Proyecto 1: Ensamblador de X86_64 bits

Gutiérrez Mata Carlos Andrés EL4314 – Arquitectura de Computadoras
Primer Semestre 2025

Abstract

Este proyecto describe la implementación de un programa en ensamblador x86-64 que lee nombres y notas desde archivos, los almacena en memoria, los ordena (tanto alfabéticamente como numéricamente usando Bubble Sort) y genera un histograma basado en las notas. Además, se incluye la lectura de un archivo de configuración para parámetros de presentación.

I. Repositorio

https://github.com/cguttierrez/Proyecto_1_Esamblador_de_X86_64_bits

II. Descripción del Diseño de Software

El programa se compone de varias secciones y rutinas en ensamblador, que se describen a continuación.

A. Gestión de Archivos y Configuración

Lógica Utilizada:

Se abre el archivo de configuración (`Ruta_configuracion.txt`) en modo solo lectura. Se carga su contenido en un buffer para, posteriormente, imprimirlo en pantalla. En caso de error al abrir el archivo, se muestra un mensaje de error. Además, se utiliza un archivo de datos (`Ruta_datos.txt`) que contiene los nombres y notas.

Código representativo:

```
; Abrir archivo de configuración
mov rdi, filename_config     ; Nombre del archivo
mov rsi, 0                   ; Modo: solo lectura
mov rax, 2                   ; syscall: open
syscall                      ; Abre el archivo
mov rbx, rax                 ; Guardar el descriptor

; Leer el contenido completo del archivo (hasta 1024 bytes)
mov rdi, rbx                 ; Descriptor del archivo
mov rsi, config_buffer       ; Buffer donde se almacenará el contenido
mov rdx, 1024                ; Tamaño máximo a leer
mov rax, 0                   ; syscall: read
syscall                      ; Se guarda en rax el número de bytes leídos
```

B. Almacenamiento de Nombres y Notas

Lógica Utilizada:

Se abre el archivo de datos y se lee completamente en un buffer. Luego, se recorre el buffer carácter a carácter. Cuando se detecta un salto de línea (ASCII 10), se considera el final de un nombre, se termina la cadena con un byte nulo y se almacena la dirección inicial en el arreglo `nombres`. Se ignoran líneas vacías.

Código representativo:

```
leer_nombres:
    cmp byte [rdi], 0          ; Si es fin del archivo
    je fin_lectura
    cmp byte [rdi], 10         ; Si hay salto de línea
    je guardar_nombre

    mov al, [rdi]
    mov [rsi], al
    inc rsi
    inc rdi
```

```
    cmp rsi, almacenamiento + 8192
    jge fin_lectura
    jmp leer_nombres

guardar_nombre:
    cmp rsi, rbx
    je ignorar_nombre
    mov byte [rsi], 0          ; Termina la cadena
    mov rax, rbx
    mov [nombres + rdx*8], rax ; Guarda la dirección del nombre
    inc rdx
    cmp rdx, 1024
    jge fin_lectura
    inc rsi                    ; Saltar el carácter de salto de línea
    inc rdi
    mov rbx, rsi
    jmp leer_nombres
```

