Práctica final 2021: CDB

Estructura de computadores I
Curso: 1º - 2020/2021
Grupo 03

Alejandro Rodríguez Arguimbau
alejandro.rodriguez7@estudiant.uib.cat
Sergi Mayol Matos
sergi.mayol1@estudiant.uib.cat

Índice

1. Introducción

Un emulador permite que un ordenador pueda ejecutar un programa para una máquina diferente. En esta práctica se pide crear un programa, en ensamblador del 68k, que sea capaz de emular la ejecución de un conjunto de instrucciones dadas. Eso implica que la máquina a emular realice las fases de fetch, decodificación y ejecución. En concreto, hemos emulado la CBD (Computador Didáctico Balear), esta máquina cuenta con un conjunto de registros e instrucciones de 16 bits, que son los siguientes: seis registros, R0-R5, de propósito general para operaciones de tipo ALU; los registros T6 y T7, que se utilizan como interfaz con la memoria; el registro ESR el cual contiene los flags Z, N y C, que se irán actualizando en la ejecución según la instrucción emulada; y un vector EPROG que contiene las instrucciones codificadas del emulador indicadas por el usuario, que se irán ejecutando según el contador del programa del emulador, EPC.

2. Explicación general del trabajo realizado

En el emulador de la máquina CDB, para emular correctamente las instrucciones es necesario la implementación de tres fases, fase de fetch, fase de decodificación y fase de ejecución.

2.1 Fase de Fetch

En la fase de fetch el emulador tiene que ocuparse de cargar la siguiente instrucción a ejecutar, indicada por el contador del programa (PC), en el registro de instrucción y actualizar el valor del contador de programa del emulador (EPC) para que la siguiente vez que se acceda a la instrucción a realizar sea la siguiente instrucción. Para la implementación de la emulación de la fase de fetch en nuestro programa, hemos utilizado el registro de datos D4 para cargar el contenido de la posición de memoria del EPC, para poder multiplicar este registro por dos. Para cargar en el EIR la siguiente instrucción a ejecutar, indicada por el EPC, hemos empleado un direccionamiento indexado completo que suma el contenido del registro de datos D4 al registro de direcciones A0, el cual contiene la dirección de la primera instrucción del EPROG, de donde obtendremos la dirección a cargar en el EIR. Finalmente, el valor del EPC es incrementado en uno, para que ya apunte a la siguiente einstrucción a realizar.

2.2 Fase de Decodificación

La fase de decodificación consiste en preparar la pila y saltar a la subrutina de librería llamada DECOD, en la que analizamos los bits de la einstrucción contenida en EIR. Esta einstrucción la introducimos en la pila correctamente, reservando un espacio, y la analizamos mediante la instrucción BTST. La instrucción BTST comprueba el valor de cada uno de los bits que nosotros queremos. Dependiendo del valor de cada bit, se hará un salto a diferentes etiquetas hasta coincidir con el valor de la instrucción Id.

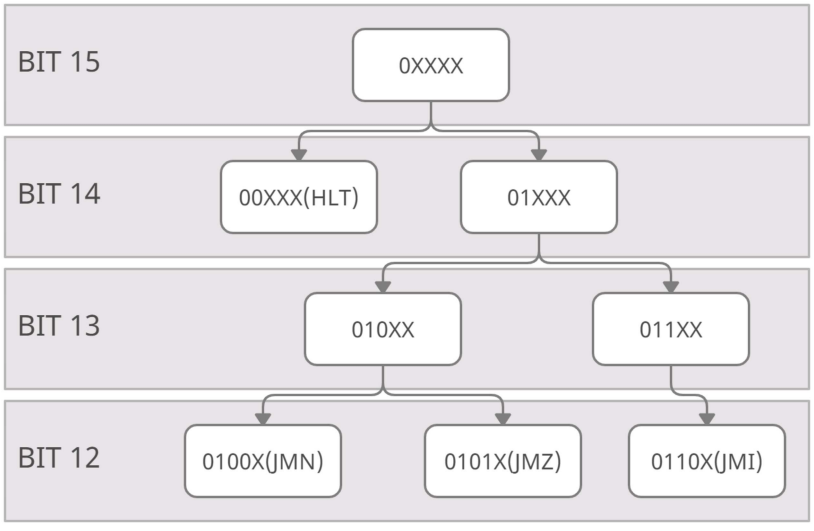
2.3 Fase de Ejecución

En la fase de ejecución, se realiza la ejecución de las instrucciones de la máquina CDB a partir de las etiquetas del repertorio de la CDB, donde dependiendo del resultado de la decodificación se realizará un salto a la etiqueta correspondiente a la instrucción a ejecutar. Dependiendo de las instrucciones de cada instrucción se han implementado una serie de subrutinas y acciones necesarias para el funcionamiento correspondiente a la CDB. En concreto, para hacerlo posible hemos implementado las subrutinas: Xa y Xb, para obtener la dirección de los operandos, y EFLAGZ, EFLAGC, EFLAGN, para la actualización de los flags del emulador.

