

PRÁCTICA II

EMULACIÓN MÁQUINA JARVIS

ESTRUCTURA DE COMPUTADORES I

Curso: 2021-2022

Grupo: 103

Integrantes e información:

Nicolás Sanz Tuñón

Constantino Byelov Serdiuk

ÍNDICE

Introducción.....	2
Explicación general del trabajo.....	3
Rutina de decodificación.....	4
Tabla subrutinas.....	5
Tabla de registros.....	8
Conjunto de pruebas.....	9
Conclusión.....	12
Código fuente.....	13

INTRODUCCIÓN

Programamos una emulación de una máquina JARVIS (Just A Rather Very Intelligent System) en el entorno de programación Easy68k en ensamblador, con dicha máquina podemos ejecutar diversas funciones que están definidas en la tabla a continuación, además de presentar la codificación de la misma.

Id	Mnemónico	Codificación	Acción	Flags
0	TRA Xa,Xb	00001bbbxaaaaxxxx	$Xb \leftarrow [Xa]$	C = n.s.a., Z y N = s.v.Xb
1	ADD Xa,Xb	00010bbbxaaaaxxxx	$Xb \leftarrow [Xb] + [Xa]$	C, Z y N = s.v.r.
2	SUB Xa,Xb	00011bbbxaaaaxxxx	$Xb \leftarrow [Xb] - [Xa]$	C, Z y N = s.v.r.
3	NAN Xa,Xb	00100bbbxaaaaxxxx	$Xb \leftarrow [Xb] \text{ nand } [Xa]$	C = n.s.a., Z y N = s.v.r.
4	STC #k,Xb	00101bbbkkkkkkkkk	$Xb \leftarrow k$ (Ext. signo)	C = n.s.a., Z y N = s.v.Xb
5	INC #k,Xb	00110bbbkkkkkkkkk	$Xb \leftarrow [Xb] + k$ (Ext. Signo)	C, Z y N = s.v.r.
6	LOA M	0100mmmmmmmmxxxx	$T6 \leftarrow [M]$	C = n.s.a., Z y N = s.v.T6
7	LOAX M(Bi),Tj	0101mmmmmmmmijxx	$Tj \leftarrow [M + [Bi]]$	C = n.s.a., Z y N = s.v.Tj
8	STO M	0110mmmmmmmmxxxx	$M \leftarrow [T6]$	n.s.a.
9	STOX Tj,M(Bi)	0111mmmmmmmmijxx	$M + [Bi] \leftarrow [Tj]$	n.s.a.
10	BRI M	1000mmmmmmmmxxxx	$PC \leftarrow M$	n.s.a.
11	BRZ M	1001mmmmmmmmxxxx	Si Z = 1, $PC \leftarrow M$	n.s.a.
12	BRN M	1010mmmmmmmmxxxx	Si N = 1, $PC \leftarrow M$	n.s.a.
13	STP	11xxxxxxxxxxxxxxxx	Detiene la máquina	n.s.a.

LEYENDA

x: Bit no utilizado (*don't care*).

mmmmmmmm: Dirección de memoria (emulada) de 8 bits.

Xa, Xb: Cualquier registro B, R o T, ver aaa y bbb.

Bi: B0 o B1, ver i.

Tj: T6 o T7, ver j.

aaa y bbb: Índice del registro según: $\begin{cases} 000 - B0, 001 - B1, 010 - R2 \\ 011 - R3, 100 - R4, 101 - R5 \\ 110 - T6, 111 - T7 \end{cases}$

i: Índice del registro B0 (i=0) o del registro B1 (i=1).

j: Índice del registro T6 (j=0) o del registro T7 (j=1).

kkkkkkkk: Constante de 8 bits en complemento a 2, $k \in \{-128, \dots, +127\}$.

n.s.a.: No se actualizan.

s.v.r.: Según el valor del resultado de la operación.

s.v.Xb: Según el valor del registro Xb después de realizar la operación.

s.v.Tj: Según el valor del registro Tj después de realizar la operación.

s.v.T6: Según el valor del registro T6 después de realizar la operación.

Dicha máquina en el código está estructura en diversas funciones tales como el **FETCH**, **BRDECOD** (se prepara la pila para el DECOD), **BREXEC** (encargada del salto a la parte de ejecución del programa), **EXEC** (encargada de de las fases de ejecución), **SUBR** (parte donde van todas las subrutinas de usuario o librería) y **DECOD** (dónde está la parte de decodificación del programa).

EXPLICACIÓN GENERAL DEL TRABAJO

Para solucionar la práctica la hemos planteado siguiendo la estructura del programa proporcionado:

En la parte del FETCH hemos planteado la manera de pasar a la siguiente instrucción a ejecutar mediante un indexado al EMEM y la suma del EPC para ir pasando a la siguiente instrucción, tras esto preparamos la pila y saltamos al DECOD.

En el DECOD, vamos comprobando bit a bit con el BTST para encontrar de qué instrucción se trata, si la hemos encontrado la devolvemos por la pila.

Tras el DECOD pasamos a la parte de ejecución el EXEC, en dicha fase saltamos a la instrucción que hemos encontrado en el DECOD y obtenemos todos los parámetros aplicando máscaras, leyendo así la parte de la instrucción que nos interese.