Diagrama de flujo del funcionamiento

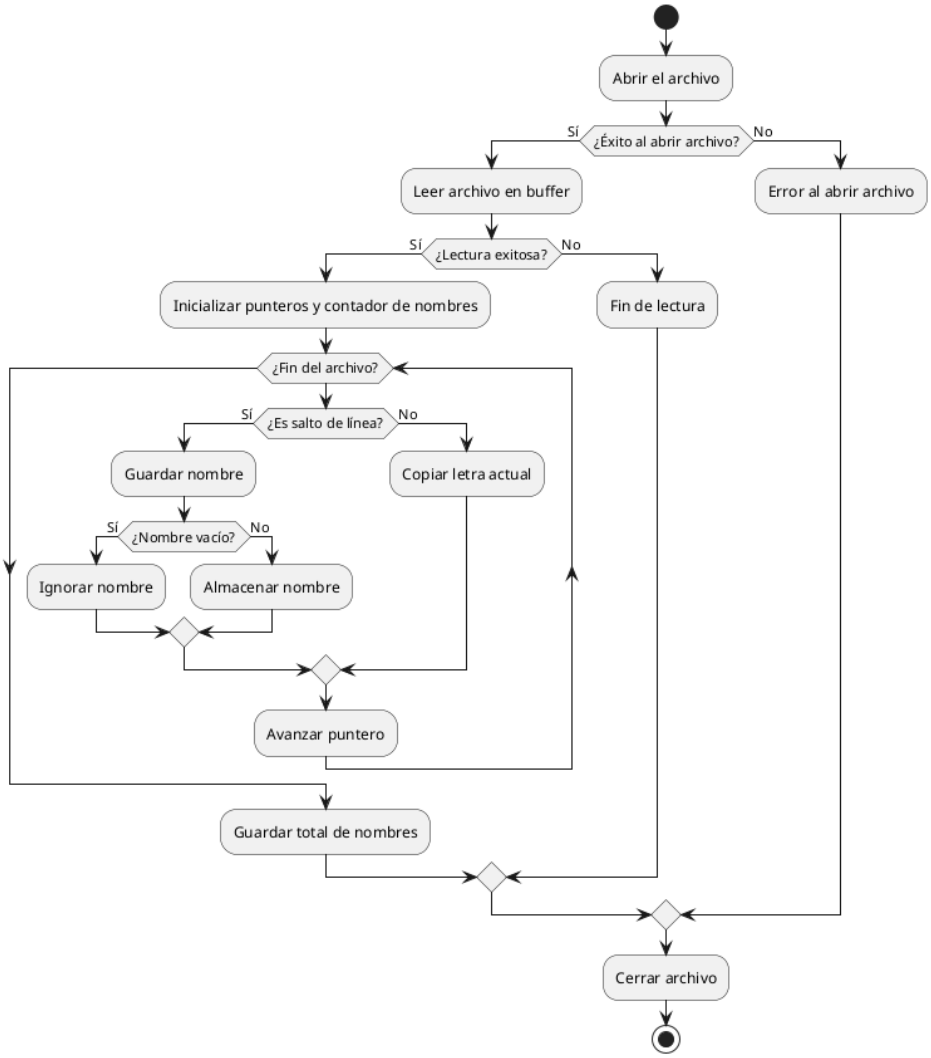

Fig. 1. Diagrama de flujo para el almacenamiento.

C. Extracción y Conversión de Notas

Lógica Utilizada:

La rutina `obtener_nota` busca la secuencia `"nota:"` en cada cadena. Una vez localizada, salta dicha etiqueta y convierte la secuencia de dígitos ASCII a su valor numérico acumulándolos en `rax`.

Código representativo:

```
obtener_nota:
    push rdi
    push rsi
    push rdx
    push rcx
buscar_nota:
    mov al, [rdi]
    cmp al, 0
    je fin_obtener
    mov rsi, rdi
    cmp byte [rsi], 'n'
    jne siguiente_char
    cmp byte [rsi+1], 'o'
    jne siguiente_char
    cmp byte [rsi+2], 't'
    jne siguiente_char
    cmp byte [rsi+3], 'a'
    jne siguiente_char
    cmp byte [rsi+4], ':'
    jne siguiente_char
    add rsi, 5              ; Saltar "nota:"
    xor rax, rax            ; Inicializar acumulador
convertir_digitos:
    movzx rcx, byte [rsi]
    cmp rcx, '0'
    jb fin_convertir
    cmp rcx, '9'
    ja fin_convertir
    imul rax, rax, 10
    sub rcx, '0'
    add rax, rcx
    inc rsi
    jmp convertir_digitos
fin_convertir:
    jmp fin_obtener
siguiente_char:
    inc rdi
    jmp buscar_nota
fin_obtener:
    pop rcx
    pop rdx
    pop rsi
    pop rdi
    ret
```