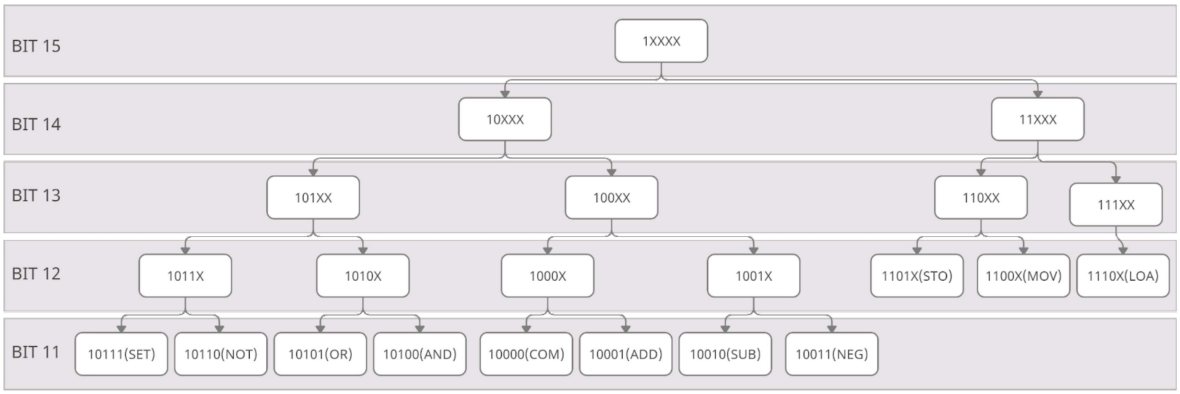
3. Descripción subrutina de decodificación

La subrutina de decodificación es una subrutina de librería, que tiene como función hacer el análisis bit a bit de la instrucción codificada de la máquina CDB para, dependiendo del contenido, determinar de qué instrucción se trata. Para decodificar las instrucciones hay que analizar, como máximo, los cinco bits más significativos de cada instrucción. A través de la pila se obtiene la instrucción a analizar, donde previamente a la llamada de esta subrutina se habrá reservado espacio para almacenar el resultado de la decodificación y se habrá guardado el EIR, para saber qué instrucción hay que examinar, el cual se copiará en el registro de datos D0 y lo primero que hacemos es mirar el bit 15, en el caso que sea 1 se realiza un salto al bucle de decodificación para las instrucciones de bit 15 = 1, en caso contrario se procederá a saltar al bucle de decodificación para bit 15 = 0, una vez ya sepa el valor de bit más significativo se procederá a analizar el siguiente bit, esto se efectuará múltiples veces hasta obtener el resultado. A continuación, en los apartados 3.1 y 3.2, mediante dos representaciones gráficas se muestra el funcionamiento de esta subrutina. En los casos que aparezca una X en los diagramas significa que el bit no ha sido analizado y no se sabe su valor, en caso contrarío, si el bit ya ha sido analizado se muestra su valor. Como se ha mencionado anteriormente solo es necesario mirar los cinco primeros bits de cada instrucción del CDB.

3.1 Si BIT 15 = 0



3.2 Si BIT 15 = 1



4. Tabla de subrutinas

Etiqueta:	Función:	Entrada:	Salida:
EFLAGZ	Subrutina de usuario que actualiza el flag N. Antes de ejecutar esta subrutina, SR se almacena en D2 y ESR en D3. En esta subrutina se copia el valor del bit de D2 correspondiente a N a su posición en D3.	D2 y D3	D3
EFLAGC	Subrutina de usuario que actualiza el flag C. Antes de ejecutar esta subrutina, SR se almacena en D2 y ESR en D3. En esta subrutina se copia el valor del bit de D2 correspondiente a C a su posición en D3.	D2 y D3	D3
EFLAGN	Subrutina de usuario que actualiza el flag N. Antes de ejecutar esta subrutina, SR se almacena en D2 y ESR en D3. En esta subrutina se copia el valor del bit de D2 correspondiente a N a su posición en D3.	D2 y D3	D3
Xa	Subrutina de usuario que decodifica qué registro es el operando A, a partir del registro D0 (contiene la einstrucción), y lo guarda en A4.	D0	A4
Xb	Subrutina de usuario que decodifica qué registro es el operando B, a partir del registro D0 (contiene la einstrucción), y lo guarda en A3.	D0	A3
DECOD	Subrutina de librería encargada de la decodificación de la einstrucción a realizar. La explicación detallada se encuentra en el apartado 3.	A7	A7