RUTINA DE DECODIFICACIÓN

Vamos mirando bit por bit y avanzando en la secuencia para poder determinar de qué instrucción se trata.

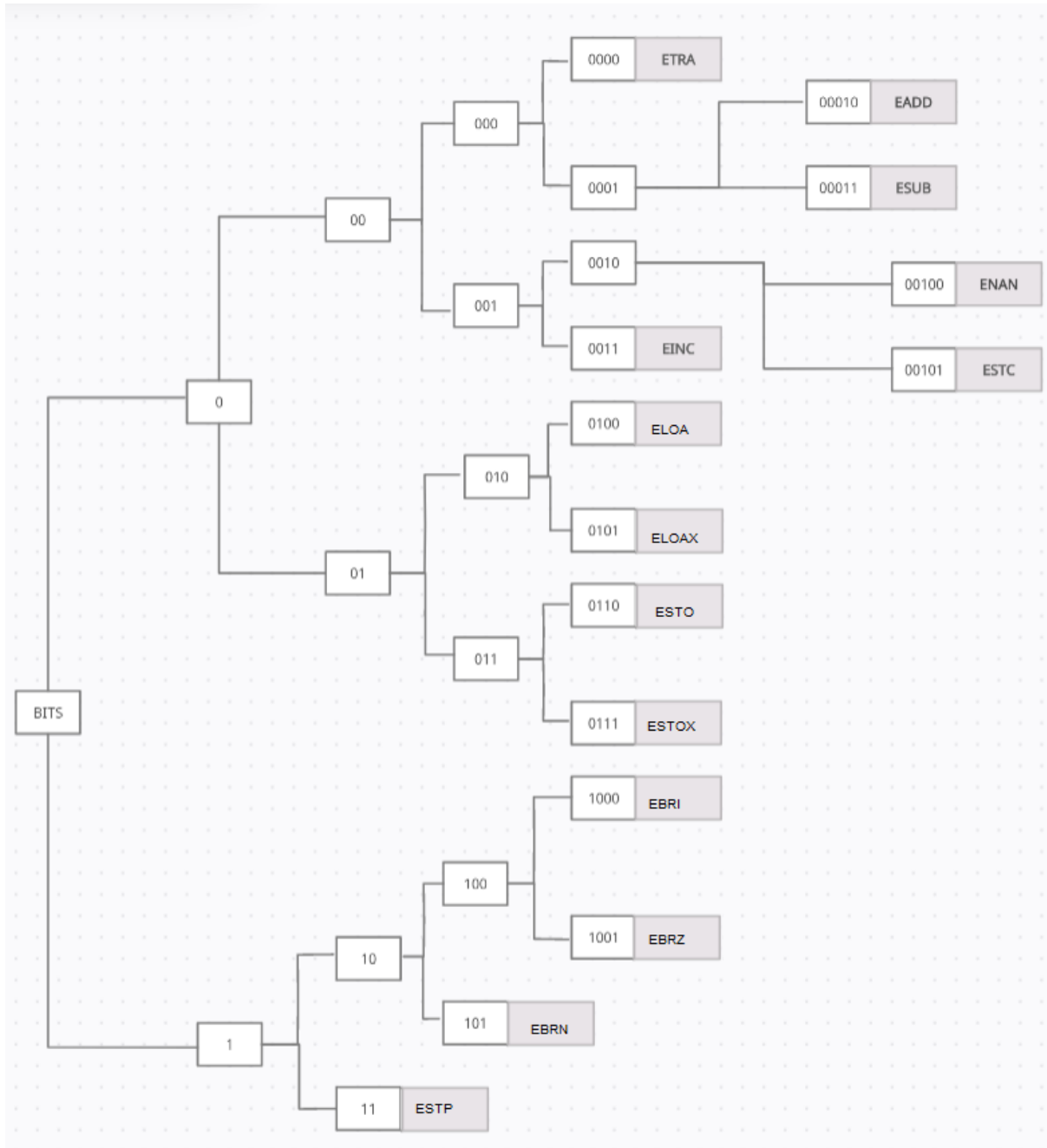


TABLA DE SUBROUTINAS

4.1 Subrutina: **OBTENER_Xa**

Descripción	Mediante la AND aplicamos una máscara, sacamos los bits que nos interesan y asignamos la secuencia a D0 para después mover la misma 4 bits a la derecha y salimos de la subrutina.
Interfaces in/out	D0
Libreria/Usuario	Subrutina de usuario

4.2 Subrutina: **OBTENER_Xb**

Descripción	Mediante la AND aplicamos una máscara, sacamos los bits que nos interesan y asignamos la secuencia a D1 y salimos de la subrutina.
Interfaces in/out	D1
Libreria/Usuario	Subrutina de usuario

4.3 Subrutina: **ENCONTRAR_REGISTRO_Xa**

Descripción	Recibe en D0 el valor del registro Xa al cual aplicamos un BTST y así vamos reduciendo poco a poco de qué registro se trata. Por ejemplo, si Xa=ET5 → D0=0005 , dado que el valor de D0 estaría asociado a dicho registro.
Interfaces in/out	D0
Libreria/Usuario	Subrutina de usuario

