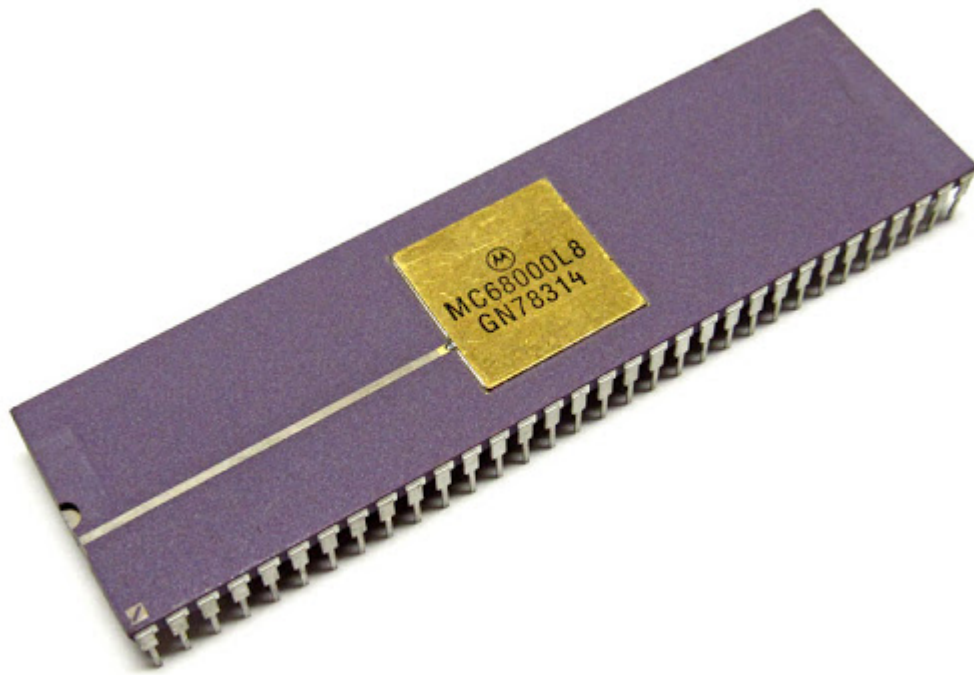


SUIBC: Super UIB Computer



PRÁCTICA II ESTRUCTURA DE COMPUTADORES I

BENÍTEZ QUINTANA, DAVID
GARCÍA ORTIZ, ALEX

ÍNDICE

I INTRODUCCIÓN

II RESUMEN GENERAL

III RUTINA DE DECODIFICACIÓN

IV TABLA DE SUBROUTINA

V TABLA DE REGISTRO DEL 68K

VI TABLA DE VARIABLES ADICIONALES

VII CONJUNTO DE PRUEBAS

VIII CONCLUSIONES

XI CÓDIGO FUENTE

INTRODUCCIÓN

Durante la segunda práctica de la asignatura de Estructura de Computadores I, se nos ha pedido hacer una máquina elemental dentro del procesador Motorola 68K utilizando el Ensamblador EASy68K.

La máquina que nos piden emular se llama **SUIBC (Super UIB Computer)**, la cual se caracteriza por tener una memoria de 2^8 posiciones y 4 registros de 16 bits, 2 de los cuales sirven la función de acumuladores (R0 y R1), mientras que los otros 2 solo pueden almacenar valores (R2 y R3).

Las operaciones que convierten esta máquina en una máquina elemental son las siguientes:

Id	Mnemonic	Codificación	Acción	Flags
0	STO Rj,M	100X jXXX mmmm mmmm	$M \leftarrow [Rj]$	n.s.a.
1	LOA Rj,M	101X jXXX mmmm mmmm	$Rj \leftarrow [M]$	C = n.s.a., Z y N = s.v.Rj
2	CMP Ra, Rj	0100 00XX Xjaa XXXX	$[Rj] - [Ra]$	C, Z y N = s.v.r.
3	ADD Ra, Rj	0100 01XX Xjaa XXXX	$Rj \leftarrow [Rj] + [Ra]$	C, Z y N = s.v.Rj
4	SUB Ra, Rj	0100 10XX Xjaa XXXX	$Rj \leftarrow [Rj] - [Ra]$	C, Z y N = s.v.Rj
5	NOR Ra, Rj	0100 11XX Xjaa XXXX	$Rj \leftarrow [Rj] \text{ nor } [Ra]$	C = n.s.a., Z y N = s.v.Rj
6	SET #c, Rb	0101 XXcc cccc ccbb	$Rb \leftarrow c$ (Ext. signo)	C = n.s.a., Z y N = s.v.Rb
7	ADQ #c, Rb	0110 XXcc cccc ccbb	$Rb \leftarrow [Rb] + c$ (Ext. signo)	C, Z y N = s.v.Rb
8	TRA Ra, Rb	0111 XXXX XXaa XXbb	$Rb \leftarrow [Ra]$	C = n.s.a., Z y N = s.v.Rb
9	JMZ M	0000 mmmm mmmm XXXX	Si Z = 1, $PC \leftarrow M$	n.s.a.
10	JMN M	0001 mmmm mmmm XXXX	Si N = 1, $PC \leftarrow M$	n.s.a.
11	JMI M	0010 mmmm mmmm XXXX	$PC \leftarrow M$	n.s.a.
12	HLT	11XX XXXX XXXX XXXX	Detiene la máquina	n.s.a.

Es una máquina elemental ya que nos permite realizar operaciones de almacenamiento de registro a memoria, de memoria a registro y de registro a registro (**STO, LOA y TRA**), operaciones aritméticas que abarcan todas las operaciones posibles (**ADD, SUB**), operaciones lógicas que permiten realizar cualquier otra operación lógica (**NOR**) y operaciones de salto condicionales e incondicionales (**JMZ, JMN y JMI**). Además, el SUIBC permite realizar operaciones para cambiar los flags (**CMP**) y operaciones de

almacenamiento y suma de constantes (**SET Y ADQ**). Por último, como es obvio la máquina posee una instrucción para detener el funcionamiento de la máquina (**HLT**).

LEYENDA

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

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

Ra, Rb: Cualquier registro R; ver aa y bb.*

Rj: Rb o R1, ver j.*

aa y bb: Índice del registro según: $\left\{ \begin{array}{ll} 00 - R0 & 01 - R1 \text{ }^* \\ 10 - R2 & 11 - R3 \end{array} \right.$

j: Índice del registro **R0** (**j = 0**) o del registro **R1** (**j = 1**). *

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

n.s.a.: No se actualizan

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

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

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

* **Estos registros pueden operar con un máximo de 16 bits.**

II RESUMEN GENERAL

Fase Fetch

La fase **Fetch** en nuestro programa inicia en la primera iteración con un borrado del contenido del **EPC**, el contador de programa (**PC**) del **SUIBC**, este borrado no aporta gran ayuda más que prevenir cualquier error, por muy improbable que sea.

La verdadera fase **Fetch** inicia con la transferencia del contenido en **EPC** al registro de direcciones del 68K, **A0**. El **EPC** avanza de 1 en 1, lo cual fuerza a doblar el valor de **A0**, ya que este registro se utilizará para acceder al componente o **einstrucción** siguiente del vector **EMEM**, el cual toma el papel de memoria principal del **SUIBC**.

Una vez se ha obtenido el valor pertinente del **EPC** en **A0**, se obtiene la siguiente **einstrucción** contenida en **EMEM**, el cual es un vector de datos de tamaño **word**. Una vez se ha obtenido la **einstrucción**, esta se almacena en **EIR**, el registro de instrucciones (**IR**) del **SUIBC**. Después de todo esto el **EPC** se incrementa en 1.

Fase Decodificación

Al iniciar la práctica, en la sesión 1 se nos pedía crear un vector de datos tamaño **word** llamado **CODE**, cuyo propósito era el de almacenar la **einstrucción** leída decodificada, es decir, obtenía el **opcode** de la instrucción a ejecutar.