5. Tabla de registros del 68k

Registro:	Función:
D0	Registro de datos empleado para el almacenamiento de las instrucciones del emulador a ejecutar del CDB. Concretamente, se utiliza en varias instrucciones durante la fase de ejecución, para poder operar sobre el EIR sin modificarlo, y en la fase de decodificación para poder decodificar la instrucción a ejecutar.
D1	Registro de datos donde se almacena el resultado de la decodificación para saltar a la fase de ejecución.
D2	Registro de datos empleado como interfaz entre el programa principal y las subrutinas de actualización de flags (EFLAGZ, EFLAGC, EFLAGN). En este se guarda el registro de estados SR del 68000 para después de la operación realizada para su posterior uso en las subrutinas.
D3	Registro de datos empleado como interfaz entre el programa principal y las subrutinas de actualización de flags (EFLAGZ, EFLAGC, EFLAGN). En este se guarda el registro de estados ESR del emulador, para posteriormente ser modificado según los flags obtenidos tras la operación realizada en la ejecución.
D4	Registro de datos empleado en la fase de fetch. En este se guarda el contenido del EPC, para más adelante multiplicarlo por dos, así obteniendo el valor correcto. Seguidamente se utiliza en un direccionamiento indexado completo para acceder a la instrucción indicada por el EPC.
D5	Registro de datos empleado como registro de datos auxiliar para operaciones durante la fase de ejecución.
D6	Registro de datos empleado como registro de datos auxiliar para operaciones durante la fase de ejecución.
D7	Sin uso.
A0	Registro de direcciones empleado para el almacenamiento de la dirección de memoria del vector de instrucciones EPROG durante la fase de fetch. Al inicio del programa se inicializa con la dirección del EPROG como valor, para posteriormente usarlo en un direccionamiento indexado completo para mover el contenido de la dirección obtenida al sumar el contador de programa del emulador EPC y el valor de A0.
A1	Registro de direcciones empleado en el salto a la fase de ejecución para ir a la instrucción correspondiente.
A2	Sin uso.
A3	Registro de direcciones empleado para el almacenamiento del operando B (Xb) en la subrutina Xb y en la fase de ejecución en instrucciones que necesitan un operando B.
A4	Registro de direcciones empleado para el almacenamiento del operando A

	(Xa) en la subrutina Xa y en la fase de ejecución en instrucciones que necesitan un operando A.
A5	Registro de direcciones empleado para el guardado de la dirección de M en las instrucciones LOA y STO durante la fase de ejecución.
A6	Registro de direcciones empleado para el guardado del EPROG durante las operaciones realizadas en las instrucciones LOA y STO durante la fase de ejecución.
A7	Registro de direcciones empleado para el acceso a la pila.

6. Tabla de variables adicionales

No hemos implementado ninguna variable adicional.

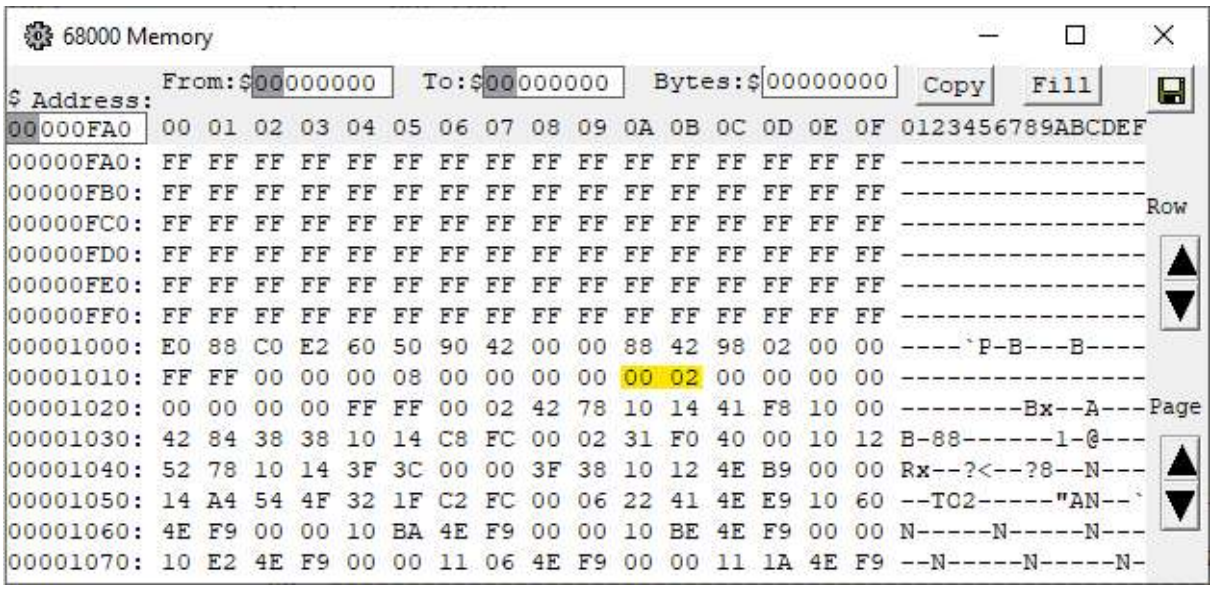
7. Conjunto de pruebas

A continuación se mostrará el conjunto de pruebas que hemos realizado para asegurarnos de que el emulador está correctamente implementado.
En este ejemplo, ejecutamos sobre nuestro emulador el siguiente programa:

Dirección @CDB	Ensamblador sin etiquetas	Instrucciones codificadas	Hex
0:	LOA 8,T7	1110000010001000	E088
1:	MOV T7,R2	1100000011100010	C0E2
2:	JMI 5	0110000001010000	6050
3:	SUB R2,R2	1001000001000010	9042
4:	HLT	0000000000000000	0000
5:	ADD R2,R2	1000100001000010	8842
6:	NEG R2	1001100000000010	9802
7:	HLT	0000000000000000	0000
8:	-1	1111111111111111	FFFF

En este programa, en el LOA se carga la posición de memoria 8 en T7, después mueve T7 a R2, del JMI hace un salto a la posición 5 de la memoria. Ahora suma los registros R2 y R2, hace la operación de negación de R2 y finalmente, se para la máquina.
Como resultado de este programa, en la posición de memoria del 68K correspondiente a ER2, (en este caso, @101AHex) debe contener el valor 2 Dec = 0002 Hex.

