Proyecto de Arquitectura de Computadores
Sistemas de Entrada/Salida
Curso 2024/2025

Santiago Rodríguez de la Fuente

Ángel Grover Pérez Muñoz

Departamento de Arquitectura y Tecnología de Sistemas Informáticos
Facultad de Informática
Universidad Politécnica de Madrid
Versión 1.0

Febrero 2025

Índice general

1. Microprocesador MC68000 **1**

1.1. Modelo de programación . . . 1
1.1.1. Modelo de programación del modo usuario . . . 1
1.1.2. Modelo de programación del modo supervisor . . . 1
1.1.3. Registro de estado . . . 3
1.2. Tipos de operandos y modos de direccionamiento . . . 3
1.3. Organización de datos en registros . . . 5
1.3.1. Registros de datos . . . 5
1.3.2. Registros de direcciones . . . 5
1.4. Mapa de direcciones . . . 5
1.4.1. Organización de datos en memoria . . . 5
1.5. Excepciones e interrupciones . . . 6
1.5.1. Procesamiento de excepciones . . . 7
1.5.2. Tabla de vectores de excepción . . . 7
1.5.3. Procesamiento de interrupciones . . . 7
1.5.4. Excepción de *reset* . . . 9
1.6. Juego de instrucciones . . . 9
1.6.1. Abreviaturas . . . 9
1.6.2. Transferencia de datos . . . 13
1.6.3. Aritméticas . . . 17
1.6.4. Lógicas . . . 23
1.6.5. Desplazamientos . . . 25
1.6.6. Manejo de bits . . . 31
1.6.7. Control del programa . . . 32
1.6.8. Control del procesador . . . 34

2. Controlador de líneas serie MC68681 **35**

2.1. Transmisión asíncrona de caracteres por una línea serie . . . 35
2.2. Características de la DUART MC68681 . . . 36
2.3. Descripción de los registros . . . 36
2.3.1. Registros de modo 1 (MR1A y MR1B) . . . 38
2.3.2. Registros de modo 2 (MR2A y MR2B) . . . 38
2.3.3. Registros de estado (SRA y SRB) . . . 38
2.3.4. Registros de selección de reloj (CSRA y CSRB) . . . 39
2.3.5. Registros de control (CRA y CRB) . . . 39
2.3.6. Registros del buffer de recepción (RBA y RBB) . . . 40

2.3.7. Registros del buffer de transmisión (TBA y TBB) . . . 40
2.3.8. Registro de control auxiliar (ACR) . . . 40
2.3.9. Registro de estado de interrupción (ISR) . . . 40
2.3.10. Registro de máscara de interrupción (IMR) . . . 41
2.3.11. Registro del vector de interrupción (IVR) . . . 41

3. Programa ensamblador 68kasm **43**
3.1. Llamada al programa ensamblador . . . 43
3.2. Formato del código fuente . . . 44
3.2.1. Campo de etiquetas . . . 44
3.2.2. Campo de operación . . . 44
3.2.3. Campo de operandos . . . 44
3.2.4. Campo de comentario . . . 44
3.2.5. Símbolos . . . 45
3.2.6. Expresiones . . . 45
3.2.7. Especificación de los modos de direccionamiento . . . 45
3.3. Instrucciones de bifurcación . . . 45
3.4. Pseudoinstrucciones . . . 46
3.4.1. Set origin `ORG` . . . 46
3.4.2. Equate `EQU` . . . 46
3.4.3. Define constant `DC` . . . 47
3.4.4. Define storage `DS` . . . 48
3.4.5. Set symbol `SET` . . . 48
3.4.6. Define register set `REG` . . . 48
3.4.7. Define constant block `DCB` . . . 48
3.4.8. `INCLUDE` . . . 49
3.5. Formato del listado ensamblador . . . 49
3.6. Ejemplos . . . 49

4. Simulador BSVC **51**
4.1. Carga de computador virtual . . . 51
4.2. Carga de un programa objeto . . . 53
4.3. Menús de la ventana de manejo del simulador . . . 53
4.4. Botones de la ventana de manejo del simulador . . . 54
4.5. Resultados de la ejecución de un programa . . . 54
4.6. Errores conocidos . . . 55

5. Enunciado del proyecto: E/S mediante interrupciones **57**
5.1. Normas de presentación . . . 77

A. Conexión a los computadores de prácticas de la Escuela **85**
A.1. Utilización de un computador Linux . . . 85
A.2. Utilización de escritorios remotos . . . 85

B. Instalación del entorno de la práctica en un computador con sistema operativo Linux **89**
B.1. Instalación en Linux 89
B.1.1. Obtención del entorno 89
B.1.2. Instalación del paquete 89
B.2. Compilación de bsvc en Linux 90
B.2.1. Obtención del entorno 90
B.2.2. Instalación del paquete 90

C. Depuración de fallos que se manifiesten como excepciones en el procesador MC68000 **93**
C.1. Identificación la instrucción que provocó la excepción 94
C.2. Ejemplo 95

Capítulo 1

Microprocesador MC68000

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.

1.1. Modelo de programación

El MC68000 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: este modo proporciona el entorno de ejecución para los programas de aplicación.

Modo supervisor: en este modo se proporcionan algunas instrucciones privilegiadas que no están disponibles en el modo usuario. El software de sistema y el sistema operativo ejecuta en este modo privilegiado.

1.1.1. Modelo de programación del modo usuario

El modelo de programación del modo usuario es común para todos los microprocesadores de la familia M68000. Este modelo contiene (ver figura 1.1) 16 registros de 32 bits de propósito general (D0–D7, A0–A7), un contador de programa (PC, Program Counter) de 32 bits y un registro de estado de 8 bits (CCR, Condition Code Register).

Los primeros 8 registros (D0–D7) se usan para almacenar operandos de un octeto (byte = 8 bits), una palabra (word = 16 bits) o de una palabra larga (long–word = 32 bits). El otro grupo de 7 registros (A0–A6) y el puntero de pila de usuario (A7 ó USP, User Stack Pointer) se pueden usar como registros de direcciones o punteros de pila de usuario, en particular el registro A6 se suele utilizar como puntero de marco (FP, Frame Pointer). Estos registros de direcciones sólo se pueden usar para operandos de una palabra o de una palabra larga, pero no de un octeto. Todos estos 16 registros se pueden usar como registros índices en los desplazamientos múltiples.

1.1.2. Modelo de programación del modo supervisor

En el modelo de programación del modo supervisor están disponibles todos los registros del modo usuario y otros adicionales. Los registros adicionales de este modo para el procesador

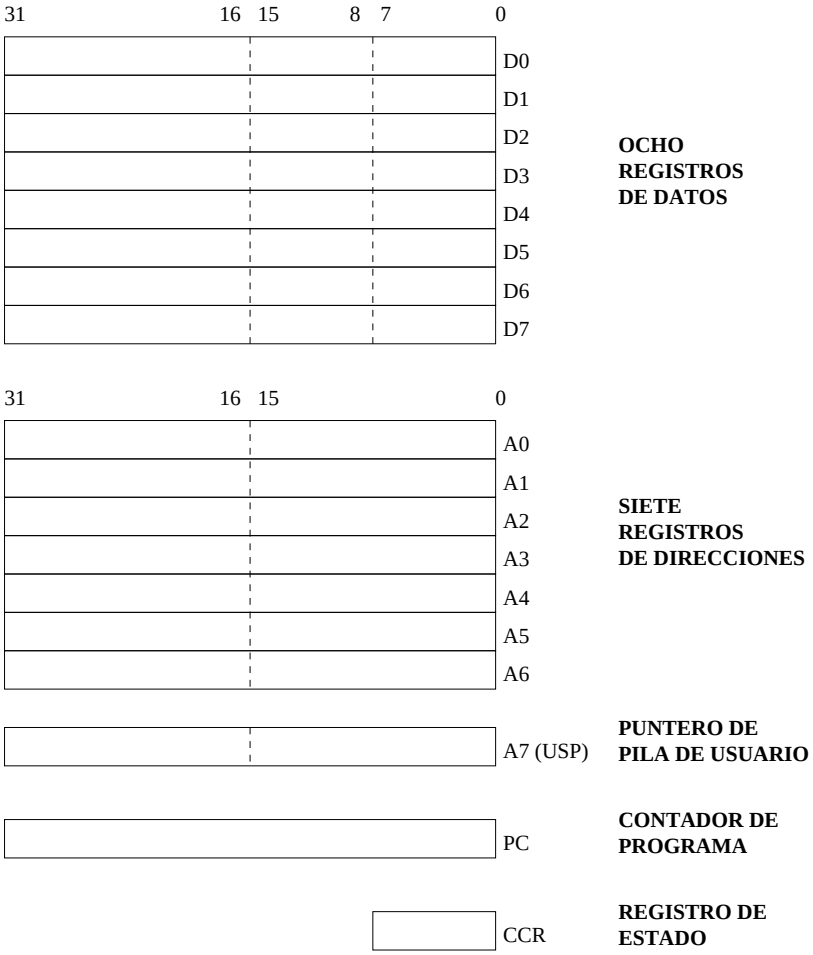


Figura 1.1: Modelo de programación de usuario

MC68000 se muestran en la figura 1.2.

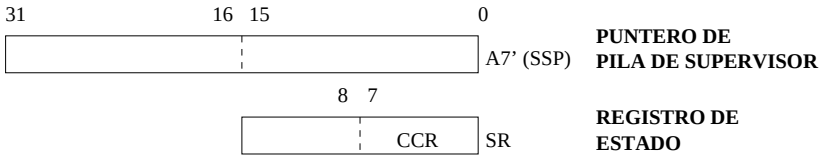


Figura 1.2: Modelo de programación de supervisor

En este modo se tiene acceso al octeto de mayor peso del registro de estado y se dispone de un puntero de pila de supervisor (A7' ó Supervisor Stack Pointer). De este modo, si se referencia el registro A7 se referencia el USP o el SSP dependiendo del modo de ejecución. Como es habitual, desde el modo supervisor se tiene acceso al puntero de pila de usuario, pero no a la inversa.

1.1.3. Registro de estado

El registro de estado (ver figura 1.3) tiene 16 bits, aunque los bits 5, 6, 7, 11, 12 y 14 no se usan en el MC68000 y están reservados para uso futuro. El octeto de menor peso (octeto de usuario) contiene los siguientes biestables de estado: desbordamiento (V, oVerflow), cero (Z, Zero), negativo (N, Negative), acarreo (C, Carry) y precisión extendida (X, Extend). El octeto de mayor peso (octeto de sistema) contiene biestables de control de modo para modo traza (T, Trace) y supervisor (S, supervisor), así como la máscara de interrupción (I_2,I_1,I_0).

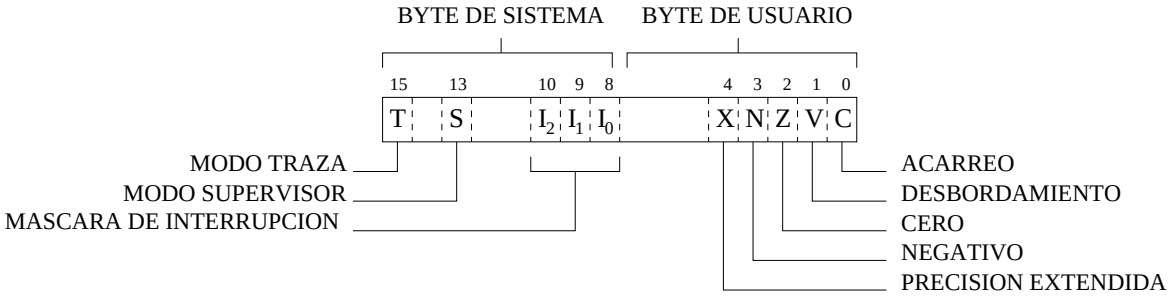


Figura 1.3: Registro de estado

1.2. Tipos de operandos y modos de direccionamiento

El procesador MC68000 soporta los siguientes tipos de operandos:

1. Bits.
2. Decimal Codificado en Binario (BCD) de 4 bits por dígito.
3. Octetos (8 bits)
4. Palabras (16 bits)
5. Palabras largas (32 bits)

Existen nemónicos específicos para operandos de bits y BCD. Sin embargo, para distinguir entre octetos, palabras y palabras largas se añade un sufijo al nemónico en cuestión. De este modo, `MOVE.B` indica operandos de 8 bits (`B`, Byte), `MOVE.W` operandos de 16 bits (`W`, Word) y `MOVE.L` operandos de 32 bits (`L`, Longword).

En cuanto a los modos de direccionamiento, el MC68000 proporciona 14 modos de direccionamiento que se pueden agrupar en los siguientes 7 tipos básicos:

1. Directo a registro
2. Absoluto
3. Indirecto a registro
4. Relativo a contador de programa.

5. Relativo a registro base.
6. Inmediato
7. Implícito

La tabla 1.1 detalla cada uno de los modos de direccionamientos del MC68000.

Modo	Cálculo de la dirección	Sintaxis
Directo a registro		
de datos	EA=Registro de dato n	Dn
de dirección	EA=Registro de dirección n	An
Absoluto		
corto	EA=Siguiente palabra en CO	(XXX).W
largo	EA=Siguientes dos palabras en CO	(XXX).L
Indirecto a registro	EA=An	(An)
con postincremento	EA=An, An ← An+N	(An)+
con predecremento	An ← An–N, EA=An	-(An)
Relativo a PC	EA=PC+d_{16}	(d_{16},PC)
con desplazamiento múltiple	EA=PC+Xn+d_8	(d_8,PC,Xn)
Relativo a registro base	EA=An+d_{16}	(d_{16},An)
con desplazamiento múltiple	EA=An+Xn+d_8	(d_8,An,Xn)
Inmediato		
byte, palabra o palabra larga	DATA=Siguientes bytes en CO	#< *data* >
corto	DATA=3 bits en CO	#< *data* >
Implícito a registro	EA=SR,USP,SSP,PC	SR,USP,SSP,PC

Leyenda: EA = Dirección efectiva
Dn = Registro de datos
An = Registro de direcciones
CO = Código de operación
d_8 = Desplazamiento de 8 bits
d_{16} = Desplazamiento de 16 bits
N = 1 para byte, 2 para palabra y 4 para palabra larga
← = Sustituye
Xn = Registro de datos o direcciones usado como índice en los desplazamientos múltiples
SR = Registro de estado
PC = Registro contador de programa
USP= Registro puntero de pila de usuario
SSP = Registro puntero de pila de supervisor

Tabla 1.1: Modos de direccionamiento

1.3. Organización de datos en registros

Los ocho registros de datos proporcionan soporte para operandos de 1, 8, 16 o 32 bits. Los siete registros de direcciones y el puntero de pila activo proporcionan soporte para direcciones de 32 bits.

1.3.1. Registros de datos

Cada registro de datos contiene 32 bits. Los operandos de octeto ocupan los 8 bits de menor peso, los de palabra los 16 bits de menor peso y los de palabra larga los 32 bits. El bit menos significativo se direcciona como bit 0 y el más significativo como bit 31.

Si alguno de estos registros se usa como fuente o destino de una operación de tamaño inferior a palabra larga, sólo se modifica la parte de menor peso correspondiente y la parte de mayor peso permanece inalterada.

1.3.2. Registros de direcciones

Estos registros de direcciones (junto con el puntero de pila) tienen 32 bits y contiene una dirección completa de 32 bits. Los registros de direcciones no proporcionan soporte para operandos de byte. A diferencia de los registros de datos, si un registro de direcciones se usa como destino, el registro completo se modifica aunque la operación tenga un tamaño de palabra. En este caso, se extiende el signo de los operandos antes de la realización de la operación.

1.4. Mapa de direcciones

Los procesadores de la familia M68000 tienen un único espacio o mapa de direcciones. De este modo, en dicho mapa de direcciones se deben ubicar tanto los dispositivos que configuran la memoria principal como los controladores de periféricos. Por lo tanto, dentro del juego de instrucciones, no existen instrucciones específicas de Entrada/Salida, sino que todas las instrucciones se podrán utilizar para direccionar tanto memoria principal como los controladores de periférico.

Aunque el MC68000 posee registros de direcciones de 32 bits de ancho, solo dispone 24 bits en el bus de direcciones[1]. Como posee direccionamiento a nivel de byte, puede direccionar 2^{24} Bytes es decir 16 Megabytes. Lo habitual es reservar las últimas direcciones del mapa para ubicar los controladores de periféricos.

1.4.1. Organización de datos en memoria

El MC68000 proporciona direccionamiento a nivel de byte con una organización *big–endian* como se muestra en la figura 1.4. Como se observa, el byte de mayor peso tiene asignado una dirección menor que la del byte de menor peso.

Las instrucciones y operandos de varios bytes se deben almacenar en direcciones pares. De otro modo, se genera la excepción *Address Error* si se intenta acceder a un operando de más de un byte con una dirección impar. En particular, los operandos de tamaño de palabra

[1]En realidad, existen desde el A_1 hasta el A_{23} para direccionar palabras y dos líneas adicionales UDS y LDS (Upper y Lower Data Strobes) para seleccionar el byte de mayor peso, el de menor peso o ambos de cada palabra.

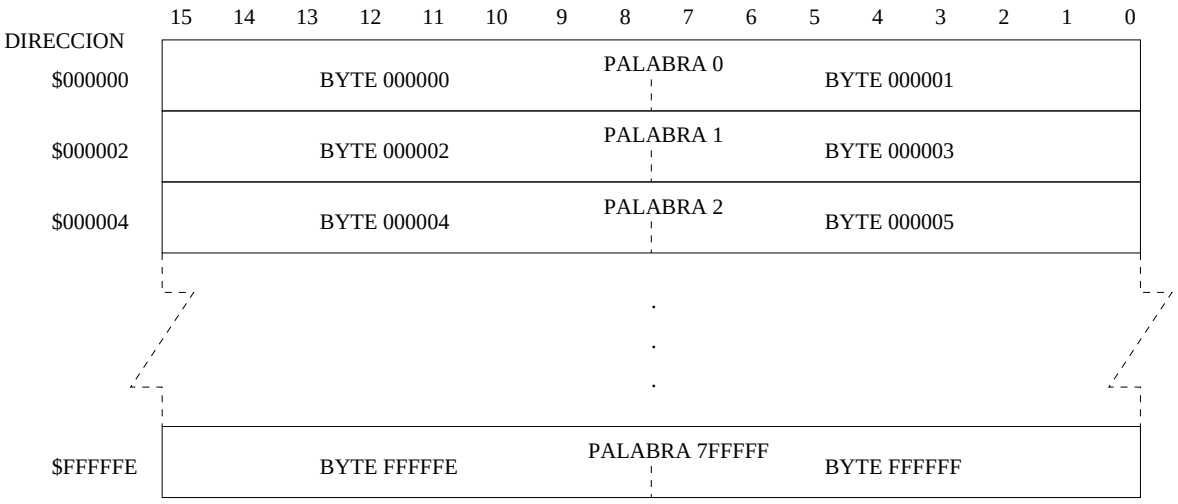


Figura 1.4: Organización de palabras en memoria

larga se almacenan en dos palabras consecutivas. Es decir, si la primera palabra se ubica en la dirección n (n debe ser par), la segunda palabra se ubica en la dirección n+2.

1.5. Excepciones e interrupciones

El término excepción denota comportamientos anómalos o no deseados por el programa en ejecución, tal es el caso de los cepos o *traps* y las interrupciones. Cuando se presenta un suceso como una división por cero, un error en el bus, una violación de privilegios o un controlador de periférico solicita una interrupción, se abandona el procesamiento de las instrucciones del programa en ejecución y se ejecuta la secuencia o ciclo de reconocimiento de excepciones.

Por lo tanto, las interrupciones que solicitan los controladores de periférico son un caso particular de las excepciones. Entre las excepciones que trata el MC68000 se encuentran:

Excepción de reset Es la que se procesa al activar $\overline{RESET}$ y $\overline{HALT}$ de forma simultánea y supone una reinicialización del procesador.

Errores de bus y de dirección Un error de bus se produce cuando desde el exterior se activa la señal $\overline{BERR}$ en un ciclo de bus, y un error de dirección, cuando se intenta acceder con dirección impar a palabras o dobles palabras. En ambos casos se suspende el ciclo de memoria y se trata la excepción, en este caso no se puede completar la ejecución de la instrucción.

Traza Se procesa al final de cada instrucción si está activo el bit de traza del registro de estado. Se utiliza para la depuración de programas.

Instrucción ilegal Se procesa cuando se intenta ejecutar un código de instrucción no implementado. Dos casos especiales son los códigos de instrucción destinados a los coprocesadores de coma flotante y memoria paginada, que tienen vectores específicos. Estos códigos comienzan con las secuencias 1111 y 1010.

Violación de privilegio Se produce cuando se intenta ejecutar en modo usuario instrucciones que sólo se pueden ejecutar en modo supervisor.

TRAP La propia instrucción TRAP procesa una de las 16 excepciones TRAP que expresa un operando inmediato. Se utiliza para solicitar servicios del sistema operativo.

TRAPV Se procesa la excepción si se cumple la condición de overflow al ejecutar una instrucción TRAPV.

CHK Se procesa la excepción si ejecutando la instrucción CHK se detecta un desbordamiento de rango. Se utiliza para dar soporte a los lenguajes de alto nivel[2] que poseen rangos en tipos enteros, como por ejemplo el índice de un vector.

División por 0 Se provoca la excepción si en una instrucción de división el cociente es 0.

Interrupción Se procesa al final de una instrucción si se solicita una interrupción por las líneas de interrupción $\overline{IPL2}$, $\overline{IPL1}$ e $\overline{IPL0}$, con una prioridad mayor a la codificada en los bits I_2, I_1 y I_0 del registro de estado.

1.5.1. Procesamiento de excepciones

En procesamiento de cualquier excepción, supone almacenar en la pila el registro de estado y contador de programa[3]. A continuación, se lleva el contenido de la entrada correspondiente de la tabla de vectores al PC y se procede a ejecutar instrucciones. Por lo tanto, en la entrada correspondiente de la tabla de vectores debe de almacenarse previamente la dirección de la rutina de tratamiento de la excepción. Además, se coloca a 1 el bit S del registro de estado y por lo tanto las rutinas de tratamiento de excepciones se ejecutan en modo privilegiado.

Existe una instrucción específica para retornar de rutina de excepción (`RTE`). Esta instrucción restaura el contador de programa y el registro de estado que se almacenaron en la pila durante el procesamiento de la excepción. Por lo tanto, al restaurar el registro de estado, restaura el bit S y se restaura el modo de ejecución anterior.

1.5.2. Tabla de vectores de excepción

La entrada de la tabla de vectores cuyo contenido se carga en el PC, viene predeterminada para todas las excepciones excepto para las interrupciones de los periféricos. El MC68000 tiene asignadas entradas distintas para cada tipo de excepción, según se muestra en la figura 1.5.

Cada entrada de esta tabla tiene 4 bytes, ya que almacena una dirección. Por lo tanto ocupa 1 Kbytes. Además y a diferencia de los otros procesadores de la familia M68000, el MC68000 no posee un registro base de la tabla de vectores. Por lo tanto, esta tabla siempre debe estar ubicada a partir de la dirección `000000` y abarca hasta la dirección `0003FF`.

1.5.3. Procesamiento de interrupciones

Las interrupciones son un caso particular de excepciones y su procesamiento es ligeramente diferente. Las principales diferencias se refieren a:

[2] Es decir, todos menos C

[3] En el procesamiento de las excepciones de dirección y bus, el procesador almacena en pila más información.

Número vector	Dirección		Excepción
	Dec	Hex	
0	0	000	*Reset: SSP inicial*
-	4	004	*Reset: PC inicial*
2	8	008	*Error de bus*
3	12	00C	*Error de dirección*
4	16	010	*Instrucción ilegal*
5	20	014	*División por 0*
6	24	018	*Instrucción CHK*
7	28	01C	*Instrucción TRAPV*
8	32	020	*Violación de privilegio*
9	36	024	*Traza*
10	40	028	*Emulador instrucción 1010*
11	44	02C	*Emulador instrucción 1111*
12-14	48	030	*Reservado*
15	60	03C	*Vector Int. no inicializado*
16-23	64	040	*Reservado*
24	96	060	*Interrupción espúrea*
25-31	100	064	*Autovectores 1-7*
32-47	128	080	*Instrucciones TRAP 0-15*
48-63	192	0C0	*Reservado*
64-255	256	100	*Vectores de interrupción*

Figura 1.5: Tabla de vectores de excepción del MC68000

Enmascaramiento selectivo. Solo se procesa la interrupción, al final de la ejecución de la instrucción en curso, si el nivel de la solicitud[4] es estrictamente superior al nivel almacenado en los bits I_2, I_1 y I_0 del registro de estado. Además del bit S, también se modifican estos bits almacenando el nivel de la interrupción que se procesa. Así se enmascaran las interrupciones de nivel igual o menor a la que se ha reconocido. La instrucción (`RTE`) restaura el registro de estado, por lo tanto restaura el nivel de interrupción anterior.

Vectorización. A diferencia de las excepciones propiamente dichas, los controladores de periféricos deben suministrar un vector de interrupción de un byte[5]. Este vector es el número de la entrada de la tabla de vectores donde está almacenada la dirección de su rutina de tratamiento.

Por lo tanto, durante la secuencia de reconocimiento de interrupciones se producen varios ciclos de bus para obtener la dirección de la rutina de tratamiento de la interrupción.

1. El ciclo de bus de reconocimiento de interrupciones cuyo objetivo es leer el vector de interrupción. Este vector debe estar comprendido entre 64 y 255 (ver tabla 1.5).
2. El vector de interrupción leído se multiplica por 4 para obtener la dirección de la entrada correspondiente de la tabla de vectores. Con la dirección obtenida se inicia

[4]Excepto las de nivel 7 que no con enmascarables

[5]También existe la posibilidad de interrupciones autovectorizadas para compatibilidad con periféricos de otras familias distintas a la M68000

un ciclo de bus para leer la dirección de comienzo de la rutina de interrupción. Esta dirección almacenada en la tabla de vectores se lleva al PC y se comienza a ejecutar la rutina de tratamiento.

