



UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INFORMÁTICOS**

CURSO 2021 - 2022

**“PROYECTO DE E/S MEDIANTE
INTERRUPCIONES”**

AUTORES:

Julio Manso Sánchez-Tornero c200320

Nihel Kella Bouziane c200315

IDENTIFICADOR:

mansokella

Índice

Introducción	2
Diagrama de flujo y comentario de los algoritmos utilizados.	3
Diagrama de flujo de SCAN	3
Diagrama de flujo de PRINT	4
Diagrama de flujo de RTI	5
Listado comentado de las subrutinas en ensamblador.	6
INIT	6
SCAN	7
PRINT	9
RTI	11
Histórico del desarrollo de las rutinas	14
Conjuntos de casos prueba	17
Scan	17
Caso 1: Cadena 144 caracteres por línea A	17
Caso 2: Dos bloques de 800 caracteres por línea B	18
Caso 3: Descriptor erróneo	19
Print	20
Caso 1: Cadena de 53 caracteres escrita por la línea A.	20
Caso 2: Cadena de 200 caracteres escritos por la línea B.	20
Caso 3: Descriptor erróneo.	22
Observaciones finales y comentarios personales	23

1. Introducción

El objetivo del proyecto es lograr una familiarización con la realización de operaciones de Entrada/Salida en un periférico mediante interrupciones. El dispositivo elegido es la DUART MC68681 operando ambas líneas mediante interrupciones. En el computador del proyecto la DUART está conectada a la línea de petición de interrupción de nivel 4.

El microprocesador MC68000 fue introducido en 1979 y es el primer microprocesador de la familia M68000 de Motorola. Es un procesador CISC, aunque posee un juego de instrucciones muy ortogonal, tiene un bus de datos de 16 bits y un bus de direcciones de 24 bits. Fue uno de los primeros microprocesadores en introducir un modo de ejecución privilegiado. Así, las instrucciones se ejecutan en uno de los dos modos posibles: modo usuario y modo supervisor.

Sobre el MC68681, tenemos que decir que es un módulo de entrada/salida perteneciente a la familia M68000 de Motorola. Su función es controlar dos líneas series con capacidad de transmisión y recepción asíncrona, a este tipo de dispositivo se le conoce con el nombre de DUART (Dual Universal Asynchronous Receiver/Transmitter).

En cuanto al simulador, se trata del BSVC, una plataforma para la simulación de procesadores, memoria y controladores de periféricos, desarrollada en C++ y Tcl/Tk.

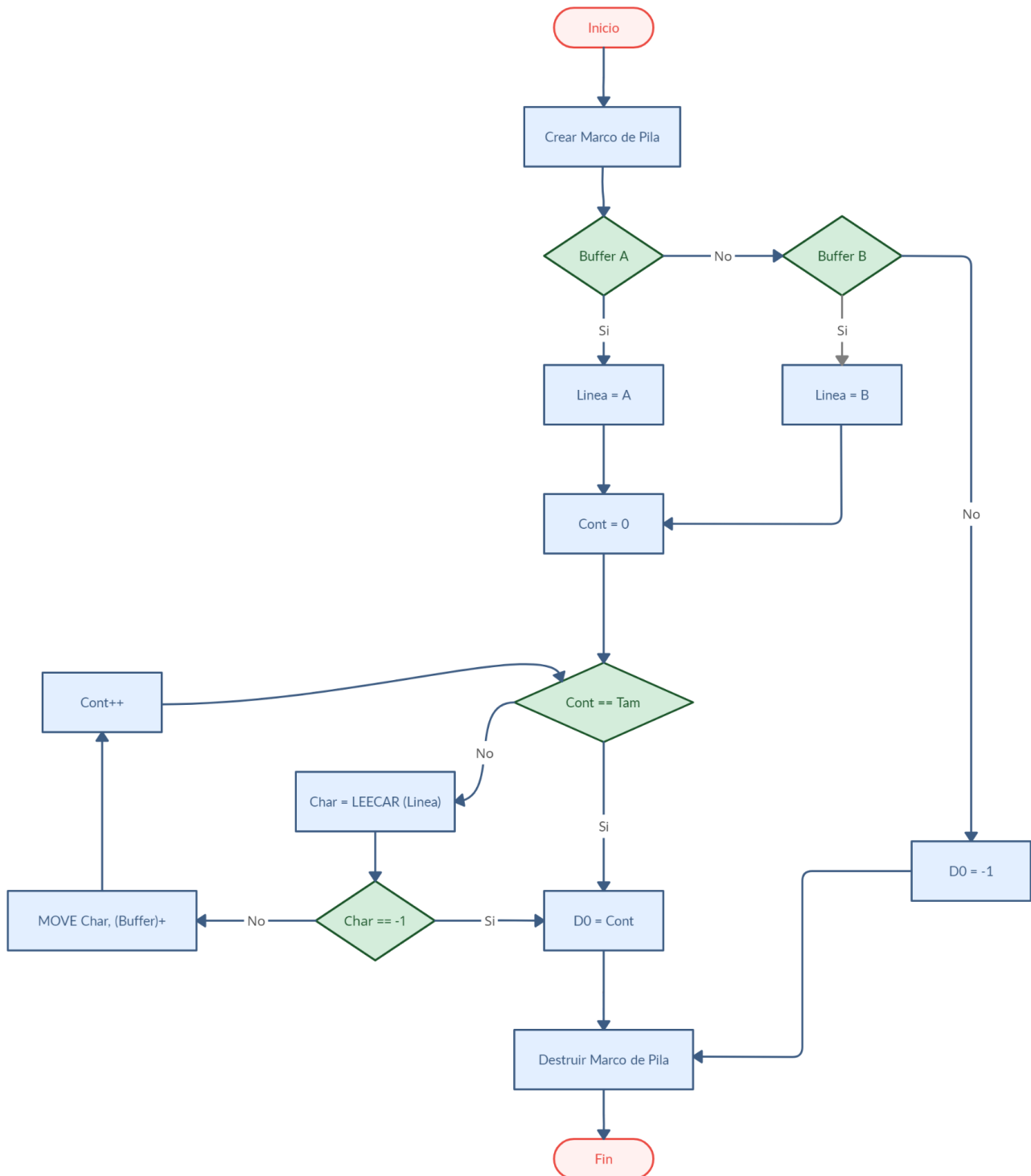
Actualmente, BSVC simula el microprocesador MC68000, el controlador de líneas DUART MC68681 y memoria RAM. Por lo tanto, permite construir computadores virtuales con procesador, memoria y unidades periféricas.