Como podemos observar en la siguiente imagen, este valor aparece en la memoria y comprobamos que el emulador funciona correctamente.



8. Conclusiones

Para la realización de esta práctica, hemos necesitado aprender, leer e informarnos sobre las instrucciones y el funcionamiento de la máquina. De esta manera, los conocimientos relacionados con el 68K se han consolidado idóneamente. Gracias a ello, hemos podido emular una máquina elemental y hacer que funcione correctamente. Respecto al trabajo realizado, nos hemos dado cuenta de que para crear algo que a simple vista parece fácil, se necesitan muchas horas de búsqueda, investigación y evaluación. También hemos podido observar más profundamente, el funcionamiento de cada fase; fase de fetch, fase de decodificación y fase de ejecución. Cabe mencionar, el aprendizaje de la subrutina de librería necesaria para poder desempeñar correctamente la práctica. Desde nuestro punto de vista, el trabajo ha sido imprescindible para reforzar y consolidar nuestros conocimientos del 68K. Por último, agradecer al profesor de prácticas, ya que, nos resolvió todas las dudas y nos ayudó en todo momento.

9. Código fuente

```
*-----------------------------------------------------------
* Title      : PRAFIN21
* Written by : Alejandro Rodriguez Arguimbau y Sergi Mayol Matos
* Date       : 31/05/2021
* Description: Emulador de la CDB
*-----------------------------------------------------------
    ORG $1000
EPROG:  DC.W $B803,$BFFC,$E0F0,$C0C0,$50C0,$E108,$C0E1,$50C0,$8803
        DC.W $8881,$50C0,$6080,$C066,$D110,$0000,$0004,$0003,$0000
EIR:    DC.W 0 ;eregistro de instruccion
EPC:    DC.W 0 ;econtador de programa
ER0:    DC.W 0 ;eregistro R0
```

```
ER1:    DC.W 0 ;eregistro R1
ER2:    DC.W 0 ;eregistro R2
ER3:    DC.W 0 ;eregistro R3
ER4:    DC.W 0 ;eregistro R4
ER5:    DC.W 0 ;eregistro R5
ET6:    DC.W 0 ;eregistro T6
ET7:    DC.W 0 ;eregistro T7
ESR:    DC.W 0 ;eregistro de estado (00000000 00000NCZ)

START:
   CLR.W EPC
   LEA.L EPROG,A0          ;Cargamos @ del EPROG(1000)
FETCH:
   ;--- IFETCH: INICIO FETCH
      ;*** En esta seccion debeis introducir el codigo necesario para cargar
      ;*** en el EIR la siguiente instruccion a ejecutar, indicada por el EPC
            ;*** y dejar listo el EPC para que apunte a la siguiente instruccion
            CLR.L D4            ;Borramos [D4] para no tener valor el anterior
            MOVE.W EPC,D4       ;Movemos EPC a un registro de datos
            MULU #2,D4          ;Multiplicamos por 2 para obtener el valor correcto
            MOVE.W 0(A0,D4),EIR ;Calcula el desplazamiento del EPROG según el EPC
            ADDQ.W #1,EPC       ;Sumamos 1 al EPC
   ;--- FFETCH: FIN FETCH
   ;--- IBRDECOD: INICIO SALTO A DECOD
      ;*** En esta seccion debeis preparar la pila para llamar a la subrutina
      ;*** DECOD, llamar a la subrutina, y vaciar la pila correctamente,
      ;*** almacenando el resultado de la decodificacion en D1
   MOVE.W #0,-(A7)     ;Reservar espacio en la pila para el resultado
   MOVE.W EIR,-(A7)    ;Paso de parametro EIR a la pila
   JSR DECOD           ;Salto a la subrutina
   ADDQ.W #2,A7        ;vaciar pila
   MOVE.W (A7)+,D1     ;Guardar resultado en D1
   ;--- FBRDECOD: FIN SALTO A DECOD
   ;--- IBREXEC: INICIO SALTO A FASE DE EJECUCION
      ;*** Esta seccion se usa para saltar a la fase de ejecucion
   MULU #6,D1
   MOVEA.L D1,A1
   JMP JMPLIST(A1)