1.5.4. Excepción de *reset*

Cuando se reinicia el computador, se produce la excepción de *RESET*. Durante el procesamiento de esta excepción no se almacena nada en la pila, ya que se usa para iniciar o reiniciar el procesador.

Los pasos más destacables de esta excepción son:

- Se lee el contenido de los 4 primeros bytes de la memoria y se llevan al SSP (puntero de pila de supervisor).
- Se lee el contenido de los siguientes 4 bytes y se llevan al PC.
- Se colocan a 1 los bits S, I_2, I_1 y I_0 del registro de estado. Por lo tanto, se va iniciar la ejecución de instrucciones en modo privilegiado y con las interrupciones inhibidas.
- Se realiza el *fetch* de la instrucción apuntada por el PC y se comienza la ejecución de instrucciones.

Por lo tanto, se debe colocar la dirección de inicio del programa en las direcciones `000004 - 000007` y el puntero de pila inicial en las direcciones `000000 - 000003`. Además, cuando se esté en condiciones de reconocer interrupciones se debe colocar `000` en los biestables de máscara de interrupción I_2, I_1 y I_0 del registro de estado.

1.6. Juego de instrucciones

Las instrucciones se resumen en el conjunto de tablas ubicadas a partir de la página 13. En ellas se representan el nemónico de las operaciones, los operandos posibles, los bytes que ocupan, el número de ciclos de reloj que consumen, los bits de estado que se modifican y un pequeño resumen de la operación que ejecutan.

Los posibles operandos se representan de forma abreviada. La descripción de estas abreviaturas se encuentra en la sección 1.6.1.

Se incluye una columna que determina el número de ciclos de reloj que emplea cada instrucción. El número de ciclos aparece con el siguiente formato: **t**(**r**/**w**), donde la t determina el número total de ciclos de reloj que emplea el procesador en ejecutar esa instrucción, incluido el ciclo de fetch. **r** determina cuántos ciclos de memoria se realizan en la instrucción, y **w**, cuántos de escritura. En algunos casos aparece un signo +, que indica que hay que sumar ciclos a los que aparecen, y que se emplean para calcular la dirección efectiva según sea ésta. Lo que hay que sumar depende del direccionamiento, y se obtendrá de la tabla 1.6. En ella todas las **w** valen 0.

1.6.1. Abreviaturas

Cuando aparece una X en un bit de estado, quiere decir que la operación modifica el bit de estado: 0 lo pone a 0 y 1 lo pone a 1. U quiere decir que quedará indefinido. Las abreviaturas empleadas son:

Modo de direccionamiento		**Pala.,byte**	**Larga**
Dn	Directo a registro datos	0(0/0)	0(0/0)
An	Directo a registro direcciones	0(0/0)	0(0/0)
(An)	Indirecto a registro	4(1/0)	8(2/0)
(An)+	Indirecto a registro con postincremento	4(1/0)	8(2/0)
-(An)	Indirecto a registro con predecremento	6(1/0)	10(2/0)
$Desp(An)	Relativo a registro base	8(2/0)	12(3/0)
$Desp(An,I)	Relativo a registro base con desplazamiento múltiple	10(2/0)	14(3/0)
xxx.W	Absoluto corto	8(2/0)	12(3/0)
xxx.L	Absoluto largo	12(3/0)	16(4/0)
$Desp(PC)	Relativo a PC	8(2/0)	12(3/0)
$Desp(An,I)	Relativo a PC con desplazamiento múltiple	10(2/0)	14(3/0)
#xxx	Inmediato	4(1/0)	8(2/0)

Figura 1.6: Cálculo de ciclos a partir de direcciones efectivas

addr: Dirección absoluta de 16 o 32 bits.

An: Registro de dirección A0, A1,... A7.

bitb: Número de bit de byte 0..7.

bitl: Número de bit de doble palabra 0..31.

cc: Código de condición:

CC: Carry Clear: $\overline{C}$

CS: Carry Set: C

EQ: Equal: Z

F: False: 0

GE: Greater Than or Equal: $(N \wedge V) \vee (\overline{N} \wedge \overline{V})$

GT: Greater Than: $(N \wedge V \wedge \overline{Z}) \vee (\overline{N} \wedge \overline{V} \wedge \overline{Z})$

HI: High: $\overline{C} \wedge \overline{Z}$

LS: Low or Same: $C \vee Z$

LE: Less Than or Equal: $Z \vee (N \wedge \overline{V}) \vee (\overline{N} \wedge V)$

LT: Less Than: $(N \wedge \overline{V}) \vee (\overline{N} \wedge V)$

MI: Minus: N

NE: Not Equal: $\overline{Z}$

PL: Plus: $\overline{N}$

T: True: 1

VC: Overflow Clear: $\overline{V}$

VS: Overflow Set: V

CCR: Condition Code Register. Byte de usuario del registro de estado.

count: Contador de desplazamiento 1-8.

count o Dn: Operando que puede ser count o Dn, indistintamente.

dadr: Dirección destino, modos de direccionamiento posibles:

- Indirecto a registro (An).
- Indirecto a registro con postincremento (An)+.

- Indirecto a registro con predecremento -(An).
- Relativo a registro base d16(An).
- Indirecto a registro indexado con desplazamiento d8(An,i).
- addr Dirección Absoluta.

dAn: Registro de dirección destino.

dDn: Registro de datos destino.

data3: 3 bits de dato inmediato.

data8: 8 bits de dato inmediato.

data16: 16 bits de dato inmediato.

data32: 32 bits de dato inmediato.

Dn: Registro de datos, n=0..7 (tamaños de 8, 16 o 32).

d8: Desplazamiento de 8 bits.

d16: Desplazamiento de 16 bits.

i: Registro índice (An o Dn, en desplazamiento múltiple).

jadr: Dirección de salto. Igual que **sadr**, excluidos (An)+ y -(An).

label: Etiqueta o dirección absoluta.

madr: Dirección múltiple. Igual que **dadr**, excluidos (An)+ y -(An).

reglist: Lista de registros, incluye varios registros separados por comas o rangos de registros Rn-Rm separados por '/'

rd: Registro destino (dDn o dAn).

rs: Registro fuente (sDn o sAn).

sadr: Dirección fuente, puede ser una de las siguientes:

- Indirecto a registro (An).
- Indirecto a registro con postincremento (An)+.
- Indirecto a registro con predecremento -(An).
- Relativo a registro base d16(An).
- Indirecto a registro indexado con desplazamiento d8(An,i).
- Dirección absoluta **addr**.
- **label** o **addr** empleadas en direccionamiento relativo a PC.
- Direccionamiento relativo a PC indexado **label**(i).

sAn: Registro de dirección fuente.

sDn: Registro de datos fuente.

SR: Registro de estado.

USP: Puntero de pila de usuario (no es el registro A7).

vector: Dirección del vector de TRAP

[ ]: Contenido de la posición de memoria cuya dirección está contenida en el registro especificado.

- $Dn \leftarrow [An]$ Indica que se almacena en Dn el contenido de la posición de memoria cuya dirección está cargada en An.
- $Dn \leftarrow An$ Indica que se carga en Dn el contenido de An.

$\overline{x}$: Complemento del valor de x.

$x < y - z >$: Bits del y al z de x. Por ejemplo, $Dn < 0 - 7 >$ representa los 8 bits bajos de Dn.

$+$:	Suma.	$\wedge$:	AND lógico.
$-$:	Resta.	$\vee$:	OR lógico.
$\times$:	Multiplicar.	$\oplus$:	OR Exclusivo lógico.
$\div$:	Dividir.	$=$:	Igual.

$\leftarrow$: Movimiento de datos en el sentido de la flecha.

$\leftarrow\rightarrow$: Intercambio de datos entre dos posiciones.

En las siguientes instrucciones se comentan las siguientes cuatro notas, señaladas en los casos necesarios mediante una indicación del tipo $comentario^i$, donde i puede ser 1, 2, 3 o 4:

1. Los postincrementos suman 1 y los predecrementos restan 1, salvo que el registro de dirección que aparezca sea el A7, en cuyo caso se trabaja con 2 en vez de 1, para que A7 siempre tenga una dirección par.
2. La dirección efectiva debe ser par.
3. El postincremento o predecremento se hace sumando o restando 2.
4. El postincremento o predecremento se hace sumando o restando 4.

1.6.2. Transferencia de datos

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
MOVE.B	(An),Dn	2	8(2/0)		X	X	0	0	Dn<0−7>←[An]
	(An)+,Dn	2	8(2/0)		X	X	0	0	Dn<0−7>←[An], An←An+1[1]
	-(An),Dn	2	10(2/0)		X	X	0	0	An←An−1, Dn<0−7>←[An][1]
	d16(An),Dn	4	12(3/0)		X	X	0	0	Dn<0−7>←[An+$d16$]
	d8(An,i),Dn	4	14(3/0)		X	X	0	0	Dn<0−7>←[An+$d8$+i]
	addr,Dn	4,6	4(1/0)+		X	X	0	0	Dn<0−7>←[$addr$]
	label,Dn	4	12(3/0)		X	X	0	0	Dn<0−7>←[PC+$d16$]
	label(i),Dn	4	14(3/0)		X	X	0	0	Dn<0−7>←[PC+$d8$+i]
MOVE.B	Dn,(An)	2	9(1/1)		X	X	0	0	[An]←Dn<0−7>
	Dn,(An)+	2	9(1/1)		X	X	0	0	[An]←Dn<0−7>, An←An+1[1]
	Dn,-(An)	2	9(1/1)		X	X	0	0	An←An−1, [An]←Dn<0−7>[1]
	Dn,d16(An)	4	13(2/1)		X	X	0	0	[An+$d16$]←Dn<0−7>
	Dn,d8(An,i)	4	15(2/1)		X	X	0	0	[An+$d8$+i]←Dn<0−7>
	Dn,addr	4,6	5(0/1)+		X	X	0	0	[$addr$]←Dn<0−7>
MOVE.B	sadr,dadr	2,4,6,8,10	5(1/1)+		X	X	0	0	[$dadr$]←[$sadr$][1]
MOVE.W	sadr,Dn	2,4,6	4(1/0)+		X	X	0	0	Dn<0−15>←[$sadr$][2,3]
MOVE.W	sadr,An	2,4,6	4(1/0)+		X	X	0	0	An<0−15>←[$sadr$], An<16−31>←An<15>[2,3]
MOVE.W	rs,dadr	2,4,6	5(0/1)+		X	X	0	0	[$dadr$]←rs<0−15>[2,3]
MOVE.W	sadr,dadr	2,4,6,8,10	5(0/1)+		X	X	0	0	[$dadr$]←[$sadr$][2,3]
MOVE.L	sadr,Dn	2,4,6	4(1/0)+		X	X	0	0	Dn<0−31>←[$sadr$][2,4]
MOVE.L	sadr,An	2,4,6	8(2/0)+		X	X	0	0	An<0−31>←[$sadr$][2,4]
MOVE.L	rs,dadr	2,4,6	10(0/2)+		X	X	0	0	[$dadr$]←rs<0−31>[2,4]
MOVE.L	sadr,dadr	2,4,6,8,10	14(1/2)+		X	X	0	0	[$dadr$]←[$sadr$][2,4]
MOVE.W	data16,An	4	8(2/0)+		X	X	0	0	An<0−15>←$data16$, An<16−31>←An<15>
MOVE.W	data16,dadr	4,6,8	9(1/1)+		X	X	0	0	[$dadr$]←$data16$
MOVE.L	data32,Dn	6	12(3/0)		X	X	0	0	Dn<0−31>←$data32$
MOVE.L	data32,An	6	12(3/0)+		X	X	0	0	An<0−31>←$data32$
MOVE.L	data32,dadr	6,8,10	18(2/2)+		X	X	0	0	[$dadr$]←$data32$
MOVE.B	sDn,dDn	2	4(1/0)		X	X	0	0	dDn<0−7>←sDn<0−7>
MOVE.W	rs,Dn	2	4(1/0)		X	X	0	0	Dn<0−15>←rs<0−15>
MOVE.W	rs,An	2	4(1/0)						An<0−15>←rs<0−15>, An<16−31>←An<15>
MOVE.L	rs,Dn	2	4(1/0)		X	X	0	0	Dn<0−31>←rs<0−31>
MOVE.L	rs,An	2	4(1/0)						An<0−31>←rs<0−31>
MOVE	An,USP	2	4(1/0)						USP←An Instrucción ejecutable en supervisor
MOVE	USP,An	2	4(1/0)						An←USP Instrucción ejecutable en supervisor

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
EXG	rs,rd	2	6(1/0)						$rs \leftarrow\rightarrow rd$
LINK	An,d16	4	18(2/2)						$A7 \leftarrow A7-4$, $[A7] \leftarrow An$ $An \leftarrow A7$, $A7 \leftarrow A7+d16$
UNLK	An	2	12(3/0)						$A7 \leftarrow An$, $An \leftarrow [A7]$, $A7 \leftarrow A7+4$
MOVEM.W	jadr,reglist	4,6,8	8+4n(1+n/0)+		X	X	0	0	$reg1<0-15> \leftarrow [jadr]$, $reg1<16-31> \leftarrow reg1<15>$ $reg2<0-15> \leftarrow [jadr+4]$, $reg2<16-31> \leftarrow reg2<15>$: $regn<0-15> \leftarrow [jadr+2n-2]$, $regn<16-31> \leftarrow regn<15>$[2] Carga palabras almacenadas secuencialmente en memoria en una lista de registros el orden en que se cargan es D0-D7,A0-A7
MOVEM.W	(An)+,reglist	4	8+4n(2+n/0)+						$reg1<0-15> \leftarrow [An]$, $reg1<16-31> \leftarrow reg1<15>$, $An \leftarrow An+2$ $reg2<0-15> \leftarrow [An]$, $reg2<16-31> \leftarrow reg2<15>$, $An \leftarrow An+2$: $regn<0-15> \leftarrow [An]$, $regn<16-31> \leftarrow regn<15>$, $An \leftarrow An+2$[2,3] Igual que la anterior, pero con postincremento
MOVEM.W	reglist,madr	4,6,8	4+5n(1/n)+						$[madr] \leftarrow reg1<0-15>$ $[madr+2] \leftarrow reg2<0-15>$ $[madr+4] \leftarrow reg3<0-15>$: : $[madr+2n-2] \leftarrow$ $regn<0-15>$[2] Almacena palabras de lista de registros en espacio secuencial de memoria, orden de almacenamiento D0-D7,A0-A7

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
MOVEM.W	reglist,-(An)	4	4+5n(1/n)+						$An \leftarrow An-2$, $[An] \leftarrow regn<0-15>$: $An \leftarrow An-2$, $[An] \leftarrow reg3<0-15>$ $An \leftarrow An-2$, $[An] \leftarrow reg2<0-15>$ $An \leftarrow An-2$, $[An] \leftarrow reg1<0-15>$ [2,3] Almacena palabras de lista de registros en espacio secuencial de memoria el orden de almacenamiento A7-A0,D7-D0
MOVEM.L	jadr,reglist	4,6,8	8+8n						
	(An)+,reglist	4	(2+2n/0)						
	reglist,madr	4,6,8	4+10n						
	reglist,-(An)	4	(1+n)						Igual que MOVEM.W, excepto que ahora se mueven los 32 bits de registros [2,4]
MOVEP.W	d16(An),Dn	4	16(4/0)						$Dn<8-15> \leftarrow [An+d16]$, $An \leftarrow An+2$, $Dn<0-7> \leftarrow [An+d16]$[3] Carga datos de periféricos de direcciones alternas. La dirección es de byte y lee bytes
MOVEP.W	Dn,d16(An)	4	18(2/2)						$[An+d16] \leftarrow Dn<8-15>$, $An \leftarrow An+2$, $[An+d16] \leftarrow Dn<0-7>$[3] Almacena los bytes de registros en direcciones alternas de periféricos. La dirección es de byte y escribe bytes
MOVEP.L	d16(An),Dn	4	24(6/0)						$Dn<24-31> \leftarrow [An+d16]$, $An \leftarrow An+2$ $Dn<16-23> \leftarrow [An+d16]$, $An \leftarrow An+2$ $Dn<8-15> \leftarrow [An+d16]$, $An \leftarrow An+2$ $Dn<0-7> \leftarrow [An+d16]$[3] Carga datos de periféricos de direcciones alternas. La dirección es de byte y lee bytes

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
MOVEP.L	Dn,d16(An)	4	28(2/4)						$[An+d16] \leftarrow Dn<24-31>$, $An \leftarrow An+2$, $[An+d16] \leftarrow Dn<16-23>$, $An \leftarrow An+2$ $[An+d16] \leftarrow Dn<8-15>$, $An \leftarrow An+2$ $[An+d16] \leftarrow Dn<0-7>$ Almacena los bytes de registros en direcciones alternas de periféricos. La dirección es de byte y escribe bytes
MOVEQ	data8,Dn	2	4(1/0)		X	X	0	0	$Dn<0-7> \leftarrow data8$, $Dn<8-31> \leftarrow Dn<7>$
MOVE.B	data8,Dn	4	8(2/0)		X	X	0	0	$Dn<0-7> \leftarrow data8$
MOVE.B	data8,dadr	4,6,8	9(1/1)+		X	X	0	0	$[dadr] \leftarrow data8$
MOVE.W	data16,Dn	4	8(2/0)		X	X	0	0	$Dn<0-15> \leftarrow data16$
LEA	jadr,An	2,4,6	2(0/0)+						$An \leftarrow jadr^2$ Carga una dirección efectiva en un registro de dirección
PEA	jadr	2,4,6	10(1/2)+						$A7 \leftarrow A7-4$, $[A7] \leftarrow jadr^4$ Apila dirección efectiva
CLR.B	dadr	2,4,6	9(1/1)+		0	1	0	0	$[dadr] \leftarrow 0^1$
CLR.W	dadr	2,4,6	9(1/1)+		0	1	0	0	$[dadr] \leftarrow 0^{2,3}$
CLR.L	dadr	2,4,6	14(1/2)+		0	1	0	0	$[dadr] \leftarrow 0^{2,4}$
CLR.B	Dn	2	4(1/0)+		0	1	0	0	$Dn<0-7> \leftarrow 0^1$
CLR.W	Dn	2	4(1/0)+		0	1	0	0	$Dn<0-15> \leftarrow 0^{2,3}$
CLR.L	Dn	2	6(1/0)+		0	1	0	0	$Dn<0-31> \leftarrow 0^{2,4}$
SWAP	Dn	2	4(1/0)		X	X	0	0	$Dn<0-7> \leftarrow \rightarrow Dn<8-15>$
Scc	dadr	2,4,6	9(1/1)+						$[dadr] \leftarrow todo1s$ si cc = TRUE $[dadr] \leftarrow$ todo 0s si cc = FALSE[1]
Scc	Dn	2	4(1/0)						$Dn<0-7> \leftarrow todo1s$ si cc = TRUE $Dn<0-7> \leftarrow todo0s$ si cc = FALSE
TAS	dadr	2,4,6	11(1/1)+		X	X	0	0	$[dadr<7>] \leftarrow 1^1$ Instrucción para multiprocesador
TAS	Dn	2	4(1/0)		X	X	0	0	$Dn<7> \leftarrow 1$

1.6.3. Aritméticas

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
ADD.B	sadr,Dn	2,4,6	4(1/0)+	X	X	X	X	X	$Dn<0-7>\leftarrow Dn<0-7>+[sadr]^{1}$
ADD.B	Dn,dadr	2,4,6	9(1/1)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]+Dn<0-7>^{1}$
ADD.W	sadr,Dn	2,4,6	4(1/0)+	X	X	X	X	X	$Dn<0-15>\leftarrow Dn<0-15>+[sadr]^{2,3}$
ADD.W	sadr,An	2,4,6	8(1/0)+						$An<0-31>\leftarrow An<0-31>+[sadr](expande\ signo)^{2,3}$
ADD.W	Dn,dadr	2,4,6	9(1/1)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]+Dn<0-15>^{2,3}$
ADD.L	sadr,Dn	2,4,6	6(1/0)+	X	X	X	X	X	$Dn<0-31>\leftarrow Dn<0-31>+[sadr]^{2,4}$
ADD.L	sadr,An	2,4,6	6(1/0)+						$An<0-31>\leftarrow An<0-31>+[sadr]^{2,4}$
ADD.L	Dn,dadr	2,4,6	14(1/2)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]+Dn<0-31>^{2,4}$
ADD.B	data8,Dn	4	8(2/0)	X	X	X	X	X	$Dn<0-7>\leftarrow Dn<0-7>+data8$
ADD.B	data8,dadr	4,6,8	13(2/1)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]+data8^{1}$
ADD.W	data16,Dn	4	8(2/0)	X	X	X	X	X	$Dn<0-15>\leftarrow Dn<0-15>+data16$
ADDA.W	data16,An	4	8(2/0)+						$An<0-31>\leftarrow An<0-31>+data16(expande\ signo)$
ADD.W	data16,dadr	4,6,8	13(2/1)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]+data16^{2,3}$
ADD.L	data32,Dn	6	16(3/0)	X	X	X	X	X	$Dn<0-31>\leftarrow Dn<0-31>+data32$
ADDA.L	data32,An	6	16(3/0)						$An<0-31>\leftarrow An<0-31>+data32$
ADD.L	data32,dadr	6,8,10	22(3/2)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]+data32^{2,4}$
ADD.B	sDn,dDn	2	4(1/0)	X	X	X	X	X	$dDn<0-7>\leftarrow dDn<0-7>+sDn<0-7>$
ADD.W	rs,Dn	2	4(1/0)	X	X	X	X	X	$Dn<0-15>\leftarrow [Dn<0-15>+rs<0-15>$
ADD.W	rs,An	2	8(1/0)+						$An<0-31>\leftarrow An<0-31>+rs<0-15>(expande\ signo)$
ADD.L	rs,Dn	2	8(1/0)	X	X	X	X	X	$Dn<0-31>\leftarrow Dn<0-31>+rs<0-31>$
ADD.L	rs,An	2	8(1/0)						$An<0-31>\leftarrow An<0-31>+rs<0-31>$
ADDQ.B	data3,Dn	2	4(1/0)	X	X	X	X	X	$Dn<0-7>\leftarrow Dn<0-7>+data3$
ADDQ.B	data3,dadr	2,4,6	9(1/0)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]+data3^{1}$
ADDQ.W	data3,Dn	2	4(1/0)	X	X	X	X	X	$Dn<0-15>\leftarrow Dn<0-15>+data3$
ADDQ.W	data3,An	2	4(1/0)+						$An<0-31>\leftarrow An<0-31>+data3$
ADDQ.W	data3,dadr	2,4,6	9(1/1)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]+data3^{2,3}$

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
ADDQ.L	data3,Dn	2	8(1/0)	X	X	X	X	X	Dn<0−31>← Dn<0−31>+$data3$
ADDQ.L	data3,An	2	8(1/0)						An<0−31>← An<0−31>+$data3$
ADDQ.L	data3,dadr	2,4,6	14(1/2)+	X	X	X	X	X	$[dadr]$←$[dadr]$+$data3^{2,4}$
ADDX.B	-(sAn),-(dAn)	2	19(3/1)	X	X	X	X	X	sAn←sAn−1, dAn←dAn−1 $[dAn]$←$[dAn]$+$[sAn]$+X^1
ADDX.W	-(sAn),-(dAn)	2	19(3/1)	X	X	X	X	X	sAn←sAn−2, dAn←dAn−2 $[dAn]$←$[dAn]$+$[sAn]$+$X^{2,3}$
ADDX.L	-(sAn),-(dAn)	2	35(5/2)	X	X	X	X	X	sAn←sAn−4, dAn←dAn−4 $[dAn]$←$[dAn]$+$[sAn]$+$X^{2,4}$
ADDX.B	sDn,dDn	2	4(1/0)	X	X	X	X	X	dDn<0−7>←dDn<0−7> +sDn<0−7>+X
ADDX.W	sDn,dDn	2	4(1/0)	X	X	X	X	X	dDn<0−15>←X+ dDn<0−15>+sDn<0−15>
ADDX.L	sDn,dDn	2	8(1/0)	X	X	X	X	X	dDn<0−31>←X+ dDn<0−31>+sDn<0−31>
ABCD	-(sAn),-(dAn)	2	19(3/1)	X	U	X	U	X	sAn←sAn−1, dAn←dAn−1 $[dAn]$←$[dAn]$+$[sAn]$+X^1 Suma números en BCD empleando bit eXtendido, las dos direcciones son de byte
ABCD	sDn,dDn	2	6(1/0)	X	U	X	U	X	dDn<0−7>←dDn<0−7> +sDn<0−7>+X
NEG.B	dadr	2,4,6	9(1/1)+	X	X	X	X	X	$[dadr]$←0−$[dadr]^1$
NEG.W	dadr	2,4,6	9(1/1)+	X	X	X	X	X	$[dadr]$←0−$[dadr]^{2,3}$
NEG.L	dadr	2,4,6	14(1/2)+	X	X	X	X	X	$[dadr]$←0−$[dadr]^{2,4}$
NEG.B	Dn	2	4(1/0)	X	X	X	X	X	Dn<0−7>←0−Dn<0−7>
NEG.W	Dn	2	4(1/0)	X	X	X	X	X	Dn<0−15>←0−Dn<0−15>
NEG.L	Dn	2	6(1/0)	X	X	X	X	X	Dn<0−31>←0−Dn<0−31>
NEGX.B	dadr	2,4,6	9(1/1)+	X	X	X	X	X	$[dadr]$←0−$[dadr]$−X^1
NEGX.W	dadr	2,4,6	9(1/1)+	X	X	X	X	X	$[dadr]$←0−$[dadr]$−$X^{2,3}$
NEGX.L	dadr	2,4,6	14(1/2)+	X	X	X	X	X	$[dadr]$←0−$[dadr]$−$X^{2,4}$
NEGX.B	Dn	2	4(1/0)	X	X	X	X	X	Dn<0−7>← 0−Dn<0−7>−X
NEGX.W	Dn	2	4(1/0)	X	X	X	X	X	Dn<0−15>← 0−Dn<0−15>−X
NEGX.L	Dn	2	6(1/0)	X	X	X	X	X	Dn<0−31>← 0−Dn<0−31>−X
NBCD	dadr	2,4,6	9(1/1)+	X	U	X	U	X	$[dadr]$←0−$[dadr]$−X^1 Complementa a 10 en BCD si X = 0 o a 9 si si X = 1
NBCD	Dn	2	6(1/0)	X	U	X	U	X	Dn<0−7>←[D<0−7>]−X

