Editor de texto: Funcionalidades extras

Nicolás Arato, Martín Córdoba, Darién Ramírez y Jorge Freyre

—Este informe corresponde a la segunda parte de la entrega del editor de texto desarrollado en lenguaje ensamblador que tuvo lugar durante el cursado del periodo 2013.

Habiendo cumplido con los requisitos de la primer parte, necesaria para la regularización, en esta ocasión detallaremos la forma en que están implementadas las nuevas exigencias, dentro de las cuales reconocemos: Procedimientos para Copiar/Pegar texto, Guardar/Abrir archivos, implementación del Scroll y por último, la función de Impresión.

Al igual que en la primer parte, explicaremos detalladamente la construcción de cada una de las rutinas principales, como así también otras auxiliares que nos fueron necesarias para desagregar en lo que refiere en complejidad, cada una de las tareas.

—editor de texto, assembler, extensión, impresora, scroll, archivos.

I. Introducción

El principal objetivo de este trabajo es continuar con el aprendizaje de nuevas interrupciones para poder lograr un mayor entendimiento sobre la forma en que el sistema se comunica y controla los diferentes componentes de hardware.

Las nuevas interrupciones que se utilizan en esta versión extendida en capacidades del básico editor presentado en la primer parte, corresponden al manejo de archivos en disco durante la lectura y escritura de datos, y a la salida de caracteres por impresora a través de una comunicación mediante el puerto paralelo.

A continuación comenzaremos a listar cada una de las nuevas funcionalidades, explicando lo más detalladamente posible la lógica que siguen nuestros algoritmos, los problemas encontrados durante el desarrollo, y las soluciones que llevamos a cabo para lograr el resultado deseado.

A parte de satisfacer los requisitos primordiales propuestos por los docentes, añadimos algunas características que facilitan el uso y aportan una experiencia positiva al momento de usar el editor. Entre ellas se encuentran: Ventanas informativas sobre errores o acciones completadas, confirmar descartar cambios realizados en un documento, la opción para cortar texto y la supresión de texto seleccionado, y por último, la implementación de los atajos que habitualmente utilizamos en cualquier programa de la actualidad.

II. Nuevas Funcionalidades de la versión

A. Copiar/Pegar/Cortar

Previo al desarrollo de estas funciones, fue necesario posibilitar al usuario la acción de *seleccionar texto* (Línea 246). Básicamente lo que se tuvo en cuenta, fue la limitación del área de selección dentro de lo que es la zona editable de la pantalla. También la posibilidad de seleccionar texto con un movimiento ascendente o descendiente del mouse. Y la eliminación de la selección (reestablecer a aspecto normal) tras la pulsación de un botón del mouse.

Esta función reemplaza a la encargada de *cambiar la posición del cursor* de la versión anterior del editor. Ya que cuando detecta que se realizó un clic izquierdo, pero no se arrastró el puntero, solo se limita a mover el cursor a la zona apuntada, sin realizar ninguna selección.

Para resaltar el texto seleccionado, lo que se hace es cambiar el formato del fondo desde la posición con menor posición hacia la de mayor posición.

Esta rutina se llama continuamente en cada iteración del bucle principal del programa. Comienza por leer el estado del mouse y posición. Si el botón izquierdo se encontraba pulsado fuera del área del texto editable, termina la rutina y sigue en curso el resto del programa. En otro caso, guarda las posiciones en memoria para luego entrar en un bucle, que se corta cuando se suelta el botón presionado (b. izq.). Si se soltó el botón fuera del área de texto, la rutina termina sin más. De lo contrario se guardan las posiciones finales.

Para todas las capturas del estado del mouse se utiliza el servicio 3h de la interrupción 33h.

Como ya tiene las coordenadas iniciales y finales del movimiento que se hizo con el botón izquierdo del mouse pulsado, las compara. Si son iguales, mueve el cursor a la posición común, limpia toda selección existente y termina la rutina. Si fueran distintas, toma la que se encuentra más próxima al borde superior de la ventana, y comienza a cambiar el atributo de los caracteres comprendidos entre las dos posiciones incluido el inicio y el fin (Esto lo hace sin modificar los bordes de la ventana).

Un aspecto particular de esta rutina, es que si como resultado de su ejecución se selecciona texto, mueve el cursor a la primera posición de lo seleccionado. Esto es necesario ya que las acciones cortar y suprimir texto seleccionado se valen de la función *Delete*.

Una función auxiliar de la antes mencionada es *Limpiar Selección*, que recorre cada posición dentro del área de texto editable, y pone sus atributos al usado para escribir comúnmente.