Diagrama de flujo del funcionamiento

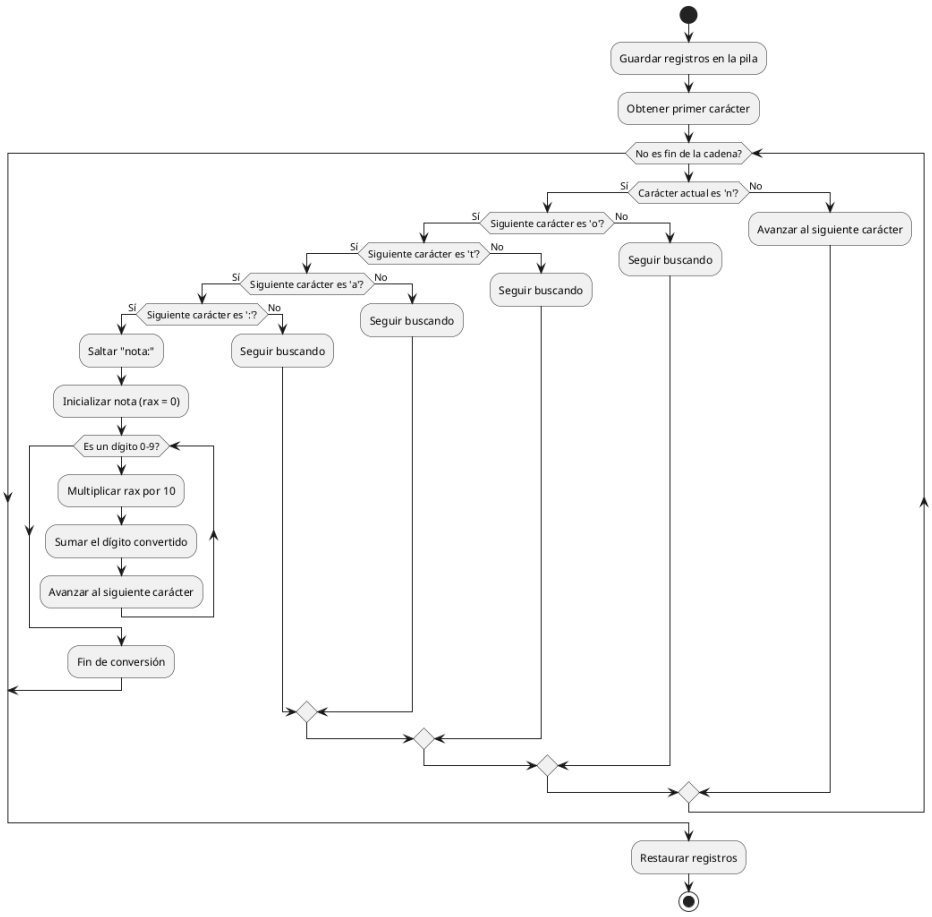

Fig. 2. Diagrama de flujo para el ordenamiento para obtener nota.

D. Ordenamiento de Datos

El programa realiza dos ordenamientos mediante el algoritmo Bubble Sort, el cual consiste en:

- Comparación y cambio de posición: Compara el primer par de elementos. Si el primero es mayor que el segundo, los intercambia.
- Iteración: Luego, compara el siguiente par de elementos, y repite el proceso hasta el final de la lista.
- Repetir: Al final de cada pasada por la lista lo hace hasta que este sea ordenado completamente.

1) Ordenamiento Alfabético: **Lógica Utilizada:**

Se utiliza la rutina `strcmp` para comparar dos cadenas. Si la primera es mayor, se intercambian sus posiciones en el arreglo `nombres`.

Código representativo:

```
bubble_sort_alfabetico:
    mov rcx, [total_nombres]
    cmp rcx, 1
    jle fin_bubble_alfabetico
    dec rcx                      ; rcx = total_nombres - 1
ordenar_alfabetico:
    mov rdx, 0
    mov r8, 0                   ; Flag de intercambio
alfabetico_loop:
    cmp rdx, rcx
    jge alfabetico_check
    mov rdi, [nombres + rdx*8]
    mov rsi, [nombres + (rdx+1)*8]
    call strcmp
```

```
    cmp rax, 0                ; Si s1 > s2, se realiza el intercambio
    jle alfabetico_no_swap
    mov rax, [nombres + rdx*8]
    mov rbx, [nombres + (rdx+1)*8]
    mov [nombres + rdx*8], rbx
    mov [nombres + (rdx+1)*8], rax
    mov r8, 1
alfabetico_no_swap:
    inc rdx
    jmp alfabetico_loop
alfabetico_check:
    cmp r8, 1
    je ordenar_alfabetico
fin_bubble_alfabetico:
    ret
```