En la sesión 1, la fase de decodificación se hacía mediante una **subrutina de usuario**, la cual fue transformada en una de **librería** más tarde, precisamente en la sesión 2. En esta subrutina se utilizan una serie de saltos que siguen un **árbol de decodificación** que abarca los 6 bits más significativos de cada **einstrucción (15 - 10)**. Cuando se llegaba a un **opcode**, este enviaba el opcode de la **einstrucción** a ejecutar al principio de la pila, el cual era recogido por el vector **CODE**. Este vector **CODE** solo tenía como función la de verificar que se había realizado correctamente la decodificación, ya que en la sesión 3 este fue suprimido e intercambiado por un valor del **0** al **12**, el cual determinaba el salto que debía realizar la sección de **JMPLIST**, la cual contiene todos los saltos a las etiquetas de cada **einstrucción**. Esta sección tenía un orden determinado, donde el primer salto era a la **einstrucción ESTO** y el último salto a **EHLT**.

Fase Ejecución

La fase de ejecución se inicia con la sección **JMPLIST**, la cual, como se ha mencionado anteriormente, es la responsable directa de realizar el salto adecuado, siguiendo el orden establecido.

Al hacer el salto a la ejecución de la **instrucción** a ejecutar, este realiza la operación pertinente, haciendo uso de registros del 68K y subrutinas diseñadas con un propósito general para aumentar la eficiencia del programa. Al finalizar la ejecución, se realiza un salto incondicional a la fase de **Fetch**, iniciando el ciclo y siguiendo las mismas instrucciones, siguiendo así hasta que se ejecute la **instrucción EHLT**.

III RUTINA DE DECODIFICACIÓN

En esta sección de la memoria se observa el **árbol de codificación** empleado para diseñar la subrutina de librería **DECOD**, la cual se encarga de decodificar la **einstrucción** pasada por parámetro.

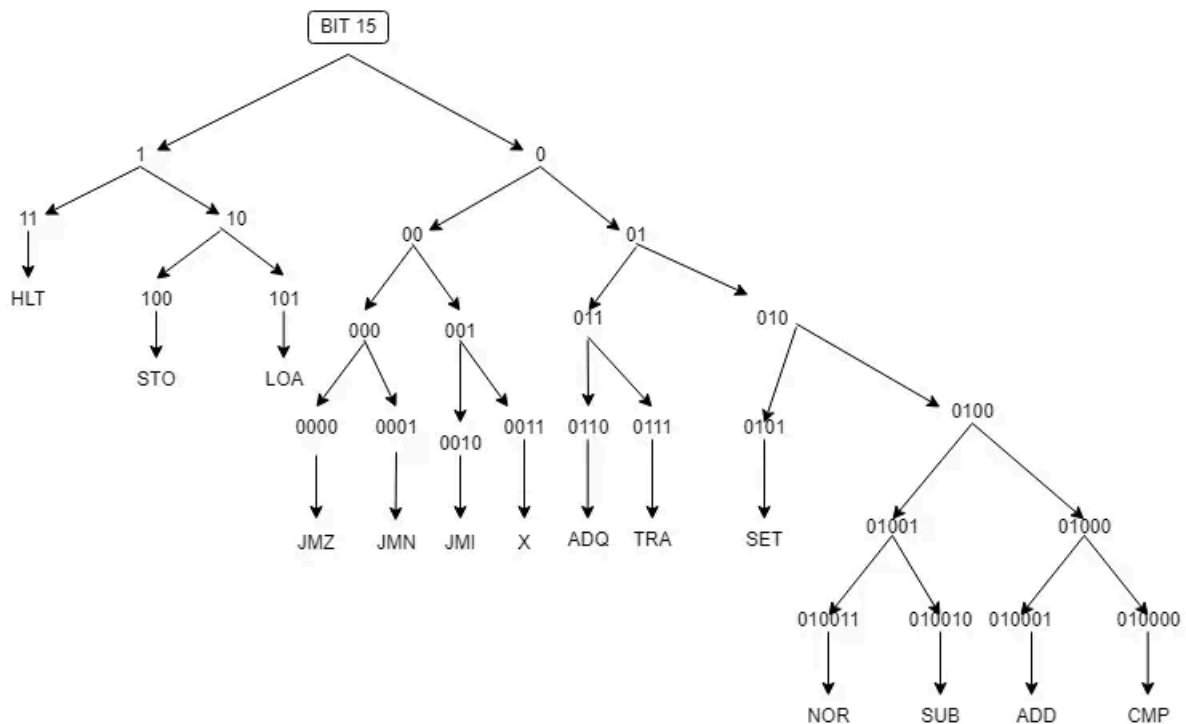


Gráfico del árbol de decodificación empleado

Como se observa, el **árbol de decodificación** está compuesto de un máximo de 6 niveles, los cuales se corresponden a un bit de la **einstrucción**. Este proceso inicia en el **bit más significativo**, el bit 15, y a partir de ahí se ramifica en dos ramas madre, la rama **0** y la rama **1**. La rama **1**, es considerablemente más pequeña, esto se debe a que solo hay **3 instrucciones** cuyo bit más significativo es **1**, estas instrucciones son **HLT**, **STO** y **LOA**. En cambio la rama **0**, contiene las **10 instrucciones restantes**.