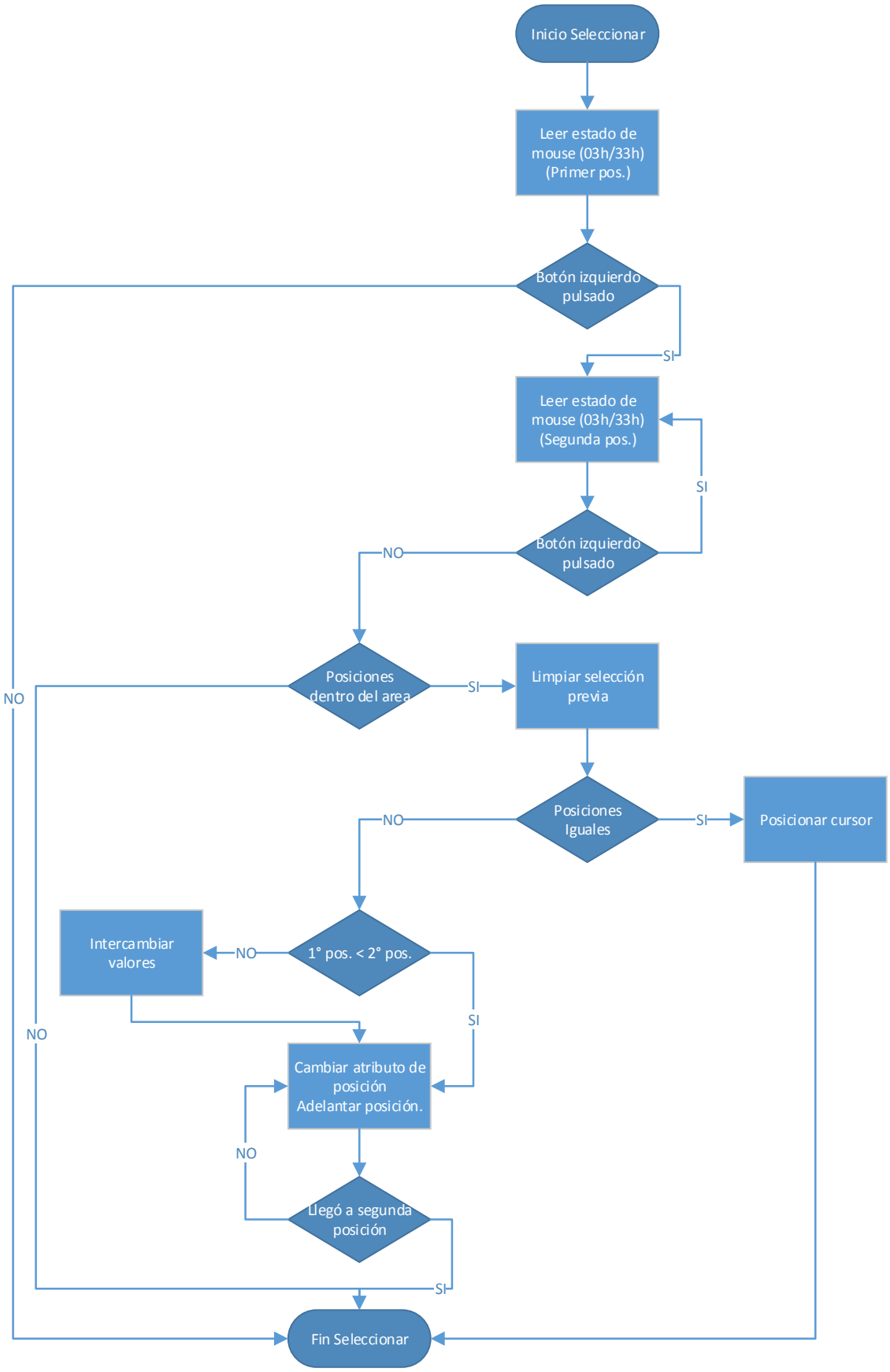

Fig. 1: Diagrama de flujo de “Seleccionar”.

La copia (guardado) y posterior recuperación (pegado) del texto, se realiza utilizando una porción de memoria reservada llamada *PortaPapeles*, de unos 1561 bytes. Tiene una capacidad de almacenamiento de 20 renglones (la ventana de edición completa).

La rutina encargada de *Copiar* (Línea 566), al igual que cuando se limpia la selección, recorre toda el área de texto, pero leyendo el carácter y el atributo. Si encuentra un atributo correspondiente al resaltado, sale del bucle que recorre toda el área para pasar a uno más pequeño que va copiando todo carácter que lee al *PortaPapeles*, y corta solo cuando vuelve a leer un atributo normal nuevamente (no considera atributos de los caracteres que forman los bordes de ventana, los saltea). En ese momento sale de la rutina después de limpiar la selección.