4.4 Subrutina: **ENCONTRAR_REGISTRO_Xb**

Descripción	Recibe en D1 el valor del registro Xb al cual aplicamos un BTST y así vamos reduciendo poco a poco de qué registro se trata. Por ejemplo, si Xb=ET6 → D0=0006 , puesto que este es el valor correspondiente al registro ET6, tras eso almacenamos ET6 haciendo uso de la instrucción LEA .
Interfaces in/out	D1 y A1
Libreria/Usuario	Subrutina de usuario

4.5 Subrutina: **ACTU_FLAG_Z_A1**

Descripción	En la subrutina se mueve el contenido de A1 a D2 y va mirando el bit del flag “ Z ” si es 1 no salta a la subrutina secundaria y con el BSET ponemos el flag Z a 1 en caso contrario se salta a la subrutina “ FLAG_Z0 ” donde se hace un BCLR y actualiza en el registro de estado.
Interfaces in/out	D3 , D2 y A1
Libreria/Usuario	Subrutina de usuario

4.6 Subrutina: **ACTU_FLAG_N_A1**

Descripción	En la subrutina se mueve el contenido de A1 a D2 y va mirando el bit del flag “ N ” si es 1 no salta a la subrutina secundaria y con el BSET ponemos el flag N a 1 en caso contrario se salta a la subrutina “ FLAG_No ” donde se hace un BCLR y actualiza en el registro de estado.
Interfaces in/out	D3 , D2 y A1
Libreria/Usuario	Subrutina de usuario

4.7 Subrutina: **ACTU_FLAG_Z_RESULTADO**

Descripción	En la subrutina se mira el bit del flag “ Z ” si es 1 salta a la subrutina secundaria y con el BSET ponemos el flag Z a 1 del registro de estado, en caso contrario se hace un BCLR y se actualiza en el registro de estado.
Interfaces in/out	D3 y D2
Libreria/Usuario	Subrutina de usuario

4.8 Subrutina: **ACTU_FLAG_C_RESULTADO**

Descripción	En la subrutina se mira el bit del flag “ C ” si es 1 salta a la subrutina secundaria y con el BSET ponemos el flag C a 1 del registro de estado, en caso contrario se hace un BCLR y se actualiza en el registro de estado.
Interfaces in/out	D3 y D2
Libreria/Usuario	Subrutina de usuario

4.9 Subrutina: **ACTU_FLAG_N_RESULTADO**

Descripción	En la subrutina se mira el bit del flag “N” si es 1 salta a la subrutina secundaria y con el BSET ponemos el flag N a 1 del registro de estado, en caso contrario se hace un BCLR y se actualiza en el registro de estado.
Interfaces in/out	D3 y D2
Libreria/Usuario	Subrutina de usuario

TABLA DE REGISTROS

Registros	Funcionamiento
D0	Utilizado para coger el contenido de Xa
D1	Utilizado para los BTST y la pila
D2	Utilizado para coger el contenido de Xb
D3	Utilizado para actualizar los flags
A0	Solo utilizado para la fase de fetch
A1	Guardar el contenido de Xb

CONJUNTO DE PRUEBAS

Para comprobar el correcto funcionamiento del programa, hemos probado una serie de conjunto de pruebas en la máquina emulada, para así ver si se comporta como una máquina JARVIS original:

Eprograma 1:

Como está permitido utilizar eprogramas del enunciado de la práctica, vamos a probar primero con otorgado como condición mínima para corregir la práctica.

Este eprograma está formado por la siguiente cabecera:

```
ORG $1000
EMEM: DC.W $4070,$0A60,$8050,$1A20,$C000,$1220,$C000,$0001
EIR: DC.W 0 ;registro de instruccion
EPC: DC.W 0 ;contador de programa
EB0: DC.W 0 ;registro B0
EB1: DC.W 0 ;registro B1
ER2: DC.W 0 ;registro R2
ER3: DC.W 0 ;registro R3
ER4: DC.W 0 ;registro R4
ER5: DC.W 0 ;registro R5
ET6: DC.W 0 ;registro T6
ET7: DC.W 0 ;registro T7
ESR: DC.W 0 ;registro de estado (00000000 00000ZCN)
```

Una forma de comprobar su funcionamiento es mirando la posición de memoria del registro ER2 (@1018 Hex), dado que al finalizar la ejecución, esta posición deberá poseer el valor de 0002 Hex. Esto es debido al vector de words \$1220, que corresponde a la instrucción ADD R2,R2.

Por tanto, nos disponemos a insertar la cabecera y ejecutar el eprograma. La memoria se queda de tal forma:

\$ Address:	From:\$00000000	To:\$00000000	Bytes:\$00000000	Copy	Fill	
00001000	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	0123456789ABCDEF				
00001000:	40 70 0A 60 80 50 1A 20 C0 00 12 20 C0 00 00 01	@p-`-P- --- ---				
00001010:	C0 00 00 07 00 00 00 00 00 02 00 00 00 00 00	-----				
00001020:	00 01 00 00 00 00 42 78 10 12 30 78 10 12 D0 F8	-----Bx--0x----				Row

Y como se puede ver en la parte subrayada, el contenido de ER2 vale 2 Dec.