Este árbol avanza en base al bit leído, es decir, a cada nivel se lee un bit menos significativo, iniciando en el bit **15** y acabando en el bit **10**.

Como se puede observar, la única combinación imposible es **0011**, el cual no pertenece a ninguna **einstrucción** del **SUIBC**, si este caso se diese, únicamente se podría utilizar como una variable.

IV TABLA DE SUBROUTINA

Esta sección contiene las subrutinas utilizadas para llevar a cabo la ejecución del **eprograma** introducido en el vector **EMEM**. En este programa se han utilizado un total de **11** subrutinas, contando **DECOD**, las cuales son de **librería**. Estas subrutinas han sido reutilizadas en varias fases de ejecución de las diferentes **einstrucciones**.

Estas son las subrutinas utilizadas en orden de arriba a abajo según han sido diseñadas:

MEM_DECOD_1:

Esta subrutina tiene como función la de decodificar la posición de memoria **m** codificada dentro de las **einstrucciones ESTO y ELOA** entre los **7** bits **menos significativos**. Para decodificar esta posición de memoria se ha hecho uso de una **máscara** cuyo valor era **00FF** en **hexadecimal**.

Parámetros de entrada: Un espacio al inicio de la pila y el **EIR**, contenido en **D0**.

Registros utilizados: Para esta subrutina se ha utilizado únicamente el registro **D0**, el cual almacena la instrucción a decodificar la posición de memoria **m**.

Parámetros de salida: Posición de memoria **m** en el espacio al inicio de la pila.

MEM_DECOD_2:

Esta subrutina tiene como función la de decodificar la posición de memoria **m** codificada dentro de las **einstrucciones de salto JMZ , JMN y JMI** entre los bits **11** y **7**. Para decodificar esta posición de memoria se ha utilizado una **máscara** cuyo valor era **0FF0** en **hexadecimal**, más tarde se ha utilizado la instrucción **LSR** para mover el contenido 2 bits y poder manipular el valor verdadero.

Parámetros de entrada: Un espacio al inicio de la pila y el **EIR**, contenido en **D0**.

Registros utilizados: Para esta subrutina se ha utilizado únicamente el registro **D0**, el cual almacena la instrucción a decodificar la posición de memoria **m**.

Parámetros de salida: Posición de memoria **m** en el espacio al inicio de la pila.

REG_LOA_1:

Esta subrutina tiene como propósito el de obtener el valor contenido en un registro **Rj (ER1/0)** codificado en las **einstrucciones ECMP, EADD, ESUB y ENOR**, el cual está codificado en el bit **6**. Esta subrutina se ejecuta siguiendo un árbol de decodificación de **2** ramas únicamente, la rama **1** y la rama **0**. Para ello se ha realizado la operación **BTST** en el bit **6**, y dependiendo del resultado, se obtiene el contenido de **ER1** o **ER0**. Esta instrucción no se ha utilizado en **ESTO**, ya que el bit **j** se encuentra en otra posición.

Parámetros de entrada: Un espacio al inicio de la pila y el **EIR**, contenido en **D0**.

Registros utilizados: Para esta subrutina se utilizado únicamente el registro **D0**, el cual contiene la instrucción anteriormente introducida en la pila.

Parámetros de salida: Contenido del registro **Rj** en el espacio al inicio de la pila.

REG_LOA_2:

Esta subrutina tiene como propósito el de obtener el valor contenido en un registro **Raa (ER3/2/1/0)** codificado en las **einstrucciones ECMP, EADD, ESUB, ENOR y ETRA**, el cual está codificado en los bits **5** y **4**. Esta subrutina se ejecuta siguiendo un árbol de decodificación de **2** ramas, las cuales se dividen de acuerdo con el bit **5**, dividiéndose en ramas **3,2** y ramas **1,0**. Para ello se ha realizado la operación **BTST** en los bits **5** y **4**, y dependiendo del resultado final, se obtiene el contenido de **ER3**, **ER2**, **ER1** o **ER0**.

