Práctica Final 2018

Estructura de Computadores I

Grupo 1

2017/2018

Gian Lucas Martín Chamorro, [redacted]

gian.martin1@estudiant.uib.cat

Alejandro Cortés Fernández, [redacted]

alejandro.cortes4@estudiant.uib.cat

1 INTRODUCCIÓN

La USC-1(UIB Simple Computer-1) es la máquina que hemos emulado para la práctica final. La máquina consta de un registro de instrucciones de tres direcciones y de registros de datos los cuales ocupan 16 bits en la memoria. Los registros de la máquina son: el registro T0, el cual sirve como operando para operaciones de tipo ALU y como interfaz con la memoria; los registros R1-R4, que sirven para operaciones de tipo ALU, tanto para el operando fuente como el operando destino; y los registros B5 y B6, que actuarán como los registros de direcciones.

El problema propuesto consiste en emular la máquina USC-1 con el programa que imita al 68k. Eso conlleva a que la máquina a emular realice las fases de fetch, decodificación y ejecución. La máquina tiene los flags Z, N y C, los cuales se deben actualizar con las instrucciones correspondientes. El emulador tiene que realizar las instrucciones codificadas indicadas por el usuario en el vector EPROG. La máquina emulada permite diversos direccionamientos: inmediato, el indexado básico, el directo por registro y el absoluto.

2 EXPLICACIÓN GENERAL

El emulador de la máquina USC-1 necesita para su correcto funcionamiento la implementación de tres fases: la fase de fetch, la fase de decodificación y la fase de ejecución. En la fase de fetch lo que tiene que hacer el emulador es ocuparse de cargar la siguiente instrucción a ejecutar, indicada por el contador de programa, en el registro de instrucción y dejar listo el EPC para que apunte a la siguiente instrucción. Lo que se hace en la fase de decodificación es transcribir la instrucción que se va a ejecutar para averiguar de qué tipo es y qué es lo que se tiene que hacer para que se lleve a cabo. En la fase de ejecución el emulador realiza los procesos necesarios para imitar el comportamiento que tendría que tener la máquina original (USC-1).

2.1 FASE DE *FETCH*

Para la emulación de la fase de fetch lo que hemos hecho es utilizar el registro de datos D4, al que se mueve el contenido de la posición de memoria EPC (Contador de programa del emulador), que es un word. Después este número se multiplica por 2 debido a que en la máquina USC-1 las posiciones de memoria van de 1 en 1, no como en la máquina 68000 en la que las instrucciones (de este programa) ocupan 2 bytes. Para mover la instrucción que marca el EPC al registro EIR utilizamos un direccionamiento indexado completo que le suma el contenido de D4 al registro de direcciones A0, que anteriormente ha sido iniciado con la dirección de EPROG, y pasa el contenido de la dirección obtenida al EIR. Para finalizar se incrementa en 1 el valor del EPC. Esto es todo lo que se hace en la fase de fetch.

2.2 FASE DE DECODIFICACIÓN

De forma resumida, lo que hemos hecho para implementar la fase de decodificación es, teniendo como parámetro la instrucción, utilizar una estructura de árbol que se va separando, dependiendo del valor de cada bit. Se empieza por el MSB (Bit más significativo) de la instrucción y se va haciendo el análisis, con las correspondientes separaciones o branches, hasta que se llega al quinto bit empezando por la izquierda. Sin embargo, la decodificación puede acabar sin mirar todos esos bits dado a que algunas instrucciones utilizan combinaciones únicas en sus primeros bits como STP (11 XXXXXXXXXXXXXX).

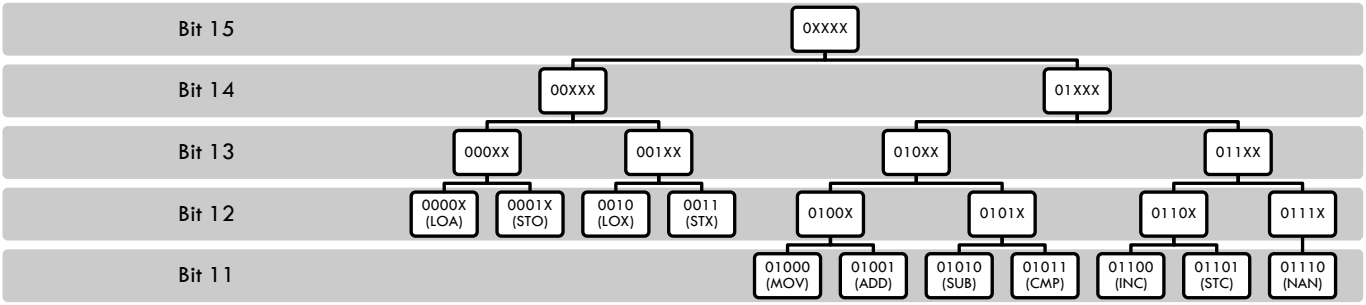
2.3 FASE DE EJECUCIÓN

En la fase de ejecución están separadas con etiquetas todas las instrucciones del repertorio de la máquina USC-1. Dependiendo de los "modos de direccionamiento" o de la estructura de cada instrucción se han usado diversas subrutinas e implementado las acciones necesarias para que todo funcionase como si fuera una USC-1. Usamos subrutinas para la obtención de los operandos: DIREC, para las instrucciones con direcciones de memoria codificadas; BRDIREC, para las instrucciones de salto; K, para las instrucciones que usan direccionamiento inmediato; tres subrutinas para la obtención de operandos que sean registros; y tres subrutinas para la actualización de flags.