Para la realización del proyecto se necesitan unos buffers internos para almacenar los caracteres que se reciben asíncronamente por las líneas. Del mismo modo, se necesitan sendos buffers internos para almacenar los caracteres pendientes de transmitirse por las líneas. Además, existe una única rutina de tratamiento de las interrupciones que será la encargada de transferir la información. El proyecto implica la programación de la rutina de tratamiento de las interrupciones (RTI) así como de las subrutinas SCAN y PRINT.

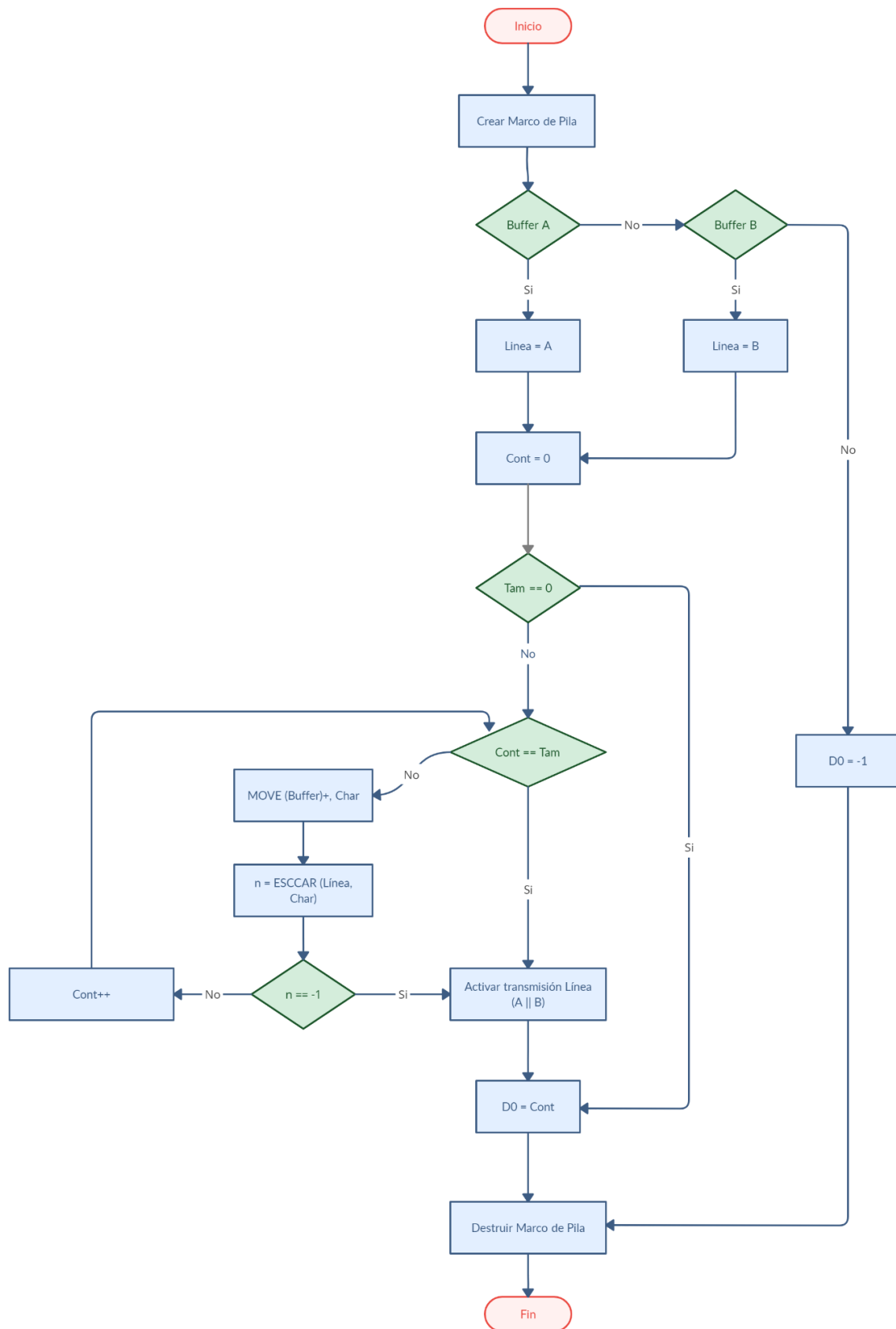
Además, entre los objetivos del proyecto se encuentra aprender a depurar nuestras rutinas para comprobar el funcionamiento correcto del programa mediante la elaboración de un conjunto de casos de prueba. También se llevará a cabo un informe que incluya la bitácora de trabajo, así como lo aprendido y destacable durante el desarrollo del proyecto.