Parámetros de entrada: Un espacio al inicio de la pila y el **EIR**, contenido en **D0**.

Registros utilizados: Para esta subrutina se utilizado únicamente el registro **D0**, el cual contiene la instrucción anteriormente introducida en la pila.

Parámetros de salida: Contenido del registro **Ra** en el espacio al inicio de la pila.

REG_LOA_3:

Esta subrutina tiene como propósito el de obtener el valor contenido en un registro **Rbb (ER3/2/1/0)** codificado en las **einstrucciones ESET, EADQ y ETRA**, el cual está codificado en los bits **1 y 0**. Esta subrutina se ejecuta siguiendo un árbol de decodificación de **2** ramas, las cuales se dividen de acuerdo con el bit **1**, dividiéndose en ramas **3,2** y ramas **1,0**. Para ello se ha realizado la operación **BTST** en los bits **1 y 0**, y dependiendo del resultado final, se obtiene el contenido de **ER3 , ER2, ER1 o ER0**.

Parámetros de entrada: Un espacio al inicio de la pila y el **EIR**, contenido en **D0**.

Registros utilizados: Para esta subrutina se utilizó únicamente el registro **D0**, el cual contiene la instrucción anteriormente introducida en la pila.

Parámetros de salida: Contenido del registro **Ra** en el espacio al inicio de la pila.

REG_STO_1:

Esta subrutina permite almacenar el valor pasado por parámetro al registro **Rj (ER1/0)** codificado en las **einstrucciones EADD, ESUB y ENOR** en el bit **6**. Esta subrutina se ejecuta siguiendo el mismo árbol que **REG_LOA_1**, lo único que cambia es el hecho de que almacena el valor pasado por parámetro al registro **Rj** codificado.

Parámetros de entrada: Un espacio al inicio de la pila, el valor a almacenar, contenido en **D4** y el **EIR**, contenido en **D0**.

Registros utilizados: Para esta subrutina se utilizó únicamente el registro **D0**, el cual contiene la instrucción anteriormente introducida en la pila.

Parámetros de salida: No tiene, debido a que almacena el valor en los **registros ER1 o ER0** según resulte la decodificación.

REG_STO_2:

Esta subrutina permite almacenar el valor pasado por parámetro al registro **Rb (ER3/2/1/0)** codificado en las **einstrucciones ESET, EADQ y ETRA** en los bits **1 y 0**. Esta subrutina se ejecuta siguiendo el mismo árbol que **REG_LOA_3**, lo único que cambia es el hecho de que almacena el valor pasado por parámetro al registro **Rb** codificado.

Parámetros de entrada: Un espacio al inicio de la pila, el valor a almacenar, contenido en **D4** y el **EIR**, contenido en **D0**.

Registros utilizados: Para esta subrutina se utilizado únicamente el registro **D0**, el cual contiene la instrucción anteriormente introducida en la pila.

Parámetros de salida: No tiene, debido a que almacena el valor en los **eregistros ER3, ER2, ER1 o ER0** según resulte la decodificación.

CONSTANT_DECOD:

Esta subrutina permite almacenar el valor de la constante **c** codificada en las **einstrucciones ESET y EADQ**. Esta constante se encuentra entre los bits **9 y 2**. Para ello se ha utilizado una máscara cuyo valor es **03FC** en **hexadecimal**, la cual apunta exactamente a esos bits mencionados.

Parámetros de entrada: Un espacio al inicio de la pila y el **EIR**, contenido en **D0**.

Registros utilizados: Para esta subrutina se utilizado únicamente el registro **D0**, el cual contiene la instrucción anteriormente introducida en la pila.

Parámetros de salida: El valor de la constante **c** en el espacio del inicio de la pila.