Eprograma 2:


Para el segundo eprograma, utilizaremos la otra cabecera otorgada en el enunciado de la práctica:

```
ORG $1000
EMEM: DC.W
$2800,$2A03,$50E0,$0B60,$5114,$0C70,$1430,$0E40,$7140,$3001,$32FF,$90D0
      DC.W $8020,$C000,$0002,$0003,$0001,$0003,$0002,$0004,$0000,$0000,$0000
EIR: DC.W 0 ;registro de instruccion
EPC: DC.W 0 ;contador de programa
EB0: DC.W 0 ;registro B0
EB1: DC.W 0 ;registro B1
ER2: DC.W 0 ;registro R2
ER3: DC.W 0 ;registro R3
ER4: DC.W 0 ;registro R4
ER5: DC.W 0 ;registro R5
ET6: DC.W 0 ;registro T6
ET7: DC.W 0 ;registro T7
ESR: DC.W 0 ;registro de estado (00000000 00000ZCN)
```

Claramente se puede ver como esta contiene mucho más vectores de words y, por tanto, muchas más instrucciones para nuestra máquina emulada.

Mediante instrucciones que utilizan el modo indexado se accede a los vectores A (\$0002,\$0003,\$0001) y B (\$0003,\$0002,\$0004). El resultado de estas operaciones se almacenan en el vector C (\$0000,\$0000,\$0000). Por tanto, podemos comprobar los valores del vector C para saber si el test ha funcionado correctamente.

Los valores de C tras la ejecución deberían de ser 3 vectores de words con valor de 5 Hex cada uno. Teniendo esto en cuenta, ejecutamos el eprograma y comprobamos la memoria:

\$ Address:	From: \$00000000	To: \$00000000	Bytes: \$00000000	Copy	Fill	
00001000	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	0123456789ABCDEF				
00001000:	28 00 2A 03 50 E0 0B 60 51 14 0C 70 14 30 0E 40	(--P--`Q--p-0-@				
00001010:	71 40 30 01 32 FF 90 D0 80 20 C0 00 00 02 00 03	q@0-2-----				
00001020:	00 01 00 03 00 02 00 04 00 05 00 05 00 05 C0 00	-----				

Efectivamente, el vector C contiene los valores deseados y esperados por la ejecución de nuestras instrucciones.

Eprograma 3:

Finalmente, para el último test vamos a utilizar una cabecera no proporcionada en el enunciado de la práctica. Consiste en la siguiente:

```
ORG $1000
EMEM: DC.W $1903,$A295,$6A91,$4471,$4721,$1110,$C000,$0362
EIR: DC.W 0 ;registro de instruccion
EPC: DC.W 0 ;contador de programa
EB0: DC.W 0 ;registro B0
EB1: DC.W 0 ;registro B1
ER2: DC.W 0 ;registro R2
ER3: DC.W 0 ;registro R3
ER4: DC.W 0 ;registro R4
ER5: DC.W 0 ;registro R5
ET6: DC.W 0 ;registro T6
ET7: DC.W 0 ;registro T7
ESR: DC.W 0 ;registro de estado (00000000 00000ZCN)
```

En la cual el vector \$4721 trata de una instrucción LOA que supuestamente ha de modificar el contenido de ET6 (@1020 Hex) y convertirlo en 12E6 Hex.

Comprobemos si realmente funciona mirando la memoria tras la ejecución:

\$ Address:	From:\$00000000	To:\$00000000	Bytes:\$00000000	Copy	Fill	
00001000	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	0123456789ABCDEF				
00001000:	19 03 A2 95 6A 91 44 71 47 21 11 10 C0 00 03 62	----j-DqG!-----b				
00001010:	C0 00 00 07 00 00 00 00 00 00 00 00 00 00 00 00	-----				
00001020:	12 E6 00 00 00 04 42 78 10 12 30 78 10 12 D0 F8	-----Bx--0x-----				Row

Y como se puede ver en la parte subrayada, se ha cumplido la condición.

CONCLUSIÓN

Hemos adquirido conocimientos tales como el uso correcto de las subrutinas de librería, como diferenciar una instrucción de otra mediante el BTST y la ayuda que nos pueden aportar el uso de las subrutinas.

Aunque la práctica haya costado mucho y haya supuesto un reto, gracias a las sesiones de prácticas donde se daba la práctica dividida en diferentes etapas, hemos sabido solucionar cada parte por separado llegando al resultado deseado. El equipo coincide que unos de los mayores retos de la práctica han sido plantear la manera de decodificar e identificar cada instrucción además de su fase de ejecución. En cambio, tras un duro esfuerzo realizado hemos podido llegar al resultado que queríamos, el cual es solucionar la práctica.

Cabe destacar que con todo el tiempo que le hemos dedicado a la práctica hemos podido repasar y aprender nuevas instrucciones de las cuales no sabíamos su función.

En conclusión, sentimos que ha sido una buena práctica para el nivel de ensamblador que sabemos y nos ha encantado el enfoque de la misma, además creemos que el Easy68k es una máquina con muchas posibilidades que aún no hemos descubierto, y nos encantaría poder descubrir todas las posibles funciones que nos puede dar este entorno de programación.

