Explicación detallada del Juego de la Vida en NASM

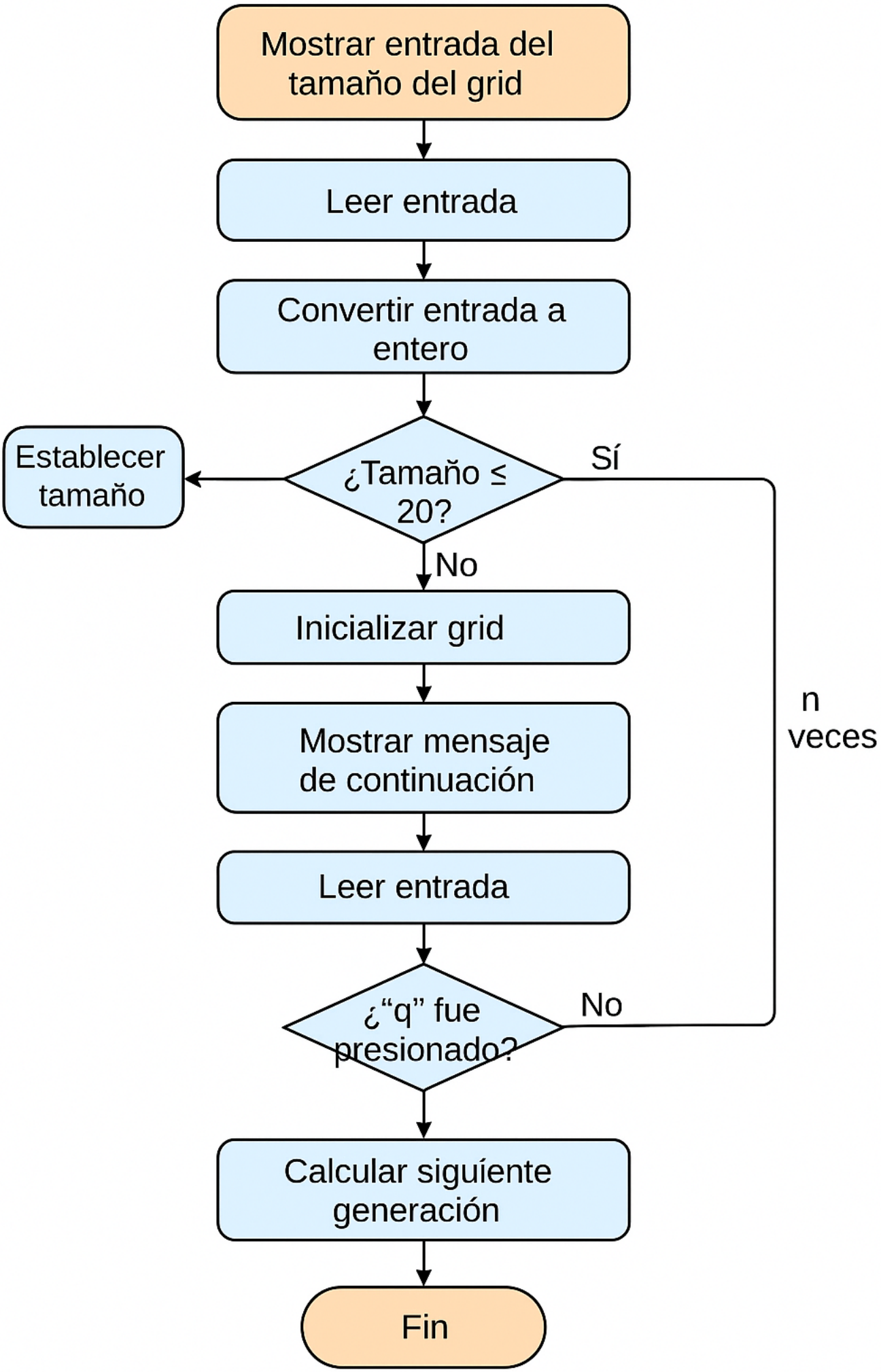
Este programa en ensamblador NASM implementa una versión sencilla del famoso Juego de la Vida de Conway. Básicamente, el usuario ingresa el tamaño del tablero (entre 5 y 20), y el programa simula cómo las células van naciendo o muriendo en función de sus vecinos. Al inicio, se muestra un mensaje pidiéndome el tamaño del grid, el cual se lee desde la entrada estándar. Luego, esa entrada en texto se convierte a número con la subrutina str2int. Si el tamaño ingresado es mayor a 20, automáticamente lo ajusta al máximo permitido. Una vez hecho esto, se inicializa todo el grid con ceros, es decir, todas las células están muertas. Después de eso, el programa llama a la función init_pattern que se encarga de colocar un patrón inicial dentro del grid. Si el tamaño es de al menos 5, se coloca un patrón tipo glider, que es una figura que se va desplazando con el tiempo. Si el tamaño es más pequeño, solo se activa una célula para que al menos se vea algún cambio en generaciones futuras.

El ciclo principal del programa se basa en mostrar el estado actual del grid en consola, luego esperar a que yo presione Enter para continuar o escriba 'q' para salir. En cada iteración, la función show_grid recorre todo el tablero y muestra una representación del mismo, usando '1' para indicar que una célula está viva y '0' si está muerta. Si decido seguir, se calcula la siguiente generación usando la subrutina next_generation. Esta parte es fundamental, ya que hace una copia del grid original y trabaja con ella para evitar modificar valores mientras se procesan. Para cada célula, se llama a la función count_neighbors, que cuenta cuántos de sus ocho vecinos están vivos (siempre y cuando estén dentro de los límites del tablero). Dependiendo del número de vecinos vivos y del estado actual de la célula, se decide si vive, muere o nace en la siguiente generación, siguiendo las reglas originales de Conway: una célula viva con menos de dos vecinos muere por soledad, con más de tres muere por sobrepoblación, y una célula muerta nace si tiene exactamente tres vecinos vivos. El bucle sigue corriendo hasta que presione 'q', momento en el cual el programa termina con

una llamada a la syscall sys_exit.

Además del código, también creé un diagrama de flujo que representa todo este proceso. Comienza con el inicio del programa, seguido de la entrada del tamaño del grid y su conversión a número. Luego viene la inicialización del grid y del patrón inicial. Después de eso, el flujo principal está claramente representado en un ciclo: se muestra el tablero, se espera mi decisión (seguir o salir), se calcula la siguiente generación y se repite todo. El diagrama también incluye las condiciones (por ejemplo, si el tamaño es menor a 5, no se coloca el glider), y muestra de forma clara las llamadas a subrutinas como show_grid, count_neighbors y next_generation. En resumen, el diagrama me ayudó a visualizar mejor cómo fluye el programa paso a paso, y cómo cada parte se conecta entre sí. Me pareció una excelente manera de complementar el análisis del código ensamblador y entender cómo se estructura un juego de simulación basado en autómatas celulares.

Diagrama de flujo del programa:


Mostrar entrada del tamaño del grid
Leer entrada
Convertir entrada a entero
¿Tamaño ≤ 20?
Establecer tamaño
Sí
No
Inicializar grid
Mostrar mensaje de continuación
Leer entrada
¿“q” fue presionado?
No
n veces
Calcular siguíente generación
Fin