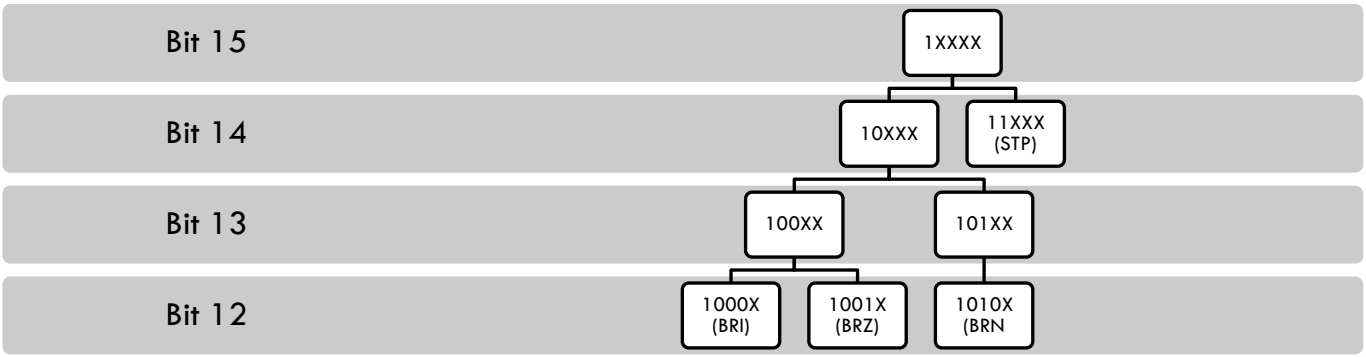
3 SUBRUTINA DE DECODIFICACIÓN

En la rutina de decodificación lo que se hace es analizar bit a bit la instrucción codificada para, dependiendo de los valores, determinar cuál es. Como máximo se llegan a analizar los cinco bits más significativos de la instrucción. El funcionamiento es muy sencillo, si el bit analizado vale 0 se hace un branch a la etiqueta que corresponda y después se repite este proceso hasta que se pueda saber de qué instrucción se trata. A continuación, está representado gráficamente mediante diagramas de árbol el funcionamiento de esta subrutina. Si el bit ya ha sido analizado se muestra su valor, si por el contrario todavía no ha sido analizado aparece una X en su posición. Solo se muestran los cinco primeros bits de la instrucción porque no se necesitan más para la decodificación. Están separados en 2 diagramas en función del valor del bit 15.

3.1 BIT 15 ES 0



3.2 BIT 15 ES 1



4 SUBRUTINAS

Subrutina	Función	Interfaz entrada	Interfaz salida
DECOD	Subrutina de librería que sirve para gestionar el parámetro pasado por la pila y determinar qué instrucción es. Tiene como parámetro de entrada, que se pasa por la pila, el contenido de EIR y para sacar el resultado se usa un espacio inicializado con anterioridad a 0 en la pila. Como utiliza D0, lo primero que hace es guardar su valor para después devolverlo como estaba.	Pila	Pila
DIREC	Subrutina de usuario que devuelve una posición de memoria, especificada en la instrucción, lista para usarse en A2.	D0	A2
BRDIREC	Subrutina de usuario específica para instrucciones de branch que devuelve la dirección que hay que poner en el EPC. Interactúa con los mismos registros que DIREC.	D0	A2
K	Subrutina de usuario que guarda el operando k (en direccionamiento inmediato) en el registro de datos D0 y hace una extensión de signo.	D0	D0
Xb	Subrutina de usuario que decodifica qué registro es el operando B y lo guarda en A3.	D0	A3
Xa	Subrutina de usuario que decodifica qué registro es el operando A y lo guarda en A4.	D0	A4
Xc	Subrutina de usuario que decodifica qué registro es el operando C y lo guarda en A5.	D0	A5
ACTZ	Subrutina de usuario que actualiza el flag Z. Antes de ejecutar esta subrutina SR y ESR son almacenados en D2 y D3 respectivamente. En la subrutina se copia el valor del bit de D2 correspondiente a Z a su posición en D3.	D2 y D3	D3
ACTN	Subrutina de usuario que actualiza el flag N. Antes de ejecutar esta subrutina SR y ESR son almacenados en D2 y D3 respectivamente. En la subrutina se copia el valor del bit de D2 correspondiente a N a su posición en D3.	D2 y D3	D3
ACTC	Subrutina de usuario que actualiza el flag C. Antes de ejecutar esta subrutina SR y ESR son almacenados en D2 y D3 respectivamente. En la subrutina se copia el valor del bit de D2 correspondiente a C a su posición en D3.	D2 y D3	D3