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
SUB.B	sadr,Dn	2,4,6	4(1/0)+	X	X	X	X	X	$Dn<0-7>\leftarrow Dn<0-7>-[sadr]^{1}$
SUB.B	Dn,dadr	2,4,6	9(1/1)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]-Dn<0-7>^{1}$
SUB.W	sadr,Dn	2,4,6	4(1/0)+	X	X	X	X	X	$Dn<0-15>\leftarrow [Dn<0-15>-[sadr]^{2,3}$
SUB.W	sadr,An	2,4,6	8(1/0)+	X	X	X	X	X	$An<0-31>\leftarrow An<0-31> -[sadr](expande\ signo)^{2,3}$
SUB.W	Dn,dadr	2,4,6	9(1/1)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]-Dn<0-15>^{2,3}$
SUB.L	sadr,Dn	2,4,6	6(1/0)+	X	X	X	X	X	$Dn<0-31>\leftarrow Dn<0-31>-[sadr]^{2,4}$
SUB.L	sadr,An	2,4,6	6(1/0)+	X	X	X	X	X	$An<0-31>\leftarrow An<0-31>-[sadr]^{2,4}$
SUB.L	Dn,dadr	2,4,6	14(1/2)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]-Dn<0-31>^{2,4}$
SUB.B	data8,Dn	4	8(2/0)	X	X	X	X	X	$Dn<0-7>\leftarrow Dn<0-7>-data8$
SUB.B	data8,dadr	4,6,8	13(2/1)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]-data8^{1}$
SUB.W	data16,Dn	4	8(2/0)	X	X	X	X	X	$Dn<0-15>\leftarrow [Dn<0-15>-data16$
SUBA.W	data16,An	4	8(2/0)+						$An<0-31>\leftarrow An<0-31> -data16(expande\ signo)$
SUB.W	data16,dadr	4,6,8	13(2/1)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]-data16^{2,3}$
SUB.L	data32,Dn	6	16(3/0)	X	X	X	X	X	$Dn<0-31>\leftarrow [Dn<0-31>-data32$
SUBA.L	data32,An	6	16(3/0)						$An<0-31>\leftarrow An<0-31>-data32$
SUB.L	data32,dadr	6,8,10	22(3/2)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]-data32^{2,4}$
SUB.B	sDn,dDn	2	4(1/0)	X	X	X	X	X	$dDn<0-7>\leftarrow dDn<0-7>-sDn<0-7>$
SUB.W	rs,Dn	2	4(1/0)	X	X	X	X	X	$Dn<0-15>\leftarrow Dn<0-15>-rs<0-15>$
SUB.W	rs,An	2	8(1/0)+						$An<0-15>\leftarrow An<0-15>- rs<0-15>(expande\ signo)$
SUB.L	rs,Dn	2	8(1/0)	X	X	X	X	X	$Dn<0-31>\leftarrow Dn<0-31>-rs<0-31>$
SUB.L	rs,An	2	8(1/0)						$An<0-31>\leftarrow An<0-31>-rs<0-31>$
SUBQ.B	data3,Dn	2	4(1/0)	X	X	X	X	X	$Dn<0-7>\leftarrow Dn<0-7>-data3$
SUBQ.B	data3,dadr	2,4,6	9(1/0)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]-data3^{1}$
SUBQ.W	data3,Dn	2	4(1/0)	X	X	X	X	X	$Dn<0-15>\leftarrow Dn<0-15>-data3$
SUBQ.W	data3,An	2	4(1/0)+						$An<0-15>\leftarrow An<0-15>-data3$
SUBQ.W	data3,dadr	2,4,6	9(1/1)+	X	X	X	X	X	$[dadr]\leftarrow [dadr]-data3^{2,3}$

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
SUBQ.L	data3,Dn	2	8(1/0)	X	X	X	X	X	$Dn<0-31>\leftarrow Dn<0-31>-data3$
SUBQ.L	data3,An	2	8(1/0)						$An<0-31>\leftarrow An<0-31>-data3$
SUBQ.L	data3,dadr	2,4,6	14(1/2)+	X	X	X	X	X	$[dadr]\leftarrow[dadr]-data3^{2,4}$
SUBX.B	-(sAn),-(dAn)	2	19(3/1)	X	X	X	X	X	$sAn\leftarrow sAn-1$, $dAn\leftarrow dAn-1$ $[dAn]\leftarrow[dAn]-[sAn]+X^{1}$
SUBX.W	-(sAn),-(dAn)	2	19(3/1)	X	X	X	X	X	$sAn\leftarrow sAn-2$, $dAn\leftarrow dAn-2$ $sAn\leftarrow sAn-2$ $[dAn]\leftarrow[dAn]-[sAn]+X^{2,3}$
SUBX.L	-(sAn),-(dAn)	2	35(5/2)	X	X	X	X	X	$sAn\leftarrow sAn-4$, $dAn\leftarrow dAn-4$ $[dAn]\leftarrow[dAn]-[sAn]+X^{2,4}$
SUBX.B	sDn,dDn	2	4(1/0)	X	X	X	X	X	$dDn<0-7>\leftarrow dDn<0-7>-sDn<0-7>+X$
SUBX.W	sDn,dDn	2	4(1/0)	X	X	X	X	X	$dDn<0-15>\leftarrow X+dDn<0-15>-sDn<0-15>$
SUBX.L	sDn,dDn	2	8(1/0)	X	X	X	X	X	$dDn<0-31>\leftarrow X+dDn<0-31>-sDn<0-31>$
SBCD	-(sAn),-(dAn)	2	10(3/1)	X	U	X	U	X	$sAn\leftarrow sAn-1$, $dAn\leftarrow dAn-1$ $[dAn]\leftarrow[dAn]-[sAn]-X^{1}$ Resta en BCD de byte de memoria
SBCD	sDn,dDn	2	6(1/0)	X	U	X	U	X	$dDn<0-7>\leftarrow dDn<0-7>-sDn<0-7>-X$ Resta en BCD con bit de extendido
MULS	asdr,Dn	2,4,6	<70(1/0)+		X	X	0	0	$Dn<0-31>\leftarrow Dn<0-15>\times[sadr]^{2,3}$ Multiplica dos números 16 bits con signo y genera uno de 32
MULS	data16,Dn	4	≤70(1/0)+		X	X	0	0	$Dn<0-31>\leftarrow Dn<0-15>\times data16$ Multiplica dos números 16 bits con signo y genera uno de 32
MULS	sDn,dDn	2	≤70(1/0)		X	X	0	0	$dDn<0-31>\leftarrow dDn<0-15>\times sDn<0-15>$ Multiplica dos números 16 bits con signo y genera uno de 32

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado					Operación realizada
				X	N	Z	V	C	
MULU	asdr,Dn	2,4,6	<74(2/0)+		X	X	0	0	Dn<0−31>← Dn<0−15>×[$sadr$]^{2,3} Multiplica dos números 16 bits sin signo y genera uno de 32
MULU	data16,Dn	4	≤74(2/0)+		X	X	0	0	Dn<0−31>← [Dn<0−15>×$data$16 Multiplica dos números 16 bits sin signo y genera uno de 32
MULU	sDn,dDn	4	≤70(1/0)		X	X	0	0	dDn<0−31>← dDn<0−15>×sDn<0−15> Multiplica dos números 16 bits sin signo y genera uno de 32
DIVS	asdr,Dn	2,4,6	<158(1/0)+		X	X	0	0	Dn<0−15>← Dn<0−31>÷[$sadr$], Dn<16−31>←$resto^{2,3}$ Divide números con signo y genera cociente y resto en 16 bits
DIVS	data16,Dn	4	≤162(2/0)+		X	X	0	0	Dn<0−15>← Dn<0−31>÷$data$16, Dn<16−31>←$resto$ Divide números con signo y genera cociente y resto en 16 bits
DIVS	sDn,dDn	2	≤158(1/0)		X	X	0	0	dDn<0−15>← dDn<0−31>÷sDn<0−15>, dDn<16−31>←$resto$ Divide números con signo y genera cociente y resto en 16 bits
DIVU	asdr,Dn	2,4,6	≤140(1/0)+		X	X	0	0	Dn<0−15>← Dn<0−31>÷[$sadr$], Dn<16−31>←$resto^{2,3}$ Divide dos números sin signo y genera cociente y resto en 16 bits
DIVU	data16,Dn	4	≤74(2/0)+		X	X	0	0	Dn<0−15>← Dn<0−31>÷$data$16, Dn<16−31>←$resto$ Divide dos números sin signo y genera cociente y resto en 16 bits

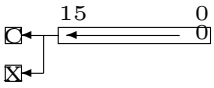
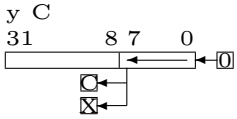
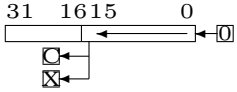
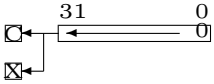
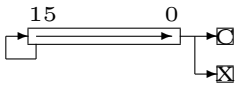
Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado					Operación realizada
				X	N	Z	V	C	
DIVU	sDn,dDn	4	≤74(2/0)+		X	X	0	0	$dDn<0-15> \leftarrow dDn<0-31> \div sDn<0-15>$, $dDn<16-31> \leftarrow resto$ Divide dos números sin signo y genera cociente y resto en 16 bits
EXT.W	Dn	2	4(1/0)		X	X	0	0	$Dn<8-15> \leftarrow Dn<7>$ Extiende signo
EXT.L	Dn	2	4(1/0)		X	X	0	0	$Dn<16-31> \leftarrow Dn<15>$ Extiende signo
CMP.B	sadr,Dn	2,4,6	4(1/0)+		X	X	X	X	$Dn<0-7> - [sadr]^{1}$
CMP.W	sadr,Dn	2,4,6	4(1/0)+		X	X	X	X	$Dn<0-15> - [sadr]^{2,3}$
CMP.W	sadr,An	2,4,6	6(1/0)+		X	X	X	X	$An<0-15> - [sadr]^{2,3}$
CMP.L	sadr,Dn	2,4,6	6(1/0)+		X	X	X	X	$Dn<0-31> - [sadr]^{2,4}$
CMP.L	sadr,An	2,4,6	6(1/0)+		X	X	X	X	$An<0-31> - [sadr]^{2,4}$
CMPM.B	(sAn)+,(dAn)+	2	12(3/0)		X	X	X	X	$[dAn] - [sAn]$ $dAn \leftarrow dAn+1$, $sAn \leftarrow sAn+1^{1}$
CMPM.W	(sAn)+,(dAn)+	2	12(3/0)		X	X	X	X	$[dAn] - [sAn]$ $dAn \leftarrow dAn+2$, $sAn \leftarrow sAn+2^{2,3}$
CMPM.L	(sAn)+,(dAn)+	2	20(5/0)		X	X	X	X	$[dAn] - [sAn]$ $dAn \leftarrow dAn+4$, $sAn \leftarrow sAn+4^{2,4}$
CMP.B	data8,Dn	4	8(2/0)		X	X	X	X	$Dn<0-7> - data8$
CMP.B	data8,dadr	4,6,8	8(2/0)+		X	X	X	X	$[dadr] - data8^{1}$
CMP.W	data16,Dn	4	8(2/0)		X	X	X	X	$Dn<0-15> - data16$
CMP.W	data16,An	4	8(2/0)		X	X	X	X	$An<0-15> - data16$
CMP.W	data16,dadr	4,6,8	8(2/0)		X	X	X	X	$[dadr] - data16^{2,3}$
CMP.L	data32,Dn	4	14(3/0)		X	X	X	X	$Dn<0-31> - data16$
CMP.L	data32,An	4	14(3/0)		X	X	X	X	$An<0-31> - data16$
CMP.L	data32,dadr	4,6,8	12(3/0)		X	X	X	X	$[dadr] - data16^{2,4}$
CMP.B	sDn,dDn	2	4(1/0)		X	X	X	X	$dDn<0-7> - sDn<0-7>$
CMP.W	rs,Dn	2	4(1/0)		X	X	X	X	$Dn<0-15> - rs<0-15>$
CMP.W	rs,An	2	6(1/0)		X	X	X	X	$An<0-15> - rs<0-15>$
CMP.L	rs,Dn	2	6(1/0)		X	X	X	X	$Dn<0-31> - rs<0-31>$
CMP.L	rs,An	2	6(1/0)		X	X	X	X	$An<0-31> - rs<0-31>$
TST.B	dadr	2,4,6	4(1/0)+		X	X	0	0	$[dadr] - 0^{1}$
TST.W	dadr	2,4,6	4(1/0)+		X	X	0	0	$[dadr] - 0^{2}$
TST.L	dadr	2,4,6	4(1/0)+		X	X	0	0	$[dadr] - 0^{2}$
TST.B	Dn	2	4(1/0)		X	X	0	0	$Dn<0-7> - 0$
TST.W	Dn	2	4(1/0)		X	X	0	0	$Dn<0-15> - 0$
TST.L	Dn	2	4(1/0)		X	X	0	0	$Dn<0-31> - 0$

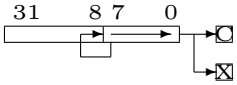
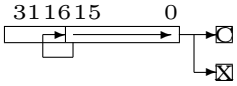
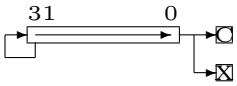
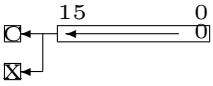
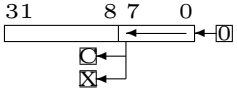
1.6.4. Lógicas

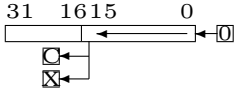
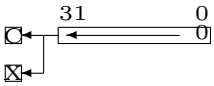
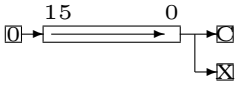
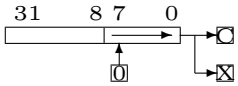
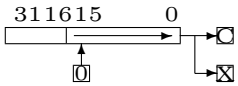
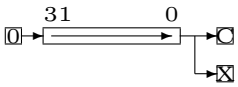
Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
NOT.B	dadr	2,4,6	9(1/1)+		X	X	0	0	$[dadr] \leftarrow [\overline{dadr}]^{1}$
NOT.W	dadr	2,4,6	9(1/1)+		X	X	0	0	$[dadr] \leftarrow [\overline{dadr}]^{2,3}$
NOT.L	dadr	2,4,6	14(1/2)+		X	X	0	0	$[dadr] \leftarrow [\overline{dadr}]^{2,4}$
NOT.B	Dn	2	4(1/0)+		X	X	0	0	$Dn<0-7> \leftarrow \overline{Dn<0-7>}$
NOT.B	Dn	2	6(1/0)+		X	X	0	0	$Dn<0-15> \leftarrow \overline{Dn<0-15>}$
NOT.L	Dn	2	6(1/0)+		X	X	0	0	$Dn<0-31> \leftarrow \overline{Dn<0-31>}$
AND.B	sadr,Dn	2,4,6	4(1/0)+		X	X	0	0	$Dn<0-7> \leftarrow Dn<0-7> \wedge [sadr]^{1}$
AND.B	Dn,dadr	2,4,6	9(1/1)+		X	X	0	0	$[dadr] \leftarrow [dadr] \wedge Dn<0-7>^{1}$
AND.W	sadr,Dn	2,4,6	4(1/0)+		X	X	0	0	$Dn<0-15> \leftarrow Dn<0-15> \wedge [sadr]^{2,3}$
AND.W	Dn,dadr	2,4,6	9(1/1)+		X	X	0	0	$[dadr] \leftarrow [dadr] \wedge Dn<0-15>^{2,3}$
AND.L	sadr,Dn	2,4,6	6(1/0)+		X	X	0	0	$Dn<0-31> \leftarrow Dn<0-31> \wedge [sadr]^{2,4}$
AND.L	Dn,dadr	2,4,6	14(1/2)+		X	X	0	0	$[dadr] \leftarrow [dadr] \wedge Dn<0-31>^{2,4}$
AND.B	data8,Dn	4	8(2/0)		X	X	0	0	$Dn<0-7> \leftarrow Dn<0-7> \wedge data8$
AND.B	data8,dadr	4,6,8	13(2/1)+		X	X	0	0	$[dadr] \leftarrow [dadr] \wedge data8^{1}$
AND.W	data16,Dn	4	8(2/0)		X	X	0	0	$Dn<0-15> \leftarrow Dn<0-15> \wedge data16$
AND.W	data16,dadr	4,6,8	13(2/1)+		X	X	0	0	$[dadr] \leftarrow [dadr] \wedge data16^{2,3}$
AND.L	data32,Dn	6	16(3/0)		X	X	0	0	$Dn<0-31> \leftarrow Dn<0-31> \wedge data32$
AND.L	data32,dadr	6,8,10	22(3/2)+		X	X	0	0	$[dadr] \leftarrow [dadr] \wedge data32^{2,4}$
AND.B	sDn,dDn	2	4(1/0)		X	X	0	0	$dDn<0-7> \leftarrow dDn<0-7> \wedge sDn<0-7>$
AND.W	sDn,dDn	2	4(1/0)		X	X	0	0	$Dn<0-15> \leftarrow dDn<0-15> \wedge sDn<0-15>$
AND.L	sDn,dDn	2	8(1/0)		X	X	0	0	$Dn<0-31> \leftarrow dDn<0-31> \wedge sDn<0-31>$
OR.B	sadr,Dn	2,4,6	4(1/0)+		X	X	0	0	$Dn<0-7> \leftarrow Dn<0-7> \vee [sadr]^{1}$
OR.B	Dn,dadr	2,4,6	9(1/1)+		X	X	0	0	$[dadr] \leftarrow [dadr] \vee Dn<0-7>^{1}$
OR.W	sadr,Dn	2,4,6	4(1/0)+		X	X	0	0	$Dn<0-15> \leftarrow Dn<0-15> \vee [sadr]^{2,3}$
OR.W	Dn,dadr	2,4,6	9(1/1)+		X	X	0	0	$[dadr] \leftarrow [dadr] \vee Dn<0-15>^{2,3}$
OR.L	sadr,Dn	2,4,6	6(1/0)+		X	X	0	0	$Dn<0-31> \leftarrow Dn<0-31> \vee [sadr]^{2,4}$
OR.L	Dn,dadr	2,4,6	14(1/2)+		X	X	0	0	$[dadr] \leftarrow [dadr] \vee Dn<0-31>^{2,4}$

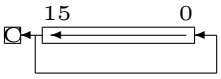
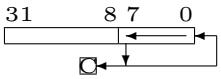
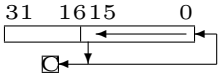
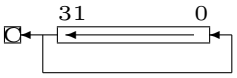
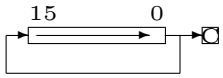
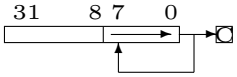
Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
OR.B	data8,Dn	4	8(2/0)		X	X	0	0	$Dn<0-7> \leftarrow Dn<0-7> \vee data8$
OR.B	data8,dadr	4,6,8	13(2/1)+		X	X	0	0	$[dadr] \leftarrow [dadr] \vee data8^{1}$
OR.W	data16,Dn	4	8(2/0)		X	X	0	0	$Dn<0-15> \leftarrow$ $Dn<0-15> \vee data16$
OR.W	data16,dadr	4,6,8	13(2/1)+		X	X	0	0	$[dadr] \leftarrow [dadr] \vee data16^{2,3}$
OR.L	data32,Dn	6	16(3/0)		X	X	0	0	$Dn<0-31> \leftarrow$ $Dn<0-31> \vee data32$
OR.L	data32,dadr	6,8,10	22(3/2)+		X	X	0	0	$[dadr] \leftarrow [dadr] \vee data32^{2,4}$
OR.B	sDn,dDn	2	4(1/0)		X	X	0	0	$dDn<0-7> \leftarrow$ $dDn<0-7> \vee sDn<0-7>$
OR.W	sDn,dDn	2	4(1/0)		X	X	0	0	$dDn<0-15> \leftarrow$ $dDn<0-15> \vee sDn<0-15>$
OR.L	sDn,dDn	2	8(1/0)		X	X	0	0	$dDn<0-31> \leftarrow$ $dDn<0-31> \vee sDn<0-31>$
EOR.B	sadr,Dn	2,4,6	4(1/0)+		X	X	0	0	$Dn<0-7> \leftarrow Dn<0-7> \oplus [sadr]^{1}$
EOR.B	Dn,dadr	2,4,6	9(1/1)+		X	X	0	0	$[dadr] \leftarrow [dadr] \oplus Dn<0-7>^{1}$
EOR.W	sadr,Dn	2,4,6	4(1/0)+		X	X	0	0	$Dn<0-15> \leftarrow$ $Dn<0-15> \oplus [sadr]^{2,3}$
EOR.W	Dn,dadr	2,4,6	9(1/1)+		X	X	0	0	$[dadr] \leftarrow$ $[dadr] \oplus Dn<0-15>^{2,3}$
EOR.L	sadr,Dn	2,4,6	6(1/0)+		X	X	0	0	$Dn<0-31> \leftarrow$ $Dn<0-31> \oplus [sadr]^{2,4}$
EOR.L	Dn,dadr	2,4,6	14(1/2)+		X	X	0	0	$[dadr] \leftarrow$ $[dadr] \oplus Dn<0-31>^{2,4}$
EOR.B	data8,Dn	4	8(2/0)		X	X	0	0	$Dn<0-7> \leftarrow Dn<0-7> \oplus data8$
EOR.B	data8,dadr	4,6,8	13(2/1)+		X	X	0	0	$[dadr] \leftarrow [dadr] \oplus data8^{1}$
EOR.W	data16,Dn	4	8(2/0)		X	X	0	0	$Dn<0-15> \leftarrow$ $Dn<0-15> \oplus data16$
EOR.W	data16,dadr	4,6,8	13(2/1)+		X	X	0	0	$[dadr] \leftarrow [dadr] \oplus data16^{2,3}$
EOR.L	data32,Dn	6	16(3/0)		X	X	0	0	$Dn<0-31> \leftarrow$ $Dn<0-31> \oplus data32$
EOR.L	data32,dadr	6,8,10	22(3/2)+		X	X	0	0	$[dadr] \leftarrow [dadr] \oplus data32^{2,4}$
EOR.B	sDn,dDn	2	4(1/0)		X	X	0	0	$dDn<0-7> \leftarrow$ $dDn<0-7> \oplus sDn<0-7>$
EOR.W	sDn,dDn	2	4(1/0)		X	X	0	0	$dDn<0-15> \leftarrow$ $dDn<0-15> \oplus sDn<0-15>$
EOR.L	sDn,dDn	2	8(1/0)		X	X	0	0	$dDn<0-31> \leftarrow$ $dDn<0-31> \oplus sDn<0-31>$

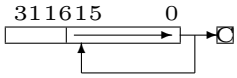
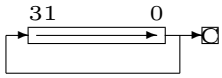
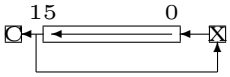
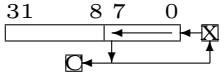
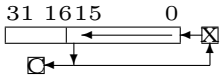
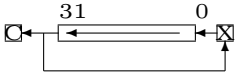
1.6.5. Desplazamientos

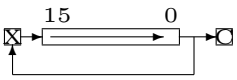
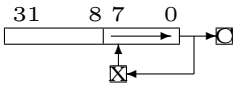
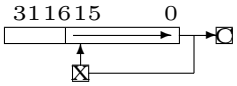
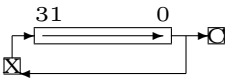
Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación Realizada
ASL	dadr	2,4,6	9(1/1)+	X	X	X	X	X	Desplazamiento aritmético un bit a izquierda, bit 0 se pone a 0. El bit 15 pasa a X y C[2,3] 
ASL.B	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	X	X	Desplazamiento aritmético de varios bit a izquierda, de byte. La cantidad la determina el contador (1-8) o el registro (1-63). Entran 0s al bit 0 el bit 7 pasa a X y C 
ASL.W	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	X	X	Igual que anterior, pero con palabra 
ASL.L	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	X	X	Igual que anterior, pero con doble palabra 
ASR	dadr	2,4,6	9(1/1)+	X	X	X	X	X	Desplazamiento aritmético un bit a derecha. El bit 15 queda como estaba y pasa al 14. El 0 se lleva a X y C 

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
ASR.B	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	X	X	Desplazamiento aritmético de varios bit a derecha de byte. El número de bit lo determina un contador (1-8) o un registro (1-63). El bit 7 conserva su valor y se propaga. El bit 0 pasa a C y X 
ASR.W	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	X	X	Igual que anterior con palabra 
ASR.L	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	X	X	Igual que anterior con doble palabra 
LSL	dadr	2,4,6	9(1/1)+	X	X	X	0	X	Desplazamiento lógico un bit a izquierda. El bit 0 queda a 0 y el 15 pasa a C y X. Es igual que el ASL, pero V siempre vale 0 [2,3] 
LSL.B	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Desplazamiento lógico varios bit a izquierda de byte. El número de bits desplazado lo determina un contador (1-8) o un registro (1-63). Entran 0s por el 0 y el bit 7 pasa a X y C 