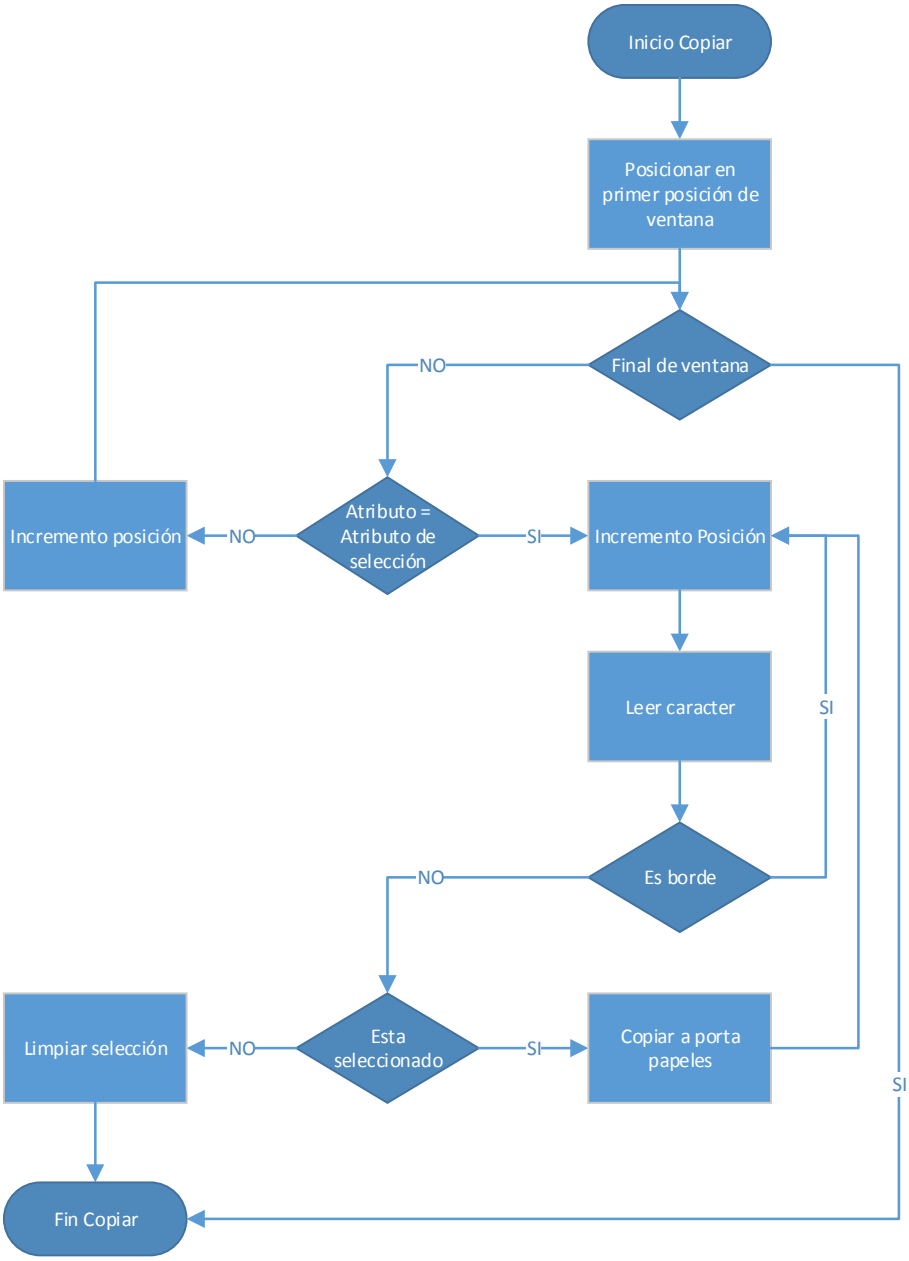

Fig. 2: Diagrama de flujo de “Copiar”.

Al llamar a la rutina *Pegar* (Línea 639), se recorre desde el inicio el segmento de memoria llamado *PortaPapeles*, hasta que se lee un carácter NULL (no hay más nada que recuperar). A medida que se recupera un carácter, se reescribe con NULL, de tal manera que al terminar, ese segmento de memoria queda sin asignar nuevamente (evitando que en futuras “pegadas” no se introduzcan caracteres que no correspondan). Esta rutina reutiliza parte del código de las funciones *Escribir* y *Siguiente Posición*, que fueron explicadas en detalle en la primer entrega.

Por último, la utilidad de la opción de *Cortar* (Línea 543), realiza el procedimiento *Copiar*, y luego entra en un bucle que llama a la rutina *Delete* tantas veces como caracteres hay seleccionados. Se supone que este comando es usado justo después de haber seleccionado texto, ya que de haber cambiado de lugar el cursor, esta rutina borraría caracteres que no corresponden.

B. Guardar/Abrir/Nuevo

Visto que para realizar las operaciones de guardado y de posterior apertura se necesitaba tener los datos guardados en la memoria, se realizaron unas cuantas funciones auxiliares para poder atacar mejor al problema. Algunas son específicas de cada función, otras son compartidas.

Para el uno de la función *Guardar* (Línea 2689) se utilizó:

1. *Texto a Buffer* (Línea 2108): copia todo el texto que corresponde al archivo a un sector de memoria de 12000 bytes llamado Buffer Archivo (12000 caracteres. Con lo que se pueden guardar hasta siete pantallas completas y poco más). Los pasos que realiza para llegar a su fin se pueden dividir en tres:
   Primero recorre toda posición de la "pila" armada en memoria donde se guardan los renglones superiores no visibles, administrada por las funciones de Scroll, de las que hablaremos más adelante. Copia cada carácter de esta pila hasta llegar a leer un NULL (llego al final del contenido de la pila). Cuando esto ocurre salta al segundo paso, que consiste en copiar todos los caracteres visibles, lo que realiza mediante un recorrido de la memoria de video (no copia bordes). Y por último paso, realiza la misma tarea que en la pila de scroll correspondiente a la parte superior de la pantalla, pero en la que guarda líneas inferiores.
   Este último paso tiene que realizarse en sentido contrario comparado con la el recorrido de la pila superior. Lo que se debe al orden en que se guardan las líneas: En ambas una nueva línea se añade al final, por lo que la pila superior siempre presenta el texto en el orden original. En cambio en la pila inferior, las líneas están en orden inverso.
2. *Vaciar Buffer Archivo* (Línea 1913): Esta rutina es utilizada por la anterior. Con su aplicación obtenemos que el *BufferArchivo* se llene de caracteres nulos. Para ello, se asigna posición por posición dicho carácter.
3. *ErroresEnGuardar* (Línea 2580): Esta rutina se encarga de manejar los errores durante la creación/sobrescritura de archivos. Lee diversas banderas en memoria para ver el resultado de la creación del archivo, y en caso de encontrar errores, muestra el mensaje correspondiente en una ventana. En caso de no encontrar errores solo muestra el mensaje "Guardado". Luego de mostrar cualquier mensaje espera que el usuario presione enter y así continuar con la ejecución del Editor.
   Los errores que contemplamos en esta operación son: RUTA INVALIDA, para cuando no existe la ruta especificada; y ACCESO DENEGADO, cuando el sistema no puede crear el archivo por algún motivo.