JMPLIST:
   JMP EHLT
   JMP EJMN
   JMP EJMZ
   JMP EJMI
   JMP ECOM
   JMP EADD
   JMP ESUB
   JMP ENEG
   JMP EAND
```

```
    JMP EOR
    JMP ENOT
    JMP ESET
    JMP EMOV
    JMP ESTO
    JMP ELOA
    ;--- FBREXEC: FIN SALTO A FASE DE EJECUCION
    ;--- IEXEC: INICIO EJECUCION
        ;*** En esta seccion debeis implementar la ejecucion de cada einstr.
;DETIENE LA MÁQUINA
EHLT:
    SIMHALT
;SALTO SI EL FLAG N = 1
EJMN:
    MOVE.W ESR,D3    ;Movemos el ESR a D3 para operar
    BTST.L #2,D3     ;Comprobamos el valor de N
    BNE Nuno         ;Luego cambio a subrutina de salto flag N
    JMP FETCH        ;N=0
Nuno: ;N=1
    AND.W #$0FF0,EIR ;Mascara para obtener M
    MOVE.W EIR,D5    ;EIR se copia en D5 para operar
    LSR.L #4,D5      ;Desp. der., obtenemos la @ de M
    MOVE.W D5,EPC    ;Cargar @ de M en el EPC
    JMP FETCH
;SALTO SI EL FLAG Z = 1
EJMZ:
    MOVE.W ESR,D3    ;Movemos el ESR a D3 para operar
    BTST.L #0,D3     ;Comprobamos valor de Z
    BNE Zuno
    JMP FETCH        ;Z=0
Zuno: ;Z=1
    AND.W #$0FF0,EIR ;Mascara para obtener M
    MOVE.W EIR,D5    ;EIR se copia en D5 para operar
    LSR.L #4,D5      ;Desp. der., obtenemos la @ de M
    MOVE.W D5,EPC    ;Cargar @ de M en el EPC
    JMP FETCH
;SALTO INCONDICIONAL
EJMI:
    AND.W #$0FF0,EIR ;Mascara para obtener M
    MOVE.W EIR,D5    ;EIR se copia en D5 para operar
    LSR.L #4,D5      ;Desp. der., obtenemos la @ de M
    MOVE.W D5,EPC    ;Cargar @ de M en el EPC
    JMP FETCH
;CMP RESTANDO [B]-[A]
ECOM:
    MOVE.W EIR,D0    ;Copia de EIR a D0 para operar
    JSR Xb           ;Subrutinas Xa y Xb para decodificar operandos
    JSR Xa
```

```
    MOVE.W (A3),D5   ;Guardamos el [A3] en D5 para operar
    CMP.W (A4),D5    ;CMP de los operandos
    MOVE.W SR,D2     ;Guardamos el SR en D2 para operar
    MOVE.W ESR,D3    ;Guardamos el ESR en D3 para operar
    ;ACTUALIZACIÓN DE LOS EFLAGS
    JSR EFLAGN       ;Actualización eflag N
    JSR EFLAGC       ;Actualización eflag C
    JSR EFLAGZ       ;Actualización eflag Z
    MOVE.W D3,ESR    ;Guardamos los eflags actualizados en ESR
    JMP FETCH
;SUMA [B]+[A] Y LO GUARDA EN B
EADD:
   ;OPERACIÓN B + A
    MOVE.W EIR,D0    ;Copia de EIR a D0 para operar
    JSR Xb           ;Subrutinas Xa y Xb para decodificar operandos
    JSR Xa
    MOVE.W (A3),D5   ;Guardamos el [A3] en D5 para operar
    ADD.W (A4),D5    ;Sumamos el [A4] con D5
    ;ACTUALIZACIÓN DE LOS EFLAGS
    MOVE.W SR,D2     ;Guardamos el SR en D2 para operar
    MOVE.W ESR,D3    ;Guardamos el ESR en D3 para operar
    JSR EFLAGN       ;Actualización eflag N
    JSR EFLAGC       ;Actualización eflag C
    JSR EFLAGZ       ;Actualización eflag Z
    MOVE.W D3,ESR    ;Guardamos los eflags actualizados en ESR
    MOVE.W D5,(A3)   ;Movemos el resultado operación
    JMP FETCH
;RESTA [B]-[A] Y LO GUARDA EN B
ESUB:
    ;OPERACIÓN B - A
    MOVE.W EIR,D0    ;Copia de EIR a D0 para operar
    JSR Xb           ;Subrutinas Xa y Xb para decodificar operandos
    JSR Xa
    MOVE.W (A3),D5   ;Guardamos el [A3] en D5 para operar
    NOT.W D5         ;Cambio de signo del operando B
    ADDQ.W #1,D5     ;(B¯ + 1)
    ADD.W (A4),D5    ;Sumamos,A + (B¯ + 1), el [A4] con D5
    ;ACTUALIZACIÓN DE LOS EFLAGS
    MOVE.W SR,D2     ;Guardamos el SR en D2 operar
    MOVE.W ESR,D3    ;Guardamos el ESR en D3 operar
    JSR EFLAGN       ;Actualización eflag N
    JSR EFLAGC       ;Actualización eflag C