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
LSL.W	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Igual que anterior con palabra 
LSL.L	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Igual que anterior con doble palabra 
LSR	dadr	2,4,6	9(1/1)+	X	X	X	0	X	Desplazamiento Lógico un bit a Derecha. El bit 15 pasa a valer 0 y el valor del 0 pasa a X y C 
LSR.B	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Desplazamiento lógico varios bit a derecha de byte. El número de bits lo determina un contador (1-8) o un registro (1-63). Entran 0s al bit 7 y el bit 0 pasa a X y C 
LSR.W	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Igual que anterior, pero con palabra 
LSR.L	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Igual que anterior, pero con doble palabra 

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
ROL	dadr	2,4,6	9(1/1)+		X	X	0	X	Rotación a al Izquierda de un bit. El bit 15 pasa al 0 y a C[2,3] 
ROL.B	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Rotación varios bit a izquierda de byte. El número de bits rotados lo determina un contador (1-8) o un registro (1-63). El bit 7 pasa al bit 0 y a C 
ROL.W	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Igual que anterior con palabra 
ROL.L	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Igual que anterior con doble palabra 
ROR	dadr	2,4,6	9(1/1)+	X	X	X	0	X	Rotación de un bit a derecha. El bit 15 se carga con el bit 0 que pasa también al C 
ROR.B	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Rotación de varios bit a derecha de byte. El número de bits lo determina un contador (1-8) o un registro (1-63). El bit 0 pasa bit 7 y a C 

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
ROR.W	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Igual que anterior, pero con palabra 
ROR.L	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Igual que anterior, pero con doble palabra 
ROXL	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Rotación a izquierda de un bit empleando extensión. El bit 15 pasa al X y a C y el X al 0[2,3] 
ROXL.B	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Rotación varios bit a izquierda de byte empleando bit eXtensión. El número de bits desplazado lo determina un contador (1-8) o un registro (1-63). El bit X pasa al bit 0 y el bit 7 pasa a X y a C 
ROXL.W	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Igual que anterior, pero con palabra 
ROXL.L	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Igual que anterior, pero con doble palabra 

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
ROXR	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Rotación de un bit a derecha empleando bit de extensión. El bit 15 se carga con el bit X y el bit 0 pasa a X y C, y el bit 15 se carga con X  .
ROXR.B	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Rotación de varios bit a derecha de byte empleando bit de eXtensión. El número de bits lo determina un contador (1-8) o un registro (1-63). El bit 0 pasa al bit X y a C, y el bit 7 se carga con X 
ROXR.W	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Igual que anterior, pero con palabra 
ROXR.L	count o Dn,Dn	2	6+2n(1/0)+	X	X	X	0	X	Igual que anterior, pero con doble palabra 

1.6.6. Manejo de bits

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
BTST	bitl,Dn	4	10(2/0)			X			$Z \leftarrow \overline{Dn<bitl>}$
	Dn,dDn	2	6(1/0)			X			$Z \leftarrow \overline{dDn<Dn>}$ Consulta el contenido de un bit de un registro
BTST	bitl,dadr	4,6,8	8(2/0)+			X			$Z \leftarrow [\overline{dadr<bitl>}]$
	Dn,dadr	2,4,6	4(1/0)+			X			$Z \leftarrow [\overline{dadr<Dn>}]$[1] Consulta el contenido de un bit de memoria
BSET	bitl,Dn	4	12(2/0)			X			$Z \leftarrow \overline{Dn<bitl>}$, $Dn<bitl> \leftarrow 1$
	Dn,dDn	2	8(1/0)			X			$Z \leftarrow \overline{dDn<Dn>}$, $dDn<Dn> \leftarrow 1$ Consulta el contenido de un determinado bit de un registro y lo pone a 1
BSET	bitl,dadr	4,6,8	13(2/1)+			X			$Z \leftarrow [\overline{dadr<bitl>}]$, $[dadr<bitl>] \leftarrow 1$
	Dn,dadr	2,4,6	9(1/1)+			X			$Z \leftarrow [\overline{dadr<Dn>}]$, $[dadr<Dn>] \leftarrow 1$[1] Consulta el contenido de un bit de memoria y lo pone a 1
BCLR	bitl,Dn	4	14(2/0)			X			$Z \leftarrow \overline{Dn<bitl>}$, $Dn<bitl> \leftarrow 0$
	Dn,dDn	2	8(1/0)			X			$Z \leftarrow \overline{dDn<Dn>}$, $dDn<Dn> \leftarrow 0$ Consulta el contenido de un determinado bit de un registro y lo pone a 0
BCLR	bitl,dadr	4,6,8	13(2/1)+			X			$Z \leftarrow [\overline{dadr<bitl>}]$, $[dadr<bitl>] \leftarrow 0$
	Dn,dadr	2,4,6	9(1/1)+			X			$Z \leftarrow [\overline{dadr<Dn>}]$, $[dadr<Dn>] \leftarrow 0$[1] Consulta el contenido de un bit de memoria y lo pone a 0

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
BCHG	bitl,Dn	4	12(2/0)			X			$Z \leftarrow \overline{Dn<bitl>}$, $Dn<bitl> \leftarrow \overline{Dn<bitl>}$
	Dn,dDn	2	8(1/0)			X			$Z \leftarrow \overline{dDn<Dn>}$, $dDn<Dn> \leftarrow \overline{dDn<Dn>}$ Consulta el contenido de un bit de un registro y lo pone al valor contrario
BCHG	bitl,dadr	4,6,8	13(2/1)+			X			$Z \leftarrow [\overline{dadr<bitl>}]$, $[dadr<bitl>] \leftarrow [\overline{dadr<bitl>}]$
	Dn,dadr	2,4,6	9(1/1)+			X			$Z \leftarrow [\overline{dadr<Dn>}]$, $[dadr<Dn>] \leftarrow [\overline{dadr<Dn>}]$[1] Consulta el contenido de un bit de memoria y lo pone al valor contrario

1.6.7. Control del programa

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
BRA	label	2,4	10(2/0)						$PC \leftarrow label$ Salto incondicional corto relativo a PC
JMP	jadr	2,4,6	4(1/0)+						$PC \leftarrow jadr$ Salto incondicional absoluto
BSR	label	2,4	10,8(1/0)						$A7 \leftarrow A7-4$, $[A7] \leftarrow PC$
			10,12(2/0)						$PC \leftarrow label$ Salto subrutina corto relativo a PC
JSR	jadr	2,4,6	14(1/2)+						$A7 \leftarrow A7-4$, $[A7] \leftarrow PC$ $PC \leftarrow jadr$ Salto subrutina absoluto
RTS		2	16(4/0)						$PC \leftarrow [A7]$, $A7 \leftarrow A7+4$ Retorno subrutina
RTR		2	20(5/0)						$SR<0-4> \leftarrow [A7<0-4>]$, $A7 \leftarrow A7+2$ $PC \leftarrow [A7]$, $A7 \leftarrow A7+4$ Restaura estado y retorna subrutina

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
RTE		2	20(5/0)	X	X	X	X	X	$SR \leftarrow [A7]$, $A7 \leftarrow A7+2$ $PC \leftarrow [A7]$, $A7 \leftarrow A7+4$ Retorno de excepción
Bcc	label	2,4	10,8(1/0) 10,12(2/0)						$PC \leftarrow label$, Si condición Salto condicional corto relativo a PC
DBcc	Dn,label	4	12,10(2/0), 14(3/0)						Si cumple condición nada. No cumple condición: $Dn<0-15> \leftarrow Dn<0-15>-1$ Si $Dn<0-15>=-1$ siguiente instrucción $PC \leftarrow label$ Salto condicional corto relativo a PC, con decremento
CHK	data16,Dn	4	49(6/3), 12(2/0)		X	U	U	U	Si $Dn<0-15><0 \vee Dn<0-15>>data16$ entonces $PC \leftarrow$ vector CHK
	Dn,dDn	2	45(5/3) 8(1/0)		X	U	U	U	Si $dDn<0-15><0 \vee dDn<0-15>>Dn$ entonces $PC \leftarrow$ vector CHK
	sadr,dDn	2,4,6	45(5/3) 8(1/0)		X	U	U	U	Si $dDn<0-15><0 \vee dDn<0-15>>[sadr]$ entonces $PC \leftarrow$ vector CHK Comprobación de rangos
TRAP	vector	2	37(4/3)						$A7 \leftarrow A7-4$, $[A7] \leftarrow PC$, $A7 \leftarrow A7-2$, $[A7] \leftarrow SR$, $PC \leftarrow vector$ Desvío al S.O.
TRAPV		2	37(5/3)						Si V = 1 TRAP vector TRAPV Comprobación bit overflow

1.6.8. Control del procesador

Nemónico	Operandos	Bytes	Ciclos reloj	Registro estado X	N	Z	V	C	Operación realizada
MOVE	Dn,CCR	2	12(2/0)	X	X	X	X	X	$SR\langle 0{-}4\rangle \leftarrow Dn\langle 0{-}4\rangle$ Actualiza estado
MOVE	sadr,CCR	2,4,6	12(2/0)+	X	X	X	X	X	$SR\langle 0{-}4\rangle \leftarrow [sadr\langle 0{-}4\rangle]^{2,3}$
MOVE	data8,CCR	4	16(3/0)	X	X	X	X	X	$SR\langle 0{-}4\rangle \leftarrow data8\langle 0{-}4\rangle$
MOVE	Dn,SR	2	12(2/0)	X	X	X	X	X	$SR \leftarrow Dn\langle 0{-}15\rangle$ Ejecutable en supervisor
MOVE	sadr,SR	2,4,6	12(2/0)+	X	X	X	X	X	$SR \leftarrow [sadr]^{2,3}$ Ejecutable en supervisor
MOVE	data16,SR	4	16(3/0)	X	X	X	X	X	$SR \leftarrow data16\langle 0{-}8\rangle$ Ejecutable en supervisor
MOVE	SR,Dn	2	6(1/0)						$Dn\langle 0{-}15\rangle \leftarrow SR$
MOVE	SR,dadr	2,4,6	9(1/1)+						$[dadr] \leftarrow SR^{2,3}$
ANDI.B	data8,SR	4	20(3/0)	X	X	X	X	X	$SR\langle 0{-}7\rangle \leftarrow SR\langle 0{-}7\rangle \wedge data8$ AND inmediato a SR
ANDI.W	data16,SR	4	20(3/0)	X	X	X	X	X	$SR \leftarrow SR \wedge data16$ Ejecutable en supervisor
EORI.B	data8,SR	4	20(3/0)	X	X	X	X	X	$SR\langle 0{-}7\rangle \leftarrow SR\langle 0{-}7\rangle \oplus data8$ EOR inmediato a SR
EORI.W	data16,SR	4	20(3/0)	X	X	X	X	X	$SR \leftarrow SR \oplus data16$ Ejecutable en supervisor
ORI.B	data8,SR	4	20(3/0)	X	X	X	X	X	$SR\langle 0{-}7\rangle \leftarrow SR\langle 0{-}7\rangle \vee data8$ OR inmediato a SR
ORI.W	data16,SR	4	20(3/0)	X	X	X	X	X	$SR \leftarrow SR \vee data16$ Ejecutable en supervisor
NOP		2	4(1/0)						No hace nada
RESET		2	132(1/0)						Activa RESET. Ejecutable en supervisor
STOP	data16	4	8(2/0)	X	X	X	X	X	$SR \leftarrow data16$ Bloqueado en espera de interrupción. Ejecutable en supervisor

Capítulo 2

Controlador de líneas serie MC68681

El MC68681 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/Transmiter).

El calificativo Universal le viene dado porque admite una gran variedad de formatos de datos y modos de operación. El modo de operación y formato de datos de cada línea se programa independientemente. Además, la MC68681 DUART tiene un mecanismo de interrupciones bastante versátil, ya que puede proporcionar un vector durante el ciclo de reconocimiento de interrupción y puede solicitar interrupción por 8 sucesos distintos enmascarables individualmente.

A pesar de que un dispositivo de estas características puede parecer complejo, las UART's son uno de los controladores de dispositivos más sencillos. Además, hay que tener en cuenta que el dispositivo simulado no realiza ningún tipo de control de errores, ni interbloqueo hardware para el control de flujo de las líneas y no tiene simulado el temporizador programable para la generación de nuevas velocidades de transmisión y recepción.

Por lo tanto, la descripción que se hace en este capítulo es incompleta y se refiere solo a las funciones que realiza la DUART MC68681 simulada que proporciona el simulador BSVC. Así, solo se describen los registros y bits de registros que proporcionan la funcionalidad simulada.

2.1. Transmisión asíncrona de caracteres por una línea serie

En el modo de transmisión asíncrono, el envío de un carácter constituye una transferencia elemental. El carácter se codifica en un formato de codificación de caracteres (habitualmente código ASCII) y se incluye en una trama cuyo formato se muestra en la figura 2.1.

Como se puede observar en la figura 2.1, el estado inactivo de la línea es a 1. Cada trama comienza con un bit de inicio (Start), que naturalmente es un 0. A continuación se transmiten los bits de datos que pueden ser desde 5 hasta 8, en la figura se utilizan 8. A continuación se puede incluir un bit de paridad, par o impar. Este bit, que es opcional, sirve para detectar errores en la transmisión y no está disponible en la MC68681 simulada, por lo que se transmitirá sin paridad. Por último, se incluye un número de bits de parada (Stop) a 1 que sirven para que el receptor se recupere antes del inicio de una nueva trama. El número

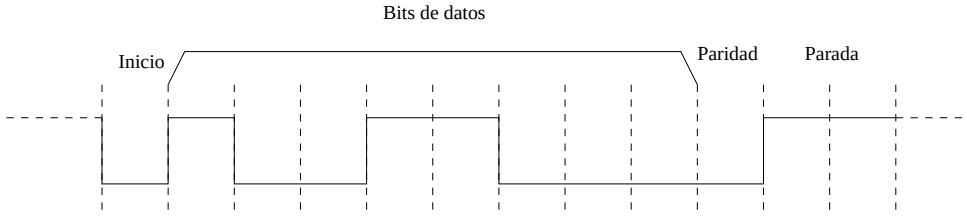


Figura 2.1: Transferencia de un carácter en modo asíncrono con 8 bits de datos

de bits de parada es variable (desde 0,5 a 2) y en la MC68681 simulada no están disponibles ya que no se necesitan al no haber una línea serie física.

El tiempo de transmisión o recepción de una trama depende de la velocidad a la que se utilice la línea y del número de bits que contenga la trama. Así, una trama como la de la figura que se compone de 12 bits y una velocidad de 9600 bits/seg, tarda en transmitirse 1,25 milisegundos pudiéndose alcanzar una velocidad de transferencia máxima de 800 bytes/seg. La velocidad de transmisión se selecciona de acuerdo a la calidad y longitud del cable y a la norma utilizada, por ejemplo RS-232, RS-422, etc.

2.2. Características de la DUART MC68681

El controlador de líneas serie simulado tiene las siguientes características:

- Dos líneas serie *full–duplex* (posibilidad de recepción y transmisión simultánea) independientes con formato de recepción/transmisión asíncrono.
- Cada línea se puede conectar a un proceso Unix.
- 18 velocidades distintas para cada transmisor y receptor, desde 50 hasta 38400 baudios.
- Formato de datos programable desde 5 hasta 8 bits por carácter.
- Modo de canal programable:
  - Normal (*full–duplex*).
  - Eco automático.
- Sistema de interrupciones versátil:
  - Posibilidad de petición de interrupción por 4 condiciones distintas enmascarables individualmente.
  - Proporciona el vector de interrupción almacenado en el registro correspondiente durante el ciclo de reconocimiento de interrupción.

2.3. Descripción de los registros

El modo de operación de la DUART se programa a través de sus registros de control. Asimismo, se pueden leer sus registros de estado para conocer el estado de operación de

la unidad. La tabla 2.1 muestra los registros de la DUART junto con su acrónimo de su denominación en inglés, la dirección que tienen asignada en el computador simulado de la práctica y si son específicos de la línea serie A, de la B o son comunes.

Dirección	Registro en lectura	Registro en escritura	Línea
EFFC01	de modo A (MR1A, MR2A)	de modo A (MR1A, MR2A)	A
EFFC03	de estado A (SRA)	de selección de reloj A (CSRA)	A
EFFC05	Sin acceso	de control A (CRA)	A
EFFC07	buffer de recepción A (RBA)	buffer de transmisión A (TBA)	A
EFFC09	Sin implementar	de control auxiliar (ACR)	Ambas
EFFC0B	de estado de interrupción (ISR)	de máscara de interrupción (IMR)	Ambas
EFFC0D	Sin implementar	Sin implementar	Ambas
EFFC0F	Sin implementar	Sin implementar	Ambas
EFFC11	de modo B (MR1B, MR2B)	de modo B (MR1B, MR2B)	B
EFFC13	de estado B (SRB)	de selección de reloj B (CSRB)	B
EFFC15	Sin acceso	de control B (CRB)	B
EFFC17	buffer de recepción B (RBB)	buffer de transmisión B (TBB)	B
EFFC19	del vector de interrupción (IVR)	del vector de interrupción (IVR)	Ambas
EFFC1B	Sin implementar	Sin implementar	Ambas
EFFC1D	Sin implementar	Sin implementar	Ambas
EFFC1F	Sin implementar	Sin implementar	Ambas

Tabla 2.1: Registros de la DUART MC68681

Todos los registros son de 8 bits y el ancho del bus de datos de esta unidad es también de 8 bits. Por lo tanto, se deberán usar instrucciones de tamaño de operando de 8 bits para leer y escribir en estos registros.

Como es habitual en los controladores de periféricos, en muchos casos 2 registros distintos comparten el mismo puerto, de forma que cuando se escribe se accede a un registro y cuando se lee se accede a otro. Además, los 2 registros de modo de cada línea comparten la misma dirección. De forma que, en el primer acceso se direcciona el registro de modo 1 y en el segundo el registro de modo 2. Todos los accesos posteriores direccionarán el registro de modo 2, a menos que se reinicialice el puntero interno que los discrimina. Este puntero se reinicializa al realizar el RESET del sistema o bien ordenando una reinicialización del puntero en el registro de control correspondiente.

A continuación se describen cada uno de los registros de la DUART MC68681 simulada. En muchos casos aparece la notación N/A que indica que la funcionalidad proporcionada por el bit correspondiente del registro no está disponible en la unidad simulada

2.3.1. Registros de modo 1 (MR1A y MR1B)

N/A	Selección RxIRQ	N/A	N/A	N/A	Bits por carácter
Bit 7	Bit 6	Bit 5	Bit 4 Bit 3	Bit 2	Bit 1 Bit 0
	0 = RxRDY 1 = FFULL				00 = 5 01 = 6 10 = 7 11 = 8

A través de este registro se le ordena a la DUART el número de bits por carácter de la línea correspondiente y cuando ha de solicitar una interrupción en recepción. En este último caso existen dos posibilidades:

RxRDY (*Receiver ReaDY*) Se solicita una interrupción cada vez que llega un carácter.

FFULL (*Fifo FULL*) Cada línea posee 3 registros de recepción organizados como una pila FIFO (*First–Input–First–Output*), lo que permite recibir 3 caracteres consecutivos. Esta opción supone por lo tanto que se solicite la interrupción cuando la pila se llene, es decir cada 3 caracteres.

2.3.2. Registros de modo 2 (MR2A y MR2B)

Modo de la línea	N/A	N/A	N/A
Bit 7 Bit 6	Bit 5	Bit 4	Bit 3 Bit 2 Bit 1 Bit 0
00 = Normal 01 = Eco 10 = N/A 11 = N/A			

A través de este registro se le ordena a la DUART el modo de operación de la línea correspondiente. En el modo eco, la DUART retransmite automáticamente cada carácter que recibe.

2.3.3. Registros de estado (SRA y SRB)

N/A	N/A	N/A	N/A	TxEMT	TXRDY	FFULL	RxRDY
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
				0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes	0 = No 1 = Yes

En este registro se consulta el estado de la línea correspondiente. Los sucesos que registra son:

TxEMT (*Transmitter EMpTy*) Si está habilitada la transmisión por la línea correspondiente, se pone a 1 cuando se puede cargar un nuevo carácter en el buffer de transmisión (TBA o TBB) y el carácter anterior se ha terminado de enviar.

TXRDY (*Transmitter ReaDY*) Si está habilitada la transmisión por la línea correspondiente, se pone a 1 cuando se puede cargar un nuevo carácter en el buffer de transmisión (TBA o TBB), pero puede que aún no se haya completado el envío del carácter anterior.

FFULL (*Fifo FULL*) Hay 3 caracteres en la pila FIFO de registros de recepción.

RxRDY (*Receiver ReaDY*) Se ha recibido un carácter y está en la pila FIFO de registros de recepción.

2.3.4. Registros de selección de reloj (CSRA y CSRB)

Selección de velocidad de recepción			Selección de velocidad de transmisión		
Bit 7 Bit 6 Bit 5 Bit 4			Bit 3 Bit 2 Bit 1 Bit 0		
	Conjunto 1 ACR Bit 7 = 0	Conjunto 2 ACR Bit 7 = 1		Conjunto 1 ACR Bit 7 = 0	Conjunto 2 ACR Bit 7 = 1
0000	50	75	0000	50	75
0001	110	110	0001	110	110
0010	134.5	134.5	0010	134.5	134.5
0011	200	150	0011	200	150
0100	300	300	0100	300	300
0101	600	600	0101	600	600
0110	1200	1200	0110	1200	1200
0111	1050	2000	0111	1050	2000
1000	2400	2400	1000	2400	2400
1001	4800	4800	1001	4800	4800
1010	7200	1800	1010	7200	1800
1011	9600	9600	1011	9600	9600
1100	38400	19200	1100	38400	19200
1101	N/A	N/A	1101	N/A	N/A
1110	N/A	N/A	1110	N/A	N/A
1111	N/A	N/A	1111	N/A	N/A

Mediante este registro en combinación con el bit 7 del registro auxiliar de control (ACR), se elige la velocidad de transmisión y recepción de la línea correspondiente.

2.3.5. Registros de control (CRA y CRB)

N/A	Misceláneos	Transmisión	Recepción
Bit 7	Bit 6 Bit 5 Bit 4	Bit 3 Bit 2	Bit 1 Bit 0
	000 = Sin efecto 001 = Reinicializar puntero a MR1 010 = Reinicializar recepción 011 = Reinicializar transmisión 100 = Reinicializar estado de error 101 = N/A 110 = N/A 111 = N/A	00 = Sin efecto 01 = Habilitar 10 = Inhibir 11 = N/A	00 = Sin efecto 01 = Habilitar 10 = Inhibir 11 = N/A

Mediante este registro de control se habilita o inhibe la transmisión y/o recepción en la línea correspondiente. Además, se puede reinicializar el puntero para poder acceder al registro de modo 1 (MR1A o MR1B), la recepción, la transmisión y los estados de error de la línea correspondiente.

2.3.6. Registros del buffer de recepción (RBA y RBB)

En cada uno de estos puertos hay 3 registros organizados como una pila FIFO, donde se almacenan los caracteres que se reciben por la línea correspondiente. De modo que, si se ha recibido más de un carácter desde la última vez que se leyó, éstos se pueden leer en accesos consecutivos.

2.3.7. Registros del buffer de transmisión (TBA y TBB)

En este registro se escribe el carácter que se va a enviar por la línea correspondiente.

2.3.8. Registro de control auxiliar (ACR)

Selección del conjunto de velocidades	N/A	N/A	N/A	N/A	N/A
Bit 7	Bit 6 Bit 5 Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0 = Conjunto 1 1 = Conjunto 2					

Este registro sirve para seleccionar uno de los dos conjuntos de velocidades de recepción y transmisión. La velocidad elegida dentro del conjunto se especifica a través de los registros de selección de reloj correspondiente (CSRA o CSRB).

2.3.9. Registro de estado de interrupción (ISR)

N/A	N/A	RxRDYB FFULLB	TxRDYB	N/A	N/A	RxRDYA FFULLA	TxRDYA
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
		0 = No 1 = Yes	0 = No 1 = Yes			0 = No 1 = Yes	0 = No 1 = Yes

Este registro proporciona el estado de todas las fuentes posibles de interrupción. Los contenidos de este registro se enmascaran con los contenidos del registro de máscara de interrupción (IMR). De modo que, se activará la petición de interrupción si y solo si existe algún bit de este registro a 1 y el bit correspondiente del registro de máscara de interrupción también está a 1.

Existen dos fuentes posibles de interrupción para cada línea:

RxRDY o FFULL (*Receiver ReaDY o Fifo FULL*) Se colocará a 1 si se ha recibido un carácter o la pila de registros FIFO éste llena, según lo elegido mediante el bit 6 del registro de modo 1 correspondiente (MR1A o MR1B).

TxRDY (*Transmiter ReaDY)* Es un duplicado del bit TxRDY del registro de estado correspondiente (SRA o SRB). Por lo tanto, se pondrá a 1 siempre que el registro buffer de transmisión esté vacío.

2.3.10. Registro de máscara de interrupción (IMR)

N/A	RxRDYB FFULLB	TxRDYB	N/A	RxRDYA FFULLA	TxRDYA
Bit 7 Bit 6	Bit 5	Bit 4	Bit 3 Bit 2	Bit 1	Bit 0
	0 = Inhibida 1 = Habilitada	0 = Inhibida 1 = Habilitada		0 = Inhibida 1 = Habilitada	0 = Inhibida 1 = Habilitada

Como se ha dicho, este registro se usa para habilitar o inhibir la petición de interrupción de cada una de las 4 fuentes posibles. En particular, si alguno de los bit TxRDYA y TxRDYB está a 1 se solicitará una interrupción siempre que se esté en condiciones de transmitir un nuevo carácter. Es decir, tras mandar el último carácter de un paquete se deberá colocar a 0. Si no, hasta que no se cargue un nuevo carácter en el registro del buffer de transmisión correspondiente (TBA o TBB) se solicitarán interrupciones de forma ininterrumpida, valga la redundancia.

2.3.11. Registro del vector de interrupción (IVR)

En este registro se almacenará el vector de interrupción que debe proporcionar el controlador MC68681 durante el ciclo de bus de reconocimiento de interrupción. Por lo tanto, la dirección de la rutina de tratamiento se deberá almacenar a partir de la posición de memoria resultante de multiplicar por 4 el contenido de este registro (véase sección 1.5.3).