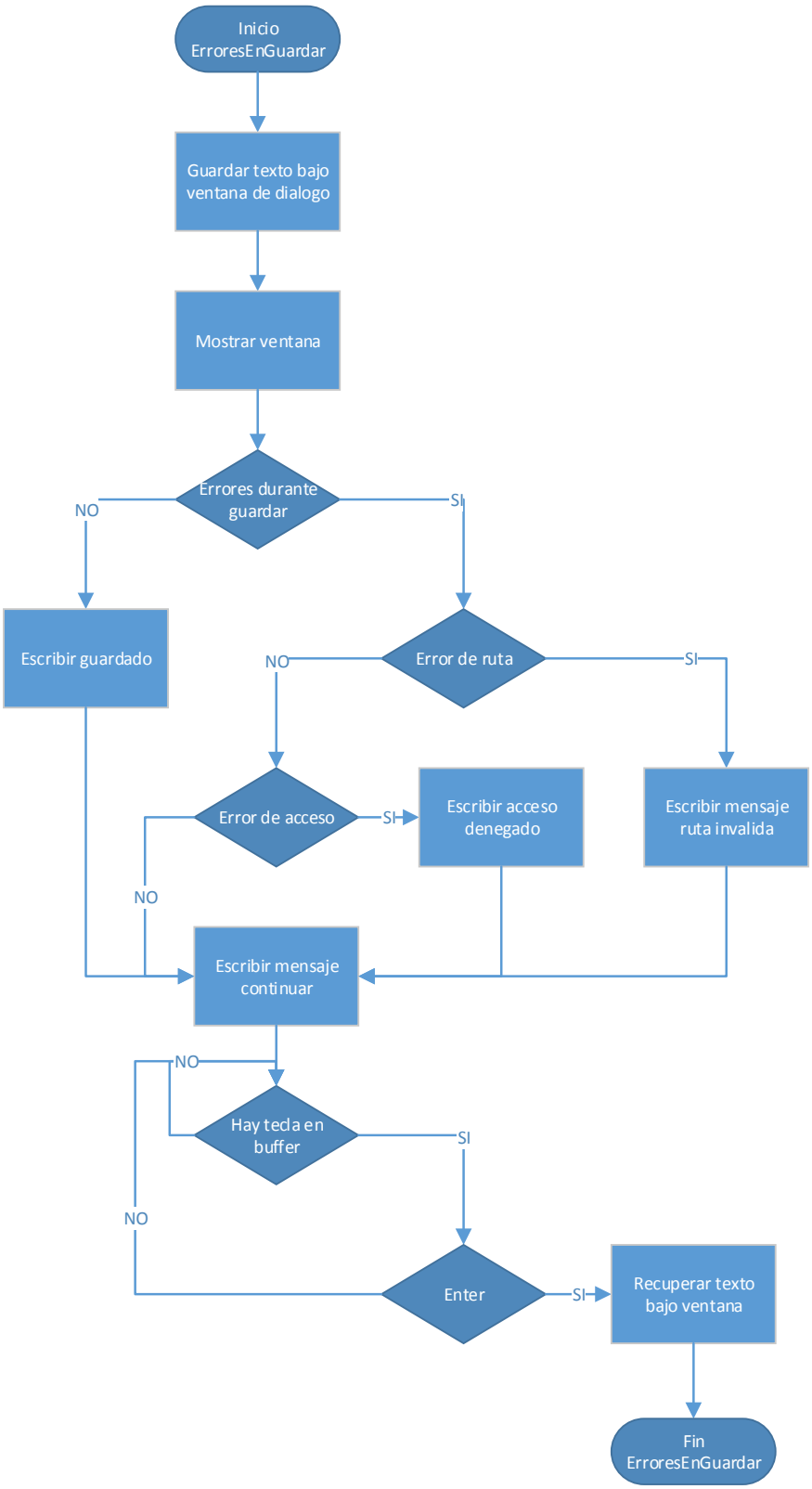

Fig. 3: Diagrama de flujo de “ErroresEnGuardar”.

Para el uso de la función *Abrir*:

1. *Buffer a Texto* (Línea 2267): Copia todo contenido del sector de memoria *BufferArchivo* a pantalla. Para esto recorre cada byte de este sector hasta llegar a leer un NULL, con lo que termina el proceso. Para introducirlo dentro de la ventana del editor, se posiciona el cursor en la primer posición, y mediante el uso de parte de código de las rutinas *Escribir* y *Siguiente Posición* (presentadas en la primer entrega), se escribe uno por uno utilizando el servicio 0Eh de la interrupción 10h. Del avance del Scroll se encarga *Siguiente Posición.*
2. *Vaciar Texto* (Línea 1841): Se ejecuta antes de traer un nuevo archivo desde el disco, o al clickear sobre el botón “nuevo” del menú Archivo. Como resultado, despejamos todo rastro del documento que estaba siendo editado.

Realiza el mismo recorrido de *Texto a Buffer*, solo que en vez de copiar, reemplaza cada carácter con un NULL (carácter 00h).

3. *Leer Nombre de Archivo* (Línea 1942): Lee el nombre de archivo, el cual es ingresado mediante el teclado por el usuario. El espacio destinado para insertar el nombre acepta el uso de teclas como backspace, delete, flechas, enter para seguir con la instrucción y escape para cancelarla. Se utiliza tanto en la función *Guardar* como en *Abrir*. Con su llamada aparece un mensaje en la segunda barra del menú solicitando la cadena identificadora del archivo. Si no se especifica una ruta, el archivo se busca/guarda en el mismo directorio donde este el programa.
   Para realizar su cometido, se borra el submenú, y se imprime en pantalla un mensaje que consta de tres partes: El texto “Nombre archivo:” tras el cual el usuario tiene un campo en donde debe ingresar lo solicitado. Y por último un “botón” con el texto “Aceptar”.