5 REGISTROS DEL 68K

Registro	Función
D0	Registro interfaz para subrutinas que cogen las direcciones o los datos inmediatos codificados en las instrucciones. También es el registro que se utiliza en la subrutina de librería de DECOD y en las subrutinas de usuario para la obtención de los operandos de algunas instrucciones.
D1	Registro de datos en el que se almacena el resultado de la decodificación de la instrucción para usarse en el salto a la fase de ejecución.
D2	Utilizado como interfaz entre el programa principal y las subrutinas de actualización de flags (ACTZ, ACTN y ACTC). En este registro se guarda el SR del 68K después de la operación para su posterior uso en las subrutinas.
D3	Utilizado como interfaz entre el programa principal y las subrutinas de actualización de flags (ACTZ, ACTN y ACTC). En este registro se pone el ESR y luego en la subrutina se modifica acorde a la situación para obtener el valor que tendría que tener el ESR de la máquina.
D4	Utilizado en la fase de *fetch*, se le pasa el contenido del EPC y después se multiplica por 2 para obtener el valor correcto. Más tarde se utiliza en un direccionamiento indexado para acceder a la instrucción indicada por el EPC.
D5	Utilizado como registro de datos auxiliar para cálculos que necesitan que uno de los dos operandos sea un registro de datos.
D6	SIN USO
D7	SIN USO
A0	Utilizado en la fase de *fetch*, al comenzar el programa se inicializa con la dirección de EPROG como valor. Después se usa en un direccionamiento indexado completo para mover el contenido de la dirección obtenida al sumar el EPC y el valor de A0.
A1	Utilizado en el salto a la fase de ejecución para ir a la instrucción correcta.
A2	Utilizado como interfaz de salida por la subrutina de obtención de direcciones y, como consecuencia, en la ejecución de las instrucciones que tienen codificadas posiciones de memoria.
A3	Utilizado para guardar la dirección del operando B (Xb) en la subrutina Xb y en instrucciones que necesitan un operando B.
A4	Utilizado para guardar la dirección del operando A (Xa) en la subrutina Xa y en instrucciones que necesitan un operando A.
A5	Utilizado para guardar la dirección del operando C (Xc) en la subrutina Xc y en instrucciones que necesitan un operando C.
A6	SIN USO
A7/SP	Utilizado para acceder a la pila.

6 PRUEBAS

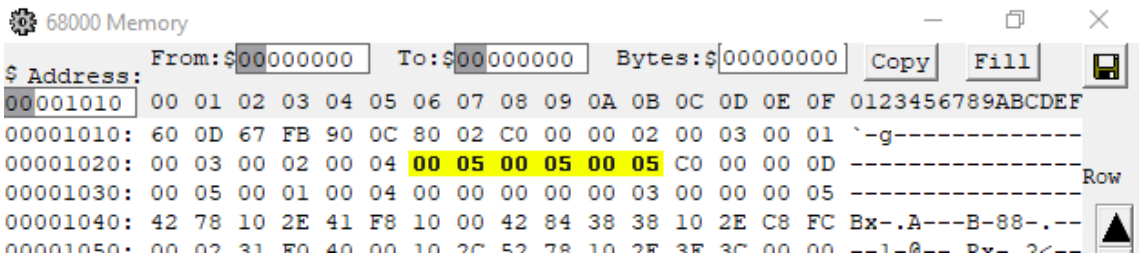
Este es el conjunto de pruebas que hemos realizado para asegurarnos de que el emulador funciona correctamente y que las instrucciones y subrutinas están bien implementadas.

6.1 EJEMPLO: RECORRIDO VECTORES

Ejemplo de programa que recorre dos vectores de tamaño 3, A y B, suma sus elementos componente a componente, y almacena el resultado en otro vector, C. El acceso a los vectores se hace siempre usando el modo indexado (instrucciones LOX y STX). Tras la ejecución del programa, C = (5, 5, 5). En este caso la posición de memoria de C en 68K es @1026.

@USC-1	Ensamblador sin etiquetas	Instrucciones codificadas	Hex
0:	STC #0, B5	01101 00000000 101	6805
1:	STC #3, R3	01101 00000011 011	681B
2:	LOX 13(B5)	0010 000 0 00001101	200D
3:	MOV T0, R1	01000 00000 000 001	4001
4:	LOX 16(B5)	0010 000 0 00010000	2010
5:	MOV T0, R2	0100 000 0 00000010	4002
6:	ADD R1, R2, T0	01001 00 000 001 010	480A
7:	STX 19(B5)	0011 000 0 00010011	3013
8:	INC #1, B5	01100 00000001 101	600D
9:	INC #-1, R3	01100 11111111 011	67FB
10:	BRZ 12	1001 0000 00001100	900C
11:	BRI 2	1000 0000 00000010	8002
12:	STP	11 00000000000000	C000
13:	2	0000 0000 0000 0010	0002
14:	3	0000 0000 0000 0011	0003
15:	1	0000 0000 0000 0001	0001
16:	3	0000 0000 0000 0011	0003
17:	2	0000 0000 0000 0010	0002
18:	4	0000 0000 0000 0100	0004
19:	0	0000 0000 0000 0000	0000
20:	0	0000 0000 0000 0000	0000
21:	0	0000 0000 0000 0000	0000

Y este es el resultado:

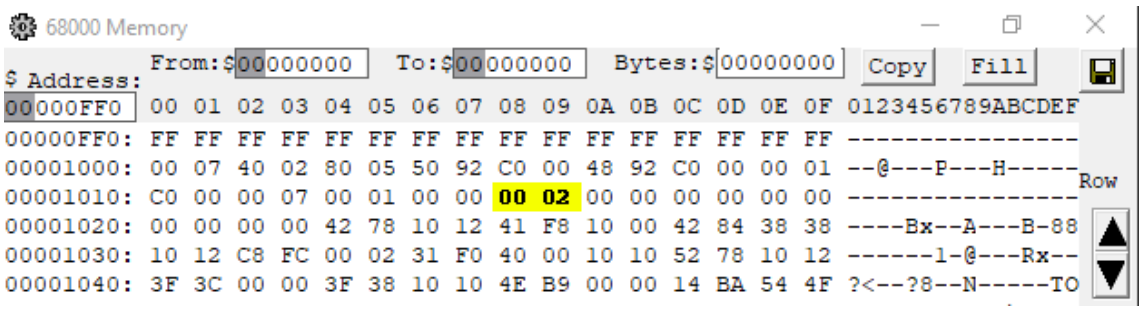


6.2 EJEMPLO: BÁSICO

Ejemplo de programa que carga el valor de la posición de memoria 7 (en USC-1) en el registro T0. Después mueve su contenido, que es el valor 1, al registro R2. Más tarde hace un branch incondicional hasta una instrucción que lo suma a sí mismo y lo guarda también en R2. Tras la ejecución del programa, la posición de memoria del 68K correspondiente a R2 (en este caso, @1018Hex) debería contener el valor 2Dec = 0002Hex.

@USC-1	Ensamblador sin etiquetas	Instrucciones codificadas	Hex
0:	LOA 7	0000 0000 00000111	0007
1:	MOV T0, R2	01000 00000 000 010	4002
2:	BRI 5	1000 0000 00000101	8005
3:	SUB R2, R2, R2	01010 00 010 010 010	5092
4:	STP	11 00000000000000	C000
5:	ADD R2, R2, R2	01001 00 010 010 010	4892
6:	STP	11 00000000000000	C000
7:	1	0000 0000 0000 0001	0001

Y este es el resultado:

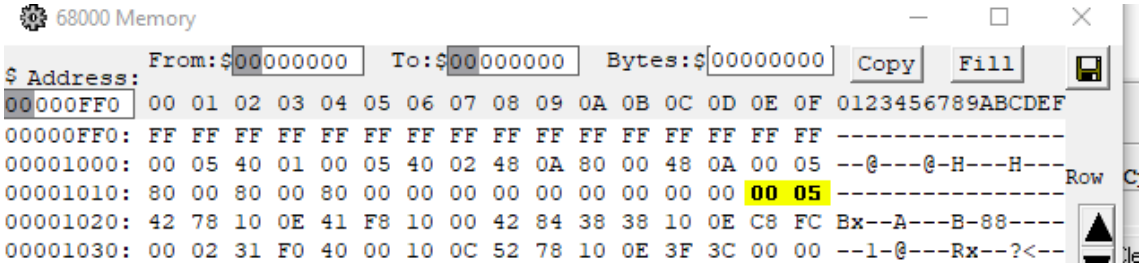


6.3 EJEMPLO: FLAGS

Ejemplo de programa que carga el valor de la posición de memoria 5, que es -32768, en el registro T0 y después lo mueve a R1 y a R2. A continuación suma los registros R1 y R2 y guarda el resultado en el registro T0. Después de la ejecución de la suma, el registro SR (ESR en la máquina 68K y cuya posición de memoria en este caso es @101E) tiene que tener el valor 5Dec=0005Hex, que significa que Z=1, N=0 y C=1.

@USC-1	Ensamblador sin etiquetas	Instrucciones codificadas	Hex
0:	LOA 5	0000 0000 00000101	0005
1:	MOV T0, R1	01000 00000 000 001	4001
2:	MOV T0, R2	01000 00000 000 010	4002
3:	ADD R1, R1, T0	01001 00 000 001 010	480A
4:	STP	11 00000000000000	C000
5:	-32768	1000 0000 0000 0000	8000

Y este es el resultado:

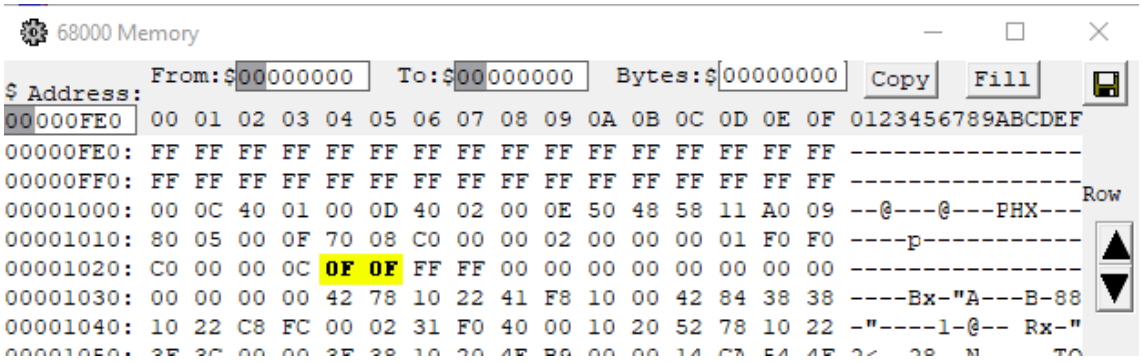


6.4 EJEMPLO: INSTRUCCIONES RESTANTES

Ejemplo de programa que realiza un bucle restando hasta que el número sea negativo y después hace una operación NAND. En primer lugar, carga en T0 el contenido de la posición de memoria 12, que contiene el valor 2Dec=0002Hex, y lo mueve al registro R1. Después hace lo mismo con un 0Dec=0000Hex de la posición de memoria 13 guardándolo en R2. Más tarde carga en T0 un 1Dec=0001Hex de la posición de memoria 14. A continuación, entra en un bucle que le resta a R1 el contenido de T0 hasta que R1 sea un número negativo mediante el uso de un CMP, con el que se compara los valores de R1 y R2, un BRN para salir del bucle y un BRI para volver al principio si no se cumple la condición. Después de salir del bucle se carga en T0 desde la posición de memoria 15 el valor -3856Dec=F0F0Hex y se hace una operación NAND con R1 y almacena el resultado en T0. Tras la ejecución del programa, la posición de memoria del 68K correspondiente a T0 (en este caso, @1024Hex) debería contener el valor 0F0FHex.