Diagrama de flujo del funcionamiento

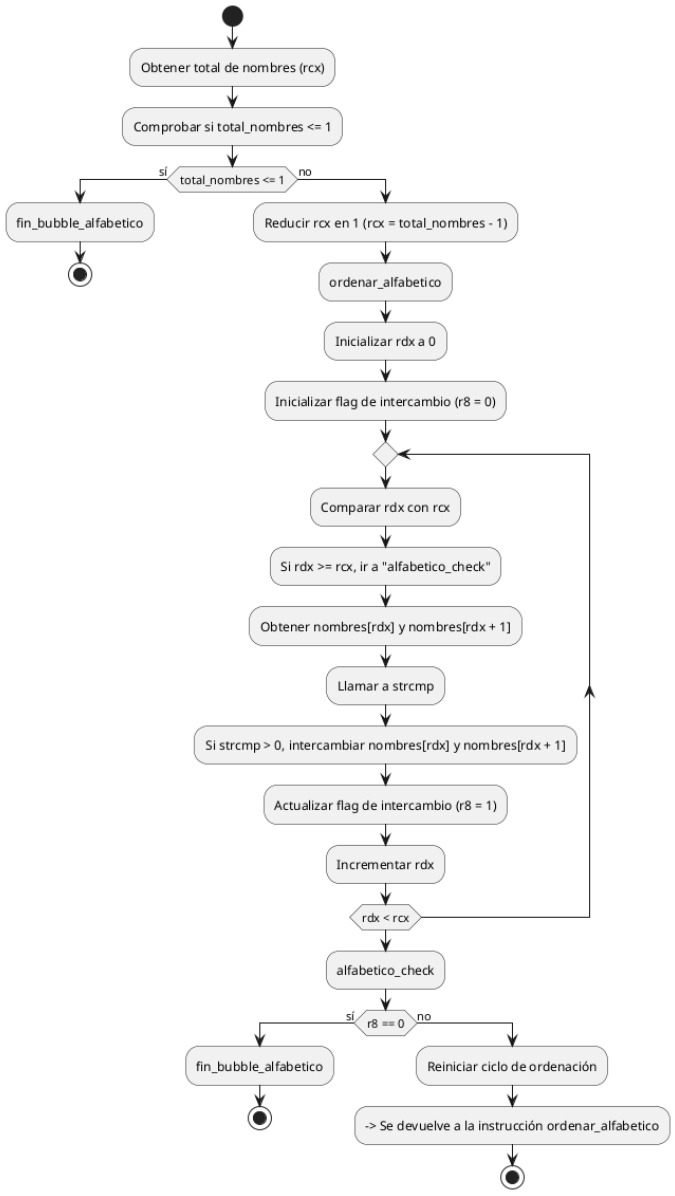

Fig. 3. Diagrama de flujo para el ordenamiento por orden alfabético.

Diagrama de flujo del funcionamiento

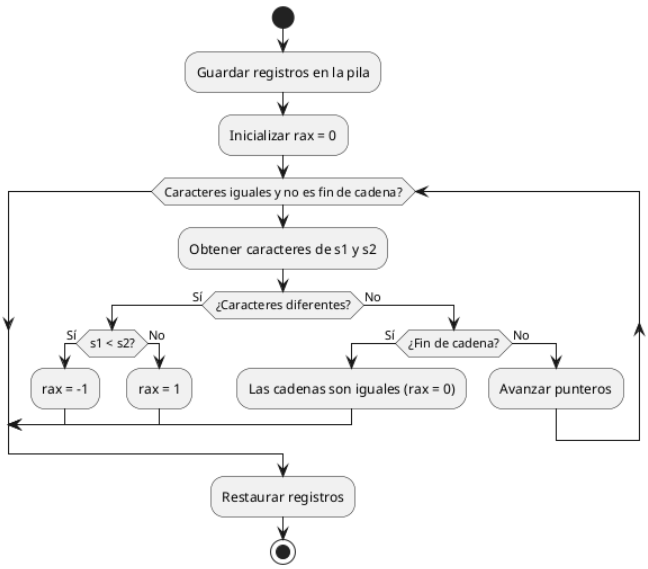

Fig. 4. Diagrama de flujo de strcmp

2) Ordenamiento Numérico (por Nota): **Lógica Utilizada:**
Se extrae la nota de cada cadena usando `obtener_nota`. Se comparan dos notas consecutivas y, si la nota actual es menor que la siguiente, se intercambian los punteros en el arreglo, de forma que el arreglo quede ordenado de mayor a menor.