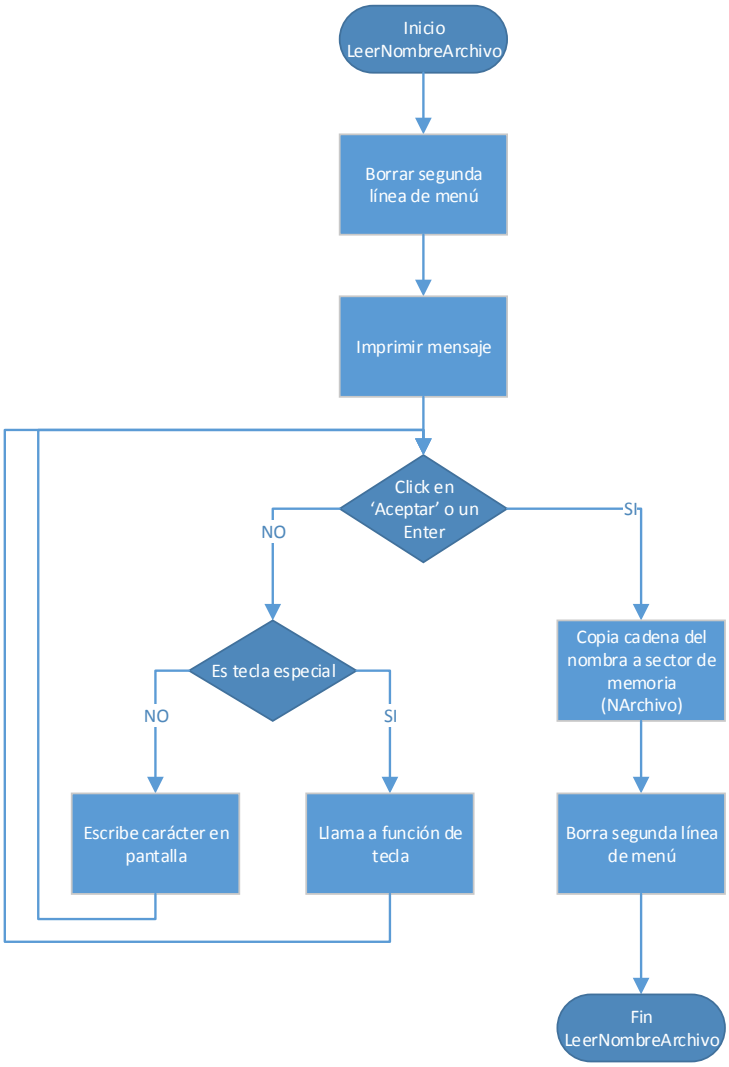

Fig. 4: Diagrama de flujo de “LeerNombreArchivo”.

   Para el ingreso del nombre, se utilizan códigos devenidos de las rutinas “Escribir”.
   Luego de presionarse el botón “Aceptar” (O haber presionado enter), se copia el contenido del campo a un sector de memoria reservada (“NArchivo”).
   Los nombres de los archivos tienen que tener un largo máximo de 25 caracteres (capacidad del campo habilitado para la escritura del mismo).
4. *ErroresEnAbrir* (Línea 2389): Muy similar al procedimiento del mismo fin en la rutina *Guardar*, solo que si no hay errores no realiza nada. Los casos que contempla son los mismos que los mencionados en la otra función, con un agregado: ARCHIVO NO ENCONTRADO, cuando el nombre especificado no coincide con los archivos existentes.

Ahora pasaremos a explicar las rutinas principales:

1. *Abrir* (Línea 2486)*:* Como primer paso, se llama al proceso *Leer Nombre de Archivo*, la dirección de la cadena obtenida se copia a DX. Luego, haciendo uso del servicio 3Dh de la interrupción 21h se abre un archivo en modo lectura. Este llamado retorna en AX un handle, es decir, un identificativo del archivo que está siendo utilizado por el

programa, que se usa para pasarlo a otros llamados a interrupciones y poder indicarle sobre que archivo se quiere que trabajen.
Si ocurrió un error en la apertura, el CF (CarryFlag) se pone a 1, lo que indica que en AX no está el handle, sino el código identificativo del error, ante el cual se informa debidamente al usuario.
Paso seguido, se utiliza el servicio 3Fh (int. 21h) para leer desde el archivo. Como parámetros del servicio se le debe pasar: en BX el handle, en CX la cantidad de Bytes a leer y en DX la dirección de memoria donde se guardara lo leído.
Hecho esto, se llama a *Vaciar Texto* y a *Buffer a Texto* con lo que ya queda el archivo totalmente visible en pantalla.
Para terminar, se cierra el archivo con el servicio 3Eh (int. 21h).

2. *Guardar* (Línea 2689)*:* Este procedimiento es muy similar al anterior. Primero hace la lectura del nombre del archivo, y lo crea (si es que no existe) o lo abre vaciando todo su contenido. Esta tarea la realiza el servicio 3Ch (int. 21h), que recibe en CX el tipo de archivo (00h, normal, es el que usamos) y en DX la dirección del nombre del archivo.
   Una vez abierto para escritura, se llama a la rutina *Texto a Buffer, y* con el servicio 40h (int. 21) se guarda todo el sector de memoria (DX = Buffer), correspondiente al archivo visible, en el archivo identificado por el handle pasado en BX (si no hubo errores, todo funciona al igual que en la función anterior).
   Se cierra el archivo, y se retorna al curso natural del programa.