    JSR EFLAGZ       ;Actualización eflag Z
    MOVE.W D3,ESR    ;Guardamos los eflags actualizados en ESR
    MOVE.W D5,(A3)   ;Movemos el resultado operación
    JMP FETCH
;CAMBIA EL SIGNO [B] Y LO GUARDA EN B
ENEG:
```

```
    ;NEGACIÓN OPERANDO B
    MOVE.W EIR,D0    ;Copia de EIR a D0 para operar
    JSR Xb           ;Subrutina Xb para decodificar operando B
    MOVE.W (A3),D5   ;Guardamos el [A3] en D5 para operar
    NEG.W D5         ;Cambio de signo del operando B
    ;ACTUALIZACIÓN DE LOS EFLAGS
    MOVE.W SR,D2     ;Guardamos el SR en D2 para operar
    MOVE.W ESR,D3    ;Guardamos el ESR en D3 para operar
    JSR EFLAGN       ;Actualización eflag N
    JSR EFLAGZ       ;Actualización eflag Z
    MOVE.W D3,ESR    ;Guardamos los eflags actualizados en ESR
    MOVE.W D5,(A3)   ;Movemos el resultado operación
    JMP FETCH
;AND BIT A BIT DEL [B] CON EL [A]
EAND:
    MOVE.W EIR,D0    ;Copia de EIR a D0 para operar
    JSR Xb           ;Subrutinas Xa y Xb para decodificar operandos
    JSR Xa
    MOVE.W (A3),D5   ;Guardamos el [A3] en D5 para operar
    MOVE.W (A4),D6   ;Guardamos el [A4] en D6 para operar
    AND.W D5,D6      ;AND bit a bit de operandos A y B
    ;ACTUALIZACIÓN DE LOS EFLAGS
    MOVE.W SR,D2     ;Guardamos el SR en D2 para operar
    MOVE.W ESR,D3    ;Guardamos el ESR en D3 para operar
    JSR EFLAGN       ;Actualización eflag N
    JSR EFLAGZ       ;Actualización eflag Z
    MOVE.W D3,ESR    ;Guardamos los eflags actualizados en ESR
    MOVE.W D6,(A3)   ;Movemos el resultado operación
    JMP FETCH
;OR BIT A BIT DEL [B] CON EL [A]
EOR:
    MOVE.W EIR,D0    ;Copia de EIR a D0 para operar
    JSR Xb           ;Subrutinas Xa y Xb para decodificar operandos
    JSR Xa
    MOVE.W (A3),D5   ;Guardamos el [A3] en D5 para operar
    MOVE.W (A4),D6   ;Guardamos el [A4] en D6 para operar
    OR.W D5,D6       ;OR bit a bit de operandos A y B
    ;ACTUALIZACIÓN DE LOS EFLAGS
    MOVE.W SR,D2     ;Guardamos el SR en D2 para operar
    MOVE.W ESR,D3    ;Guardamos el ESR en D3 para operar
    JSR EFLAGN       ;Actualización eflag N
    JSR EFLAGZ       ;Actualización eflag Z
    MOVE.W D3,ESR    ;Guardamos los eflags actualizados en ESR
    MOVE.W D6,(A3)   ;Movemos el resultado operación
    JMP FETCH
;CAMBIA CADA BIT HACIENDO UN NOT DEL [B]
ENOT:
    MOVE.W EIR,D0    ;Copia de EIR a D0 para operar
```

```
    JSR Xb              ;Subrutina Xb para decodificar operando B
    MOVE.W (A3),D5   ;Guardamos el [A3] en D5 para operar
    NOT.W D5         ;NOT del registro D5(Operando B)
   ;ACTUALIZACIÓN DE LOS EFLAGS
    MOVE.W SR,D2     ;Guardamos el SR en D2 para operar
    MOVE.W ESR,D3    ;Guardamos el ESR en D3 para operar
    JSR EFLAGN       ;Actualización eflag N
    JSR EFLAGZ       ;Actualización eflag Z
    MOVE.W D3,ESR    ;Guardamos los eflags actualizados en ESR
    MOVE.W D5,(A3)   ;Movemos el resultado operación
    JMP FETCH
;EXTENSIÓN DE SIGNO
ESET:
    MOVE.W EIR,D0    ;Copia de EIR a D0 para operar
    JSR Xb           ;Subrutina Xb para decodificar operando B
    MOVE.W EIR,D5    ;Copia de EIR en D5 para operar
    AND.W #$0FFF,D5  ;Mascara a D5
    LSL.L #1,D5      ;Desp. izq. de un bit
    AND.W #$0FF0,D5  ;Mascara a D5
    LSR.L #4,D5      ;Desp. der. y obtenemos K en D5
    BTST.L #7,D5     ;Comprobamos valor bit más significativo
    BNE EXT          ;Bit más sig. = 1, ext. signo
    MOVE.W D5,(A3)   ;Movemos el resultado operación