2. Diagrama de flujo y comentario de los algoritmos utilizados.

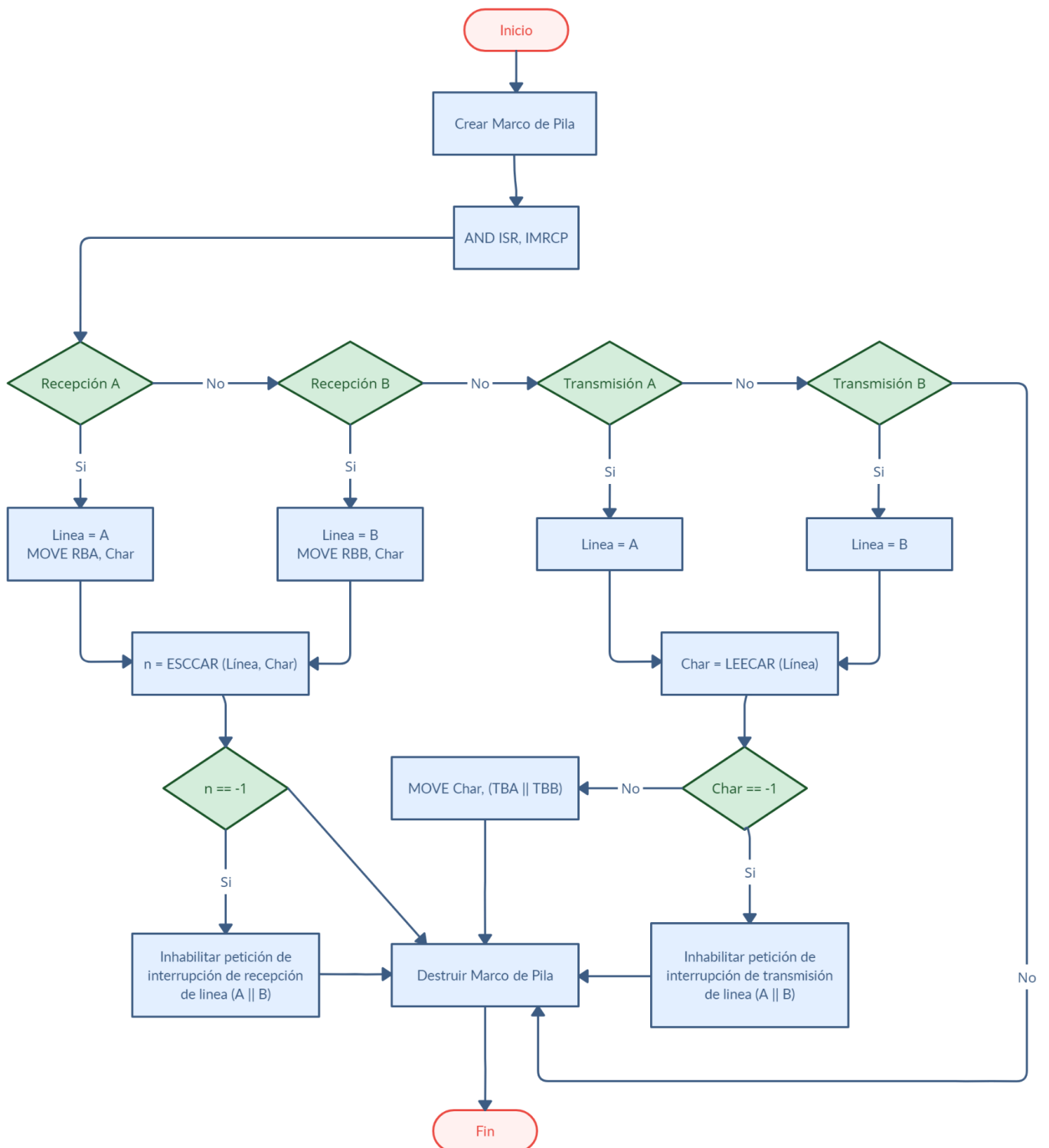
2.1. Diagrama de flujo de SCAN



2.2. Diagrama de flujo de PRINT



2.3. Diagrama de flujo de RTI



3. Listado comentado de las subrutinas en ensamblador.

Antes de pasar con el código respectivo a las subrutinas, aclaramos que previamente creamos la copia de la IMR que posteriormente se utilizará para la lectura de la IMR (ya que esta no se puede leer directamente). Además, movemos el punto de inicio del programa para evitar desalineamiento.

```
IMR_COPIA: DC.B 0      * Copia del IMR para lecturas
ORG $2400
```

3.1. INIT

En la subrutina *INIT* simplemente se definen las equivalencias de la DUART con el procesador, cumpliendo con los parámetros dados.

INIT:

MOVE.B	##00010000,CRA	* Reinicia el puntero MR1
MOVE.B	##00010000,CRB	* Reinicia el puntero MR1
MOVE.B	##00000011,MR1A	* 8 bits por caracter.
MOVE.B	##00000011,MR1B	* 8 bits por caracter.
MOVE.B	##00000000,MR2A	* Eco desactivado.
MOVE.B	##00000000,MR2B	* Eco desactivado.
MOVE.B	##11001100,CSRA	* Velocidad = 38400 bps.
MOVE.B	##11001100,CSRB	* Velocidad = 38400 bps.
MOVE.B	##00000000,ACR	* Velocidad = 38400 bps.
MOVE.B	##00000101,CRA	* Transmision y recepcion activadas A
MOVE.B	##00000101,CRB	* Transmisión y recepción activadas B
MOVE.B	##00100010,IMR	* Registro de máscara de interrupción
MOVE.B	##00100010,IMR_COPIA	* Copia del reg. de máscara de interrupción
MOVE.B	##01000000,IVR	* Vector de interrupción a H'40
MOVE.L	#RTI,\$100	* Act. tabla de rutinas de interrupción
BSR	INI_BUFS	
RTS		

3.2. SCAN

La subrutina *SCAN* se encarga de la transferencia de un bloque de caracteres de un tamaño determinado desde un buffer de recepción (el de la línea A o el de la línea B) a un buffer determinado. Se pasan como parámetros un descriptor que especifica la línea a la que se quiere acceder, el tamaño del bloque a transferir y la dirección del buffer donde se deben volcar los caracteres leídos del respectivo buffer de recepción. Esta subrutina hace uso de *LEECAR* para leer los caracteres de los buffers de recepción para luego poder volcarlos sobre el buffer introducido por dirección.

SCAN:

LINK	A6,#-16	* Reserva de espacio en pila para 4 variables locales
MOVE.L	A0,-4(A6)	* Salvado de los registros que utilizamos en SCAN
MOVE.W	D1,-8(A6)	
MOVE.W	D2,-12(A6)	
MOVE.L	D3,-16(A6)	
MOVE.L	8(A6),A0	* Dirección del Buffer
MOVE.W	12(A6),D1	* Dato de Descriptor
MOVE.W	14(A6),D2	* Dato de Tamaño
EOR.L	D3,D3	* Inicializamos a cero el contador de caracteres
EOR.L	D0,D0	* Inicializamos el resultado a cero
CMP.W	#0,D1	* Si el descriptor es 0 entonces la lectura será por la línea A
BEQ	LEC_A	
CMP.W	#1,D1	* Si el descriptor es 1 entonces la lectura será por la línea B
BEQ	LEC_B	
SUB.L	#1,D0	
BRA	FIN_SERR	* Si no es ni 0 ni 1 entonces se produce un error

LEC_A:

CMP.W	D3,D2	* Se comprueba si se han leído todos los caracteres
BEQ	FIN_SCAN	
MOVE.L	#0,D0	* Buffer A de recepción
BSR	LEECAR	
CMPL	#\$ffffff,D0	* Miramos si hay más caracteres a leer o no
BEQ	FIN_SCAN	
ADD.L	#1,D3	* Sumamos uno al contador de caracteres
MOVE.B	D0,(A0)+	* Metemos el carácter en el buffer
BRA	LEC_A	

LEC_B:

CMP.W	D3,D2	* Se comprueba si se han leído todos los caracteres
BEQ	FIN_SCAN	
MOVE.L	#1,D0	* Buffer B de recepción
BSR	LEECAR	
CMPL	#\$ffffff,D0	* Miramos si hay más caracteres a leer o no
BEQ	FIN_SCAN	
ADD.L	#1,D3	* Sumamos uno al contador
MOVE.B	D0,(A0)+	* Metemos el carácter en el buffer
BRA	LEC_B	