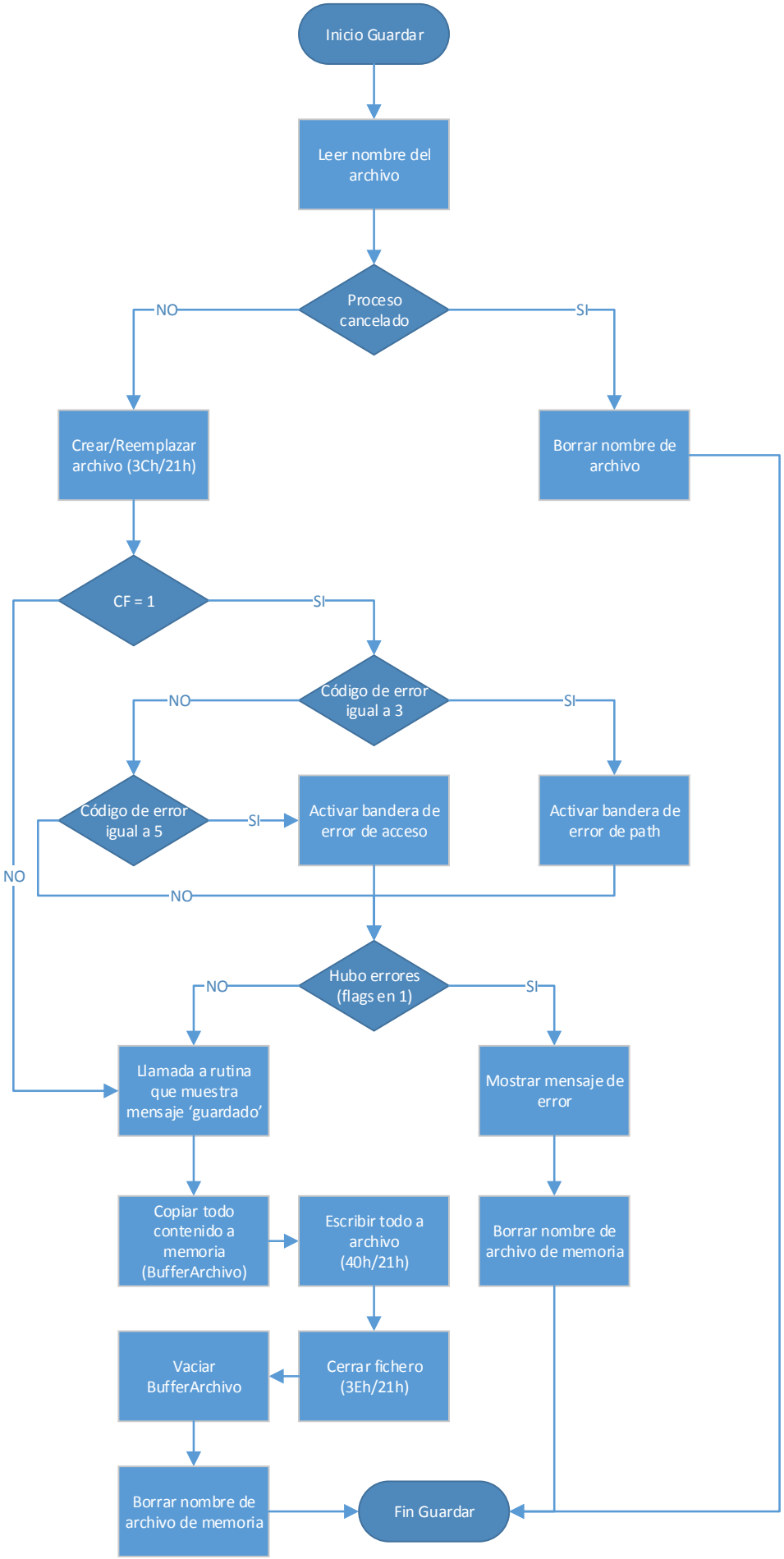

Fig. 5: Diagrama de flujo de "Guardar".

3. *Nuevo* (Línea 1760): Lo que solo hace es un llamado a vaciar texto, y luego pone el cursor al inicio de la ventana. Antes de realizar esto muestra un cuadro de dialogo en el que advierte que se perderán todos los cambios sin guardar. Para continuar tienes que especificarlo con una ‘s’ o ‘S’, en caso de querer volver hacia atrás, con una ‘n’ o ‘N’.

Otro cuadro de dialogo que responde a la pulsación S/N por parte del usuario es el chequeo que se realiza al pulsar Salir, en el menú Archivo. Si se detectan cambios (bandera que se pone a 1 al ejecutarse las funciones *Escribir* o *Pegar*), se crea una ventana que pregunta si quieres guardarlos. Si se contesta afirmativamente se llama a la función *Guardar*. En otro caso sale sin más.

C. Scroll

Hay dos rutinas que se encargan de desplazar el texto hacia arriba o hacia abajo maximizando el largo posible de los archivos que son creados mediante nuestro editor a un total de 152 líneas (ya que usamos espacios de 11856 bytes/caracteres, y hay 78 caracteres por línea).

Solo explicaremos una de las dos funciones, ya que realizan prácticamente los mismos pasos.

La función *Scroll Arriba* (Línea 817) mueve todo el contenido de la ventana, una línea hacia arriba. La línea superior que deja de ser visible se copia a la memoria reservada de nombre *PilaArriba* (11856 bytes, se eligió esta cantidad ya que es el espacio reservado para el buffer que utilizamos para guardar archivos. No tendría sentido hacer ese buffer de tantas líneas de capacidad, y poder crear y editar mucho menos de la mitad).

Consta de tres pasos. Primero se guarda la línea superior en memoria, luego se hace el scroll con el uso del servicio 06h (int. 10h), y por último se recupera (si es que hubiere alguna) una línea del sector de memoria *PilaAbajo*. Todo esto se hace recorriendo cada uno de los sectores como describimos ya en otras funciones.

Lo nuevo de esta función, que la diferencia del resto es la administración de las pilas mencionadas. Para calcular la posición (desplazamiento desde el inicio) del comienzo de la última línea guardada (es a la única que se debería poder acceder), se va actualizando el valor de la cantidad de líneas que contiene la pila. A ese valor se lo multiplica por 78 (cantidad de bytes que se guardan por línea), con lo que obtenemos el desplazamiento del primer carácter de la última línea almacenada (o en otro caso, la posición donde debería almacenarse).