Código representativo:

```
bubble_sort_numerico:
    mov r10, [total_nombres]
    cmp r10, 1
    jle fin_bubble_num
    dec r10                      ; r10 = total_nombres - 1
ordenar_num:
    mov rdx, 0
    mov r8, 0                   ; Flag de intercambio
num_loop:
    cmp rdx, r10
    jge num_check
    mov rdi, [nombres + rdx*8]
    call obtener_nota
    mov rbx, rax              ; Nota actual
    mov rdi, [nombres + (rdx+1)*8]
    call obtener_nota
    mov r9, rax               ; Nota del siguiente elemento
    cmp rbx, r9
    jge num_no_swap         ; No se intercambia si la nota actual es mayor o igual
    mov rax, [nombres + rdx*8]
    mov rbx, [nombres + (rdx+1)*8]
    mov [nombres + rdx*8], rbx
    mov [nombres + (rdx+1)*8], rax
    mov r8, 1
num_no_swap:
    inc rdx
    jmp num_loop
num_check:
    cmp r8, 1
    je ordenar_num
fin_bubble_num:
```

```
    ret
```

Diagrama de flujo del funcionamiento

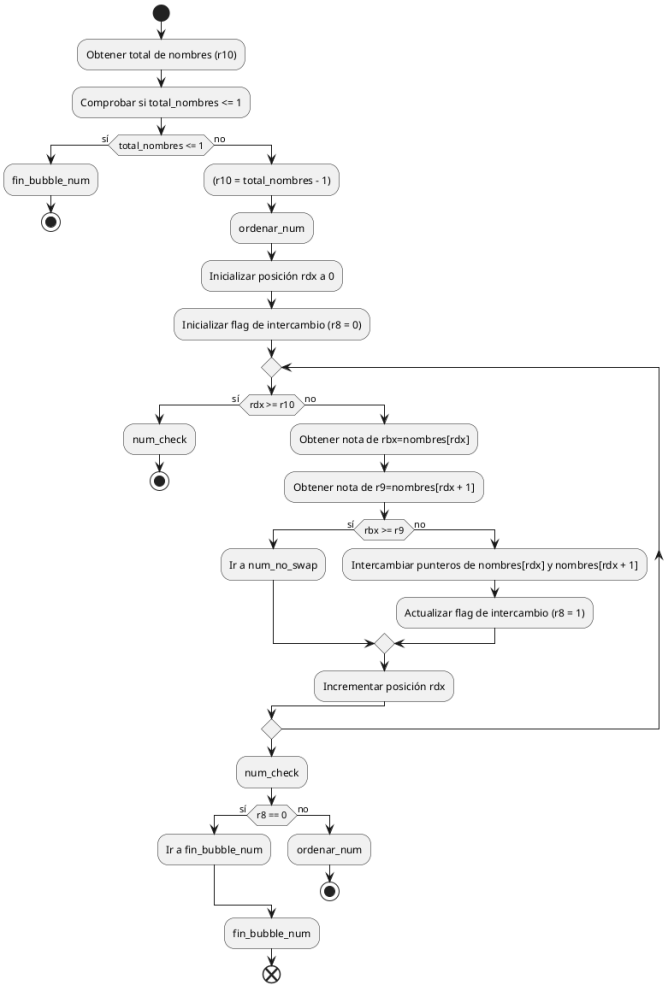

Fig. 5. Diagrama de flujo para ordenamiento por notas.

E. Generación del Histograma

Lógica Utilizada:

Se cuenta la cantidad de notas en intervalos (bins) de 10 unidades. Si la nota es menor que 10 o mayor o igual a 100, se fuerza su asignación al primer o último bin, respectivamente. Se incrementa el contador correspondiente en el arreglo `frec_count`.

Código representativo:

```
contar_intervalos_notas:
    xor rbx, rbx          ; Índice para recorrer nombres
    mov rcx, [total_nombres]
    test rcx, rcx
    jle fin_contar_notas

    ; Limpiar el arreglo de bins
    xor rax, rax
limpiar_bin_count:
    mov qword [frec_count + rax*8], 0
    inc rax
    cmp rax, 10
    jl limpiar_bin_count
```

```
    xor rbx, rbx         ; Reiniciar rbx

bucle_contar:
    cmp rbx, rcx
    jge fin_contar_notas

    mov rdi, [nombres + rbx*8]
    call obtener_nota       ; La nota queda en rax
    mov r8, rax             ; r8 = nota actual

    cmp r8, 10            ; Si nota < 10
    jl forzar_primer_bin

    cmp r8, 100
    jge forzar_ultimo     ; Si nota >= 100

    mov rax, r8
    xor rdx, rdx
    mov r9, 10
    div r9                ; rax = nota / 10
    mov r8, rax          ; r8 es el índice del bin
    jmp incrementar_bin

forzar_ultimo:
    mov r8, 9
    jmp incrementar_bin

forzar_primer_bin:
    mov r8, 0

incrementar_bin:
    ; Incrementa frec_count[r8]
    mov rax, [frec_count + r8*8]
    add rax, 1
    mov [frec_count + r8*8], rax
    inc rbx
    jmp bucle_contar

fin_contar_notas:
    ret
```