Tras un *reset* contiene el vector `0F`, que según la tabla 1.5 es el correspondiente a vector de interrupción no inicializado.

Capítulo 3

Programa ensamblador 68kasm

El programa que se describe es un ensamblador básico de dos pasadas para los microprocesadores MC68000 y MC68010. Este ensamblador traduce el juego completo de instrucciones de ambos procesadores a la vez que proporciona un juego de directivas para el ensamblador o pseudoinstrucciones. La salida que genera se compone de un fichero con el listado de ensamblaje y otro que contiene el código objeto en el formato *S-record* de Motorola.

Paul McKee escribió la versión 1 de este programa en 1986, Mark Hollomon la versión 2.0 y Bradford W. Mott la versión 2.1.

3.1. Llamada al programa ensamblador

La sintaxis para invocar al programa ensamblador 68kasm es la siguiente:

```
68kasm [-clna] <fichero fuente>

Options: -c  Show full constant expansions for DC directives
         -l  Produce listing file (infile.lis)
         -n  Produce NO object file (infile.h68)
         -a  Produce long word absolute addresses only (infile.h68)
```

La sintaxis recomendada es:

```
68kasm -l practica.s
```

Lo que producirá un fichero con el listado de ensamblaje llamado `practica.lis` y, si no ha habido errores, un fichero con el código objeto llamado `practica.h68`. Si no ha habido errores el programa mostrará el mensaje siguiente:

```
68000 Assembler by PGM

No errors detected
No warnings generated
```

Se recomienda utilizar la opción `-l` ya que, por un lado permitirá estudiar los posibles errores en el fichero de listado y, además, permitirá al simulador BSVC mostrar el código de programa en la ventana correspondiente. Si no se utiliza esta opción el simulador no mostrará el código ensamblador del programa y no se podrán colocar puntos de ruptura con una simple pulsación del ratón.

3.2. Formato del código fuente

La entrada al programa ensamblador es un fichero que contiene instrucciones, pseudoinstrucciones y comentarios. Cada línea puede tener hasta un máximo de 256 caracteres y el programa no distingue entre mayúsculas y minúsculas.

Cada línea de código fuente puede tener los siguientes campos:

```
ETIQUETA  OPERACION      OPERANDO,,     COMENTARIO
```

Por ejemplo:

```
LOOP:     MOVE.L         (A0)+,(A1)+    *Ejemplo
```

Los campos se pueden separar por cualquier combinación de blancos y tabuladores. Sin embargo, no se permite el uso de blancos y tabuladores dentro de un campo, a excepción del campo de comentario y en tiras de caracteres entre comillas.

3.2.1. Campo de etiquetas

El campo de etiquetas se distingue, bien porque comienza en la primera columna o bien porque se termina con una coma, que no forma parte de la etiqueta sino que solo la finaliza. Una etiqueta puede tener cualquier longitud, pero **solo se tienen en cuenta los ocho primeros caracteres**.

Se recomienda finalizar las etiquetas con dos puntos para mejorar la legibilidad del código.

3.2.2. Campo de operación

Este campo específica la instrucción que se va a ensamblar o la pseudoinstrucción para el ensamblador. Si la operación lo permite, se puede añadir el sufijo que indica el tamaño de la operación (.B, .W, .L o .S) para indicar byte, palabra, palabra larga o desplazamiento corto. Este campo no puede empezar en la primera columna.

3.2.3. Campo de operandos

Los operandos se separan de la operación por espacios o tabuladores. Si la operación tiene más de un operando, éstos se deben separar por comas. Los operandos no pueden contener espacios a menos que se entrecomillen. Además, no se pueden incluir espacios antes o detrás de la coma que separa dos operandos.

3.2.4. Campo de comentario

Un comentario es cualquier combinación de caracteres que esté en la misma línea tras el campo de operandos. Además, una línea que comience por un `*` se considera como línea de comentario.

Se recomienda comenzar todos los comentarios con un asterisco para mejorar la legibilidad del código.

3.2.5. Símbolos

Los símbolos aparecen en el código como etiquetas, constantes u operandos. El primer carácter de un símbolo debe de ser una letra o un punto. Los restantes pueden ser: letras, `$`, `.` ó `_`. Pueden ser de cualquier longitud **aunque solo se distinguen por sus 8 primeros caracteres**. Se recuerda que el programa ensamblador no distingue entre mayúsculas y minúsculas.

3.2.6. Expresiones

Se pueden utilizar expresiones en lugar de números. Una expresión consiste en uno o más operandos (números o símbolos) y operadores de uno o dos operandos. Los cálculos se realizan con 32 bits y no se obtiene ningún resultado en caso de *overflow*. Aunque, la división por cero produce un error.

Operandos de expresiones

Un operando de una expresión puede ser un símbolo o una constante de cualquiera de los tipos siguientes:

Números decimales Están constituidos por una secuencia de dígitos decimales.

Números hexadecimales Están constituidos por una secuencia de dígitos hexadecimales precedidos por `$`.

Números binarios Están constituidos por una secuencia de dígitos binarios precedidos por `%`.

Números octales Están constituidos por una secuencia de dígitos octales precedidos por `@`.

Constantes ASCII Están constituidas por una tira de caracteres ASCII, hasta un máximo de 4, entrecomilladas por comillas simples (`'`). Si se quiere introducir una comilla simple en la tira hay que colocar dos consecutivas.

Operadores de expresiones

Los operadores se muestran en la tabla 3.1 en orden de precedencia decreciente. Los de igual precedencia se evalúan de izquierda a derecha (a excepción de los del grupo 2).

3.2.7. Especificación de los modos de direccionamiento

La tabla 3.2 detalla como se especifican los modos de direccionamientos en el programa ensamblador.

3.3. Instrucciones de bifurcación

Las instrucciones `Bcc, BRA y BSR` son las únicas a las que se les puede añadir el sufijo `.S`. Este sufijo obliga a ensamblar estas instrucciones con direcciones de salto de 1 byte (rango -128..127). Si se usa y la dirección de destino cae fuera de rango se produce un error. Si se añade el sufijo `.L`, se usa una palabra para la dirección de salto puede estar en el rango

Precedencia	Operador	Descripción
1	`()`	Paréntesis
2	`-`	Negación (complementa a dos)
	`~`	NOT (complementa a uno)
3	`<<`	Desplaza a la izquierda (x<<y significa que x se desplaza y bits y se introducen ceros)
	`>>`	Desplaza a la derecha (x>>y significa que x se desplaza y bits y se introducen ceros)
4	`&`	AND
	`!`	OR
5	`*`	Multiplicación
	`/`	División entera
	`\`	Módulo
6	`+`	Suma
	`-`	Resta

Tabla 3.1: Operadores de expresiones

-32768..32767. Si no se usa ningún sufijo el ensamblador intenta utilizar un salto corto `.S`, pero si la dirección destino no está en el rango correspondiente o no se conoce tras la primera pasada, utilizará un salto largo `.L`.

3.4. Pseudoinstrucciones

3.4.1. Set origin `ORG`

Especifica en qué posiciones de memoria se ubicarán las variables o el código que aparece a continuación. El formato de una línea con esta pseudoinstrucción es:

```
<ETIQUETA>    ORG             <EXPRESION>
```

La expresión no debe contener referencias adelantadas ya que su valor se debe conocer durante la primera pasada en la línea donde `ORG` aparece. La etiqueta es opcional y si se usa, el símbolo adquiere el valor de la dirección `<EXPRESION>`.

3.4.2. Equate `EQU`

Se usa para definir símbolos cuyo valor permanece constante. El formato de una línea con esta pseudoinstrucción es:

```
<ETIQUETA>    EQU             <EXPRESION>
```

La expresión no debe contener referencias adelantadas ya que su valor se debe conocer durante la primera pasada en la línea donde `ORG` aparece. La etiqueta es obligatoria ya que se trata del símbolo que se define y adquiere el valor de `<EXPRESION>`.

Modo	Sintaxis
Directo a registro de datos	Dn
Directo a registro de dirección	An
Absoluto (el tamaño lo elige el ensamblador)	$< ex >$
Relativo a PC	$< ex16 >$(PC)
Relativo a PC con desplazamiento múltiple	$< ex8 >$(PC,Xn.s)
Indirecto a registro	(An)
Indirecto a registro con postincremento	(An)+
Indirecto a registro con predecremento	-(An)
Relativo a registro base	$< ex16 >$(An)
Relativo a registro base con desplazamiento múltiple	$< ex8 >$(An,Xn.s)
Inmediato	$\#< ex >$
Implícito	SR,USP,PC,VBR,SFC,DFC

Leyenda: Dn = Registro de datos
An = Registro de direcciones (se puede usar SP por A7)
Xn = Registro de datos o direcciones usado como índice en los desplazamientos múltiples
.s = Código de tamaño del registro índice (.W o .L, .W es la opción por defecto)
$< ex8 >$ = Expresión que se calcula a 8 bits, si no se pone se asume 0
$< ex16 >$= Expresión que se calcula a 16 bits, si no se pone se asume 0
$< ex >$ = Cualquier expresión
SR = Registro de estado
PC = Registro contador de programa
USP = Registro puntero de pila de usuario
SSP = Registro puntero de pila de supervisor
VBR = Registro base de la tabla de vectores (MC68010)
SFC = Registro fuente de código de función (MC68010)
DFC = Registro destino de código de función (MC68010)

Tabla 3.2: Especificación de los modos de direccionamiento

3.4.3. Define constant `DC`

Esta pseudoinstrucción es equivalente a la DATA del estándar IEEE–694. Se usa para almacenar tiras de caracteres y listas de constantes en memoria. El formato de una línea con esta pseudoinstrucción es:

```
<ETIQUETA>    DC.<TAMAÑO>      <ITEM>,<ITEM>,...
```

Al símbolo de la etiqueta se le asigna la dirección de memoria de comienzo de la lista de datos. El código de `<TAMAÑO>` especifica si cada `<ITEM>` tiene el tamaño de 1 byte (`.B`), 1 palabra (`.W`) o 1 palabra larga (`.L`). La opción por defecto es de palabra.

Ejemplo:

```
TEXTO         DC.B             'DC Ejemplo',$0D,$0A,0
```

Como consecuencia a partir de la dirección de memoria `TEXTO` se almacenará:

```
44 43 20 45 6A 65 6D 70 6C 6F 0D 0A 00
```

3.4.4. Define storage `DS`

Esta pseudoinstrucción es equivalente a la RES del estándar IEEE–694. Se usa para generar un bloque de bytes, palabras o palabras largas sin inicializar. El formato de una línea con esta pseudoinstrucción es:

```
<ETIQUETA>    DS.<TAMAÑO>      <LONGITUD>
```

3.4.5. Set symbol `SET`

Es similar a la anterior, pero se usa para símbolos que se pueden redefinir usando otra pseudoinstrucción `SET` (pero no usando `EQU` o `REG`). El formato de una línea con esta pseudoinstrucción es:

```
<ETIQUETA>    SET              <EXPRESION>
```

3.4.6. Define register set `REG`

Se usa para definir un conjunto de registros que se van a usar en una instrucción `MOVEM`. El formato de una línea con esta pseudoinstrucción es:

```
<ETIQUETA>    REG              <EXPRESION DE REGISTROS>
```

La expresión de registros es de la forma:

```
                               RI/RI/RI...
```

Por ejemplo:

```
CONTEXTO      REG              D0/D3-D7/A1-A5/A6
              MOVEM            CONTEXTO,-(A7)
```

equivale a:

```
              MOVEM            D0/D3-D7/A1-A5/A6,-(A7)
```

3.4.7. Define constant block `DCB`

Se usa para generar un bloque de bytes, palabras o palabras largas que se inicializan al mismo valor. El formato de una línea con esta pseudoinstrucción es:

```
<ETIQUETA>    DCB.<TAMAÑO>     <LONGITUD>,<VALOR>,...
```

3.4.8. INCLUDE

Se usa para incluir otro fichero como parte del fichero fuente ensamblador. Cuando el programa ensamblador encuentra la pseudoinstrucción `INCLUDE`, detiene el ensamblado del fichero y comienza a ensamblar el fichero incluido. De esta forma, el efecto es que se sustituye la pseudoinstruccion por el contenido completo del fichero incluido.

El formato de una línea con esta pseudoinstrucción es:

```
INCLUDE biblioteca.s
```

Hay que tener en cuenta que si una etiqueta está presente en el fichero incluido y en el fichero que contiene la pseudoinstruccion, se generará un error.

3.5. Formato del listado ensamblador

El ensamblador produce un listado del código fuente junto con su código objeto si se le invoca con la opción `-l`. El formato de una línea de este fichero es:

```
0000102E  22D8       200  LOOP  MOVE.L (A0)+,(A1)+    Sample
```

La primera columna muestra la dirección de memoria correspondiente. La segunda columna el código de operación correspondiente a la instrucción. La tercera columna la línea correspondiente del fichero de listado. Las columnas posteriores contienen los campos de la línea correspondiente del código fuente.

En caso de que se presente algún error o aviso, éste se muestra en la línea siguiente. Al final del fichero se imprime el número total de errores y avisos.

3.6. Ejemplos

En el directorio `/usr/local/bsvc/samples/m68000` existen dos programas ejemplo que se distribuyen con el simulador. Los ficheros fuentes se llaman `MatrixMultiply.s` y `example.s`. También se pueden encontrar en ese directorio los listados de ensamblaje correspondientes, cuya extensión es `.lis`.

Capítulo 4

Simulador BSVC

BSVC es una plataforma para la simulación de procesadores, memoria y controladores de periféricos, desarrollada en C++ y Tcl/Tk. BSVC fue desarrollado por Bradford W. Mott en la Universidad del Estado de Carolina del Norte para la realización de prácticas de sistemas de entrada/salida y constituyó su proyecto fin de carrera[1]. Actualmente, BSVC simula el microprocesador MC68000, el controlador de líneas series DUART MC68681 y memoria RAM. Por lo tanto, permite construir computadores virtuales con procesador, memoria y unidades periféricas.

BSVC se compila y ejecuta sobre sistemas operativos Unix, con compilador de C++ gcc 2.7.2 e interfaz gráfico Tcl 7.5 y Tk 4.1. Existe una versión para Windows95, pero no incorpora el simulador de la DUART MC68681.

En este capítulo se detalla la funcionalidad de BSVC necesaria para la realización de la práctica. El manual completo de este simulador se encuentra en el directorio `/usr/local/bsvc/doc/Manual/html/` en formato hipertexto HTML o en `/usr/local/bsvc/doc/Manual/ps` en formato PostScript.

4.1. Carga de computador virtual

Para ejecutar el simulador simplemente hay que ordenar `bsvc`. A continuación aparece el interfaz de usuario del simulador que se muestra en la figura 4.1.

Si se pulsa sobre el menú `File`, aparece una persiana con tres opciones activas:

New Setup Esta opción permite crear un computador virtual.

Open Setup Esta opción permite cargar un computador virtual previamente creado.

Quit Esta opción permite salir del simulador.

Se debe pulsar la opción `Open Setup` para cargar el computador virtual sobre el que se va a realizar la práctica. Al pulsarla aparece una nueva ventana con un contenido similar a la de la figura 4.2.

El fichero que contiene el computador de la práctica se llama `practica.setup` y está en `/usr/local/bsvc`. Este computador consta de:

- Un microprocesador MC68000.

[1] Senior design project

Figura 4.1: Ventana inicial del simulador BSVC

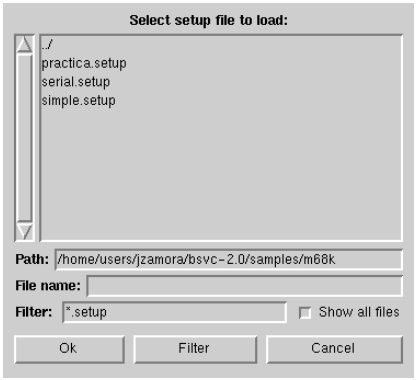


Figura 4.2: Ventana de búsqueda de ficheros

- Un módulo de memoria RAM de 32 Kbytes situado a partir de la dirección 0.
- Una DUART MC68681 situada a partir de la dirección EFFC00, conectada a la línea de petición de interrupción de nivel 4 y cuyas dos líneas series están conectadas a dos terminales alfanuméricos que a su vez se simulan con dos ventanas.

Al pulsar dos veces consecutivas sobre él se carga la configuración de dicho computador. La ventana inicial cambia de aspecto y se muestra la ventana de manejo del simulador (figura 4.3).

Además aparecen dos ventanas con las etiquetas `MC68681 Línea A` y `MC68681 Línea B`, que están conectados a las líneas correspondientes como si de terminales alfanuméricos se tratase.

Existe la posibilidad de cargar el fichero de configuración del computador (`practica.setup`) desde la línea de comandos:

```
bsvc /usr/local/bsvc/practica.setup
```

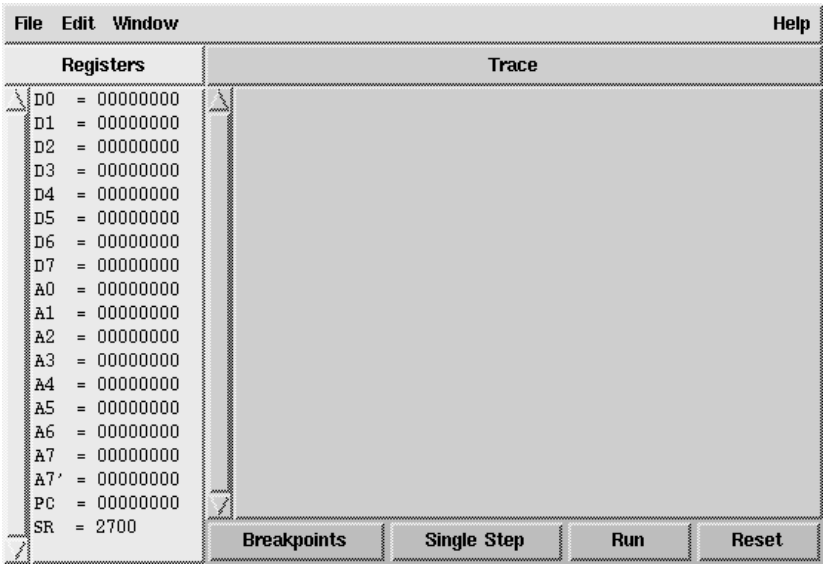


Figura 4.3: Ventana del simulador

4.2. Carga de un programa objeto

Ahora que se dispone de un computador, se puede proceder a cargar un programa en la memoria. Pulsando en la opción `File` aparece la persiana anterior pero con dos nuevas opciones activas:

Load Program Esta opción permite cargar un programa en la memoria del computador. Al pulsarla aparece una ventana como la de la figura 4.2.

Existen programas ejemplo en `/usr/local/bsvc/samples/m68000`, si se desea cargar alguno hay que cargar el programa objeto de extensión `.h68`. Para familiarizarse cargue `example.h68` pulsando dos veces consecutivas sobre él.

Save Setup Esta opción permite salvar un computador virtual.

4.3. Menús de la ventana de manejo del simulador

En la parte superior derecha de esta ventana (figura 4.3), existen tres menús:

File Es idéntica a su homónima correspondiente a la ventana inicial y sirve para para cargar, crear y salvar computadores virtuales, así como para cargar programas y salir del simulador.

Edit Sirve para ver y modificar el computador virtual que se ha cargado.

Window Sirve para generar dos nuevas ventanas. Una muestra el listado de ensamblaje del programa cargado, que contiene el código fuente. La otra permite ver y alterar el contenido del mapa de direcciones.

Registers La columna izquierda de esta ventana muestra los registros del microprocesador MC68000. En su parte superior existe un menú etiquetado `Registers` que despliega dos opciones:

Alter Permite modificar el contenido de un registro. Para ello se selecciona el registro en cuestión y a continuación esta opción. El mismo efecto se consigue pulsando dos veces consecutivas sobre el registro.

Clear All Pone a 0 todos los registros.

Trace La columna de la derecha es una zona de presentación de la traza de ejecución del programa. El menú superior etiquetado `Trace` permite seleccionar el tipo de información que se muestra y salvar la traza a fichero. Por defecto, se muestra la instrucción donde se ha detenido la ejecución y los posibles errores.

4.4. Botones de la ventana de manejo del simulador

En la parte inferior derecha, bajo la ventana de traza existen 4 botones:

Breakpoints Permite ver, añadir y quitar puntos de ruptura introduciendo una dirección de memoria. Sin embargo, para añadir un punto de ruptura basta con pulsar sobre la instrucción correspondiente, la línea de código correspondiente cambiará al color rojo. Del mismo modo, para quitar un punto de ruptura se pulsa sobre la línea correspondiente. Este simulador únicamente acepta puntos de ruptura de tipo *fetch*.

Single Step Permite ejecutar el programa instrucción a instrucción.

Run/Stop Permite iniciar o detener la ejecución del programa.

Reset Simula la excepción de `Reset` del procesador. Como se describe en la sección 1.5.4, supone la inicialización del computador virtual. El puntero de pila de supervisor (A7') se inicializa con el contenido de la palabra larga de la dirección 0, el contador de programa (PC) se inicializa con el contenido de la palabra larga de la dirección 4 y procesador se coloca en modo supervisor con las interrupciones inhibidas (2700 en el registro de estado, SR).

Antes de ejecutar el programa se debe pulsar este botón.

4.5. Resultados de la ejecución de un programa

La implementación de la herramienta bsvc original se ha modificado para permitir al usuario que le sea más fácil la depuración de un programa. Las modificaciones realizadas consisten en la generación de tres ficheros adicionales en los que se almacena información de la ejecución de un programa:

- Ficheros `puertoa` y `puertob`. Contienen la información que se transmite por las líneas serie A y B respectivamente, aunque los caracteres transmitidos no sean representables en la ventana que representa la línea serie. Estos ficheros se pueden mostrar o editar con cualquier editor de ficheros binarios o volcar su contenido con el comando `od` de linux. Por ejemplo la ejecución de `od -x puertoa` realiza un volcado hexadecimal por la pantalla de todos los caracteres contenidos en el fichero `puertoa`.

- Fichero `traza.log`. Contiene información acerca de todas las escrituras que se han realizado en memoria. Puesto que el mapa de memoria y de entrada salida es único, en este fichero también se encontrarán las escrituras en los puertos de la DUART. Para que se genere este fichero, se debe establecer la variable de entorno `TRAZA_BSVC` al valor `1`. A modo de ejemplo se muestra un posible contenido del fichero:

```
        LOG DE LAS ESCRITURAS EN MEMORIA.
Se indica la dirección de la instrucción que genera la escritura, la
dirección en la que se escribe y el valor
PC         Dirección  Dato
0x00008A9C 0x00008FFC 0x00008AA0
0x000004A2 0x00EFFC03 0xCC
0x000004AA 0x00EFFC13 0xCC
0x00008AD8 0x00008FF6 0x0000
0x00008ADE 0x00008FF4 0x0064
```

  La primera columna muestra la dirección que ocupa la instrucción que realiza la escritura. Para conocer de qué instrucción se trata se debe consultar el fichero `.lis` mediante la opción `Program Listing` del menú `Window` de la herramienta.

  La segunda columna indica la dirección de memoria sobre la que se ejecuta la escritura.

  La tercera columna contiene el valor que se ha escrito en la dirección anterior que puede ser un valor de 8, 16 o 32 bits.

  En el ejemplo, la instrucción contenida en la dirección `0x8A9C` escribe el valor de 32 bits `0x00008AA0` en la dirección `0x00008FFC`. Las instrucciones ubicadas en las posiciones `0x04A2` y `0x04AA` escriben en los puertos de E/S de la DUART CSRA y CSRB el valor `0xCC`. Las instrucciones almacenadas en `0x8AD8` y `0x8ADE` escriben los valores de 16 bits `0` y `0x0064` en las direcciones `0x00008FF6` y `0x00008FF4` respectivamente.

4.6. Errores conocidos

Problema Cuando se ejecuta instrucción a instrucción, si se modifica el registro de estado se sigue mostrando el contenido antiguo. (`Registers`).

Posible solución Colocar un punto de ruptura en la instrucción siguiente a la que modifica el registro de estado y pulsar `Run`.

Problema Cuando se carga un programa desde la ventana del simulador (figura 4.3) se debe cargar el fichero con el código objeto cuya extensión `.h68`. Sin embargo, si se carga desde la ventana de listado no se podrá ejecutar correctamente.

Capítulo 5

Enunciado del proyecto: E/S mediante interrupciones

El objetivo del proyecto es que el alumno se familiarice 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**.

La estructura del proyecto se muestra en la figura 5.1. Como puede apreciarse se necesitan unos búfferes internos para almacenar los caracteres que se reciben asíncronamente por las líneas. Del mismo modo, se necesitan sendos búfferes internos para almacenar los caracteres pendientes de transmitirse por las líneas.

Además, existe una única rutina de tratamiento de las interrupciones de las líneas que será la encargada transferir la información a o desde los mencionados búfferes internos. El proyecto implica la programación de la rutina de tratamiento de las interrupciones (RTI) así como de las subrutinas SCAN y PRINT que constituyen la interfaz:

INIT: Inicialización de los dispositivos. Preparará las dos líneas serie para recibir y transmitir caracteres y notificar los sucesos mediante la solicitud de interrupciones.

SCAN: Lectura de un dispositivo. Devolverá un bloque de caracteres que se haya recibido previamente por la línea correspondiente (A o B).

PRINT: Escritura en un dispositivo. Ordenará la escritura de un bloque de caracteres por la línea correspondiente (A o B).

Como ayuda para la implementación y prueba de los búfferes internos que deben manipular las subrutinas anteriores, se proporcionan las subrutinas auxiliares que se indican a continuación:

LEECAR: Obtención de un carácter de un buffer interno. Se encarga, junto a la función siguiente, de la gestión de los búfferes internos que permiten el comportamiento no bloqueante de las subrutinas `SCAN` y `PRINT`.