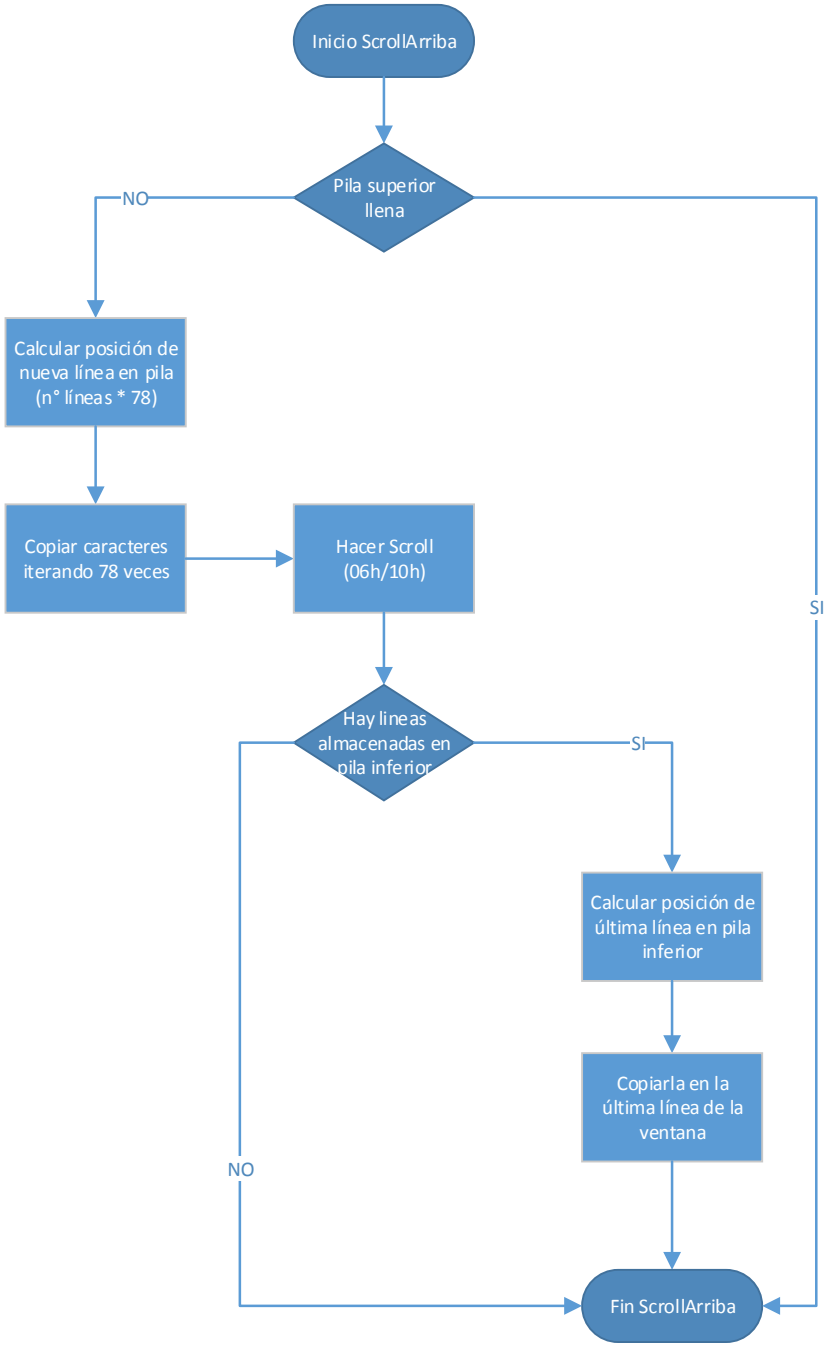

Fig. 6: Diagrama de flujo de “ScrollArriba”.

D.Imprimir

Nada más al inicio, se inicializa la impresora. Para esto, se verifica en cada puerto el bit de estado devuelto por la interrupción (AH: 01h/ INT 17h). Si ningún bit de estado devuelve que está conectada (distinto a 00110000b), se muestra una ventana refiriendo a la situación y termina la llamada.

Acto seguido, se lee el estado de la impresora para inspeccionar si hay errores, esto se realiza a través de una llamada a *LeerEstadoImpresora* (Línea 4971), que funciona de forma similar a las que manejan errores durante el manejo de las funciones de archivos. Los errores que se contemplan en esta instancia son: disponibilidad de papel, time-out y error de io.

Si se encuentra algún error se termina el llamado. De lo contrario se continúa.

Para mandar caracteres se utiliza el servicio 00h (imprimir carácter) de la interrupción 17h. Antes de comenzar a mandar realiza un llamado a *Texto a Buffer*, función utilizada cuando trabajamos con archivos, pero que es útil en este caso ya que facilita el recorrido de todo el texto: en vez de ir a buscarlo en las pilas de scroll y en pantalla, lo buscamos todo de una sola fuente: un sector de memoria.

Una vez empezado el recorrido del sector donde se encuentran los caracteres, se controla mediante un contador, el número de símbolos escritos en una línea. Cuando este llega a 78, se hace un avance de línea del cabezal de impresión imprimiendo los caracteres de control: nueva línea y retorno de carro.

Cuando lee de la memoria un carácter NULL, termina de imprimir.

Antes de mandar a imprimir cada carácter verifica nuevamente el estado de impresión por si se queda sin papel. Y luego de que cada carácter sea enviado, se verifica el bit que acusa recepción (01000000b), si no está activo, se notifica el error por pantalla y se termina la llamada a esta función.