CÓDIGO FUENTE

*-----

* Title : PRAFIN22

* Written by : Nicolás Sanz Tuñón y Constantino Byelov Serdiuk

* Date : 31/05/2022

* Description: Emulador de la JARVIS

*-----

ORG \$1000

EMEM:

DC.W

\$2800,\$2A03,\$50E0,\$0B60,\$5114,\$0C70,\$1430,\$0E40,\$7140,\$3001,\$32FF,\$90D0

DC.W \$8020,\$C000,\$0002,\$0003,\$0001,\$0003,\$0002,\$0004,\$0000,\$0000,\$0000

EIR: DC.W 0 ;registro de instruccion

EPC: DC.W 0 ;contador de programa

EB0: DC.W 0 ;registro B0

EB1: DC.W 0 ;registro B1

ER2: DC.W 0 ;registro R2

ER3: DC.W 0 ;registro R3

ER4: DC.W 0 ;registro R4

ER5: DC.W 0 ;registro R5

ET6: DC.W 0 ;registro T6

ET7: DC.W 0 ;registro T7

ESR: DC.W 0 ;registro de estado (00000000 00000ZCN)

START:

CLR.W EPC

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

; ESCRIBID VUESTRO CODIGO AQUI

MOVE.W EPC, A0

ADD.W EPC, A0

MOVE.W EMEM(A0),EIR

ADDQ.W #1, 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

; ESCRIBID VUESTRO CODIGO AQUI

MOVE.W #0,-(SP) ;Espacio extra para el decod del resultado

MOVE.W EIR,-(SP) ;EIR = parámetro entrada pila

JSR DECOD

MOVE.W (SP)+,D0

MOVE.W (SP)+,D1

;--- FBRDECOD: FIN SALTO A DECOD

;--- IBREXEC: INICIO SALTO A FASE DE EJECUCION

;*** Esta seccion se usa para saltar a la fase de ejecucion
;*** NO HACE FALTA MODIFICARLA

MULU #6,D1

MOVEA.L D1,A1

JMP JMPLIST(A1)

JMPLIST:

JMP ETRA

JMP EADD

JMP ESUB

JMP ENAN

JMP ESTC

JMP EINC

JMP ELOA

JMP ELOAX

JMP ESTO

JMP ESTOX

JMP EBRI

JMP EBRZ

JMP EBRN

JMP ESTP

;--- FBREXEC: FIN SALTO A FASE DE EJECUCION

;--- IEXEC: INICIO EJECUCION

;*** En esta seccion debeis implementar la ejecucion de cada einstr.

; ESCRIBID EN CADA ETIQUETA LA FASE DE EJECUCION DE CADA INSTRUCCION

ETRA:

```
MOVE.W D0,D1
JSR OBTENER_Xa    ;Salto a subrutina para obtener Xa en D0
JSR OBTENER_Xb    ;Obtenemos Xb en D1
JSR ENCONTRAR_REGISTRO_Xa
JSR ENCONTRAR_REGISTRO_Xb ;Y direccion Xb en A1

MOVE.W D0,(A1) ;Ejecutamos la intrucción deseada

JSR ACTU_FLAG_Z_A1
JSR ACTU_FLAG_N_A1
BRA FETCH
```

EADD:

```
MOVE.W D0,D1
JSR OBTENER_Xa    ;Salto a subrutina para obtener Xa en D0
JSR OBTENER_Xb    ;Obtenemos Xb en D1
JSR ENCONTRAR_REGISTRO_Xa
JSR ENCONTRAR_REGISTRO_Xb

ADD.W D0,(A1) ;Ejecutamos la instrucción deseada

MOVE.W SR,D2 ;Registro estados 68K a D2 para comprobar
              ;los flags del resultado de la operación
JSR ACTU_FLAG_C_RESULTADO
JSR ACTU_FLAG_Z_RESULTADO
JSR ACTU_FLAG_N_RESULTADO
BRA FETCH
```

ESUB:

```
MOVE.W D0,D1
JSR OBTENER_Xa    ;Salto a subrutina para obtener Xa en D0
JSR OBTENER_Xb    ;Obtenemos Xb en D1
JSR ENCONTRAR_REGISTRO_Xa
JSR ENCONTRAR_REGISTRO_Xb

NOT.W D0          ;D0'
ADDQ.W #1,D0      ;D0' + 1
ADD.W D0,(A1)     ;Operación resta: D1 + (D0'+1)

MOVE.W SR,D2 ;Registro estados 68K a D2 para comprobar
```



```

        ;los flags del resultado de la operación
JSR ACTU_FLAG_C_RESULTADO
JSR ACTU_FLAG_Z_RESULTADO
JSR ACTU_FLAG_N_RESULTADO
BRA FETCH

```

ENAN:

```

MOVE.W D0,D1
JSR OBTENER_Xa    ;Salto a subrutina para obtener Xa en D0
JSR OBTENER_Xb    ;Obtenemos Xb en D1
JSR ENCONTRAR_REGISTRO_Xa
JSR ENCONTRAR_REGISTRO_Xb

AND.W D0,(A1)
NOT.W (A1)

