Arquitectura de Computadoras I.

Ejercicio: Contador de Números Pares e Impares en un Arreglo

Objetivo: Escribir un programa en ensamblador que recorra un arreglo de números almacenado en memoria y cuente cuántos son pares y cuántos son impares. Los resultados deben almacenarse en memoria y mostrarse en pantalla.

Especificaciones

1. Se tiene un arreglo de **10 números enteros** almacenado en memoria.
2. El programa debe recorrer el arreglo y determinar cuántos números son **pares** y cuántos son **impares**.
3. Los resultados deben guardarse en memoria en dos variables:
    - `pares`: cantidad de números pares.
    - `impares`: cantidad de números impares.
4. Debe mostrarse el resultado en pantalla.

Ejemplo

Entrada:

Arreglo en memoria:

```
assembly
CopiarEditar
.data
numeros db 4, 7, 12, 3, 9, 8, 2, 10, 15, 6
```

Salida esperada en memoria:

```
assembly
CopiarEditar
pares    db 6
impares  db 4
```

Salida en pantalla:

```
yaml
CopiarEditar
Números pares: 6
Números impares: 4
```

Pistas para la Implementación

- Usar registros como `AL` para recorrer los valores del arreglo.
- Aplicar la operación `AND` con `1` para determinar si un número es par o impar.
- Usar un contador para cada categoría (`pares`, `impares`).
- Utilizar una interrupción (`int 21h`) para mostrar los resultados en pantalla si se trabaja en DOS.

Versión en Ensamblador x86 (MASM)

Aquí tienes un código base para guiar a los estudiantes:

```
assembly
CopiarEditar
.model small
.stack 100h
.data
    numeros db 4, 7, 12, 3, 9, 8, 2, 10, 15, 6
    pares   db 0
    impares db 0
    msg_pares  db 'Numeros pares: $'
    msg_impares db 'Numeros impares: $'
    salto_linea db 10, 13, '$'

.code
start:
    mov ax, @data
    mov ds, ax

    mov cx, 10         ; Tamaño del arreglo
    mov si, 0          ; Índice del arreglo

contador:
    mov al, numeros[si] ; Cargar número
    test al, 1          ; Comprobar si es impar (AL AND 1)
    jnz es_impar        ; Si es impar, saltar a es_impar
    inc pares           ; Si es par, incrementar pares
    jmp siguiente

es_impar:
    inc impares         ; Incrementar impares

siguiente:
    inc si             ; Siguiente número
    loop contador      ; Repetir hasta recorrer el arreglo

; Mostrar resultado en pantalla
    mov dx, offset msg_pares
    mov ah, 09h
```

```
    int 21h

    mov al, pares
    add al, '0'
    mov dl, al
    mov ah, 02h
    int 21h

    mov dx, offset salto_linea
    mov ah, 09h
    int 21h

    mov dx, offset msg_impares
    mov ah, 09h
    int 21h

    mov al, impares
    add al, '0'
    mov dl, al
    mov ah, 02h
    int 21h

    mov ah, 4Ch ; Terminar programa
    int 21h

end start
```

Preguntas

1. ¿Cómo podría optimizarse el programa para trabajar con arreglos más grandes?
2. ¿Cómo podría adaptarse este código para ensamblador RISC o ARM?
3. ¿Qué otras operaciones se podrían aplicar a los números del arreglo?