en la dirección 200h
MOV CX, 100           ; Número de elementos a ordenar

ordena:
   MOV DI, SI         ; Reiniciar el índice DI en el inicio de la lista
   DEC CX             ; Bajar el contador en cada pasada

   mov BX, CX         ; BX servirá como contador para la pasada

   ordena_burbuja:
      MOV AL, [DI]    ; Cargar el valor en DI
      MOV DL, [DI+1] ; Cargar el valor siguiente en DI+1

      CMP AL, DL      ; Comparar los dos valores
      JBE no_intercambio ; Si ya están en orden, no intercambiar

      ; Intercambiar si AL > DL
      MOV [DI], DL    ; Poner el valor menor en DI
      MOV [DI+1], AL ; Poner el valor mayor en DI+1

   no_intercambio:
      INC DI         ; Mover a la siguiente posición
      DEC BX         ; Reducir el contador de la pasada
      JNZ ordena_burbuja ; Repetir hasta que BX llegue a 0

   LOOP ordena       ; Repetir la rutina de burbuja hasta que CX llegue a 1

INT 20h               ; Termina el programa
```

Este programa ordena en forma ascendente los valores ubicados en 100 posiciones de memoria, comenzando en la dirección 200h y terminando en 263h. Se implementa un algoritmo de ordenamiento de tipo **burbuja** (Bubble Sort), en el cual se comparan elementos consecutivos y se intercambian si están fuera de orden. Este proceso se repite hasta que toda la lista esté ordenada.

Configuración Inicial:

1. `MOV CX, 64`: Configura el registro `CX` para realizar 100 iteraciones (64 en hexadecimal).
2. `MOV SI, 200h`: Define el inicio de la lista a ordenar en la dirección `200h`.
3. **Bucle Externo de Ordenamiento (`ORDENAMIENTO`)**:
4. Este bucle externo controla las pasadas de comparación en el arreglo de datos.
5. **Bucle Interno de Comparación (`COMPARACION`)**:
6. `MOV DI, SI`: Reinicia el índice `DI` en cada pasada para recorrer el arreglo desde el principio.
7. `MOV AL, [DI]` y `MOV AH, [DI+1]`: Carga los valores consecutivos en `AL` y `AH`.
8. `CMP AL, AH`: Compara los valores. Si `AL` (actual) es mayor que `AH` (siguiente), realiza un intercambio.
9. **Intercambio**:
    a. `MOV [DI], AH` y `MOV [DI+1], AL`: Si el valor actual es mayor que el siguiente, se intercambian.
10. `INC DI`: Avanza a la siguiente posición de la lista para continuar comparando.
11. **Terminación del Bucle**:
12. `DEC CX`: Decrementa el número de pasadas restantes.
13. `JNZ ORDENAMIENTO`: Repite el bucle hasta que todos los valores estén ordenados.

Programa 2:

Escriba un programa en ensamblador que multiplique dos números ubicados en las localidades 202h y 203h, dejar el resultado en formato little endian en las localidades 200h y 201h. Aplicar el método de La Russe.

codigo:

```
; Programa para multiplicar dos números en 202h y 203h con el método de La Russe

ORG 200h

MOV AL, [202h]      ; Cargar el primer número en AL
MOV BL, [203h]      ; Cargar el segundo número en BL
MOV CX, 0           ; CX usará como acumulador del resultado

multiplicacion:
   TEST AL, 1       ; Verificar si el bit menos significativo de AL es 1
   JZ siguiente     ; Si es 0, no sumamos

   ADD CX, BX       ; Sumar el valor de BL al acumulador CX si AL es impar
```

```
siguiente:
    SHR AL, 1          ; Dividir AL por 2 (desplazamiento a la derecha)
    SHL BL, 1          ; Multiplicar BL por 2 (desplazamiento a la izquierda)
    CMP AL, 0          ; Verificar si AL es 0
    JNZ multiplicacion ; Repetir hasta que AL sea 0