FIN_SCAN:

EOR.L	D0,D0	* Inicializamos a 0 el registro resultado
OR.L	D3,D0	* Cargamos el número total de caracteres leídos en D0

FIN_SERR:

MOVE.L	-4(A6),A0	* Se recuperan los registros guardados en pila
MOVE.L	-8(A6),D1	
MOVE.L	-12(A6),D2	
MOVE.L	-16(A6),D3	
UNLK	A6	* Destrucción del marco de pila
RTS		

3.3. PRINT

La subrutina *PRINT* se encarga de la transferencia de un bloque de caracteres de un tamaño determinado almacenados en un buffer en memoria sobre el buffer de transmisión de la correspondiente línea (A o B). Para ello, esta rutina recibe en la pila los valores de tamaño, la dirección del buffer donde están los caracteres y un descriptor que indica la línea, y por tanto el buffer de transmisión, donde se quieren volcar los caracteres para su posterior transmisión. Esta rutina vuelca los caracteres de memoria al buffer de transmisión mediante sucesivas llamadas a *ESCCAR*.

PRINT:

LINK	A6,#-20	* Reserva de espacio en pila para 5 variables locales
MOVE.L	A5,-4(A6)	* Guardo los registros que utilizo en PRINT
MOVE.W	D1,-8(A6)	
MOVE.W	D2,-12(A6)	
MOVE.L	D3,-16(A6)	
MOVE.L	D4,-20(A6)	
MOVE.L	8(A6),A5	* Dirección del Buffer
MOVE.W	12(A6),D1	* Dato de Descriptor
MOVE.W	14(A6),D2	* Dato de Tamaño
EOR.L	D3,D3	* Contador de caracteres a 0
EOR.L	D0,D0	* Inicializamos el resultado a 0
CMP.W	#0,D1	* Comparamos el descriptor con 0 para saber si es la línea A
BEQ	ESC_A	
CMP.W	#1,D1	* Comparamos el descriptor con 1 para saber si es la línea B
BEQ	ESC_B	
SUB.L	#1,D0	* Se introduce el valor -1 en D0 para indicar error
BRA	FIN_PERR	* Fin de PRINT con descriptor erróneo

ESC_A:

CMP.W	D3,D2	* Se comprueba si se han transmitido todos los caracteres
BEQ	FINP_A	
MOVE.B	(A5)+,D1	* Metemos el carácter en D1 y postincr. el puntero al buffer

MOVE.L	#2,D0	* Buffer A de transmisión
BSR	ESCCAR	
CMPL	#\$ffffff,D0	* Miramos si el buffer está lleno (devuelve -1 en D0)
BEQ	FINP_A	
ADD.W	#1,D3	* Sumamos uno al contador de caracteres
BRA	ESC_A	

FINP_A:

CMP.W	#0,D2	* Miramos si el tamaño a transmitir es cero
BEQ	FIN_PRINT	
MOVE.W	SR,D4	* Almaceno el valor previo de SR
MOVE.W	#\$2700,SR	* Se inhiben las interrupciones y se activa el modo supervisor
BSET	#0,IMR_COPIA	* Cambiamos el bit 0 de la copia del IMR para activar trans.
MOVE.B	IMR_COPIA,IMR	* Introducimos el valor de la copia del IMR al IMR
MOVE.	W D4,SR	* Restauramos el registro de estado
BRA	FIN_PRINT	

ESC_B:

CMP.W	D3,D2	* Se comprueba si se han transmitido todos los caracteres
BEQ	FINP_B	
MOVE.B	(A5)+,D1	* Metemos el carácter en D1 y postincr. el puntero al buffer
MOVE.L	#3,D0	* Buffer B de transmisión
BSR	ESCCAR	
CMPL	#\$ffffff,D0	* Miramos si el buffer está lleno (devuelve -1 en D0)
BEQ	FINP_B	
ADD.W	#1,D3	* Sumamos uno al contador de caracteres
BRA	ESC_B	

FINP_B:

CMP.W	#0,D2	* Miramos si el tamaño a transmitir es 0
BEQ	FIN_PRINT	

MOVE.W	SR,D4	* Almaceno el valor previo de SR
MOVE.W	#\$2700,SR	* Se inhiben las interrupciones y se activa el modo supervisor
BSET	#4,IMR_COPIA	* Cambiamos el bit 4 de la copia del IMR para activar trans.
MOVE.B	IMR_COPIA,IMR	* Introducimos el valor de la copia del IMR al IMR
MOVE.W	D4,SR	* Restauramos el registro de estado
FIN_PRINT:		
EOR.L	D0,D0	* Inicializamos a 0 el registro resultado
OR.L	D3,D0	* Cargamos el número total de caracteres impresos en D0
FIN_PERR:		
MOVE.L	-4(A6),A5	* Retomamos los parámetros de PRINT
MOVE.L	-8(A6),D1	
MOVE.L	-12(A6),D2	
MOVE.L	-16(A6),D3	
MOVE.L	-20(A6),D4	
UNLK	A6	* Rompemos el marco de pila
RTS		

3.4. RTI

La subrutina *RTI* es la rutina de tratamiento de interrupciones y, como su nombre indica, se ejecutará cuando se detecte (y además se pueda atender) una interrupción. Estas interrupciones pueden ser de recepción (se tienen que mover los caracteres del buffer de recepción de la línea al buffer interno de recepción) o de transmisión (se tienen que mover los caracteres del buffer interno de transmisión al buffer de transmisión de la línea).