Diagrama de flujo del funcionamiento

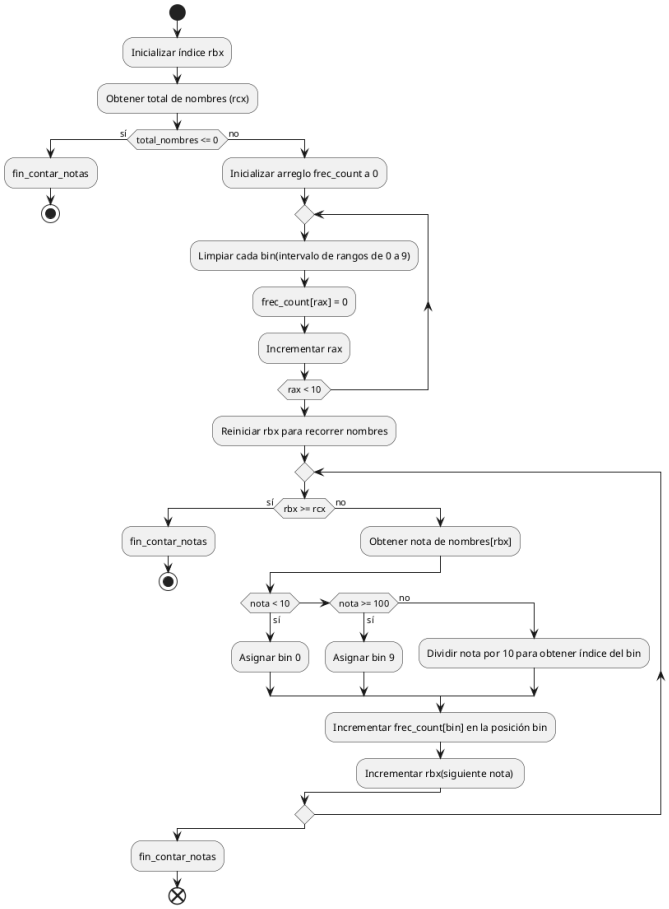

Fig. 6. Diagrama de flujo para contar las frecuencias.

F. Impresión de Resultados

El programa imprime:

- El contenido del archivo de configuración.
- El histograma de frecuencias, mostrando la cantidad de estudiantes en cada bin.
- Los nombres ordenados alfabéticamente y por nota, con un salto de línea entre cada cadena.

Para la impresión se utilizan rutinas que convierten enteros a cadena ASCII (para el histograma) y que cuentan la longitud de cada cadena antes de enviarla a la salida mediante el syscall `write`.