ESCCAR: Inserción de un carácter en un buffer interno. Se encarga, junto a la función anterior, de la gestión de los búfferes internos que permiten el comportamiento no bloqueante de las subrutinas `SCAN` y `PRINT`.

INI_BUFS: Inicialización de todos los buffers internos involucrados en el proyecto. Esta subrutina se invocará una sola vez desde `INIT`.

Estas tres subrutinas auxiliares están contenidas en el fichero `/usr/local/bsvc/bib_aux.s` de los computadores asignados al proyecto (véase el apéndice A). Este fichero se debe copiar en el directorio de trabajo del alumno, es decir, en el mismo directorio desde el que se ensambla `es_int.s` e incluirse mediante la pseudoinstrucción `INCLUDE` al **final del fichero** que se debe entregar:

```
INCLUDE bib_aux.s
```

Es importante que después de la línea que contiene el `INCLUDE` se añada al menos una línea vacía, puesto que de lo contrario generará un error de ensamblado.

Las subrutinas SCAN y PRINT deberán tener un comportamiento **no bloqueante**. Es decir, estas subrutinas se limitarán a almacenar o recuperar la información solicitada de los búfferes internos y no esperarán en ningún caso a que termine la transmisión de los caracteres o a que se reciban nuevos caracteres.

La forma de acceder a los búfferes internos y variables compartidas entre la `RTI` y `SCAN` y `PRINT` debe asegurar la integridad de los datos manejados por las subrutinas, es decir, que ningún carácter es leído dos veces y que no se pierde ninguno. Esto exige realizar un estudio de concurrencia entre la RTI y el resto de subrutinas que se pueden ejecutar de forma concurrente, teniendo en cuenta **qué información** modifica cada subrutina y **en qué momento** lo hace.

Las subrutinas anteriores se depurarán y probarán escribiendo una serie de programas principales que llamen a estas subrutinas con un conjunto de parámetros distintos. Este juego de ensayo debe asegurar el funcionamiento correcto de la RTI, SCAN y PRINT.

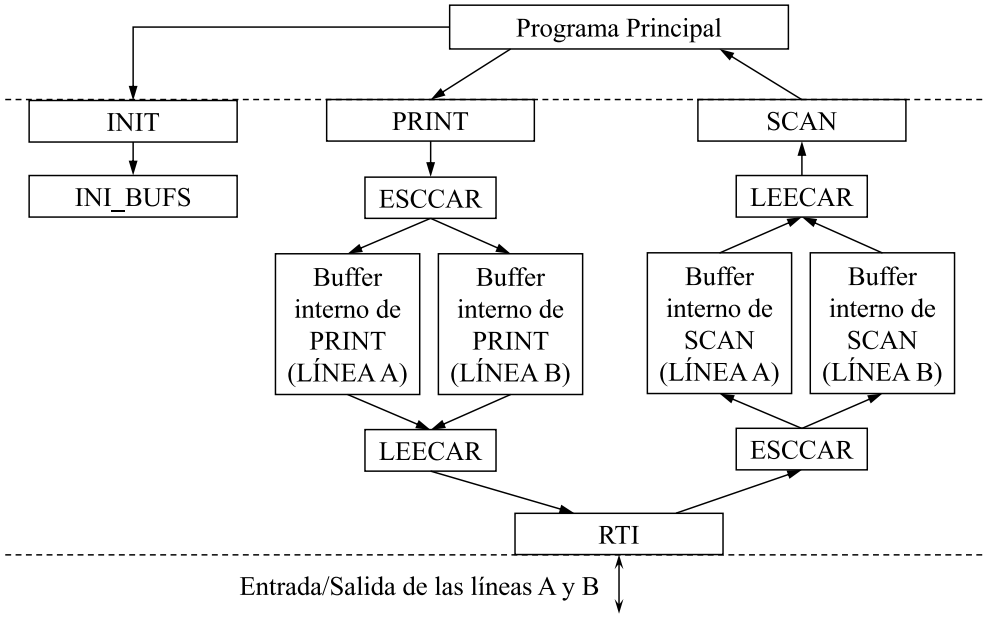


Figura 5.1: Estructura del proyecto.

Descripción de las Subrutinas

Todas las subrutinas, excepto `INI_BUFS`, `LEECAR` y `ESCCAR` reciben los parámetros en la pila y el valor de retorno, si lo tiene, se devuelve en el registro `D0`. En la figura 5.1 se muestra la relación entre las subrutinas de la práctica, excluyendo `INI_BUFS`, `LEECAR` y `ESCCAR` que se utilizarán para que `SCAN`, `PRINT` y `RTI` realicen el acceso a los búfferes internos de 2000 bytes de las líneas serie.

`INI_BUFS ( )`

Parámetros:
No tiene.

Descripción:
La rutina `INI_BUFS` realiza la inicialización de los cuatro búfferes internos y, después de la ejecución, los cuatro estarán vacíos. Esta subrutina se llamará una sola vez desde `INIT`.

`LEECAR (Buffer)`

Parámetros:

- **Buffer:** 4 bytes. Es un descriptor que indica de qué buffer interno se desea extraer el carácter. Se pasa por valor en el registro D0. Es un parámetro de entrada/salida. Tiene dos bits significativos:
  - **Bit 0:** Selecciona la línea de transmisión. Un 0 indica que se desea acceder a un buffer asociado a la línea A y un 1 a la línea B.
  - **Bit 1:** Selecciona el tipo de buffer. Un 0 indica que se desea acceder al buffer de recepción y un 1 al de transmisión.
  - El resto de bits no serán tenidos en cuenta.

Descripción:
La rutina `LEECAR` realiza la extracción de un carácter del buffer interno que se selecciona en el parámetro. Si el buffer interno está vacío, la función devolverá el valor 0xFFFFFFFF y no modificará el buffer. Si el buffer interno contiene caracteres, la función extraerá el primer carácter del buffer almacenándolo en el registro `D0` y lo "eliminará". Los posibles valores de los dos bits menos significativos del parámetro de entrada son:

- 0: indica que se desea acceder al buffer interno de recepción de la línea A.
- 1: indica que se desea acceder al buffer interno de recepción de la línea B.
- 2: indica que se desea acceder al buffer interno de transmisión de la línea A.
- 3: indica que se desea acceder al buffer interno de transmisión de la línea B.

Se supondrá que el programa que invoca a esta subrutina no deja ningún valor representativo en los registros del computador salvo el puntero de marco de pila (A6) y en el parámetro D0.
Resultado:

- **D0:** 4 bytes. Se devuelve un código que indica el resultado de la operación:
  - **D0 = 0xFFFFFFFF** si no hay ningún carácter disponible en el buffer interno seleccionado.
  - **D0 es un número entre 0 y 255.** Indicará el carácter que se ha extraído del buffer interno seleccionado.

`ESCCAR (Buffer,Caracter)`

Parámetros:

- **Buffer:** 4 bytes. Es un descriptor que indica de qué buffer interno se desea obtener el primer carácter. Se pasa por valor en el registro D0. Es un parámetro de entrada/salida. Tiene dos bits significativos:
  - **Bit 0:** Selecciona la línea de transmisión. Un 0 indica que se desea acceder a un buffer asociado a la línea A y un 1 a la línea B.
  - **Bit 1:** Selecciona el tipo de buffer. Un 0 indica que se desea acceder al buffer de recepción y un 1 al de transmisión.
  - El resto de bits no serán tenidos en cuenta.
- **Caracter:** 1 byte. Es el carácter que se desea añadir al buffer interno como último carácter. Se pasa por valor en los 8 bits menos significativos del registro D1. Es un parámetro de entrada.

Descripción:
La rutina `ESCCAR` realiza la inserción de un carácter del buffer interno que se selecciona en el parámetro. Si el buffer interno está lleno, la función devolverá el valor 0xFFFFFFFF y no modificará el buffer. Si el buffer no está lleno, la función insertará el carácter contenido en D1 como último carácter del buffer. Los posibles valores de los dos bits menos significativos de D0 son:

- 0: indica que se desea acceder al buffer interno de recepción de la línea A.
- 1: indica que se desea acceder al buffer interno de recepción de la línea B.
- 2: indica que se desea acceder al buffer interno de transmisión de la línea A.
- 3: indica que se desea acceder al buffer interno de transmisión de la línea B.

Se supondrá que el programa que invoca a esta subrutina no deja ningún valor representativo en los registros del computador salvo el puntero de marco de pila (A6) y en los parámetros D0 y D1.

Resultado:

- **D0:** 4 bytes. Se devuelve un código que indica el resultado de la operación:
  - **D0 = 0xFFFFFFFF** si el buffer interno seleccionado está lleno.
  - **D0 = 0** indicará que el carácter se ha insertado en el buffer interno correctamente.

`INIT ( )`

Parámetros:
No tiene.

Resultado:
Las líneas A y B deben quedar preparadas para la **recepción** y **transmisión** de caracteres mediante E/S por interrupciones. Al finalizar la ejecución de la instrucción `RTS`, el puntero de pila (*SP*) debe apuntar a la misma dirección a la que apuntaba antes de ejecutar la instrucción `BSR`. Debido a la particular configuración del emulador, esta subrutina no puede devolver ningún error y, por tanto, no se devuelve ningún valor de retorno. Se supondrá que el programa que invoca a esta subrutina no deja ningún valor representativo en los registros del computador salvo el puntero de marco de pila (A6).

Descripción:
La rutina `INIT` realiza la inicialización de las dos líneas disponibles en la DUART MC68681. Los parámetros de inicialización de esta subrutina son los siguientes:

- 8 bits por carácter para ambas líneas.
- No activar el *eco* en ninguna de las líneas.
- Se debe solicitar una interrupción cada vez que llegue un carácter.
- La velocidad de recepción y transmisión será de 38400 bits/s en ambas líneas.
- Funcionamiento *Full Duplex*: deben estar habilitadas la recepción y la transmisión simultáneamente.
- Establecer el vector de interrupción 40 (hexadecimal).
- Habilitar las interrupciones de recepción de las líneas correspondientes en la máscara de interrupción. Las interrupciones de transmisión sólo se activarán cuando el buffer de transmisión de la línea correspondiente contenga algún carácter.
- Actualizar la dirección de la rutina de tratamiento de interrupción en la tabla de vectores de interrupción.
- Inicializar los búfferes internos de 2000 bytes de las subrutinas indicadas anteriormente mediante una llamada a `INI_BUFS`.

Nota: se recuerda que el registro de máscara de interrupción (IMR) de la DUART MC68681 no se puede leer. Si la lógica del programa necesitase conocer su contenido, se podría mantener una copia en memoria de las escrituras sobre dicho registro.

`SCAN (Buffer, Descriptor, Tamaño)`

Parámetros:

- **Buffer:** 4 bytes. Es el buffer en el que se van a devolver los caracteres que se han leído del dispositivo. Se pasa por dirección. Es un parámetro de salida.
- **Descriptor:** 2 bytes. Es un número entero. Es un parámetro de entrada. Indica el dispositivo sobre el que se desea realizar la operación de lectura:
  - 0 indica que la lectura se realizará de la línea A.
  - 1 indica que la lectura se realizará de la línea B.
  - Cualquier otro valor provocará que la subrutina devuelva un error.
- **Tamaño:** 2 bytes. Es un número entero sin signo. Es un parámetro de entrada. Indica el número **máximo** de caracteres que se deben leer del buffer interno y copiar en el parámetro `Buffer`.

Resultado:

- **D0:** 4 bytes. Se devuelve un código que indica el resultado de la operación:
  - **D0 = 0xFFFFFFFF** si existe algún error en los parámetros pasados.
  - **D0 es un número positivo.** Indicará el número de caracteres que se han leído y se han copiado a partir de la posición de memoria indicada por el parámetro `Buffer`.

Descripción:

La rutina `SCAN` realiza la lectura de un bloque de caracteres de la línea correspondiente (A o B).

La lectura se deberá realizar de forma **no bloqueante**, es decir, la subrutina se limitará a copiar en el parámetro `Buffer` los `Tamaño` primeros caracteres almacenados en el buffer interno correspondiente y "eliminarlos" de dicho buffer interno utilizando la función `LEECAR`. Si el buffer interno contiene menos de `Tamaño` caracteres, los copiará en el parámetro `Buffer` y el buffer interno pasará a estar vacío. En **D0** se almacenará el número de caracteres que se han copiado en `Buffer`.

Además se deberá tener en cuenta lo siguiente:

- El comportamiento no bloqueante es resultado de gestionar las líneas serie mediante E/S por interrupciones. Para ello la subrutina `SCAN` dispondrá de sendos búfferes internos de 2000 bytes (a los que tiene acceso dicha subrutina a través de las subrutinas auxiliares) que contendrán los caracteres leídos de las líneas y no consumidos por ninguna llamada a `SCAN`. En particular esta subrutina debe asegurar que ningún carácter es leído dos veces y que no se pierde ninguno (véase la figura 5.1).
- En ningún caso esta subrutina debe esperar a que lleguen nuevos caracteres del dispositivo.
- Se copiarán a lo sumo tantos bytes como indique el parámetro `Tamaño`.

Esta subrutina deberá dejar el dispositivo preparado para realizar lecturas posteriores y, al igual que la subrutina de inicialización, debe dejar el puntero de pila (*SP*) apuntando a la misma posición de memoria a la que apuntaba antes de realizar la llamada a subrutina.

Se supondrá que el programa que invoca a esta subrutina habrá reservado espacio suficiente en el buffer que se pasa como parámetro (`Buffer`) y no deja ningún valor representativo en los registros del computador salvo el puntero de marco de pila (A6).

`PRINT (Buffer, Descriptor, Tamaño)`

Parámetros:

- **Buffer:** 4 bytes. Es el buffer en el que se pasa el conjunto de caracteres que se desea escribir en el dispositivo. Se pasa por dirección. Es un parámetro de entrada.
- **Descriptor:** 2 bytes. Es un número entero. Es un parámetro de entrada. Indica el dispositivo sobre el que se desea realizar la operación de escritura:
  - 0 indica que la escritura se realizará de la línea A.
  - 1 indica que la escritura se realizará de la línea B.
  - Cualquier otro valor provocará que la subrutina devuelva un error.
- **Tamaño:** 2 bytes. Es un número entero sin signo. Es un parámetro de entrada. Indica el número de caracteres que se deben leer del parámetro `Buffer` y escribir en el puerto.

Resultado:

- **D0:** 4 bytes. Se devuelve un código que indica el resultado de la operación:
  - **D0 = 0xFFFFFFFF** si existe algún error en los parámetros pasados.
  - **D0 es un número positivo.** Indicará el número de caracteres que se han aceptado para su escritura en el dispositivo.

Descripción:

La rutina `PRINT` realiza la escritura en el correspondiente buffer interno de tantos caracteres como indique el parámetro `Tamaño` contenidos en el buffer que se pasa como parámetro.

La escritura se deberá realizar de forma **no bloqueante**, es decir, la subrutina finalizará inmediatamente después de copiar los caracteres pasados en el parámetro `Buffer` al buffer interno y, si como resultado de dicha copia hay caracteres en el buffer interno, activar la transmisión de caracteres por la línea. La copia de los caracteres se realizará invocando a la función `ESCCAR`.

Además se deberá tener en cuenta lo siguiente:

- Al igual que en la subrutina `SCAN` el comportamiento no bloqueante es resultado de gestionar la línea serie mediante E/S por interrupciones. Esto indica que el MC68681 generará una interrupción cuando alguna de las líneas esté preparada para transmitir y, por tanto, la rutina de tratamiento de interrupción será la encargada de ir transmitiendo los caracteres por la línea correspondiente (véase la figura 5.1). Para permitir este comportamiento, la subrutina `PRINT` dispondrá de sendos búfferes internos de 2000 bytes (a los que tiene acceso a través de las subrutinas auxiliares) que contendrá los caracteres pendientes de ser enviados por las líneas.

- Una llamada a `PRINT` para una de las líneas se limitará a copiar del parámetro `Buffer` los datos que se desean escribir al buffer interno correspondiente y "encolarlos" al conjunto de caracteres que están pendientes de transmitirse. En el caso de que los caracteres que se desean transmitir no quepan en su totalidad en el buffer interno, se "encolarán" los que quepan y se devolverá el número de caracteres copiados en **D0**. Si se ha copiado algún carácter en el buffer, se activará la transmisión de caracteres por la línea.

- En ningún caso esta subrutina debe esperar a que finalice la transmisión de caracteres del dispositivo.

- La DUART solicitará interrupciones cada vez que la línea correspondiente esté lista para transmitir si se han activado en el registro de máscara de interrupciones `(IMR)`.

Esta subrutina deberá dejar el dispositivo preparado para realizar escrituras posteriores y, al igual que las otras subrutinas, debe dejar el puntero de pila (*SP*) apuntando a la misma posición de memoria a la que apuntaba antes de realizar la llamada a subrutina.

Se supondrá que el programa que invoca a esta subrutina habrá reservado espacio suficiente en el buffer que se pasa como parámetro (`Buffer`) y no deja ningún valor representativo en los registros del computador salvo el puntero de marco de pila (A6).

`RTI`

Descripción:

La invocación de la rutina de tratamiento de interrupción es el resultado de la ejecución de la secuencia de reconocimiento de interrupciones expuesta en la página 8. Entre otras acciones esta subrutina debe realizar las siguientes acciones:

- **Identificación de la fuente de interrupción.** Puesto que el MC68681 activa una misma señal de interrupción para las cuatro condiciones posibles, esta subrutina debe identificar cuál de las cuatro posibles condiciones ha generado la solicitud de interrupción.

- **Tratamiento de la interrupción.** Una vez identificada la fuente, se debe realizar el tratamiento de la interrupción.

  - Si la interrupción es de "recepción" indica que la cola FIFO de recepción de la línea no está vacía (véase la página 38). En este caso se debe añadir el carácter que se recibió por la línea al buffer interno correspondiente utilizando la función `ESCCAR`.
  - Si la interrupción es de "transmisión" indica que la línea está preparada para transmitir un carácter. En este caso si quedan caracteres en el buffer interno de transmisión, se debe obtener el primer carácter del buffer interno, "eliminarlo" invocando a la función `LEECAR` y transmitirlo por la línea.

- **Situaciones "especiales".** Hay situaciones en las que el tratamiento de la interrupción no se puede asociar al tratamiento general:

- Si la interrupción es de "recepción" y el buffer interno está lleno, (la llamada a la función `ESCCAR` ha devuelto 0xFFFFFFFF) no se puede añadir el carácter que se recibe por la línea, pero se debe leer el carácter del buffer de recepción del MC68681 para desactivar la petición de interrupción. En este caso el carácter no se añade al buffer interno (se "tira").
- Si la interrupción es de "transmisión" y el buffer interno de la línea está vacío (la llamada a la función `LEECAR` ha devuelto 0xFFFFFFFF) se deben deshabilitar las interrupciones de transmisión para la línea que ha interrumpido en el registro IMR del MC68681. Si no se realizara esta operación el dispositivo no desactivaría la señal de interrupción puesto que seguiría estando preparado para transmitir.

Nota: como complemento a la descripción de estas subrutinas, en la sección Ejemplos se proporcionan distintos casos de uso.

Variables locales y paso de parámetros

El procesador MC68000 no dispone de un registro de propósito específico que realice las tareas de puntero de marco de pila (`FP`). No obstante, habitualmente se suele utilizar el registro de direcciones `A6` para que realice estas funciones. El procesador MC68000 dispone de dos instrucciones que ayudan a la creación y destrucción del marco de pila de una subrutina: `LINK` y `UNLK`. Estas instrucciones permiten gestionar fácilmente la creación y destrucción de las variables locales de una subrutina.

El espacio asignado para variables locales se reserva en el marco de pila de la correspondiente rutina. Para construir dicho marco de pila basta con salvaguardar el valor que tuviera el registro que actúa como puntero al marco de pila (`A6`), crear el nuevo marco de pila y reservar espacio en la pila para las variables locales. Obsérvese que todas estas funciones son realizadas por la instrucción `LINK`.

Supóngase que una rutina `SCAN` necesita utilizar dos variables locales de 32 bits (i y j). La estructura de la pila se muestra en la figura 5.2. La reserva de este espacio de variables (8 bytes) se realizará al entrar en la rutina:

```
SCAN:     LINK A6, #-8      *Se crea el marco de pila
```

Si en el código posterior de la rutina `SCAN` se desea cargar en el registro **D2** la variable i y en **D3** la variable j se realiza con el siguiente código ensamblador:

```
SCAN:     LINK A6, #-8      *Se crea el marco de pila
          MOVE.L -8(A6),D2
          MOVE.L -4(A6),D3
```

Para deshacer el marco de pila creado a la entrada de la subrutina se deberán realizar las siguientes acciones:

- Se copia el valor del puntero de marco al puntero de pila.
- Se restaura el valor que tuviera el puntero de marco antes de entrar en la subrutina.
- Se retorna a la subrutina llamante.

Las dos primeras acciones son realizadas mediante la instrucción `UNLK`. Por tanto, el código de salida de una subrutina que utilice marco de pila se muestra a continuación.

```
UNLK A6      *Se destruye el marco de pila
RTS
```

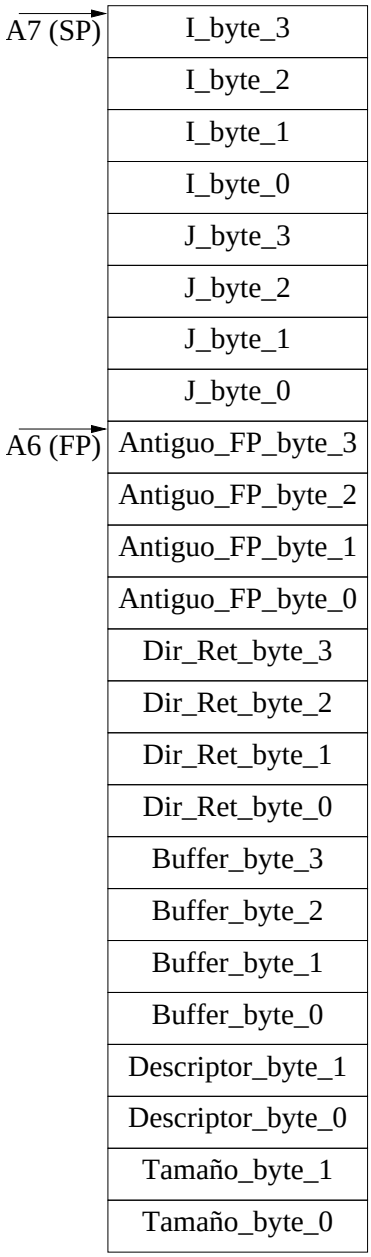


Figura 5.2: Gestión de variables locales.

Asignación de etiquetas y de memoria

Los puntos de entrada de las subrutinas deberán ir asociados a las etiquetas `INIT`, `SCAN` y `PRINT`.

El rango de direcciones **0 a la 0x00003FF** se reservarán para ubicar la tabla de vectores de interrupción. El alumno debe ubicar todo el código (datos y variables globales privadas a las subrutinas) a partir de la dirección hexadecimal 0x0000400 hasta la 0x00007FFF. La pila se situará en las **posiciones altas de memoria**.

Ejemplos

Como aclaración a la especificación de las subrutinas, a continuación se incluye una serie de ejemplos con los argumentos que se pasan a cada una de las subrutinas y direcciones de memoria que se modifican. Este conjunto de casos debe ser utilizado como ejemplo de la especificación a subrutinas, no como los casos de prueba con los que se evaluará el proyecto. Puesto que en este procesador el direccionamiento es a nivel de byte, cada una de las direcciones que se muestran en este apartado contendrán un byte. Obsérvese que los ejemplos que se muestran a continuación incluyen las subrutinas auxiliares, que no se deben implementar. El objetivo es mostrar su funcionamiento, tomando como base la especificación de las subrutinas proporcionadas.

NOTA: Los números que comienzan con 0x están representados en hexadecimal.

INIT

Caso 1.

A7 = 32000

Resultado:

A7 = 32000

Debe dejar las líneas A y B preparadas para la recepción y transmisión de caracteres, se habrá habilitado la recepción y transmisión en ambas líneas, se habrá establecido correctamente el vector de interrupción, se habrá actualizado la entrada de la tabla de vectores de interrupción y se habrá inicializado como buffer vacío dos búfferes de 2000 caracteres para cada una de las líneas.

LEECAR

Caso 2.

D0 = 0

Caracteres recibidos por la línea A y almacenados en el buffer interno	de computadores y mas
Representación ASCII:	0x64, 0x65, 0x20, 0x63, 0x6F, 0x6D, 0x70, 0x75, 0x74, 0x61, 0x64, 0x6F, 0x72, 0x65, 0x73, 0x20 0x79, 0x20, 0x6D, 0x61, 0x73

Resultado:

D0 = 0x64

Caracteres recibidos por la línea A y almacenados en el buffer interno	e computadores y mas
Representación ASCII:	0x65, 0x20, 0x63, 0x6F, 0x6D, 0x70, 0x75, 0x74, 0x61, 0x64, 0x6F, 0x72, 0x65, 0x73, 0x20 0x79, 0x20, 0x6D, 0x61, 0x73

Caso 3.

D0 = 0x03040503

Caracteres a enviar por la línea B y almacenados en el buffer interno	interrupción
Representación ASCII:	0x69, 0x6E, 0x74, 0x65, 0x72, 0x72, 0x75, 0x70, 0x63, 0x69, 0xF3, 0x6E

Resultado:

D0 = 0x69

Caracteres a enviar por la línea B y almacenados el buffer interno	nterrupción
Representación ASCII:	0x6E, 0x74, 0x65, 0x72, 0x72, 0x75, 0x70, 0x63, 0x69, 0xF3, 0x6E

Obsérvese que los 30 bits más significativos del parámetro de entrada de D0 se ignoran.

Caso 4.

D0 = 0x00000003

Caracteres a enviar por la línea B y almacenados en el buffer interno	`<vacío>`

Resultado:

D0 = 0xFFFFFFFF

Caracteres a enviar por la línea B y almacenados el buffer interno	`<vacío>`

ESCCAR

Caso 5.