    JMP ACTEFLAGS    ;Act. de eflags
EXT:
    OR.W #$FF00,D5   ;Or bit a bit a D5 para ext. signo
    MOVE.W D5,(A3)   ;Movemos el resultado operación
ACTEFLAGS:
    MOVE.W SR,D2     ;Guardamos el SR en D2 para operar
    MOVE.W ESR,D3    ;Guardamos el ESR en D3 para operar
    JSR EFLAGN       ;Actualización eflag N
    JSR EFLAGZ       ;Actualización eflag Z
    MOVE.W D3,ESR    ;Guardamos los eflags actualizados en ESR
    JMP FETCH
;COPIA EL [A] EN B
EMOV:
    MOVE.W EIR,D0    ;Copia de EIR a D0 para operar
    JSR Xb              ;Subrutinas Xa y Xb para decodificar operandos
    JSR Xa
    MOVE.W (A4),D6   ;A,Guardamos el [A4] en D6 para operar
    MOVE.W D6,(A3)   ;MOVE.W A,B
    MOVE.W SR,D2     ;Guardamos el SR en D2 para operar
    MOVE.W ESR,D3    ;Guardamos el ESR en D3 para operar
    JSR EFLAGN       ;Actualización eflag N
    JSR EFLAGZ       ;Actualización eflag Z
    MOVE.W D3,ESR    ;Guardamos los eflags actualizados en ESR
    JMP FETCH
;MUEVE EL [Ti] AL OPERANDO DESTINO(M)
```

```
ESTO:
   MOVE.W EIR,D5    ;Copia de EIR en D5 para operar
   AND.W #$0FF0,D5  ;Mascara para obtener M
   LSR.L #4,D5      ;Desp. der., obtenemos la @ de M
   MULS.W #2,D5     ;Mult. D5(M) por dos
   MOVE.W D5,A5     ;Guardamos la @ de M en A5
   MOVE.W EIR,D6    ;Copia de EIR en D6 para operar
   BTST.L #3,D6     ;Comprobamos valor de i
   BNE iT7uno       ;i = 1
   LEA.L EPROG,A6   ;Cargamos @ EPROG en A6
   ADD A6,A5        ;Obtenemos la @ de M en A5
   MOVE.W ET6,(A5)  ;Guardamos ET6 en A5
   JMP FETCH
   iT7uno: ;i = 1
   LEA.L EPROG,A6   ;Cargamos @ EPROG en A6
   ADD.W A6,A5      ;Obtenemos la @ de M en A5
   MOVE.W ET7,(A5)  ;Guardamos ET7 en A5
   JMP FETCH
;MUEVE EL [M] AL OPERANDO DESTINO(Ti)
ELOA:
   MOVE.W EIR,D5    ;Copia de EIR en D5 para operar
   AND.W #$0FF0,D5  ;Mascara para obtener M
   LSR.L #4,D5      ;Desp. der., obtenemos la @ de M
   MULS.W #2,D5     ;Mult. D5(M) por dos
   MOVE.W D5,A5     ;Guardamos la @ de M en A5
   MOVE.W EIR,D6    ;Copia de EIR en D6 para operar
   BTST.L #3,D6     ;Comprobamos valor de i
   BNE iT7dos      ;i = 1
   LEA.L EPROG,A6   ;Cargamos @ EPROG en A6
   ADD.W A6,A5      ;Obtenemos la @ de M en A5
   MOVE.W (A5),ET6  ;Guardamos [A5] en ET6
         ;ACTUALIZAMOS FLAGS
   MOVE.W SR,D2     ;Guardamos el SR en D2 para operar
   MOVE.W ESR,D3    ;Guardamos el ESR en D3 para operar
   JSR EFLAGN       ;Actualización eflag N
   JSR EFLAGZ       ;Actualización eflag Z
   MOVE.W D3,ESR    ;Guardamos los eflags actualizados en ESR
   JMP FETCH
iT7dos: ;i = 1
   LEA EPROG,A6     ;Cargamos @ EPROG en A6
   ADD.W A6,A5      ;Obtenemos la @ de M en A5
   MOVE.W (A5),ET7  ;Guardamos [A5] en ET7
;ACTUALIZAMOS FLAGS
   MOVE.W SR,D2     ;Guardamos el SR en D2 para operar
   MOVE.W ESR,D3    ;Guardamos el ESR en D3 para operar
   JSR EFLAGN       ;Actualización eflag N
   JSR EFLAGZ       ;Actualización eflag Z
   MOVE.W D3,ESR    ;Guardamos los eflags actualizados en ESR
```

```
    JMP FETCH
    ;--- FEXEC: FIN EJECUCION
    ;--- ISUBR: INICIO SUBRUTINAS
        ;*** Aqui debeis incluir las subrutinas que necesite vuestra solucion
        ;*** SALVO DECOD, que va en la siguiente seccion
;Subrutina de usuario que actualiza el eflag N
EFLAGN:
    BTST #3,D2 ;BTST bit 3 del SR, contiene el flag N
    BNE N
    BCLR #2,D3 ; N=0
    RTS
N:  BSET #2,D3 ; N=1
    RTS
;Subrutina de usuario que actualiza el eflag C