@USC-1	Ensamblador sin etiquetas	Instrucciones codificadas	Hex
0:	LOA 12	0000 0000 00001100	000C
1:	MOV T0, R1	01000 00000 000 001	4001
2:	LOA 13	0000 0000 00001101	000D
3:	MOV T0, R2	01000 00000 000 010	4002
4:	LOA 14	0000 0000 00001110	000E
5:	SUB R1, T0, R1	01010 00 001 001 000	5048
6:	CMP R2, R1	01011 00000 010 001	5811
7:	BRN 9	1010 0000 00000101	A009
8:	BRI 5	1000 0000 00000101	8005
9:	LOA 15	0000 0000 00001111	000F
10:	NAN R1, T0, T0	01110 00 000 001 000	7008
11:	STP	11 00000000000000	C000
12:	2	0000 0000 0000 0010	0002
13:	0	0000 0000 0000 0000	0000
14:	1	0000 0000 0000 0001	0001
15:	-3856	1111 0000 1111 0000	F0F0

Y este es el resultado:



7 CONCLUSIONES

En la realización de este proyecto hemos consolidado el conocimiento sobre la máquina 68K y el lenguaje ensamblador previamente adquirido en las clases teóricas de Estructura de Computadores I. Se han puesto en práctica estos conocimientos para conseguir crear un programa capaz de emular un procesador. Debido a la naturaleza de este objetivo hemos podido observar, a un cierto nivel de abstracción, cada paso que se realiza en un procesador para realizar las acciones necesarias para llevar a cabo las instrucciones. Gracias a esta práctica hemos podido comprobar que con una máquina de dos direcciones se puede emular una máquina de tres instrucciones a cambio de más líneas de código. También hemos visto con más detalle cómo se realizan las fases de fetch, decodificación y ejecución. Hemos aprendido a hacer una subrutina de librería usando la pila como interfaz y modificando el Stack Pointer. Nuestra valoración personal del trabajo es que ha sido gratificante ver que todo el trabajo realizado durante las clases prácticas ha dado sus frutos y que se haya podido ejecutar el programa sin error alguno con un funcionamiento óptimo.