; Guardar el resultado en formato little endian en 200h y 201h
MOV [200h], CL         ; Byte menos significativo en 200h
MOV [201h], CH         ; Byte más significativo en 201h
INT 20h                ; Termina el programa
```

Este programa implementa el método de multiplicación rusa (Método de La Russe) para multiplicar dos números ubicados en las direcciones de memoria `202h` (multiplicando) y `203h` (multiplicador). El resultado se almacena en formato little-endian en las direcciones `200h` (parte baja) y `201h` (parte alta).

1. **Cargar Valores de Memoria**:
2. `MOV AL, [202h]`: Carga el multiplicando desde la dirección `202h`.
3. `MOV BL, [203h]`: Carga el multiplicador desde la dirección `203h`.
4. `XOR CX, CX`: Inicializa `CX` a cero para usarlo como acumulador del resultado.
5. **Bucle de Multiplicación (`MULTIPLICACION`)**:
6. `TEST BL, 1`: Verifica si el multiplicador `BL` es impar (si su bit menos significativo es 1).
7. `ADD CX, AX`: Si `BL` es impar, suma el valor actual de `AX` (multiplicando) al acumulador `CX`.
8. `SHR BL, 1`: Divide el multiplicador `BL` por 2 (desplazamiento a la derecha).
9. `SHL AL, 1`: Multiplica el multiplicando `AL` por 2 (desplazamiento a la izquierda).
10. **Guardar el Resultado**:
11. `MOV [200h], CL` y `MOV [201h], CH`: Almacena el resultado en `200h` y `201h` en formato little-endian.

Programa 3

Escriba un programa en ensamblador que calcule la raíz cuadrada de un número ubicado en la localidad 201h, dejar el resultado en la localidad 200h. Aplique el método de la raíz aproximada.

codigo

```
; Programa para calcular la raíz cuadrada de un número en 201h

ORG 200h

MOV AL, [201h]         ; Cargar el número en AL
MOV BL, 0              ; BL servirá como aproximación inicial

raiz_aproximada:
    INC BL             ; Incrementamos la posible raíz
    MOV AH, BL         ; AH = BL (valor a verificar)
    MUL BL             ; Multiplicar BL por sí mismo (BL * BL)

    CMP AL, AH         ; Comparar con el número original en AL
    JAE raiz_aproximada ; Si el cuadrado es menor o igual, seguir buscando
```

```
DEC BL              ; BL contiene la raíz cuadrada aproximada
MOV [200h], BL        ; Guardar el resultado en 200h

INT 20h             ; Termina el programa
```

Este programa calcula la **raíz cuadrada aproximada** de un número ubicado en la dirección `201h` usando un método de aproximación sucesiva. El resultado se almacena en la dirección `200h`.

1. **Cargar Valor en Memoria**:
2. `MOV AL, [201h]`: Carga el número del cual queremos encontrar la raíz cuadrada aproximada.
3. `MOV CX, 0`: Inicializa `CX` en cero como aproximación inicial de la raíz cuadrada.
4. `MOV BL, AL`: Guarda el valor original en `BL` para comparaciones.
5. **Bucle de Aproximación (`RAIZ_APROX`)**:
6. `INC CX`: Incrementa `CX` (aproximación de la raíz) en cada iteración.
7. `MOV AX, CX` y `MUL CX`: Calcula el cuadrado de `CX` y lo guarda en `AX`.
8. `CMP AX, BL`: Compara el cuadrado con el número original.
9. `JBE RAIZ_APROX`: Si el cuadrado es menor o igual que el número original, repite el ciclo.
10. **Guardar el Resultado**:
11. `DEC CX`: Ajusta `CX` a la última raíz válida (una menor si el cuadrado actual excede el valor).
12. `MOV [200h], CL`: Guarda la raíz cuadrada aproximada en `200h`.