```

```

MOVE.W SR,D2 ;Registro estados 68K a D2 para comprobar
        ;los flags del resultado de la operación
JSR ACTU_FLAG_Z_RESULTADO
JSR ACTU_FLAG_N_RESULTADO
BRA FETCH

```

ESTC:

```

MOVE.W D0,D1
;JSR OBTENER_Xa    ;Salto a subrutina para obtener Xa en D0
JSR OBTENER_Xb    ;Obtenemos Xb en D1
;JSR ENCONTRAR_REGISTRO_Xa
JSR ENCONTRAR_REGISTRO_Xb

;Extensión de signo a Xb
MOVE.W D0,D1
AND.W #$00FF,D1 ;las k estan en los 8 primeros bits
EXT.W D1
MOVE.W D1,(A1)

```

```

JSR ACTU_FLAG_Z_A1
JSR ACTU_FLAG_N_A1
BRA FETCH

```

EINC:

```

MOVE.W D0,D1
;JSR OBTENER_Xa    ;Salto a subrutina para obtener Xa en D0
JSR OBTENER_Xb    ;Obtenemos Xb en D1
;JSR ENCONTRAR_REGISTRO_Xa
JSR ENCONTRAR_REGISTRO_Xb

;Extensión de signo a Xb

```

```

MOVE.W D0,D1
AND.W #$00FF,D1 ;las k estan en los 8 primeros bits
EXT.W D1
ADD.W D1,(A1) ;Sumamos la k extendida al contenido de Xb

MOVE.W SR,D2 ;Registro estados 68K a D2 para comprobar
        ;los flags del resultado de la operación
JSR ACTU_FLAG_C_RESULTADO
JSR ACTU_FLAG_Z_RESULTADO
JSR ACTU_FLAG_N_RESULTADO
BRA FETCH
ELOA:
    JSR OBTENER_M

    ADD.W D0,D0
    MOVE.W D0,A1

    MOVE.W EMEM(A1),ET6

    LEA.L ET6,A1      ;Para la act. de flags

    JSR ACTU_FLAG_Z_A1
    JSR ACTU_FLAG_N_A1
    BRA FETCH
ELOAX:
    BTST.L #3,D0 ;Miramos la i para saber si se trata de B0 o B1
    BEQ LOAX_B0
    MOVE.W D0,D5
    JSR OBTENER_M    ; D0 = M
    ADD.W EB1,D0     ;M+B1
    ADD.W D0,D0      ;*2
    MOVE.W D0,A1

    LOAX_T6_T7:
    BTST.L #2,D5
    BEQ LOAX_T6

    MOVE.W EMEM(A1),ET7

    LEA.L ET7,A1      ; Para la act. de flags
    JSR ACTU_FLAG_Z_A1
    JSR ACTU_FLAG_N_A1
    BRA FETCH

```

```

LOAX_B0:
MOVE.W D0,D5
JSR OBTENER_M      ; D0 = M
ADD.W EB0,D0      ;M+B0
ADD.W D0,D0      ;*2
MOVE.W D0,A1

```

```

BRA LOAX_T6_T7

```

```

LOAX_T6:
MOVE.W EMEM(A1),ET6

```

```

LEA.L ET6,A1      ; Para la act. de flags
JSR ACTU_FLAG_Z_A1
JSR ACTU_FLAG_N_A1
BRA FETCH

```

ESTO:

```

JSR OBTENER_M      ; D0 = M
ADD.W D0,D0
MOVE.W D0,A1      ; A1 = indice para @M

```

```

MOVE.W ET6,EMEM(A1)
BRA FETCH

```

ESTOX:

```

BTST.L #3,D0      ;Miramos la i para saber si se trata de B0 o B1
BEQ STOX_B0
MOVE.W D0,D5
JSR OBTENER_M      ; D0 = M
ADD.W EB1,D0      ;M+B1
ADD.W D0,D0      ;*2
MOVE.W D0,A1

```

```

STOX_T6_T7:
BTST.L #2,D5
BEQ STOX_T6

```

```

MOVE.W ET7,EMEM(A1)
BRA FETCH

```

```

STOX_B0:
MOVE.W D0,D5
JSR OBTENER_M      ; D0 = M
ADD.W EB0,D0      ;M+B0
ADD.W D0,D0      ;*2
MOVE.W D0,A1

BRA STOX_T6_T7

STOX_T6:
MOVE.W ET6,EMEM(A1)
BRA FETCH
EBRI:
JSR OBTENER_M
MOVE.W D0,EPC
BRA FETCH

EBRZ:
MOVE.W ESR,D3
BTST.L #2,D3
BNE EBRI      ; si Z=1, M -> PC
BRA FETCH
EBRN:
MOVE.W ESR,D3
BTST.L #0,D3
BNE EBRI      ; si N = 1, M -> PC
BRA FETCH
ESTP:
BRA FINAL
;--- FEEXEC: FIN EJECUCION

;--- ISUBR: INICIO SUBROUTINAS
;*** Aqui debeis incluir las subrutinas que necesite vuestra solucion
;*** SALVO DECOD, que va en la siguiente seccion

; ESCRIBID VUESTRO CODIGO AQUI
OBTENER_Xa:
AND.W #$0070,D0
LSR.L #4,D0
RTS

OBTENER_Xb:
AND.W #$0700,D1
LSR.L #8,D1
RTS