8 CÓDIGO FUENTE

```
*-----------------------------------------------------------
* Title      : PRAFIN18
* Written by : Gian Lucas Martín Chamorro y Alejandro Cortés Fernández
* Date       : 20/05/2018
* Description: Emulador de la USC-1
*-----------------------------------------------------------
    ORG $1000
EPROG: DC.W $6805,$681B,$200D,$4001,$2010,$4002,$480A
       DC.W $3013,$600D,$67FB,$900C,$8002,$C000,$0002
       DC.W $0003,$0001,$0003,$0002,$0004,$0000,$0000,$0000
EIR:   DC.W 0 ;eregistro de instrucción
EPC:   DC.W 0 ;econtador de programa
ET0:   DC.W 0 ;eregistro T0
ER1:   DC.W 0 ;eregistro R1
ER2:   DC.W 0 ;eregistro R2
ER3:   DC.W 0 ;eregistro R3
ER4:   DC.W 0 ;eregistro R4
EB5:   DC.W 0 ;eregistro B5
EB6:   DC.W 0 ;eregistro B6
ESR:   DC.W 0 ;eregistro de estado (00000000 00000ZNC)

START:
    CLR.W EPC
    LEA.L EPROG,A0

FETCH:
    ;--- IFETCH: INICIO FETCH
        ;*** En esta sección debéis introducir el código necesario para cargar
        ;*** en el EIR la siguiente instrucción a ejecutar, indicada por el
       ;*** EPC y dejar listo el EPC para que apunte a la siguiente
        ;***instrucción.
    CLR.L D4
    MOVE.W EPC,D4
    MULU #2,D4
    MOVE.W 0(A0,D4),EIR ;Calcula desplazamiento respecto a EPROG en base al EPC
    ADDQ.W #1,EPC
    ;--- FFETCH: FIN FETCH

    ;--- IBRDECOD: INICIO SALTO A DECOD
        ;*** En esta sección debéis preparar la pila para llamar a la subrutina
        ;*** DECOD, llamar a la subrutina, y vaciar la pila correctamente,
        ;*** almacenando el resultado de la decodificación en D1
        MOVE.W #0,-(A7) ;Espacio en pila para resultado
        MOVE.W EIR,-(A7) ;Parámetro en la pila
        JSR DECOD
        ADDQ.W #2,SP ;Borra de pila el parámetro
        MOVE.W (SP)+,D1 ;Resultado de decodificación
    ;--- FBRDECOD: FIN SALTO A DECOD

    ;--- IBREXEC: INICIO SALTO A FASE DE EJECUCION
        ;*** Esta sección se usa para saltar a la fase de ejecución
        ;*** NO HACE FALTA MODIFICARLA
    MULU #6,D1
    MOVEA.L D1,A1
    JMP JMPLIST(A1)
JMPLIST:
    JMP ELOA
    JMP ESTO
    JMP ELOX
    JMP ESTX
    JMP EMOV
    JMP EADD
    JMP ESUB
    JMP ECMP
    JMP EINC
```

```
    JMP ESTC
    JMP ENAN
    JMP EBRI
    JMP EBRZ
    JMP EBRN
    JMP ESTP
    ;--- FBREXEC: FIN SALTO A FASE DE EJECUCION

    ;--- IEXEC: INICIO EJECUCION
        ;*** En esta sección debéis implementar la ejecución de cada einstr.

;Decodifica la dirección de memoria en la subrutina DIREC y mueve, con indexado
;sumándole la @ de EPROG, su contenido a ET0. Después actualiza los eflags
;Z y N.
ELOA:
    MOVE.W EIR, D0
    JSR DIREC
    MOVE.W EPROG(A2),ET0
    MOVE SR,D2
    MOVE.W ESR,D3
    JSR ACTZ
    JSR ACTN
    MOVE.W D3,ESR
    JMP FETCH

;Decodifica la dirección de memoria en la subrutina DIREC y mueve
;el contenido de ET0 a dicha dirección.
ESTO:
    MOVE.W EIR, D0
    JSR DIREC
    MOVE.W ET0,EPROG(A2)
    JMP FETCH

;Mueve el contenido del registro de memoria B5 o B6(dependiendo del bit 8)
;a D5 y se lo suma a la dirección de memoria A2 para después mover, con indexado
;sumándole la @ de EPROG, su contenido a ETO. Finalmente actualiza los flags
;Z y N.
ELOX:
    MOVE.W EIR, D0
    BTST #8,D0
    BEQ ETB5
    JSR DIREC
    MOVE.W EB6,D5
    MULU.W #2,D5
    ADDA.W D5,A2
    MOVE.W EPROG(A2),ET0
    JMP ETELOX
    ETB5:
    JSR DIREC
    MOVE.W EB5,D5
    MULU.W #2,D5
    ADDA.W D5,A2
    MOVE.W EPROG(A2),ET0
    ETELOX:
    MOVE SR,D2
    MOVE.W ESR,D3
    JSR ACTZ
    JSR ACTN
    MOVE.W D3,ESR
    JMP FETCH

;Mueve el contenido del registro de memoria B5 o B6(dependiendo del bit 8)
;a D5 y se lo suma a la dirección de memoria a A2, luego de sumarle el @ de EPROG
;el contenido de ET0 se mueve a dicha dirección.
ESTX:
    MOVE.W EIR, D0
    BTST #8,D0
    BEQ ETB5_2
```

```
    JSR DIREC
    MOVE.W EB6,D5
    MULU.W #2,D5
    ADDA.W D5,A2
    MOVE.W ET0,EPROG(A2)
    JMP FETCH
    ETB5_2:
    JSR DIREC
    MOVE.W EB5,D5
    MULU.W #2,D5
    ADDA.W D5,A2