D0 = 1 **D1** = 0x41

Caracteres recibidos por la línea B y almacenados en el buffer interno	COS
Representación ASCII:	0x43, 0x4F, 0x53

Resultado:

D0 = 0

Caracteres recibidos por la línea B y almacenados en el buffer interno	COSA
Representación ASCII:	0x43, 0x4F, 0x53, 0x41

Caso 6.

D0 = 2 **D1** = 0x62

Caracteres a enviar por la línea A y almacenados en el buffer interno	`Buffer lleno (2000 caracteres)`

Resultado:

D0 = 0xFFFFFFFF

Caracteres a enviar por la línea A y almacenados en el buffer interno	`Buffer lleno (2000 caracteres)`

SCAN

Caso 7.

A7 = 32000	Direcciones de Memoria: **32000:** 0x00, 0x00, 0x13, 0x88, **32004:** 0x00, 0x01, **32006:** 0x00, 0x0F, **5000:** ??, ??, ??, ??, . . . , ??
Caracteres recibidos por la línea B y almacenados en el buffer interno	de computadores y mas
Representación ASCII:	0x64, 0x65, 0x20, 0x63, 0x6F, 0x6D, 0x70, 0x75, 0x74, 0x61, 0x64, 0x6F, 0x72, 0x65, 0x73, 0x20 0x79, 0x20, 0x6D, 0x61, 0x73

Resultado:

A7 = 32000 **D0** = 15	Direcciones de Memoria: **5000:** 0x64, 0x65, 0x20, 0x63, 0x6F, 0x6D, 0x70, 0x75, **5008:** 0x74, 0x61, 0x64, 0x6F, 0x72, 0x65, 0x73
Caracteres recibidos por la línea B y almacenados en el buffer interno	`<blanco>` y mas
Representación ASCII:	0x20, 0x79, 0x20, 0x6D, 0x61, 0x73

Caso 8.

A7 = 32000	Direcciones de Memoria: **32000:** 0x00, 0x00, 0x13, 0x88, **32004:** 0x00, 0x01, **32006:** 0x00, 0x22, **5000:** ??, ??, ??, ??, . . . , ??
Caracteres recibidos por la línea B y almacenados en el buffer interno	de
Representación ASCII:	0x64, 0x65

Resultado:

A7 = 32000 **D0** = 2	Direcciones de Memoria: **5000:** 0x64, 0x65
Caracteres recibidos por la línea B y almacenados en el buffer interno	`<vacío>`

PRINT

Caso 9.

A7 = 32000	Direcciones de Memoria: **32000:** 0x00, 0x00, 0x13, 0x88, **32004:** 0x00, 0x10, **32006:** 0x00, 0x11, **5000:** ??, ??, ??, ??, . . . , ??

Resultado:

A7 = 32000
D0 = 0xFFFFFFFF

Caso 10.

A7 = 32000	Direcciones de Memoria: **32000:** 0x00, 0x00, 0x13, 0x88, **32004:** 0x00, 0x01, **32006:** 0x00, 0x0C, **5000:** 0x69, 0x6E, 0x74, 0x65, 0x72, 0x72, **5006:** 0x75, 0x70, 0x63, 0x69, 0xF3, 0x6E
Caracteres a enviar por la línea B (buffer interno):	`<vacío>`

Resultado:

A7 = 32000
D0 = 12

Caracteres a enviar por la línea B (buffer interno):	0x69, 0x6E, 0x74, 0x65, 0x72, 0x72, 0x75, 0x70, 0x63, 0x69, 0xF3, 0x6E
Representación ASCII:	interrupción

Caso 11.

A7 = 32000	Direcciones de Memoria: **32000:** 0x00, 0x00, 0x13, 0x88, **32004:** 0x00, 0x01, **32006:** 0x00, 0x01, **5000:** 0x69
Caracteres a enviar por la línea B (buffer interno):	0x70, 0x65
Representación ASCII:	pe

Resultado:

A7 = 32000
D0 = 1

Caracteres a enviar por la línea B (buffer interno):	0x70, 0x65, 0x69
Representación ASCII:	pei

Caso 12.

A7 = 32000

Direcciones de Memoria:
32000: 0x00, 0x00, 0x13, 0x88,
32004: 0x00, 0x00,
32006: 0x00, 0x00,
5000: 0xF0, 0x6F, 0xFF, ...

Resultado:

A7 = 32000

Direcciones de Memoria:
5000: 0xF0, 0x6F, 0xFF, ..., ??

D0 = 0

Diseño y codificación de casos de prueba

Los ejemplos que se han expuesto anteriormente intentan aclarar la especificación de la práctica, pero para asegurar el correcto funcionamiento de las subrutinas de la práctica es necesario realizar un conjunto de casos de prueba que cubran el mayor número posible de situaciones que se puedan presentar.

Para mostrar cómo se puede construir un caso de prueba se incluye un ejemplo de programa principal que hace funcionar concurrentemente las rutinas de `SCAN`, `PRINT` y la `RTI`. El programa tiene un bucle principal, en el que cada iteración primero se queda en un bucle de llamadas a `SCAN` hasta que se consigue leer del puerto serie A un bloque de `TAMBS` caracteres y, a continuación, se queda un bucle de llamadas a `PRINT` hasta que se consigue imprimir en el puerto B todos los caracteres leídos, en bloques de tamaño `TAMBP`. El bucle principal se repite indefinidamente, de modo que lo que se teclea por el puerto serie A "aparece" por el puerto serie B.

La variable `BUFFER` se utiliza para almacenar todos los caracteres que se van a leer de la línea A. En la primera lectura se pasa como parámetro la dirección de comienzo de esta variable y en las sucesivas lecturas se pasará la dirección de comienzo incrementada en el número de caracteres que se han leído (valor de D0)

Las variables `PARDIR` y `PARTAM` son respectivamente la dirección y el tamaño que se pasarán como parámetro a ambas rutinas `SCAN` y `PRINT`.

La variable `CONTC` contiene el número de caracteres que quedan por imprimir.

Las constantes `DESA` y `DESB` son los descriptores de lectura y escritura que se pasarán como parámetros a `SCAN` y `PRINT`.

El programa comienza invocando a la rutina `INIT` y estableciendo los manejadores de las excepciones que se pueden producir (véase el apéndice C). Estos manejadores se limitan a parar la ejecución del programa. A continuación se realiza la iteración del bucle de `SCAN` tal como se ha indicado al comienzo de esta sección.

Una vez que se ha conseguido leer el bloque completo de caracteres se ejecuta el bucle de escrituras, que consiste en escribir todos los caracteres leídos en el bucle anterior, en bloques de `TAMBP` caracteres.

En cada iteración de este bucle se resta el número de caracteres que la subrutina `PRINT` ha aceptado para la transmisión. Si no se ha escrito todo el bloque se repite la llamada a `PRINT` con el conjunto de caracteres que no se ha transmitido.

El caso especial se trata al final del bucle. Esta situación consiste en que el número de caracteres que quedan por transmitir sea menor que el tamaño de bloque de print. En este caso se pasa como parámetro el número de caracteres que quedan por transmitir en lugar de pasar el tamaño de bloque.

```
BUFFER: DS.B    2100    * Buffer para lectura y escritura de caracteres
PARDIR: DC.L    0       * Dirección que se pasa como parámetro
PARTAM: DC.W    0       * Tamaño que se pasa como parámetro
CONTC:  DC.W    0       * Contador de caracteres a imprimir
DESA:   EQU     0       * Descriptor línea A
DESB:   EQU     1       * Descriptor línea B
TAMBS:  EQU     30      * Tamaño de bloque para SCAN
TAMBP:  EQU     7       * Tamaño de bloque para PRINT
```

```
        * Manejadores de excepciones
INICIO: MOVE.L  #BUS_ERROR,8    * Bus error handler
        MOVE.L  #ADDRESS_ER,12  * Address error handler
        MOVE.L  #ILLEGAL_IN,16  * Illegal instruction handler
        MOVE.L  #PRIV_VIOLT,32  * Privilege violation handler
        MOVE.L  #ILLEGAL_IN,40  * Illegal instruction handler
        MOVE.L  #ILLEGAL_IN,44  * Illegal instruction handler

        BSR     INIT
        MOVE.W  #$2000,SR       * Permite interrupciones

BUCPR:  MOVE.W  #TAMBS,PARTAM   * Inicializa parámetro de tamaño
        MOVE.L  #BUFFER,PARDIR  * Parámetro BUFFER = comienzo del buffer
OTRAL:  MOVE.W  PARTAM,-(A7)    * Tamaño de bloque
        MOVE.W  #DESA,-(A7)     * Puerto A
        MOVE.L  PARDIR,-(A7)    * Dirección de lectura
ESPL:   BSR     SCAN
        ADD.L   #8,A7           * Restablece la pila
        ADD.L   D0,PARDIR       * Calcula la nueva dirección de lectura
        SUB.W   D0,PARTAM       * Actualiza el número de caracteres leídos
        BNE     OTRAL           * Si no se han leído todas los caracteres
                                * del bloque se vuelve a leer

        MOVE.W  #TAMBS,CONTC    * Inicializa contador de caracteres a imprimir
        MOVE.L  #BUFFER,PARDIR  * Parámetro BUFFER = comienzo del buffer
OTRAE:  MOVE.W  #TAMBP,PARTAM   * Tamaño de escritura = Tamaño de bloque
ESPE:   MOVE.W  PARTAM,-(A7)    * Tamaño de escritura
        MOVE.W  #DESB,-(A7)     * Puerto B
        MOVE.L  PARDIR,-(A7)    * Dirección de escritura
        BSR     PRINT
        ADD.L   #8,A7           * Restablece la pila
        ADD.L   D0,PARDIR       * Calcula la nueva dirección del buffer
        SUB.W   D0,CONTC        * Actualiza el contador de caracteres
        BEQ     SALIR           * Si no quedan caracteres se acaba
        SUB.W   D0,PARTAM       * Actualiza el tamaño de escritura
        BNE     ESPE            * Si no se ha escrito todo el bloque se insiste
        CMP.W   #TAMBP,CONTC    * Si el nº de caracteres que quedan es menor que
                                * el tamaño establecido se imprime ese número
        BHI     OTRAE           * Siguiente bloque
        MOVE.W  CONTC,PARTAM
        BRA     ESPE            * Siguiente bloque

SALIR:  BRA     BUCPR

BUS_ERROR:      BREAK           * Bus error handler
                NOP
ADDRESS_ER:     BREAK           * Address error handler
                NOP
ILLEGAL_IN:     BREAK           * Illegal instruction handler
                NOP
PRIV_VIOLT:     BREAK           * Privilege violation handler
                NOP
```

5.1. Normas de presentación

Se recomienda a los alumnos que consulten periódicamente la página Web del proyecto, en la que se publicarán noticias relacionadas con este proyecto:

`http://www.datsi.fi.upm.es/docencia/Arquitectura_09/Proyecto_E_S`

La evaluación del proyecto se realizará en tres partes: pruebas de funcionamiento, memoria y examen. Para compensar el proyecto (obtener una nota mayor o igual que 3) se deberá superar un conjunto de pruebas que se indican en los párrafos siguientes y obtener la calificación de apto en la memoria y el examen. Todas las correcciones del proyecto que se citan en los párrafos siguientes se realizarán a las 21:00 en los días lectivos, salvo las excepciones que se indican expresamente.

Las fechas que aparecen en este documento son orientativas y se han establecido con las directrices proporcionadas por la Universidad en el momento de la elaboración de este documento. Si existiese algún cambio en las mismas, las fechas y las condiciones de evaluación podrán modificarse para cumplir con la nueva normativa.

CONVOCATORIA DE JUNIO

El plazo de entrega del proyecto terminará el día **21 de mayo de 2025 a las 21:00**. En este momento se realizará una corrección para todos los grupos que lo hayan entregado y no se les haya corregido, que **no será obligatoria** si el alumno considera que ha alcanzado los objetivos en entregas anteriores. A partir del 22 de mayo el sistema de entrega se configurará de tal forma que permita entregar únicamente la memoria. Dicha memoria se podrá entregar hasta el **6 de junio a las 21:00** en formato electrónico exclusivamente.

El sistema de entrega de prácticas se abrirá el 10 de marzo y para facilitar a los alumnos la planificación de su trabajo se establece **un hito** evaluable en el plazo de ejecución del proyecto que se indica a continuación. La consecución de este hito no será obligatoria para aprobar, pero sí lo será para la obtención de la máxima calificación en la ejecución y memoria del proyecto. El sistema de corrección se configurará de tal forma que se evaluarán todas las pruebas y no solo las pruebas involucradas en el hito. De esta forma si el alumno consigue alcanzar el objetivo del hito antes del plazo puede seguir avanzando en la realización del proyecto.

Hito: Supondrá el **10 %** de la calificación final del proyecto. Para alcanzarlo hay que superar todas las pruebas que involucren un solo puerto con un tamaño de transmisión de bloques inferior a 300 caracteres. El plazo final para alcanzarlo será el **22 de abril** a las 21:00. En este momento se realizará una corrección para todos los grupos que hayan entregado que servirá para la evaluación del hito. El **14 de marzo** se realizará una corrección a las 21:00 para todos los grupos que hayan entregado y desde el **17 de marzo** hasta el **11 de abril** se realizarán correcciones todos los días a las 21:00, de las que el alumno podrá elegir **cinco**. Las correcciones no consumidas para la consecución de este hito **no serán acumulables** para la evaluación final y se perderán. El sistema de entrega del proyecto (véase la página 79) se abrirá una semana antes de la primera corrección programada.

En la "Semana de actividades complementarias" se realizarán correcciones, pero no se proporcionará soporte del sistema de corrección al no ser una semana lectiva.

Si el alumno satisface el hito en alguna de las correcciones anteriores, se tomará el hito como alcanzado, independientemente de que no se satisfaga en las correcciones siguientes.

Independientemente de que el alumno haya alcanzado o no el hito anterior, podrá alcanzar los objetivos del proyecto utilizando a lo sumo **cinco correcciones** automáticas (en las fechas que el alumno estime conveniente) desde el **23 de abril** hasta el **20 de mayo**. Para solicitar una corrección basta con realizar correctamente la entrega del proyecto y la corrección se llevará a cabo todos los días lectivos a las 21:00. En este plazo se evaluarán todas las pruebas del proyecto y supondrá el 90 % de la calificación de las pruebas. La evaluación de la memoria se basará en esta calificación y podrá modificarse dependiendo de la calidad del documento presentado. La calificación de las pruebas y la memoria supone el 80 % de la calificación del proyecto y el 20 % el examen del mismo.

Para compensar el proyecto, es decir, obtener una calificación mayor o igual que 3 y que se conserve para el curso próximo, será necesario superar las pruebas asociadas al hito y todas las pruebas que involucren un solo puerto (puerto A o B) en las que se reciban o transmitan menos de 2100 caracteres. Los alumnos que cumplan esta condición podrán presentarse al examen del proyecto.

Para que el examen del proyecto pueda hacer media con la parte de la memoria y las pruebas será necesario obtener al menos un 2 en el mismo.

El examen del proyecto se realizará el mismo día que el examen final de la asignatura según la planificación que se publique en la página Web de la asignatura. En este examen se podrá utilizar el documento "Ayuda al Proyecto de Entrada/Salida"que está publicado en la web del proyecto, exclusivamente en papel y que deberá aportar el alumno:
`http://www.datsi.fi.upm.es/docencia/Arquitectura_09/Proyecto_E_S/ayuda_exa.pdf`

CONVOCATORIA DE JULIO

Puesto que en esta convocatoria no se consideran días lectivos, se considerarán como lectivos los días laborables. El plazo de entrega del proyecto terminará el día **3 de julio de 2025 a las 21:00**. En este momento se realizará una corrección del proyecto para todos los grupos que la hayan entregado y no se les haya corregido. Esta corrección **no será obligatoria** si el alumno considera alcanzados los objetivos del proyecto. El 4 de julio se configurará el sistema de entrega de tal forma que permita entregar únicamente la memoria del proyecto. Dicha memoria se podrá entregar hasta el **11 de julio a las 21:00** en formato electrónico.

Desde el 16 de junio hasta el 2 de julio se realizarán correcciones a las 21:00 para todos los alumnos que hayan realizado la entrega del proyecto correctamente. De estas correcciones, el alumno podrá disponer de **tres correcciones** (en las fechas que el alumno estime conveniente). Para solicitar una corrección basta con realizar correctamente la entrega del proyecto.

Para la evaluación del proyecto se tendrán en cuenta la calificación obtenida en el hito de la convocatoria ordinaria. La calificación obtenida en esta convocatoria ponderará el 90 % en la nota del proyecto (ejecución y memoria). Si el alumno compensa la ejecución en esta convocatoria, compensará la ejecución del proyecto, siguiendo el mismo criterio que en la convocatoria de Junio, independientemente de la calificación obtenida en el hito.

Si el alumno ha suspendido el examen del proyecto en la convocatoria de junio y no desea realizar ninguna mejora ni en el código ni en la memoria entregados, **no es necesario que vuelva a realizar ninguna entrega** y se tendrá en cuenta la última entrega realizada en la convocatoria de junio.

El examen del proyecto se realizará el mismo día que el examen final de la asignatura según la planificación que se publique en la página Web de la asignatura. En este examen se podrá utilizar el documento "Ayuda al Proyecto de Entrada/Salida"que está publicado en la web del proyecto, exclusivamente en papel y que deberá aportar el alumno:
`http://www.datsi.fi.upm.es/docencia/Arquitectura_09/Proyecto_E_S/ayuda_exa.pdf`

TUTORÍAS DEL PROYECTO

Las posibles preguntas relacionadas con la práctica se atenderán personalmente o por correo electrónico. Las consultas por correo electrónico se deberán dirigir a la dirección **pr_ent_sal@datsi.fi.upm.es**.

Si desea alguna tutoría presencial o telemática (vía Teams) con alguno de los profesores del proyecto, deberá concertar cita en la dirección de correo electrónico anterior.

ENTREGA DE LA PRÁCTICA

La entrega se compone de:

1. Una **memoria**, en formato DINA4, en la que deberán figurar claramente el nombre y apellidos de los autores de la práctica. Dicha memoria deberá contener los siguientes puntos:

   - Diagrama de flujo o pseudocódigo y comentario de los algoritmos utilizados.
   - Listado comentado de las subrutinas en ensamblador.
   - Descripción del juego de ensayo (conjunto de casos de prueba) que el alumno haya diseñado y utilizado para probar el correcto funcionamiento de la práctica.
   - Observaciones finales y comentarios personales de esta práctica, estimando asimismo el tiempo empleado en su realización.

   Esta memoria se entregará exclusivamente en formato electrónico según se indica en el siguiente apartado.

2. La entrega de los ficheros que contienen la práctica. Será obligatorio entregar los siguientes ficheros:

   - **autores**: Es un fichero ASCII que deberá contener los apellidos, nombre, número de matrícula y DNI de los autores de la práctica. La práctica se realizará individualmente o en grupos de **dos personas**. Cada línea de este fichero contendrá los datos de uno de los autores de la práctica, de acuerdo al siguiente formato:

     Nº Matrícula; DNI; apellido apellido, nombre; dir. correo electrónico

     El número de matrícula que se debe indicar en el fichero es el que **asigna la secretaría de la Facultad** (por ejemplo 990999) y no el que se utiliza como identificador para abrir cuentas en el Centro de Cálculo (por ejemplo a990999). Este fichero solo se entrega cuando se registra el grupo.

- **es_int.s**: Contendrá todas las subrutinas que componen el proyecto. Además, este fichero deberá incluir un programa principal que se haya utilizado para la depuración de la práctica. Este programa principal se utilizará para indagar sobre posibles errores que se puedan producir.
- **memoria.pdf**: Es un fichero PDF que deberá contener la memoria de la práctica. Este fichero solo se entregará al final del plazo de entrega de cada convocatoria.

IMPORTANTE: Se recomienda al alumno que antes de realizar una entrega de la práctica ensamble el fichero **es_int.s**, se asegure de que no generan ningún error y ejecute la práctica con sus propios casos de prueba.

FORMA DE ENTREGA DE LOS FICHEROS

Sistema de entrega en *triqui*.

Se utilizará un programa de entrega denominado **"ent68000"**. Para ejecutar este programa se deberá teclear, desde el intérprete de comandos de "triqui", el mandato **"/usr/local/bsvc/bin/ent68000"**. Dicho programa, permite entregar los ficheros indicados anteriormente, así como consultar los resultados de la ejecución de un conjunto de tests de pruebas utilizados por el corrector.

Al entrar en el programa, este pide la identificación del usuario. Tomaremos como identificación de usuario el número de matrícula de uno de los integrantes del grupo. El programa mostrará el mensaje:

Introduzca su identificador (Num. matricula): 990999

El usuario deberá introducir el número de matrícula de uno de los integrantes del grupo (p.e. 990999).

Si es la primera vez que el usuario entra en el sistema de entrega, el programa le invitará a que introduzca una palabra clave ("password") mostrando el siguiente mensaje:

Se va a establecer password.
Password:

El usuario deberá introducir una palabra clave (no se mostrará en pantalla). Para confirmar que no se ha producido ningún error al introducir el "password"se vuelve a pedir:

Repita el password tecleado anteriormente:

Si se ha producido algún error se reintentará establecer el password de nuevo.

Después de mostrar este mensaje el programa termina. Si el comando se ha ejecutado con éxito se mostrarán los datos que ha registrado el sistema de cada uno de los integrantes del grupo de prácticas. Seguidamente aparecerá el siguiente mensaje:

SE HAN DADO DE ALTA LOS SIGUIENTES ALUMNOS:
990999 123433342 PEREZ PEREZ JESUS

Se ha asignado password al usuario A990999.

- NO LO OLVIDE
- NO LO APUNTE
- NO LO DIVULGUE

suponiendo que el grupo de prácticas esté compuesto por un único alumno (Jesús Pérez Pérez con DNI n.º 123433342 y número de matrícula 990999) y la información que aparece en el fichero **autores** es:

990999 ; 123433342; PEREZ PEREZ JESUS

Si dicho alumno no aparece en las listas en poder del departamento o no ha introducido correctamente alguno de los datos, se mostrará un mensaje de error. A continuación se mostrará el siguiente menú:

OPCIONES:
1. Mandar Ficheros.
2. Consultar Resultados.
3. Cancelar Entregas.
4. Bloquear la Entrega.
5. Ayuda !!!!
6. Noticias.
7. Fichero Adicional.
q Abandonar.
>>>>

A continuación se explica cada una de las opciones del menú.

MANDAR FICHEROS

Esta opción permite mandar los ficheros de una práctica, que deberán estar en el directorio de trabajo del usuario. Si el comando se ejecuta correctamente se mostrarán los siguientes mensajes:

MANDANDO EL FICHERO es_int.s ...OK.

Si alguno de los ficheros no se encuentra, el programa lo comunicará al usuario. Por ejemplo:

MANDANDO EL FICHERO es_int.s ...
No se puede abrir el fichero es_int.s
Entrega abortada.

El servidor de entregas intenta asegurar que cada uno de los ficheros tiene el formato correcto. En nuestro caso esto se traduce en que el fichero se va a poder ensamblar cuando se realice la corrección. Si el comando de ensamblado no ha finalizado con éxito se mostrará un mensaje:

EL FICHERO es_int.s NO TIENE EL FORMATO CORRECTO
ENTREGA NO REALIZADA

En este caso el alumno deberá comprobar que se puede ensamblar correctamente el fichero y comprobar que contiene todas y cada una de las etiquetas que es obligatorio que aparezcan en dicho fichero.

En el caso de que se genere un error en la entrega de los ficheros, el programa de entrega termina la ejecución del programa y se muestra el "prompt"del sistema operativo. Si se desea realizar una nueva entrega se volverá a teclear el comando.

La realización de una **entrega anula todas las entregas anteriores** pendientes de corrección. Si dicha entrega es **errónea**, también se anula esta entrega.

CANCELAR ENTREGAS

Esta opción permite cancelar todas las entregas realizadas desde la última corrección. El grupo de prácticas será eliminado de la lista de proyectos pendientes de corregir. Si se han realizado varias entregas se cancelarán todas las entregas.

CONSULTAR RESULTADOS

Esta opción permite consultar los resultados de la corrección de la entrega de una proyecto. El programa pide el nombre de fichero en el que se copiarán los resultados de la ejecución del conjunto de *tests* de pruebas que componen el corrector. El programa pedirá un nombre de fichero donde escribir los datos generados. Se mostrará el siguiente mensaje:

> **La salida será redirigida a un fichero.**
> **Nombre del fichero (ENTER para salida por pantalla) ?? result.txt**

En este caso se grabarán los resultados de las pruebas en el fichero **result.txt**. Si como respuesta al mensaje se teclea ENTER, los resultados serán mostrados por pantalla. El nombre del fichero que se proporciona al programa (**result.txt**) no debe existir en el disco.

Esta opción se incluye para permitir la corrección automática de los proyectos. El alumno no debe utilizar este programa para depurar su práctica. Debe ser el propio alumno el que construya su conjunto de pruebas que le permita comprobar que la práctica funciona correctamente. Esta es la razón por la que los casos de prueba utilizados para la corrección de la práctica no se ponen a disposición del alumnado.

BLOQUEO DE LA ENTREGA

Si el usuario se compromete a no entregar más veces el proyecto, puede bloquear la entrega para mayor seguridad. Si se ejecuta esta opción no se podrá volver a realizar una nueva entrega de los ficheros asociados a la práctica. Si el comando se ejecuta satisfactoriamente se mostrará el mensaje:

> **ENTREGA BLOQUEADA.**

AYUDA

Esta opción mostrará en pantalla una breve descripción de cada una de las opciones del programa de entrega. No significa que se vaya a proporcionar ayuda para la realización de la práctica.