```

OBTENER_M:

```
AND.W #$0FF0,D0
    LSR.L #4,D0
    RTS
```

ENCONTRAR_REGISTRO_Xa:

```
BTST.L #2,D0
BEQ ET_R0_Xa    ;Salta si tercer bit 0 (B0,B1,R2,R3)
                ;Sino, tercer bit 1 (R4,R5,T6,T7)
BTST.L #1,D0
BEQ ET_R10_Xa   ;Salta si secuencia 10 (R4,R5)
                ;Sino, secuencia 11 (T6,T7)
BTST.L #0,D0
BEQ ET_R110_Xa  ;Salta si T6 (secuencia 110)

MOVE.W ET7,D0   ;Sino, se trata del registro T7 (111)
    RTS
```

ET_R0_Xa:

```
BTST.L #1,D0
BEQ ET_R00_Xa   ;Salta si secuencia 00 (B0,B1)
                ;Sino, secuencia 01 (R2,R3)
BTST.L #0,D0
BEQ ET_R010_Xa  ;Salta si R2 (010)

MOVE.W ER3,D0   ;Sino, secuencia 011 (R3)
    RTS
```

ET_R00_Xa:

```
BTST.L #0,D0
BEQ ET_R000_Xa  ;Salta si R0 (000)

MOVE.W EB1,D0   ;Sino, R1 (001)
    RTS
```

ET_R000_Xa:

```
MOVE.W EB0,D0   ;Sino, R0 (000)
    RTS
```

ET_R010_Xa:

```
MOVE.W ER2,D0   ;Sino, secuencia 010 (R2)
    RTS
```

ET_R10_Xa:

```
BTST.L #0,D0
BEQ ET_R100_Xa ;Si salta, se trata de R4
MOVE.W ER5,D0 ;Sino, secuencia 101 (R5)
RTS
```

```
ET_R100_Xa:
MOVE.W ER4,D0 ;Se trata del registro R4 (100)
RTS
```

```
ET_R110_Xa:
MOVE.W ET6,D0 ;Se trata del registro T6 (110)
RTS
```

```
ENCONTRAR_REGISTRO_Xb:
BTST.L #2,D1
BEQ ET_R0_Xb ;Salta si tercer bit 0 (B0,B1,R2,R3)
;Sino, tercer bit 1 (R4,R5,T6,T7)
BTST.L #1,D1
BEQ ET_R10_Xb ;Salta si secuencia 10 (R4,R5)
;Sino, secuencia 11 (T6,T7)
BTST.L #0,D1
BEQ ET_R110_Xb ;Salta si T6 (secuencia 110)

LEA.L ET7,A1 ;Sino, se trata del registro T7 (111)
RTS
```

```
ET_R0_Xb:
BTST.L #1,D1
BEQ ET_R00_Xb ;Salta si secuencia 00 (B0,B1)
;Sino, secuencia 01 (R2,R3)
BTST.L #0,D1
BEQ ET_R010_Xb ;Salta si R2 (010)

LEA.L ER3,A1 ;Sino, secuencia 011 (R3)
RTS
```

```
ET_R00_Xb:
BTST.L #0,D1
BEQ ET_R000_Xb ;Salta si R0 (000)

LEA.L EB1,A1 ;Sino, R1 (001)
RTS
```

```
ET_R000_Xb:
    LEA.L EB0,A1    ;Sino, R0 (000)
    RTS
```

```
ET_R010_Xb:
    LEA.L ER2,A1    ;Sino, secuencia 010 (R2)
    RTS
```

```
ET_R10_Xb:
    BTST.L #0,D1
    BEQ ET_R100_Xb ;Si salta, se trata de R4
    LEA.L ER5,A1    ;Sino, secuencia 101 (R5)
    RTS
```

```
ET_R100_Xb:
    LEA.L ER4,A1    ;Se trata del registro R4 (100)
    RTS
```

```
ET_R110_Xb:
    LEA.L ET6,A1    ;Se trata del registro T6 (110)
    RTS
```

;Actualización del flag Z según el valor del contenido
;de A1

```
ACTU_FLAG_Z_A1:
    MOVE.W ESR,D3    ;Movemos registro de estado a D3
    MOVE.W (A1),D2    ;Contenido A1 a D2
    CMP.W #0,D2      ;Si z = 0, salta y actualizamos flag z = 0
    BNE FLAG_Z0
    BSET #2,D3        ;Ponemos específicamente el flag Z a 1
    MOVE.W D3,ESR     ;Actualizamos registro de estado
    RTS
```

```
FLAG_Z0:
    BCLR #2,D3        ;Ponemos específicamente el flag Z a 0
    MOVE.W D3,ESR     ;Actualizamos registro de estado
    RTS
```

;Actualización del flag N según el valor del contenido
;de A1

```
ACTU_FLAG_N_A1:
    MOVE.W ESR,D3    ;Movemos registro de estado a D3
    MOVE.W (A1),D2    ;Contenido A1 a D2
    CMP.W #0,D2      ;Si N = 0, salta
    BGE FLAG_Z0
    BSET #0,D3        ;Ponemos específicamente el flag N a 1
```

```
MOVE.W D3,ESR    ;Actualizamos registro de estado
RTS
```