RTI:

MOVEM.L	D0-D5,-(A7)	* Reserva de espacio en pila para registros usados en RTI
MOVE.B	IMR_COPIA,D4	* Se guarda el byte del IMR
MOVE.B	ISR,D5	* Se guarda el byte del ISR
AND.B	D4,D5	* Se comparan para ver los bits que tienen en común

BTST	#1,D5	* neg(D5) -> Z / Buscamos un 1, que en Z es un 0
BNE	REC_A	* Devuelve 1 si en Z hay un 0 y si lo es es recepción de A
BTST	#5,D5	* neg(D5) -> Z / Buscamos un 1, que en Z es un 0
BNE	REC_B	* Devuelve 1 si en Z hay un 0 y si lo es es recepción de B
BTST	#0,D5	* neg(D5) -> Z / Buscamos un 1, que en Z es un 0
BNE	TRA_A	* Devuelve 1 si en Z hay un 0 y si lo es es transmisión de A
BTST	#4,D5	* neg(D5) -> Z / Buscamos un 1, que en Z es un 0
BNE	TRA_B	* Devuelve 1 si en Z hay un 0 y si lo es es transmisión de B

FIN_RTI:

MOVEM.L	(A7)+,D0-D5	* Se recuperan los registros usados en RTI guardados en pila
---------	-------------	--

RTE

REC_A:

EOR.L	D0,D0	* Descriptor línea de recepción A (0)
EOR.L	D1,D1	* Inicialización de D1 a 0
MOVE.B	RBA,D1	* Cargo el carácter en D1
BSR	ESCCAR	
CMPL	#\$ffffff,D0	* Si la subrutina devuelve -1 es porque el buffer está lleno
BEQ	REC_FA	* Buffer lleno (full) de línea A
BRA	FIN_RTI	

REC_B:

MOVE.L	#1,D0	* Descriptor línea de recepción B (1)
EOR.L	D1,D1	* Inicialización de D1 a 0
MOVE.B	RBB,D1	* Cargo el carácter en D1
BSR	ESCCAR	
CMPL	#\$ffffff,D0	* Si la subrutina devuelve -1 es porque el buffer está lleno
BEQ	REC_FB	* Buffer lleno (full) de línea B
BRA	FIN_RTI	

REC_FA:

BCLR	#1,IMR_COPIA	* Deshabilitamos la pet. de int. de recepción de la línea A
MOVE.B	IMR_COPIA,IMR	* Actualizamos el IMR
BRA	FIN_RTI	

REC_FB:

BCLR	#5,IMR_COPIA	* Deshabilitamos la pet. de int. de recepción de la línea B
MOVE.B	IMR_COPIA,IMR	* Actualizamos el IMR
BRA	FIN_RTI	

TRA_A:

MOVE.L	#2,D0	* Descriptor línea de transmisión A (2)
BSR	LEECAR	
CMP.L	#\$ffffff,D0	* Si la subrutina devuelve -1 es porque el buffer está vacío
BEQ	TRA_EA	* Buffer vacío (empty) de línea A
MOVE.B	D0,TBA	* Cargo el carácter devuelto en TBA
BRA	FIN_RTI	

TRA_B:

EOR.L	#3,D0	* Descriptor linea de transmision B (3)
BSR	LEECAR	
CMP.L	#\$ffffff,D0	* Si la subrutina devuelve -1 es porque el buffer está vacío
BEQ	TRA_EB	* Buffer vacío (empty) de línea B
MOVE.B	D0,TBB	* Cargo el caracter devuelto en TBB
BRA	FIN_RTI	

TRA_EA:

BCLR	#0,IMR_COPIA	* Deshabilitamos la pet. de int. de transmisión de la línea A
MOVE.B	IMR_COPIA,IMR	* Actualizamos el IMR
BRA	FIN_RTI	

TRA_EB:

BCLR	#4,IMR_COPIA	* Deshabilitamos la pet. de int. de transmisión de la línea B
------	--------------	---

MOVE.B IMR_COPIA,IMR * Actualizamos el IMR
BRA FIN_RTI

4. Histórico del desarrollo de las rutinas

Ambos miembros del grupo realizaron el trabajo de manera conjunta y de forma telemática. Fueron dedicadas 28 horas totales de trabajo.

En primer lugar revisamos el enunciado del proyecto así como las rutinas que había que realizar junto al juego de instrucciones.

10/04/2022 (Comienzo de Proyecto)

10/04/2022: (La duración fue de 1 hora aprox.)

Esta sesión fue dedicada al planteamiento del proyecto, en la que hablamos sobre la organización y planificación. Propusimos hacer lecturas tanto del manual como del enunciado además de ver los vídeos de la explicación del proyecto de nuevo.

11/04/2022: (la duración fue de 4 horas aprox.)

Dedicamos este día para leer el manual además de revisar los vídeos publicados en *Moodle* acerca de la implementación de *SCAN* en salida programada para ir asimilando conceptos. También realizamos la configuración del escritorio virtual además de tener una primera toma de contacto con *Putty*, *Filezilla* y el simulador. También desarrollamos el *INIT*.

13/04/2022: (la duración fue de 3 horas aprox.)

Realizamos la rutina *SCAN* para ambos puertos A y B. Aunque con dudas acerca del marco de pila y de su funcionamiento. Inicialmente realizamos una implementación sin crear el marco de pila.

14/04/2022: (la duración fue de 3 horas aprox.)

Empezamos a realizar *PRINT*, tuvimos dificultades a la hora de saber cómo inhibir las interrupciones y de como activar el modo supervisor.

Por otro lado, entendimos el funcionamiento del marco de pila así como de cómo guardar los registros en la pila y cuáles de ellos se requieren almacenar como variable local.

17/04/2022: (la duración fue de 3 horas aprox.)

En esta sesión comenzamos con el planteamiento del *RTL*. Tuvimos dificultades acerca de cómo saber si teníamos interrupciones de recepción o de transmisión por el puerto A o B. Pero tras leer el manual, nos fijamos en el apartado en el que se especifica el funcionamiento del IMR y utilizamos la instrucción BTST para saber qué tipo de interrupción teníamos.

19/04/2022: (la duración fue de 3 horas aprox.)

Para la entrega del primer hito, compilamos nuestro código y tuvimos muchos errores de sintaxis. Tras un largo periodo de tiempo revisándolo, caímos en que el compilador solo procesa los 8 primeros caracteres de las etiquetas, por lo que debido a la longitud de éstas, el compilador detectaba varias etiquetas iguales. Cambiamos, por tanto, el nombre de la mayoría de las etiquetas definidas.

ENTREGA HITO N°1 : 19/04/2022 -> Fallan 25 pruebas

09/05/2022: (la duración fue de 2 horas aprox.)

En esta sesión revisamos las pruebas que fallaban y nos dimos cuenta de que el vector de interrupciones estaba mal inicializado, ya que lo teníamos inicializado a 40 en vez de a 60.

Tras este fallo probamos *SCAN* para la línea A y entregamos la primera corrección adicional.

CORRECCIÓN ADICIONAL N°1 : 09/05/2021 -> Fallan 21 pruebas

Fallan las pruebas correspondientes a *SCAN* de la línea B y todas las de *PRINT*.

15/05/2022: (la duración fue de 4 horas aprox.)