    MOVE.W ET0,EPROG(A2)
    JMP FETCH

;Mueve el contenido del contenido de A4(Xa) al contenido de A3(Xb) y actualiza
;los flags Z y N.
EMOV:
    MOVE.W EIR,D0
    JSR Xb
    JSR Xa
    MOVE.W (A4),(A3)
    MOVE SR,D2
    MOVE.W ESR,D3
    JSR ACTZ
    JSR ACTN
    MOVE.W D3,ESR
    JMP FETCH

;Mueve el contenido del contenido de A3(Xb) a D5. Después, el contenido
;del contenido de A4(Xa) se suma a D5 y se actualizan los tres flags. Después,
;el contenido de D5 se mueva a la dirección A5(Xc).
EADD:
    MOVE.W EIR, D0
    JSR Xb
    JSR Xa
    JSR Xc
    MOVE.W (A3),D5
    ADD.W (A4),D5
    MOVE SR,D2
    MOVE.W ESR,D3
    JSR ACTZ
    JSR ACTN
    JSR ACTC
    MOVE.W D3,ESR
    MOVE.W D5,(A5)
    JMP FETCH

;Mueve el contenido de A3(Xb) a D5 y le hace una NOT bit a bit. Después se
;suma un 1 a D5 y después se suma este con el contenido del contenido de
;A4(Xa) y se actualizan los tres flags para finalmente mover el resultado a la
;dirección A5(Xc).
ESUB:
    MOVE.W EIR, D0
    JSR Xb
    JSR Xa
    JSR Xc
    MOVE.W (A3),D5
    NOT.W D5
    ADDQ.W #1,D5
    ADD.W (A4),D5
    MOVE SR,D2
    MOVE.W ESR,D3
    JSR ACTZ
    JSR ACTN
    JSR ACTC
    MOVE.W D3,ESR
    MOVE.W D5,(A5)
    JMP FETCH
```

```
;Mueve el contenido de A3(Xb) a D5 y le resta el contenido del contenido de
;A4(Xa) para después actualizar los tres flags.
ECMP:
    MOVE.W EIR, D0
    JSR Xb
    JSR Xa
    JSR Xc
    MOVE.W (A3),D5
    CMP.W (A4),D5
    MOVE SR,D2
    MOVE.W ESR,D3
    JSR ACTZ
    JSR ACTN
    JSR ACTC
    MOVE.W D3,ESR
    JMP FETCH

;Le suma D0 (cuyo contenido es el valor de k) al contenido del contenido de
;A3(Xb) y se actualizan los tres flags.
EINC:
    MOVE.W EIR, D0
    JSR Xb
    JSR K
    ADD.W D0,(A3)
    MOVE SR,D2
    MOVE.W ESR,D3
    JSR ACTZ
    JSR ACTN
    JSR ACTC
    MOVE.W D3,ESR
    JMP FETCH

;Mueve el contenido de D0 (cuyo contenido es el valor de k) y lo mueve a la
;dirección A3(Xb). Finalmente actualiza los flags Z y N
ESTC:
    MOVE.W EIR, D0
    JSR Xb
    JSR K
    MOVE.W D0,(A3)
    MOVE SR,D2
    MOVE.W ESR,D3
    JSR ACTZ
    JSR ACTN
    MOVE.W D3,ESR
    JMP FETCH

;Mueve el contenido del contenido de A3(Xb) al registro D5 para después hacer
;una AND bit a bit con el contenido del contenido de A4(Xa), hacer una NOT bit
;a bit de D5 y mover el resultado a la dirección A5(Xc). Finalmente actualiza
;los flags Z y N.
ENAN:
    MOVE.W EIR, D0
    JSR Xb
    JSR Xa
    JSR Xc
    MOVE.W (A3),D5
    AND.W (A4),D5
    NOT.W D5
    MOVE.W D5,(A5)
    MOVE SR,D2
    MOVE.W ESR,D3
    JSR ACTZ
    JSR ACTN
    MOVE.W D3,ESR
    JMP FETCH

;Salta a la subrutina BRDIREC y después coge esa dirección de salto y lo carga
```

```
;en el EPC.
EBRI:
    MOVE.W EIR, D0
    JSR BRDIREC
    MOVE.W A2,EPC
    JMP FETCH

;Dependiendo del bit 2 (eflag Z) de D3(ESR) hace el branch a EZ0 o carga en
;EPC el valor de salto.
EBRZ:
    MOVE.W ESR,D3
    BTST #2,D3
    BEQ EZ0
    MOVE.W EIR,D0
    JSR BRDIREC
    MOVE.W A2,EPC
    EZ0:
    JMP FETCH

;Dependiendo del bit 1 (eflag N) de D3(ESR) hace el branch a EN0 o carga en
;EPC el valor de salto.
EBRN:
    MOVE.W ESR,D3
    BTST #1,D3
    BEQ EN0
    MOVE.W EIR,D0
    JSR BRDIREC
    MOVE.W A2,EPC
    EN0:
    JMP FETCH

;Salta a END START y detiene la ejecución del programa
ESTP:
   JMP FINISH
    ;--- FEXEC: FIN EJECUCION

    ;--- ISUBR: INICIO SUBRUTINAS
        ;*** Aquí debeis incluir las subrutinas que necesite vuestra solución
        ;*** SALVO DECOD, que va en la siguiente sección

;Subrutina de usuario que devuelve una posición de memoria, especificada en la
;einstruccion, lista para usarse en A2.
DIREC:
    LSL.W #8,D0
    LSR.W #8,D0
    MULU.W #2,D0
    MOVEA.W D0,A2
    RTS

;Subrutina de usuario específica para einstrucciones de branch que devuelve
;la dirección que hay que poner en el EPC.
BRDIREC:
    LSL.W #8,D0
    LSR.W #8,D0
    MOVEA.W D0,A2
    RTS

;Subrutina que guarda el operando k (en dir. inmediato) en el registro
;de datos D0 y hace una extensión de signo.