FLAG_N0:

```
BCLR #0,D3    ;Ponemos especificamente el flag N a 0
MOVE.W D3,ESR    ;Actualizamos registro de estado
RTS
```

ACTU_FLAG_C_RESULTADO:

```
MOVE.W ESR,D3
BTST.L #0,D2
BNE FLAG_C1
BCLR #1,D3
MOVE.W D3,ESR
RTS
```

FLAG_C1:

```
BSET #1,D3
MOVE.W D3,ESR
RTS
```

ACTU_FLAG_Z_RESULTADO:

```
MOVE.W ESR,D3
BTST.L #2,D2
BNE FLAG_Z1
BCLR #2,D3
MOVE.W D3,ESR
RTS
```

FLAG_Z1:

```
BSET.L #2,D3
MOVE.W D3,ESR
RTS
```

ACTU_FLAG_N_RESULTADO:

```
MOVE.W ESR,D3
BTST.L #3,D2
BNE FLAG_C1
BCLR #0,D3
MOVE.W D3,ESR
RTS
```

FLAG_N1:

```
BSET.L #0,D3
MOVE.W D3,ESR
```


RTS

;--- FSUBR: FIN SUBROUTINAS

;--- 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:

; ESCRIBID VUESTRO CODIGO AQUI

MOVE.L D1,-(SP)

MOVE.W 8(SP),D1

BTST.L #15,D1

BEQ ET_0 ;Si salta, primer bit 0

;Sino, primer bit 1

BTST.L #14,D1

BEQ ET_10 ;Si salta, secuencia 10

;Sino, secuencia 11

MOVE.W #13,10(SP) ;Instruccion STP encontrada

BRA FINAL_DECOD

ET_0:

BTST.L #14,D1

BEQ ET_00 ;Si salta, secuencia 00

;Sino, secuencia 01

BTST.L #13,D1

BEQ ET_010 ;Si salta, secuencia 010

;Sino, secuencia 011

BTST.L #12,D1

BEQ ET_0110 ;Si salta, secuencia 0110

;Sino, secuencia 0111

MOVE.W #9,10(SP) ;Instruccion STOX encontrada

BRA FINAL_DECOD

ET_00:

BTST.L #13,D1

BEQ ET_000 ;Si salta, secuencia 000

```

        ;Sino, secuencia 001
BTST.L #12,D1
BEQ ET_0010    ;Si salta, secuencia 0010
        ;Sino, secuencia 0011
MOVE.W #5,10(SP) ;Instruccion INC encontrada
BRA FINAL_DECOD

```

```

ET_000:
    BTST.L #12,D1
    BEQ ET_0000    ;Si salta, secuencia 0000
        ;Sino, secuencia 0001
    BTST.L #11,D1
    BEQ ET_00010    ;Si salta, secuencia 00010
        ;Sino, secuencia 00011
    MOVE.W #2,10(SP) ;Instruccion SUB encontrada
    BRA FINAL_DECOD

```

```

ET_00010:
    MOVE.W #1,10(SP) ;Instruccion ADD encontrada
    BRA FINAL_DECOD

```

```

ET_0000:
    MOVE.W #0,10(SP) ;Instruccion TRA encontrada
    BRA FINAL_DECOD

```

```

ET_0010:
    BTST.L #11,D1
    BEQ ET_00100    ;Si salta, secuencia 00100
        ;Sino, secuencia 00101
    MOVE.W #4,10(SP) ;Instruccion STC encontrada
    BRA FINAL_DECOD

```

```

ET_00100:
    MOVE.W #3,10(SP) ;Instruccion NAN encontrada
    BRA FINAL_DECOD

```

```

ET_010:
    BTST.L #12,D1
    BEQ ET_0100    ;Si salta, secuencia 0100
        ;Sino, secuencia 0101

```

```
MOVE.W #7,10(SP) ;Instruccion LOAX encontrada  
BRA FINAL_DECOD
```

```
ET_0100:  
MOVE.W #6,10(SP) ;Instruccion LOA encontrada  
BRA FINAL_DECOD
```

```
ET_0110:  
MOVE.W #8,10(SP) ;Instruccion STO encontrada  
BRA FINAL_DECOD
```

```
ET_10:  
BTST.L #13,D1  
BEQ ET_100 ;Si salta, secuencia 100  
;Sino, secuencia 101  
MOVE.W #12,10(SP) ;Instruccion BRN encontrada  
BRA FINAL_DECOD
```

```
ET_100:  
BTST.L #12,D1  
BEQ ET_1000 ;Si salta, secuencia 1000  
;Sino, secuencia 1001  
MOVE.W #11,10(SP) ;Instruccion BRZ encontrada  
BRA FINAL_DECOD
```

```
ET_1000:  
MOVE.W #10,10(SP) ;Instruccion BRI encontrada  
BRA FINAL_DECOD
```

```
FINAL_DECOD:
```

```
MOVE.L (SP)+,D1  
RTS
```

```
FINAL:  
;--- FDECOD: FIN DECOD  
END START
```