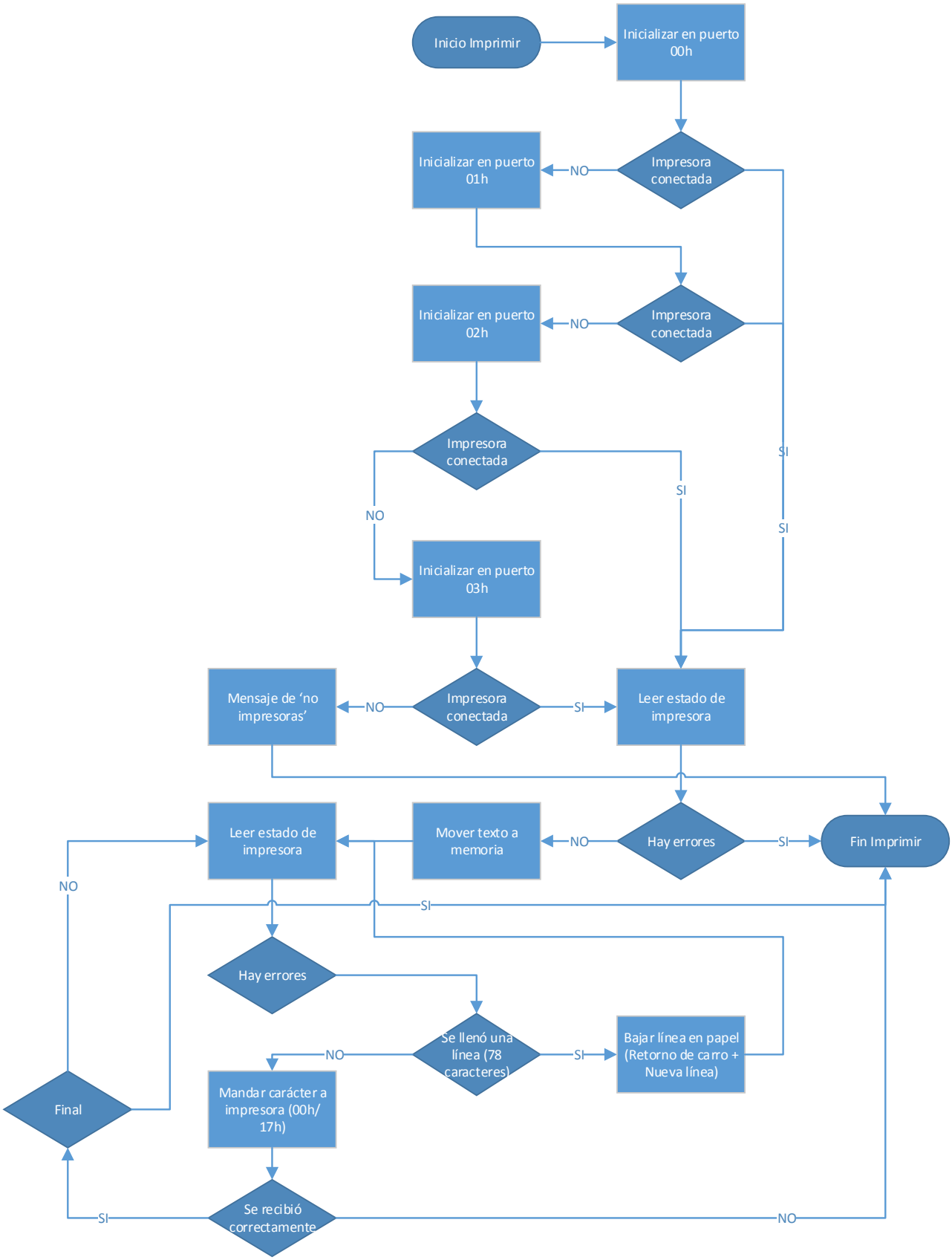

Fig. 7: Diagrama de flujo de "Imprimir".

III. CONCLUSIONES

Durante el desarrollo del editor, pero sobre todo en estas últimas funcionalidades, nos encontramos con algunos problemas. Estos fueron sin duda por la poca experiencia que teníamos en el desarrollo utilizando lenguajes de bajo nivel, como lo es el ensamblador.

La principal dificultad que nos encontramos fue la poca cantidad de registros de uso general (AX, BX, CX, DX), por lo que en ocasiones donde se realizaban muchos cálculos fue necesario reservar varios bytes en memoria con el fin de no perder ninguna información o hacer un uso intensivo del segmento de pila. Este proceso nos pareció engorroso ya que es menos directo que trabajar con una variable sin más y dejar que el sistema por su cuenta haga la gestión de memoria, como en cualquier otro lenguaje de más alto nivel.

En particular, en el desarrollo de las rutinas que realizaban el scroll, tuvimos problemas en reservar grandes cantidades de memoria para las pilas utilizadas. Como reservábamos grandes espacios en medio de la declaración del programa, se separaban demasiado entre si las declaraciones (etiquetas) que localizaban cada sub-rutina, y al hacer uso de la instrucción CALL, esta no encontraba los nombres que le pasábamos como parámetro, ya que quedaban fuera de la capacidad de salto de esta instrucción. Esto lo solucionamos reservando los grandes espacios de memoria al final.

Un problema importante fue la falta de hardware necesario para la prueba de las funciones de impresión. Mediante nuestra investigación sobre sus interrupciones pudimos entender como es la comunicación, pero al no poder hacer pruebas sobre una impresora real, que devuelva bits de estados, la implementación de sus funciones está realizada sin ningún seguro que pueda reafirmarnos en nuestro desarrollo. Por este motivo nos vimos obligados a dejar partes del código comentadas para mantener solo la funcionalidad básica: Mandar caracteres a imprimir. Esto es una lástima porque ya contemplamos la ocurrencia de errores en las funciones de archivos, y en esta parte, consideramos necesario informar al usuario que está ocurriendo, o que fallas se dieron que explique la interrupción de su trabajo de impresión.

Al terminar este trabajo, y junto con los conocimientos recibidos en tema de circuitos digitales, pudimos entender mejor el funcionamiento del procesador, así como todas las tareas que nos facilitan los softwares en lo que se refiere al manejo de recursos y comunicación con dispositivos de hardware.