Diagrama de flujo del funcionamiento

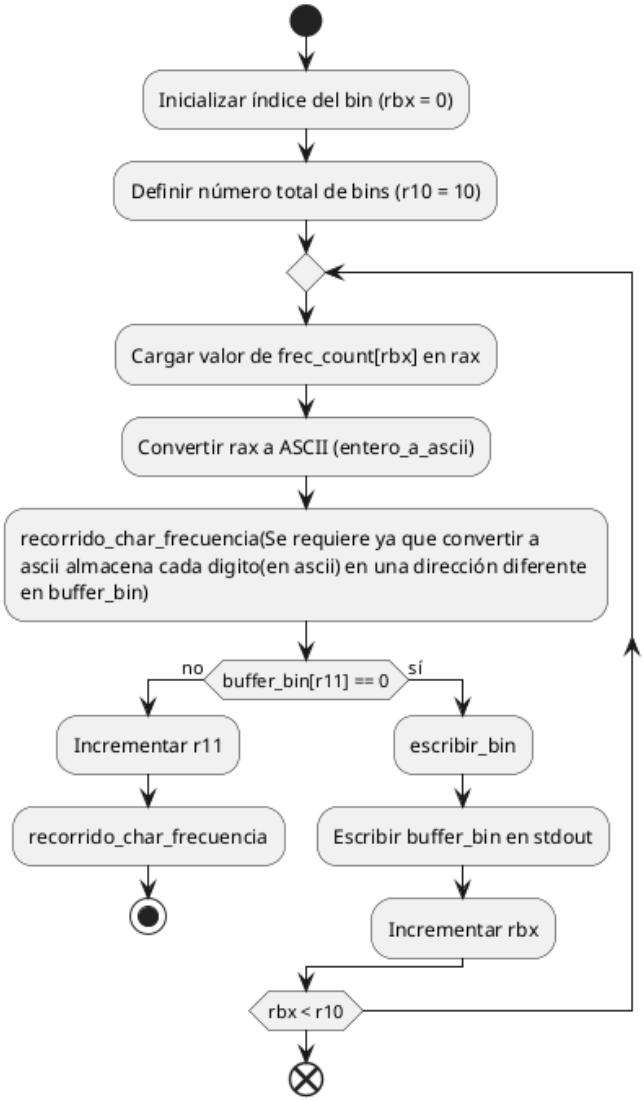

Fig. 7. Diagrama de flujo de la impresión de frecuencias.

III. ¿Cuáles fueron los principales retos a resolver y cómo se resolvieron?

- Lectura de archivos en ensamblador y su almacenamiento: Inicialmente, hubo un desafío con la lectura del archivo de texto y la manera de manejar la cada inicial y las notas. Se resolvió con el uso de un buffer que almacenó todos los caracteres del archivo, y luego se procesaron de manera individual los nombres y las notas, de forma que lo que importaba era conservar las direcciones de cada inicial para recorrer todo el nombre completo. Por lo tanto la cadena de caracteres se almacenó en memoria con terminación nula, lo que permitió procesarlos de forma más efectiva según sus direcciones de memoria.
- Ordenamiento alfabético con Bubble Sort: El desafío fue elegir el algoritmo adecuado. Aunque se descartó el uso de Merge Sort debido a su complejidad para implementar, se optó por Bubble Sort, aunque menos eficiente, ya que resultaba más sencillo de implementar su algoritmo en x86 64. Se implementó un algoritmo para comparar las iniciales de los nombres según sus direcciones y para realizar los intercambios necesarios en la memoria.
- Conversión de notas de ASCII a decimal: Otro reto fue la conversión de las notas de texto (ASCII) a valores numéricos, debido al manejo de registros. Se creó una rutina que iteraba sobre cada carácter, lo convertía a su valor numérico y lo acumulaba para obtener la nota completa.
- Frecuencia de intervalos (Bins) de notas: El problema de calcular las frecuencias de las notas en intervalos de 10 unidades fue resuelto mediante la creación de un contador de intervalos. Esto se hizo usando operaciones aritméticas y la división de

las notas para determinar en qué bin encajaban(nota/10) indicandolo su cociente. También se implementó un mecanismo para forzar las notas fuera de rango (menores de 10 o mayores de 100) a los bins 0 y 9, respectivamente.

IV. ¿Cuáles mejoras se sugieren para el programa y cómo se harían?

- Optimización en el cálculo de la frecuencia de intervalos (bins): La lógica actual para calcular los bins es funcional, pero puede ser optimizada en términos de grupos de 5 estudiantes. Por ejemplo, se podrían utilizar estructuras más eficientes para contar las frecuencias obtener su cociente de la división por 5, en dónde ése cociente me indicaría a cuál posición de estudiantes pertenece.
- Mejoras en la visualización de los resultados: Actualmente, las frecuencias de los bins se imprimen correctamente, pero la presentación es rígida y carece de algunos detalles importantes, como las notas de aprobación, reposición, el tamaño de los grupos de notas y la escala del gráfico. Para lograrlo, sería útil incorporar una forma más dinámica que considere estos factores, como un formato que permita interpretar más fácilmente la distribución de las notas. Además, de mejorar la escala y la organización de la salida.

V. Referencias

1) Evingtone, ”Assembly BUBBLE SORT using Nasm,” GitHub, 2021. https://gist.github.com/Evin-Ngoa/3431161e46b2125cbecac5eb10