K:
    LSR.W #3,D0
    EXT.W D0
    RTS

;Subrutina que decodifica que registro es el operando B y lo guarda en A3.
Xb:
    BTST.L #2,D0
    BEQ Xb0XX
```

```
    BTST #1,D0
    BEQ Xb10X
    LEA.L EB6,A3 ;B es B6
    RTS
Xb0XX:
    BTST #1,D0
    BEQ Xb00X
    BTST #0,D0
    BEQ Xb010
    LEA.L ER3,A3 ;B es R3
    RTS
Xb10X:
    BTST.L #0,D0
    BEQ Xb100
    LEA.L EB5,A3 ;B es B5
    RTS
Xb100:
    LEA.L ER4,A3 ;B es R4
    RTS
Xb00X:
    BTST.L #0,D0
    BEQ Xb000
    LEA.L ER1,A3 ;B es R1
    RTS
Xb000:
    LEA.L ET0,A3 ;B es T0
    RTS
Xb010:
    LEA.L ER2,A3 ;B es R2
    RTS

;Subrutina que decodifica que registro es el operando A y lo guarda en A4.
Xa:
    BTST.L #5,D0
    BEQ Xa0XX
    BTST #4,D0
    BEQ Xa10X
    LEA.L EB6,A4 ;A es B6
    RTS
Xa0XX:
    BTST #4,D0
    BEQ Xa00X
    BTST #3,D0
    BEQ Xa010
    LEA.L ER3,A4 ;A es R3
    RTS
Xa10X:
    BTST.L #3,D0
    BEQ Xa100
    LEA.L EB5,A4 ;A es B5
    RTS
Xa100:
    LEA.L ER4,A4 ;A es R4
    RTS
Xa00X:
    BTST.L #3,D0
    BEQ Xa000
    LEA.L ER1,A4 ;A es R1
    RTS
Xa000:
    LEA.L ET0,A4 ;A es T0
    RTS
Xa010:
    LEA.L ER2,A4 ;A es R2
    RTS

; Subrutina que decodifica que registro es el operando C y lo guarda en A5.
Xc:
```

```
    BTST.L #8, D0
    BEQ Xc0XX
    BTST #7, D0
    BEQ Xa10X
    LEA.L EB6, A5 ;C es B6
    RTS
Xc0XX:
    BTST #7,D0
    BEQ Xc00X
    BTST #6,D0
    BEQ Xc010
    LEA.L ER3,A5 ;C es R3
    RTS
Xc10X:
    BTST.L #6,D0
    BEQ Xc100
    LEA.L EB5,A5 ;C es B5
    RTS
Xc100:
    LEA.L ER4,A5 ;C es R4
    RTS
Xc00X:
    BTST.L #6,D0
    BEQ Xc000
    LEA.L ER1,A5 ;C es R1
    RTS
Xc000:
    LEA.L ET0,A5 ;C es T0
    RTS
Xc010:
    LEA.L ER2,A5 ;C es R2
    RTS

ACTZ: ;Se usa para actualizar el eflag Z.
    BTST #2,D2
    BEQ Z0
    BSET #2,D3
    RTS
Z0: BCLR #2,D3
    RTS

ACTN: ;Se usa para actualizar el eflag N.
    BTST #3,D2
    BEQ N0
    BSET #1,D3
    RTS
N0: BCLR #1,D3
    RTS

ACTC: ;Se usa para actualizar el eflag C.
    BTST #0,D2
    BEQ C0
    BSET #0,D3
    RTS
C0: BCLR #0,D3
    RTS

    ;--- FSUBR: FIN SUBRUTINAS

    ;--- IDECOD: INICIO DECOD
        ;*** Tras la etiqueta DECOD, debéis implementar la subrutina de
        ;*** decodificación, que deberá ser de librería, siguiendo la interfaz
        ;*** especificada en el enunciado
DECOD:
    MOVE.L D0,-(SP) ;Guarda D0
    MOVE.W 8(SP),D0 ;Pone el EIR en D0

    BTST #15,D0
```

```
    BEQ E0XXXX
    BTST #14,D0
    BEQ E10XXX
    MOVE.W #14,10(SP) ;EINSTRUCCION 14
    MOVE.L (SP)+,D0
    RTS
E0XXXX:
    BTST #14,D0
    BEQ E00XXX
    BTST #13,D0
    BEQ E010XX
    BTST #12,D0
    BEQ E0110X
    MOVE.W #10,10(SP) ;EINSTRUCCION 10
    MOVE.L (SP)+,D0
    RTS
E00XXX:
    BTST #13,D0
    BEQ E000XX
    BTST #12,D0
    BEQ E0010X
    MOVE.W #3,10(SP) ;EINSTRUCCION 3
    MOVE.L (SP)+,D0
    RTS
E000XX:
    BTST #12,D0
    BEQ E0000X
    MOVE.W #1,10(SP) ;EINSTRUCCION 1
    MOVE.L (SP)+,D0
    RTS
E0000X:
    MOVE.W #0,10(SP) ;EINSTRUCCION 0
    MOVE.L (SP)+,D0
    RTS
E0010X:
    MOVE.W #2,10(SP) ;EINSTRUCCION 2
    MOVE.L (SP)+,D0
    RTS
E010XX:
    BTST #12,D0
    BEQ E0100X
    BTST #11,D0
    BEQ E01010
    MOVE.W #7,10(SP) ;EINSTRUCCION 7
    MOVE.L (SP)+,D0
    RTS
E0100X:
    BTST #11,D0
    BEQ E01000
    MOVE.W #5,10(SP) ;EINSTRUCCION 5
    MOVE.L (SP)+,D0
    RTS
E01000:
    MOVE.W #4,10(SP) ;EINSTRUCCION 4
    MOVE.L (SP)+,D0
    RTS
E01010:
    MOVE.W #6,10(SP) ;EINSTRUCCION 6
    MOVE.L (SP)+,D0
    RTS
E0110X:
    BTST #11,D0
    BEQ E01100
    MOVE.W #9,10(SP) ;EINSTRUCCION 9
    MOVE.L (SP)+,D0
    RTS
E01100:
    MOVE.W #8,10(SP) ;EINSTRUCCION 8
```

```
    MOVE.L (SP)+,D0
    RTS
E10XXX:
    BTST #13,D0
    BEQ E100XX
    MOVE.W #13,10(SP) ;EINSTRUCCION 13
    MOVE.L (SP)+,D0
    RTS
E100XX:
    BTST #12,D0
    BEQ E1000X
    MOVE.W #12,10(SP) ;EINSTRUCCION 12
    MOVE.L (SP)+,D0
    RTS
E1000X:
    MOVE.W #11,10(SP) ;EINSTRUCCION 11
    MOVE.L (SP)+,D0
    RTS
    ;--- FDECOD: FIN DECOD

FINISH:
    END    START
```