NOTICIAS

Esta opción es puramente informativa. Permite notificar al alumno modificaciones en la especificación de la práctica o, en general, noticias de interés de la asignatura asociada a la práctica. El programa pide el nombre de fichero en el que se copiarán las noticias.

> **La salida será redirigida a un fichero.**
> **Nombre del fichero (ENTER para salida por pantalla) ?? noticias.txt**

En este caso se grabará la información relativa a la asignatura en el fichero **noticias.txt**. Si como respuesta al mensaje se teclea ENTER, la información será mostrada por pantalla.

FICHERO ADICIONAL

Esta opción recupera un fichero binario que es complementario al fichero de consultas. Es un fichero comprimido en formato ZIP que contiene el log de las escrituras en memoria de las pruebas que han fallado. Estas trazas permiten conocer qué dato se escribe en una posición de memoria y desde qué instrucción. Su formato se compone de tres columnas de números hexadecimales:

- Dirección de memoria que ocupa la instrucción que ha provocado la escritura. Para identificar dicha instrucción debe acceder al fichero del programa con extensión `.lis`.
- Dirección de memoria en la que se ha escrito el dato.
- Datos que se ha escrito en la dirección anterior. Dependiendo del número de dígitos hexadecimales que aparezcan en esta columna el dato sera 1 byte (dos dígitos), una palabra (cuatro dígitos) o una doble palabra (ocho dígitos).

El formato de estos ficheros es el mismo que el del fichero `traza.log` y está descrito en el capítulo 4.

ABANDONAR

Termina la ejecución del programa de entrega. Si se realiza con éxito se mostrará el mensaje:

> **Cerrando la conexion**

y a continuación aparecerá el "prompt"del sistema operativo.

NOTA: NO SE CORREGIRÁ NINGÚN PROYECTO QUE NO SE ATENGA A ESTAS NORMAS Y SE CONSIDERARÁ POR LO TANTO COMO NO PRESENTADO.

PROCEDIMIENTO DE ENTREGA VÍA WEB

Se ha desarrollado una aplicación Web que permite realizar las mismas operaciones que la aplicación descrita en el apartado anterior. La URL en la que se encuentra es

http://www.datsi.fi.upm.es/Practicas

Apéndice A

Conexión a los computadores de prácticas de la Escuela

El objetivo de este apéndice es proporcionar al estudiante una guía detallada para establecer la conexión a los computadores del centro de cálculo dedicados al proyecto y práctica de Entrada/Salida.

A.1. Utilización de un computador Linux

Si se dispone de un computador Linux conectado a una VPN de la Universidad, puede conectarse directamente a uno de los computadores asignados al proyecto. Conéctese a `triqui.fi.upm.es` y el sistema le asignará uno al azar. Para ello debe conectarse a la VPN con las credenciales de la Escuela o la Universidad. La información de instalación y conexión de las mismas están contenidas en `https://www.fi.upm.es/?pagina=373` y `https://www.upm.es/UPM/ServiciosTecnologicos/vpn`.

Una vez conectado a la VPN, en la línea de comandos del computador Linux se debe teclear el mandato: `ssh -Y alumno@computador`, en el que `alumno` es el identificador del alumno en la Universidad y el `computador` es uno de los indicados anteriormente. Por ejemplo, el mandato `ssh -Y a.b@triqui.fi.upm.es` intentará establecer una conexión con el computador `triqui` y el usuario proporcionado para el login es `a.b`. A continuación el mandato `ssh` le solicitará la clave de acceso de la Universidad.

A.2. Utilización de escritorios remotos

Si no dispone de un computador Linux o no está satisfecho con el resultado de la opción anterior puede hacer uso del servicio de escritorios virtuales que proporciona la Universidad. La documentación de este servicio está disponible en
`https://docs.cesvima.upm.es/escritorioupm/`.

- Para acceder al servicio debe conectarse a `https://escritorio.upm.es/` y proporcionar su usuario y clave de acceso corporativos de la Universidad.
- La primera vez debe descargarse e instalar en su computador el cliente UDS. El enlace está situado en la parte superior derecha de la página a la que se acaba de acceder.

Escritorio Windows 11

- Vuelva a la página `https://escritorio.upm.es/`, seleccione el escritorio "UPM (Windows 11)" y permita que abra la aplicación udss, que es el cliente que acaba de instalar.
- Una vez que autorice la conexión, accederá a un escritorio Windows 11 en el que debe terminar el servidor X que tiene arrancado, puesto que no está correctamente configurado. Para ello, en la barra de tareas busque la opción *Mostrar iconos ocultos* y pulse sobre el icono X con el botón derecho del ratón. Seleccione la opción *Exit*.
- Vuelva a arrancar el gestor de ventanas. Busque la aplicación *Xlaunch* en las aplicaciones de Windows. Arránquelo y mantenga todas las opciones por defecto excepto *Disable access control* que debe marcar.

  En este computador virtual busque la aplicación `putty` y ánclela en la barra de tareas para que sea más fácil su ejecución posterior.
- Ejecute `putty`. Busque la aplicación `putty` y ejecútela. Para poder utilizar el gestor de ventanas gráfico X-Window es necesario habilitar la opción *Enable X11 forwarding* que está disponible en la pestaña *Connection/SSH/X11*. En este entorno es habitual que la opción ya esté habilitada.
- Conéctese a cualquiera de los computadores triqui del centro de cálculo de la Escuela, accediendo a `triqui.fi.upm.es`. El sistema le asignará uno de ellos al azar. Almacene la sesión, proporcionando un nombre y pulsando el botón `Save`, para que sea más fácil conectarse en sesiones futuras. Las credenciales de acceso que debe proporcionar son las que utiliza habitualmente para acceder a los computadores del centro de cálculo de la Escuela.
- Una utilidad que le será útil será copiar y pegar texto. En este entorno deberá seleccionar con el ratón el texto que quiere copiar y para pegarlo en una de las ventanas del gestor X, deberá hacerlo con el botón central del ratón.
- En este momento puede teclear el comando del emulador `bsvc &`, el ensamblador `68kasm` o una consola de comandos `xterm &`.
- En el computador virtual tiene disponible las aplicaciones `FileZilla` y `WinScp`. Cualquiera de ellas permite transferir ficheros entre el computador triqui y el computador virtual. En el escritorio virtual están disponibles tanto el OneDrive corporativo de la Universidad como el disco local del usuario.

Escritorio Linux

- Como alternativa a las opciones anteriores, si se siente más cómodo en un escritorio Linux, puede seleccionarlo: "UPM (Ubuntu)".
- Abra un terminal pulsando el botón derecho del ratón sobre el escritorio: "Abrir un terminal aquí".
- Conéctese al triqui tal y como se indica en la sección A.1.

- En este momento puede teclear el comando del emulador `bsvc &`, el ensamblador `68kasm` o una consola de comandos `xterm &`.
- Para copiar un texto, basta con que lo seleccione con el ratón y para pegarlo actúe sobre el botón central del ratón.
- Si desea transferir ficheros entre el computador virtual utilice el comando scp.
- En la sección "Carpeta Personal→thinclient_drives"tiene disponibles los discos locales de su computador que puede utilizar para transferir ficheros entre el computador virtual y su computador.

Para despedirse de la sesión debe ejecutar los mismos pasos que siguió para realizar la conexión, pero de forma inversa.

- Cierre el emulador bsvc.
- Cierre todas las consolas que haya abierto.
- Desconecte la sesión del escritorio virtual.

Apéndice B

Instalación del entorno de la práctica en un computador con sistema operativo Linux

El entorno de la práctica compuesto por el ensamblador 68kasm y el emulador bsvc, se puede instalar en un computador personal con un sistema operativo Linux.

Además del sistema operativo Linux se debe tener instalado el sistema de ventanas X y el entorno Tcl/Tk.

B.1. Instalación en Linux

La versión que se proporciona se ha probado en algunas distribuciones de Linux, por lo que se recomienda intentar esta instalación antes de compilar los fuentes de la distribución. El entorno se ha compilado para ser instalado en el directorio `/usr/local`.

B.1.1. Obtención del entorno

La distribución de versiones compiladas para plataformas Linux se realiza conectándose al URL `http://www.datsi.fi.upm.es/docencia/Arquitectura_09/Proyecto_E_S/`

B.1.2. Instalación del paquete

Se puede instalar siguiendo los pasos siguientes:

- Obtenga el fichero con la distribución de bsvc para Linux ya compilada:

`http://www.datsi.fi.upm.es/docencia/Arquitectura_09/Proyecto_E_S/bsvc-2.1+_Estatica.tar.gz`

  y cópiela en un directorio temporal (`/tmp`).

- Instálelo utilizando la utilidad `tar` como administrador de sistema, para ello sitúese en la raíz del sistema de ficheros:

```
cd /
umask 22
sudo tar zxvf /tmp/bsvc-2.1+_Estatica.tar.gz
```

- Asegúrese de que el directorio `/usr/local/bsvc/bin` está en la variable PATH. Si no es así, no podrá ejecutar el simulador. Si utiliza el intérprete de comandos bash, se debe incorporar este directorio a dicha variable añadiendo la siguiente línea al final del fichero `.profile` o `.bashrc` de la raíz de la cuenta desde la que se desee ejecutar bsvc:

  ```
  export PATH=$PATH:/usr/local/bsvc/bin
  ```

- Abra una nueva sesión y ejecute los comandos `bsvc` y `68kasm`. Si no funciona, compruebe que los ejecutables están instalados en el directorio `/usr/local/bsvc/bin` y que tienen permisos de ejecución. Además debe tener instaladas las versiones correctas de TCL y TK. Si no las tiene correctamente instaladas, al ejecutar el programa `bsvc` aparece el error:

  ```
  /usr/local/bsvc/bin/bsvc: 7: exec: wish: not found
  ```

  Instálelas correctamente y el problema desaparecerá. En la sección de Anuncios de la página del proyecto puede encontrar soluciones a los problemas más habituales.

Si no se notifica ningún error, el entorno de prácticas queda instalado. Los programas ejecutables, la documentación y los ejemplos se encuentran en `/usr/local/bsvc`.

Si se produce algún error en la instalación o no puede ejecutar el programa siga las indicaciones de la sección B.2 o conéctese a la Facultad y realice la práctica en alguno de los computadores del centro de cálculo según se indica en el apéndice A.

B.2. Compilación de bsvc en Linux

En esta sección se describe el procedimiento para instalar el entorno a partir de los ficheros fuente. Para compilar los fuentes se necesita el compilador de C++ gcc 2.7.2 o superior y para ejecutarlo tcl 7.5 y tk 4.1. El procedimiento que se describe se ha verificado con la versión de gcc 2.7.2.1 y se ejecuta correctamente con tcl 7.6 y tk 4.2.

B.2.1. Obtención del entorno

Siga lo indicado en la sección B.1.1 pero, en este caso, el fichero se llama `bsvc-2.1-src.tar.gz`.

B.2.2. Instalación del paquete

Se puede instalar siguiendo los pasos siguientes:

- Entre en el computador Linux como administrador (root).
- Si tiene el fichero `bsvc-2.1-src.tar.gz` en un disquete con formato MS-DOS insértelo y cópielo en un directorio de trabajo (por ejemplo `/tmp`). Para ello, ya que el sistema de ficheros MS-DOS habrá acortado el nombre, teclee:

```
mcopy a:/bsvc-2~1.gz /tmp/bsvc-2.1-src.tar.gz
```

- Sitúese en el directorio de trabajo y descomprima el fichero:

```
cd /tmp
tar zxvf bsvc-2.1-src.tar.gz
```

- Sitúese en el directorio que se acaba de crear (`bsvc-2.1`):

```
cd bsvc-2.1
```

- En este directorio existe un fichero llamado `Unix.doc`, donde se describe el procedimiento de compilación, instalación y comprobación del entorno.

El procedimiento de instalación es sencillo y lo único a tener en cuenta es que, si bien el directorio de instalación (`INSTALL_DIR`) puede ser cualquiera, debe estar en el `path` para que bsvc se ejecute correctamente.

Apéndice C

Depuración de fallos que se manifiesten como excepciones en el procesador MC68000

Una de las funciones del sistema operativo es capturar las excepciones que se producen debido al mal funcionamiento de los programas. Cuando se desarrolla código para un computador sin sistema operativo ("computador desnudo") estas excepciones no se manejan a menos que lo haga el propio programa.

Las excepciones más frecuentes se pueden producir con un procesador de la familia M68000 cuando se desarrolla código para un computador desnudo son:

Bus error: se produce cuando se direcciona fuera del rango de la memoria física disponible.

Address error: se produce cuando se utiliza una dirección impar para acceder a una palabra o doble palabra (error de alineamiento).

Illegal instruction: se produce cuando se intenta ejecutar un código de instrucción inexistente. Si el programa se ha ensamblado correctamente se suele producir cuando se pretende ejecutar datos.

Privilege violation: se produce cuando se intenta ejecutar en modo usuario una instrucción privilegiada.

Una buena práctica consiste en instalar manejadores para estas excepciones. El programa de la figura C.1 instala estos manejadores y genera una excepción de error de dirección.

El manejador de instrucción ilegal se ha instalado también en las entradas de la tabla de vectores (véase la figura 1.5) correspondientes a las instrucciones para los coprocesadores, ya que el computador simulado carece de ellos.

Los manejadores se limitan a una instrucción `BREAK` que detiene el procesador simulado y una instrucción `NOP` cuyo propósito es servir de separador. De esta forma si se produce cualquiera de las excepciones manejadas se detiene el simulador y el `PC` señalará a la instrucción siguiente a `BREAK` del manejador correspondiente.

```
        ORG     $0
        DC.L    $8000           Stack pointer value after a reset
        DC.L    START           Program counter value after a reset


        ORG     $2000           Start at location 2000 Hex

START   MOVE.L #BUS_ERROR,8     Install bus error handler
        MOVE.L #ADDRESS_ER,12   Install address error handler
        MOVE.L #ILLEGAL_IN,16   Install illegal instruction handler
        MOVE.L #PRIV_VIOLT,32   Install privilege violation handler
        MOVE.L #ILLEGAL_IN,40   Install illegal instruction handler
        MOVE.L #ILLEGAL_IN,44   Install illegal instruction handler

        LEA     EVEN,A0
        ADDQ.L  #1,A0
        CLR.W   (A0)            Generate an address error exception

        ORG     $3000
EVEN    DS.W    1               Reserve one word for even

BUS_ERROR:  BREAK               Bus error handler
            NOP
ADDRESS_ER: BREAK               Address error handler
            NOP
ILLEGAL_IN: BREAK               Illegal instruction handler
            NOP
PRIV_VIOLT: BREAK               Privilege violation handler
            NOP
```

Figura C.1: Programa con código para el manejo de excepciones

C.1. Identificación la instrucción que provocó la excepción

En el programa anterior resulta muy fácil identificar la instrucción que provocó la excepción. Sin embargo en un programa más complejo esto no resulta tan obvio. La solución es bastante fácil: basta con indagar en el marco de pila que creó la secuencia de procesamiento de la excepción.

El formato del marco de pila para todas las excepciones excepto los errores de bus y dirección se muestra en la figura C.2.

En este marco de pila se almacena el `PC` que apuntará a la instrucción que provocó la excepción. Si se tratase de una interrupción el `PC` apuntaría a la siguiente instrucción. El porqué de esta diferencia es clara, el procesador espera a que se complete la instrucción para reconocer una interrupción, sin embargo si se trata de una excepción no se puede proceder a la ejecución de la instrucción. El caso más claro es quizás el de una instrucción ilegal, es obvio que el procesador no puede ejecutarla.

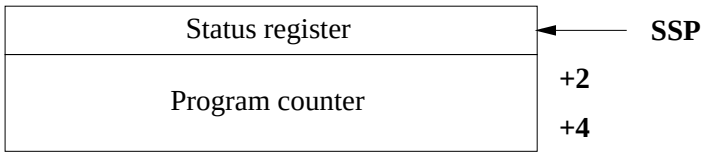


Figura C.2: Marco de pila de procesamiento de excepción

El formato del marco de pila para las **excepciones de errores de bus y dirección** se muestra en la figura C.3.

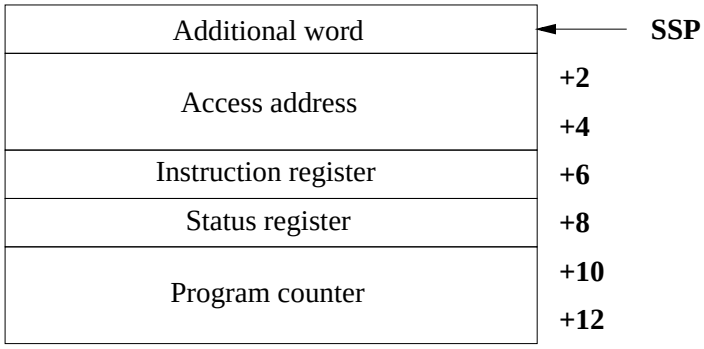


Figura C.3: Marco de pila de procesamiento de las excepciones de *bus* y *address error*

En este marco de pila se almacena mucha más información para identificar claramente cuando se produjo la excepción. Téngase en cuenta que la ejecución de una instrucción puede suponer varios accesos a memoria. Así se salva el `PC` y el `SR` como en el caso anterior pero además el contenido del registro de instrucción, la dirección que provocó la excepción y una palabra adicional donde se indica, entre otras cosas, si el ciclo de bus era de lectura o escritura.

Desafortunadamente, el simulador falla y el campo correspondiente a la dirección que provocó la excepción siempre está a cero. En cualquier caso, la información restante es suficiente para identificar el problema.

C.2. Ejemplo

Para ilustrar el procedimiento utilizaremos el programa de la figura C.1.

Si ejecutamos dicho programa obtendremos los datos en la ventana de manejo del simulador que se muestran en la figura C.4.

La zona de traza nos muestra un mensaje diciendo que la ejecución del programa se ha detenido al alcanzar una instrucción `BREAK`. Si se echa un vistazo a la ventana de listado de programa (figura C.5), se observa que el `PC` apunta a la instrucción siguiente de la instrucción `BREAK` del manejador de la excepción de error de dirección (`ADDRESS_ER`).

Lo que ha ocurrido es que se ha producido una excepción de error de dirección, se ha ejecutado el manejador correspondiente y se ha alcanzado la instrucción `BREAK` que contiene dicho manejador.

Para obtener información de dónde se ha producido dicha excepción, se puede indagar en el

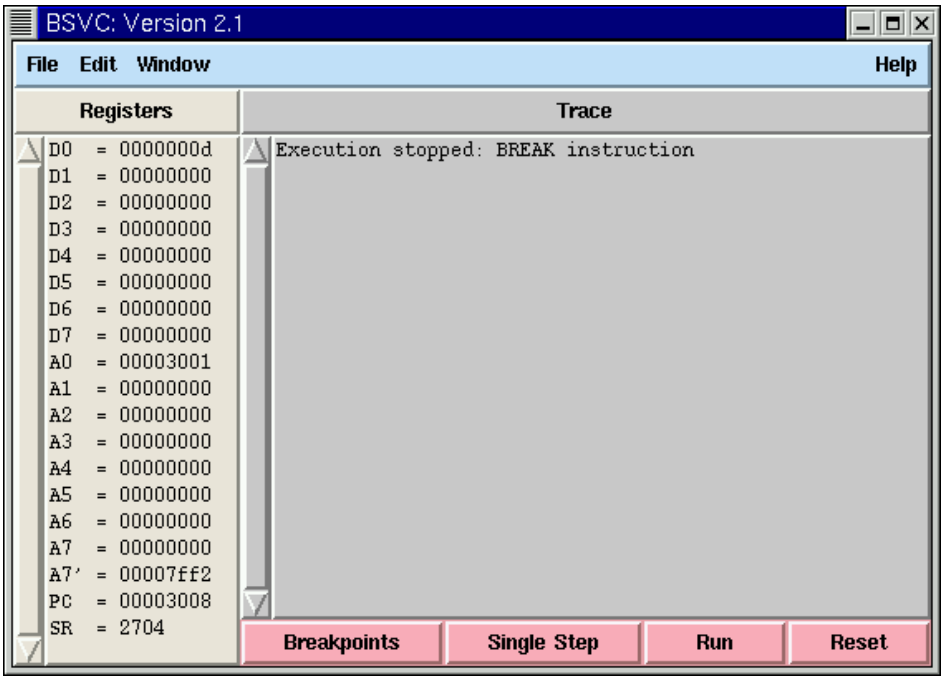


Figura C.4: Ventana del simulador

correspondiente marco de pila. En la ventana de manejo del simulador se muestra el contenido del puntero de pila de supervisor `A7'`. Por lo tanto, se puede mostrar el correspondiente marco de pila mediante la ventana de listado de posiciones de memoria. El resultado se refleja en la figura C.6.

El contenido de la posición apuntada por `A7'= 007ff2` y las 14 siguientes posiciones de memoria se corresponden con el marco de pila de procesamiento de la excepción de error de dirección que se muestra en la figura C.3.

Additional word (00 09): El bit 5 a 1 indica que se abortó un ciclo de lectura, si estuviese a 0 se trataría de un ciclo de escritura.

Access address (00 00 00 00): Un fallo en el simulador provoca que sea siempre cero.

Instruction register (42 50): Contiene el código de operación de la instrucción que provocó la excepción (`CLR.W (A0)`)

Status register (27 04): Indica que se encontraba en modo supervisor, con las interrupciones inhibidas y el flag `Z` a 1.

Program counter (00 00 20 3a): Apunta a la instrucción que provocó la excepción más dos, ya que el contador de programa se incrementa durante la ejecución de instrucciones.

Estos datos son más que suficientes para identificar la instrucción que provocó la excepción y a partir de aquí identificar el fallo del programa.

Por último, nótese que el hecho de que la dirección accedida esté siempre a cero no impide determinarla, puesto que se conoce el contenido de los registros. En este caso es obvio

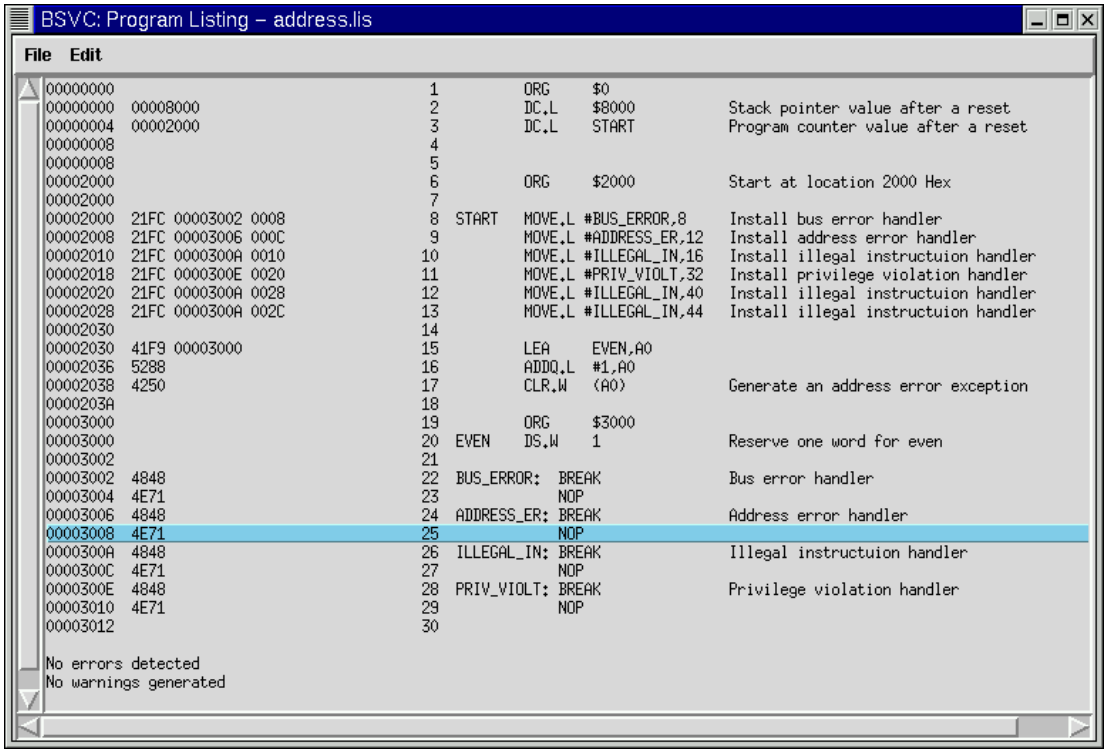


Figura C.5: Ventana de listado del programa

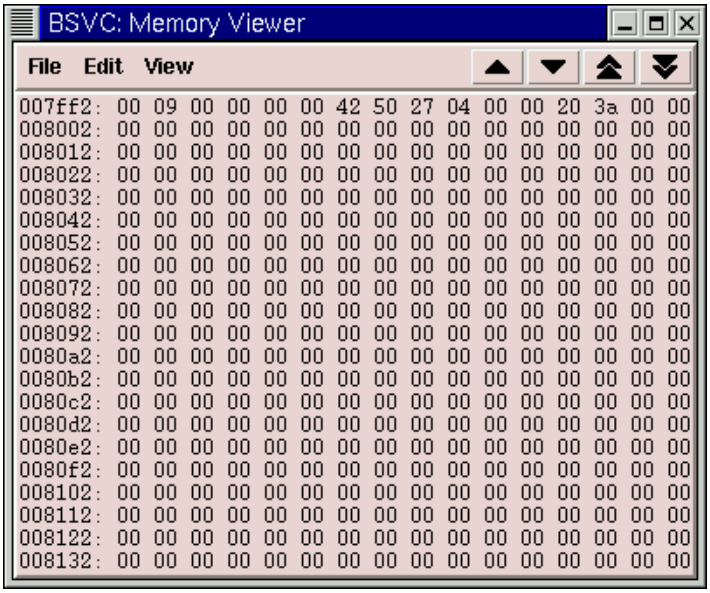


Figura C.6: Contenido del marco de pila para la excepción *Address Error*

determinar que esa dirección es `3001` pero en otros casos con un modo de direccionamiento más complejo se necesitará trabajo extra.