EFLAGC:
    BTST #0,D2 ;BTST bit 0 del SR, contiene el flag C
    BNE C
    BCLR #1,D3 ; C=0
    RTS
C:  BSET #1,D3 ; C=1
    RTS
;Subrutina de usuario que actualiza el eflag Z
EFLAGZ:
    BTST #2,D2 ;BTST bit 2 del SR, contiene el flag Z
    BNE Z
    BCLR #0,D3 ; Z=0
    RTS
Z:  BSET #0,D3 ; Z=1
    RTS
;Subrutina que decodifica operando A y guarda en A4
Xa:
    BTST.L #7,D0
    BEQ Xa0XX
    BTST.L #6,D0
    BEQ Xa10X
    BTST.L #5,D0
    BEQ Xa110
    LEA.L ET7,A4 ;A es T7 ;111
    RTS
Xa0XX:
    BTST #6,D0
    BEQ Xa00X
    BTST #5,D0
    BEQ Xa010
    LEA.L ER3,A4 ;A es R3 ;011
    RTS
Xa00X:
    BTST #5,D0
```

```
    BEQ Xa000
    LEA.L ER1,A4 ;A es R1 ;001
    RTS
Xa010:
    LEA.L ER2,A4 ;A es R2 ;010
    RTS
Xa000:
    LEA.L ER0,A4 ;A es R0 ;000
    RTS
Xa10X:
    BTST.L #5,D0
    BEQ Xa100
    LEA.L ER5,A4 ;A es R5 ;101
    RTS
Xa100:
    LEA.L ER4,A4 ;A es R4 ;100
    RTS
Xa110:
    LEA.L ET6,A4 ;A es T6 ;110
    RTS
;Subrutina que decodifica operando B y guarda en A3
Xb:
    BTST.L #2,D0
    BEQ Xb0XX
    BTST.L #1,D0
    BEQ Xb10X
    BTST.L #0,D0
    BEQ Xb110
    LEA.L ET7,A3 ;B es T7 ;111
    RTS
Xb0XX:
    BTST #1,D0
    BEQ Xb00X
    BTST #0,D0
    BEQ Xb010
    LEA.L ER3,A3 ;B es R3 ;011
    RTS
Xb00X:
    BTST #0,D0
    BEQ Xb000
    LEA.L ER1,A3 ;B es R1 ;001
    RTS
Xb010:
    LEA.L ER2,A3 ;B es R2 ;010
    RTS
Xb000:
    LEA.L ER0,A3 ;B es R0 ;000
    RTS
```

```
Xb10X:
   BTST.L #0,D0
   BEQ Xb100
   LEA.L ER5,A3 ;B es R5 ;101
   RTS
Xb100:
   LEA.L ER4,A3 ;B es R4 ;100
   RTS
Xb110:
   LEA.L ET6,A3 ;B es T6 ;110
   RTS
   ;--- FSUBR: FIN SUBRUTINAS
   ;--- IDECOD: INICIO DECOD
      ;*** Tras la etiqueta DECOD, debeis implementar la subrutina de
      ;*** decodificacion, que debera ser de libreria, siguiendo la interfaz
      ;*** especificada en el enunciado
DECOD:
   MOVE.L D0,-(A7)
   MOVE.W 8(A7),D0
   BTST #15,D0
   BNE BIT1
   BTST #14,D0       ;0
   BNE BIT1cero
   MOVE.W #0,8(A7)
   JMP FINAL         ;00 Id HLT
BIT1:
   BTST #14,D0       ;1
   BNE BIT2
   BTST #13,D0       ;10
   BNE BIT5
   BTST #12,D0       ;100
   BNE BIT6
   BTST #11,D0       ;1000
   BNE BIT7
   MOVE.W #4,8(A7)   ;10000 ID COM
   JMP FINAL
BIT2:
   BTST #13,D0       ;11
   BNE BIT3
   BTST #12,D0       ;110
   BNE BIT4
   MOVE.W #12,10(A7)
   JMP FINAL         ;1100 ID MOV
BIT3:
   MOVE.W #14,10(A7)
   JMP FINAL         ;1110 Id LOA
BIT4:
   MOVE.W #13,10(A7)
```

```
   JMP FINAL         ;1101 Id STO
BIT5:
   MOVE.W #12,D0     ;101
   BNE BIT9
   MOVE.W #11,D0     ;1010
   BNE BIT10
   MOVE.W #8,10(A7)
   JMP FINAL         ;10100 ID AND
BIT6:
   BTST #11,D0       ;1001
   BNE BIT8
   MOVE.W #6,10(A7)
   JMP FINAL         ;10010 ID SUB
BIT7:
   MOVE.W #5,10(A7)
   JMP FINAL         ;10001 ID ADD
BIT8:
   MOVE.W #7,10(A7)
   JMP FINAL         ;10011 Id NEG
BIT9:
   MOVE.W #11,D0     ;1011
   BNE BIT11
   MOVE.W #10,10(A7)
   JMP FINAL         ;10110 ID NOT
BIT10:
   MOVE.W #9,10(A7)
   JMP FINAL         ;10101 ID OR
BIT11:
   MOVE.W #11,10(A7)
   JMP FINAL         ;10111 ID SET
BIT1cero:            ;0
   BTST #13,D0       ;01
   BEQ BIT2cero
   MOVE.W #3,10(A7)  ;011
   JMP FINAL         ;0110 ID JMI M
BIT2cero:
   BTST #12,D0       ;010
   BNE BIT3cero
   MOVE.W #1,10(A7)
   JMP FINAL         ;0100 ID JMN M
BIT3cero:
   MOVE.W #2,10(A7)
   JMP FINAL         ;0101 ID JMZ M
FINAL:
   MOVE.L (A7)+,D0
   RTS
   ;--- FDECOD: FIN DECOD
   END    START
```