En esta sesión nos dimos cuenta del motivo por el que la línea B no funcionaba. Esto era debido a que en el *INIT* se nos había olvidado incluir CRB.

Tras subsanar este error, vimos otro fallo en *RTL*. Este fallo consiste en que, a la hora de comparar los bits del IMR para saber qué tipo de interrupción teníamos, nos dimos cuenta de que estábamos usando el registro que no era para hacer la comparación. Lo corregimos y realizamos pruebas en el simulador. Se entrega una corrección adicional.

CORRECCIÓN ADICIONAL N°2 : 17/05/2022 -> Fallan 8 pruebas, de la 37 a la 45

18/05/2022: (la duración fue de 1 hora aprox.)

En este punto, no sabemos muy bien por qué nuestro código no funciona para esas pruebas. Optamos por realizar otra implementación de *PRINT* y probarlo nuevamente.

CORRECCIÓN ADICIONAL N°3 : 18/05/2022 -> Fallan 8 pruebas, de la 37 a la 45

19/05/2022 por la mañana: (la duración fue de 1 hora aprox.)

En este punto, y tras no haber conseguido mejoras en la corrección, no sabemos muy bien por qué nuestro código no funciona para esas pruebas. Redactamos un correo con el objetivo de poder reconocer lo que ocurre. Se nos indica que no realizamos correctamente la comparación del contador con el tamaño a escribir/leer, ya que utilizamos `.l` en lugar de `.w`. Además de eso, se nos aconseja en *RTI* el marco de pila por la instrucción `MOVEM.L` para hacer el código más eficiente. Realizamos los correspondientes cambios además de nuevas pruebas y entregamos una nueva corrección adicional.

CORRECCIÓN ADICIONAL N°4 : 19/05/2022 -> Fallan 18 pruebas.

Fallan las pruebas correspondientes a *PRINT* porque no se deposita el valor de los caracteres leídos/escritos en `D0` pero pasan la 42 y 43.

19/05/2022 por la noche: (la duración fue de 2 horas aprox.)

Tras el resultado obtenido, quedamos sorprendidos ante el fallo de tantas pruebas. Revisamos el código entregado y vemos que los cambios realizados fueron hechos sobre un archivo antiguo en el que había una errata en el marco de pila, pues incluimos `D0` y por ese motivo, al finalizar *PRINT* se depositaba un `0` en `D0` siempre. Recuperamos el archivo sin erratas, realizamos de nuevo los cambios especificados en el día 19 de mayo y revisamos exhaustivamente el código con el objetivo de solventar cualquier tipo de falla al no disponer de más correcciones adicionales antes de la entrega definitiva.

20/05/2022: (la duración fue de 30 min. aprox.)

Volvemos a dedicar un momento para revisar el código y hacer pruebas antes de la entrega final. No se realizan más cambios.

ENTREGA DEFINITIVA: 20/05/2022 -> Fallan 4 pruebas.

Se solucionan la mayoría de las pruebas, excepto las que involucran concurrencia. No disponíamos de más tiempo ni opción para tratar ese problema. Sin embargo, se mejora la anterior corrección de 8 pruebas fallidas.

Última actualización: 21/05/2022

5. Conjuntos de casos prueba

Para la comprobación del buen funcionamiento de las subrutinas se llevaron a cabo una serie de pruebas con las que se trató de corroborar tanto el buen funcionamiento de las subrutinas de forma individual como en conjunción de las mismas. Para el total aprovechamiento de los resultados se ha de tener en cuenta que se revisaban los resultados mediante la función *Memory Viewer* del simulador, además de hacer un seguimiento del transcurso del programa mediante la opción *Program Listing*.

5.1. Scan

1. Caso 1: Cadena 144 caracteres por línea A

Llama a *SCAN* pasándole los siguientes parámetros:

- Línea de entrada: A
- Tamaño a leer: 144
- Cadena de entrada:

“El progreso no consiste en aniquilar hoy el ayer sino al revés, en conservar aquella esencia del ayer que tuvo la virtud de crear ese hoy mejor.”

- Salida esperada:

45,6c,20,70,72,6f,67,72,65,73,6f,20,6e,6f,20,63,6f,6e,73,69,73,74,65,20,65,6e,20,61,6e,
69,71,75,69,6c,61,72,20,68,6f,79,20,65,6c,20,61,79,65,72,20,73,69,6e,6f,20,61,6c,20,7
2,65,76,c3,a9,73,2c,20,65,6e,20,63,6f,6e,73,65,72,76,61,72,20,61,71,75,65,6c,6c,61,20,
65,73,65,6e,63,69,61,20,64,65,6c,20,61,79,65,72,20,71,75,65,20,74,75,76,6f,20,6c,61,
20,76,69,72,74,75,64,20,64,65,20,63,72,65,61,72,20,65,73,65,20,68,6f,79,20,6d,65,6a,
6f,72,2e

- Cadena visualizada tras la ejecución de *SCAN* desde *Memory Viewer*:

```
0026ba: ff 4e 45 6c 20 70 72 6f 67 72 65 73 6f 20 6e 6f 20 63 6f 6e 73 69 73 74
0026d2: 65 20 65 6e 20 61 6e 69 71 75 69 6c 61 72 20 68 6f 79 20 65 6c 20 61 79
0026ea: 65 72 20 73 69 6e 6f 20 61 6c 20 72 65 76 c3 a9 73 2c 20 65 6e 20 63 6f
002702: 6e 73 65 72 76 61 72 20 61 71 75 65 6c 6c 61 20 65 73 65 6e 63 69 61 20
00271a: 64 65 6c 20 61 79 65 72 20 71 75 65 20 74 75 76 6f 20 6c 61 20 76 69 72
002732: 74 75 64 20 64 65 20 63 72 65 61 72 20 65 73 65 20 68 6f 79 20 6d 65 6a
00274a: 6f 72 2e 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
002762: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Resultado: D0 = 0x90 (144)
Prueba realizada con éxito.

2. Caso 2: Dos bloques de 800 caracteres por línea B

Llama a *SCAN* pasándole los siguientes parámetros:

- Línea de entrada: B
- Tamaño a leer: 1600
- Cadena de entrada:

"El lenguaje provee a la conciencia de un cuerpo inmaterial en el que encarnarse."

Cadena escrita 10 veces por bloque (800 caracteres).

- Salida esperada:

45,6c,20,6c,65,6e,67,75,61,6a,65,20,70,72,6f,76,65,65,20,61,20,6c,61,20,63,6f,6e,63,6
9,65,6e,63,69,61,20,64,65,20,75,6e,20,63,75,65,72,70,6f,20,69,6e,6d,61,74,65,72,69,6
1,6c,20,65,6e,20,65,6c,20,71,75,65,20,65,6e,63,61,72,6e,61,72,73,65,2e

Repetida 20 veces en memoria.

- Cadena visualizada tras la ejecución de *SCAN* desde *Memory Viewer*:

Primer bloque:

```
00292a: 65 6c 20 71 75 65 20 65 6e 63 61 72 6e 61 72 73 65 2e 45 6c 20 6c 65 6e
002942: 67 75 61 6a 65 20 70 72 6f 76 65 65 20 61 20 6c 61 20 63 6f 6e 63 69 65
00295a: 6e 63 69 61 20 64 65 20 75 6e 20 63 75 65 72 70 6f 20 69 6e 6d 61 74 65
002972: 72 69 61 6c 20 65 6e 20 65 6c 20 71 75 65 20 65 6e 63 61 72 6e 61 72 73
00298a: 65 2e 45 6c 20 6c 65 6e 67 75 61 6a 65 20 70 72 6f 76 65 65 20 61 20 6c
0029a2: 61 20 63 6f 6e 63 69 65 6e 63 69 61 20 64 65 20 75 6e 20 63 75 65 72 70
0029ba: 6f 20 69 6e 6d 61 74 65 72 69 61 6c 20 65 6e 20 65 6c 20 71 75 65 20 65
0029d2: 6e 63 61 72 6e 61 72 73 65 2e 00 00 00 00 00 00 00 00 00 00 00 00 00
0029ea: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Empieza en 2702 (9986) y acaba en 29d2 (10706)

Segundo bloque:

```
002c5a: 65 2e 45 6c 20 6c 65 6e 67 75 61 6a 65 20 70 72 6f 76 65 65 20 61 20 6c
002c72: 61 20 63 6f 6e 63 69 65 6e 63 69 61 20 64 65 20 75 6e 20 63 75 65 72 70
002c8a: 6f 20 69 6e 6d 61 74 65 72 69 61 6c 20 65 6e 20 65 6c 20 71 75 65 20 65
002ca2: 6e 63 61 72 6e 61 72 73 65 2e 45 6c 20 6c 65 6e 67 75 61 6a 65 20 70 72
002cba: 6f 76 65 65 20 61 20 6c 61 20 63 6f 6e 63 69 65 6e 63 69 61 20 64 65 20
002cd2: 75 6e 20 63 75 65 72 70 6f 20 69 6e 6d 61 74 65 72 69 61 6c 20 65 6e 20
002cea: 65 6c 20 71 75 65 20 65 6e 63 61 72 6e 61 72 73 65 2e 00 00 00 00 00 00
002d02: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Empieza en 29d2 (10706) y acaba en 2cea (11498).

Resultado: Prueba realizada con éxito.

3. Caso 3: Descriptor erróneo

Llama a *SCAN* pasándole los siguientes parámetros:

- Línea de entrada: C (Se pasa el valor 2 por la pila)
- Tamaño a leer: 3000
- Cadena de entrada:

"0123456789"

- Salida esperada:

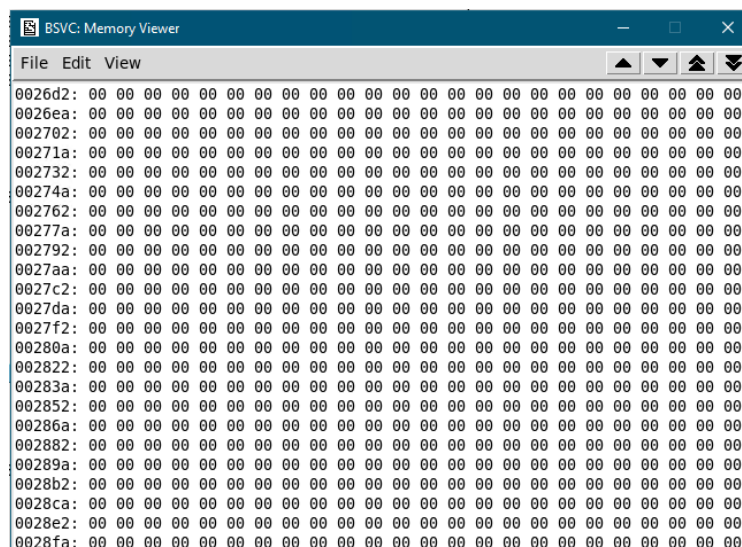
D0 = 0xFFFFFFFF (-1)

Cadena no introducida en el buffer

- Cadena visualizada tras la ejecución de *SCAN* desde *Memory Viewer*:

```
304  OTRAL:
305      MOVE.W #3000, -(A7)    * Tamano de bloque
306      MOVE.W #2, -(A7)      * Puerto A
307      MOVE.L PARDIR, -(A7)   * Direccion de lectura
308
309  ESPL:
310      BSR SCAN

156      CMP.W #0,D1           * Si el descriptor es 0 entonces la lectura sera por la linea A
157      BEQ LEC A
158      CMP.W #1,D1           * Si el descriptor es 1 entonces la lectura sera por la linea B
159      BEQ LEC B
160      SUB.L #1,D0
161      BRA FIN SERR          * Si no es ni 0 ni 1 entonces se produce un error
```



Resultado: D0 = 0xFFFFFFFF (-1)
Prueba realizada con éxito.

5.2. Print

1. Caso 1: Cadena de 53 caracteres escrita por la línea A.

Se llama a *PRINT* con los siguientes parámetros.

- Línea de Salida: A
- Tamaño a leer: 53
- Cadena de entrada:

“no se puede desatar un nudo sin saber como esta hecho”

- Salida esperada:

Por la línea A debería verse el texto anterior:

“no se puede desatar un nudo sin saber como esta hecho”

- Tras la ejecución de *PRINT* podemos ver el siguiente resultado en la línea A



Resultado: D0 = 0x35 (53 en hexadecimal)

La prueba se ha completado con éxito.

2. Caso 2: Cadena de 200 caracteres escritos por la línea B.

Se llama a *PRINT* con los siguientes parámetros.

- Línea de Salida: B
- Tamaño a leer: 200
- Cadena de entrada:

“Lo que distingue las mentes verdaderamente originales no es que sean la primeras en ver algo nuevo, sino que son capaces de ver como nuevo lo que es viejo, conocido, visto y menospreciado por todos.”

- Salida esperada:

Por la línea B debería verse el texto anterior:

“Lo que distingue las mentes verdaderamente originales no es que sean la primeras en ver algo nuevo, sino que son capaces de ver como nuevo lo que es viejo, conocido, visto y menospreciado por todos.”

La prueba va a consistir en dos bloques de 100 caracteres cada uno.

El primero será:

“Lo que distingue las mentes verdaderamente originales no es que sean la primeras en ver algo nuevo”

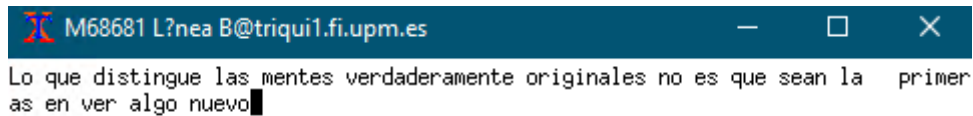
El segundo será:

“, sino que son capaces de ver como nuevo lo que es viejo, conocido, visto y menospreciado por todos.”

Ambos bloques serán escritos por la línea A.

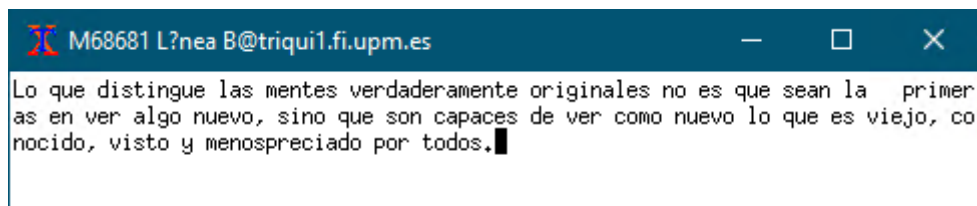
- Tras la ejecución de *PRINT* podemos ver el siguiente resultado en la línea B

1º Bloque



A screenshot of a terminal window with a dark blue title bar containing the text 'M68681 L?nea B@triqui1.fi.upm.es'. The terminal content shows the first 100 characters of the expected output: 'Lo que distingue las mentes verdaderamente originales no es que sean la primeras en ver algo nuevo'.

2º Bloque



A screenshot of a terminal window with a dark blue title bar containing the text 'M68681 L?nea B@triqui1.fi.upm.es'. The terminal content shows the next 100 characters of the expected output: ', sino que son capaces de ver como nuevo lo que es viejo, conocido, visto y menospreciado por todos.'.

Resultado: D0 = 0xC8 (200 caracteres)

Prueba realizada con éxito.

3. Caso 3: Descriptor erróneo.

- Línea de Salida: Descriptor erróneo(21)

ESPE:

```
MOVE.W #100, -(A7)    * Tamano de escritura
MOVE.W #21, -(A7)     * Puerto B
MOVE.L PARDIR, -(A7)  * Direccion de escritura
MOVE.L #$ffffffff,D2
BSR PRINT
ADD.L #8,A7           * Restablece la pila
```

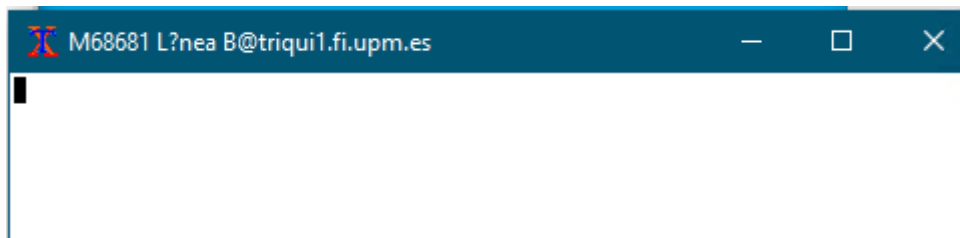
- Tamaño a leer: 200
- Cadena de entrada:

“Lo que distingue las mentes verdaderamente originales no es que sean la primeras en ver algo nuevo, sino que son capaces de ver como nuevo lo que es viejo, conocido, visto y menospreciado por todos.”

- Salida esperada:

El resultado debe ser en D0= ffffffff ya que pasamos un descriptor erróneo y por la línea B no debe de imprimir nada.

- Tras la ejecución de *PRINT* podemos ver el siguiente resultado en la línea B.



Está vacía.

D0 = ffffffff
D1 = 00000000

Resultado: D0 = 0xFFFFFFFF (-1)
Prueba realizada con éxito.

6. Observaciones finales y comentarios personales

Durante el desarrollo de este proyecto hemos realizado un sistema de entrada/salida mediante interrupciones. Gracias a esto hemos podido entender, aunque de una forma muy pincelada, cómo el procesador trata la comunicación con los periféricos en este caso mediante las interrupciones.

Para ello hemos realizado una serie de rutinas (*INIT*, *SCAN*, *PRINT*, *RTI*) en la que cada una de éstas se encargaba de una tarea esencial que, en conjunto, permite la introducción de una serie de caracteres para luego imprimirlos por la línea A y B o para simplemente guardar dichos caracteres en un buffer. Durante la realización de estas rutinas tuvimos problemas sobre todo al principio, ya que la programación en ensamblador no es la misma que utilizamos en el pasado proyecto de *Estructura de Computadores*, pero no se nos hizo bastante complicado adaptarnos a la nueva sintaxis del ensamblador MC68000.

Otro problema que tuvimos fue, en lo personal, el simulador, pues nos pareció que la máquina virtual que tuvimos que emplear no era lo suficientemente rápida para que el trabajo se hiciera ameno, ya que muchas veces se quedaba bloqueada, iba muy lenta o simplemente cerraba a sesión en mitad de una prueba.

Por otro lado, nos asustó mucho el hecho de pensar que teníamos pocas correcciones para poder completar el proyecto con éxito, además de que, tras la primera corrección que enviamos, nos pareció que las pruebas eran bastante complicadas de depurar, sobre todo cuando hablamos de muchos caracteres y de problemas concurrentes. Este problema hizo que muchas veces no supiéramos cuál era el motivo por el que las pruebas fallaban, ya que en ocasiones eran términos de eficiencia o de concurrencia.

Agradecemos también la atención del profesorado a lo largo de todo el proyecto, ya que en particular empleamos el correo electrónico para preguntar las dudas y plantear los problemas que nos surgían y la atención era, además de rápida, eficaz.

En cómputo general, consideramos que ha sido una práctica muy completa a la par que muy instructiva, pues nos ha servido para ver de primera mano cómo funciona una unidad de E/S por interrupciones (aunque sea de forma reducida).