FLAGS_1:

Esta subrutina permite actualizar los flags **Z**, **N** y **C** en el **ESR**, el cual es el registro de estado (**SR**) del **SUIBC**. Esta subrutina verifica el valor pasado por parámetro en tamaño **long**, para ello verifica unos bits concretos, los cuales son el bit **16**, el cual verifica si **C** se tiene activar o desactivar; el bit **15**, que al ser el **bit más significativo** de los registros de la máquina, determina si es un valor negativo, y por ende verifica si **N** tiene que activarse o desactivarse; y por último, se realiza una comparación del valor pasado por parámetro con el valor **0**, determinando si **Z** tiene que activarse o desactivarse. Esta subrutina se utiliza en las **instrucciones ECMP, EADD, ESUB y EADQ**, las únicas operaciones aritméticas.

Parámetros de entrada: El **ESR**, contenido en **D2** y el valor a verificar, contenido en **D4**.

Registros utilizados: Para esta subrutina se han utilizado **2** registros, **D0** para almacenar el valor a verificar pasado por parámetro, y **D2** el cual contiene el **ESR**.

Parámetros de salida: El **ESR** actualizado después de las verificaciones.

FLAGS_2:

Esta subrutina permite actualizar los flags **Z** y **N** en el **ESR**. Esta subrutina verifica el valor pasado por parámetro en tamaño **word**, contrario que la subrutina **FLAGS_1**, ya que no debe verificar el valor del bit **16**. Para verificar que flags se tienen que activar se ha utilizado el mismo proceso que en la subrutina **FLAGS_1**, lo que se ha eliminado el proceso de verificación del flag **C**, ya que esta subrutina se utiliza en **instrucciones**, que no actualizan el flag **C**. Esta subrutina se utiliza en las **instrucciones ELOA, ENOR, ESET y ETRA**, las únicas operaciones aritméticas.

Parámetros de entrada: El **ESR**, contenido en **D2** y el valor a verificar, contenido en **D4**.

Registros utilizados: Para esta subrutina se han utilizado **2** registros, **D0** para almacenar el valor a verificar pasado por parámetro, y **D2** el cual contiene el **ESR**.

Parámetros de salida: El **ESR** actualizado después de las verificaciones.

DECOD:

Esta subrutina, como se ha mencionado anteriormente, es la protagonista en la fase de decodificación, esta se caracteriza por ejecutarse siguiendo un árbol de codificación de gran tamaño.

Parámetros de entrada: Un espacio al inicio de la pila para guardar el valor del salto, el **EIR** a decodificar el opcode.

Registros utilizados: Para esta subrutina se han utilizado **2** registros, **D0** el cual almacena la **instrucción** a decodificar y **D2**, la cual almacena el bit a verificar, iniciando como 15 y decrementando a cada nivel.

Parámetros de salida: El valor de salto para **JMPLIST** localizado en el espacio de inicio de la pila, el cual se almacena en **D1**, para después ser multiplicado por **6**, debido al tamaño de cada instrucción de salto.

V TABLA DE REGISTRO DEL 68K

Los registros que se han utilizado en el **SUIBC** son los siguientes:

Registros de datos:

D0: Utilizado para almacenar el **EIR** en cada ejecución, y como auxiliar en las subrutinas.

D1: Utilizado para almacenar el valor del salto en la sección **JMPLIST**.

D2: Utilizado para almacenar el **ESR** en cada ejecución que lo requiera, y como auxiliar en las subrutinas.

D3: Utilizado para almacenar la posición de memoria a introducir en el **EPC** en las **instrucciones** de salto, **JMZ**, **JMN** y **JMI**.

D4: Utilizado como operando, almacén del contenido de los registros acumuladores, y almacén para la constante en **ESET**.

D5: Utilizado como operando.

Registros de direcciones:

A0: Utilizado para recorrer el vector **EMEM**.

A1: Utilizado para los saltos en **JMPLIST**.

A2: Utilizado para obtener el valor y almacenar en las posiciones de memoria en las **instrucciones** **ESTO** y **ELOA**, respectivamente.

VI TABLA DE VARIABLES ADICIONALES

Para la **SUIBC** no se ha declarado ninguna variable adicional.

VII CONJUNTO DE PRUEBAS

Para verificar que la máquina funciona como es debido se han realizado un total de **3** pruebas con diferentes programas:

PRUEBA 1:

EMEM: DC.W \$A807,\$7012,\$2050,\$4860,\$C000,\$4460,\$C000,\$0001

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000FE0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
00000FF0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
00001000:	A8	07	70	12	20	50	48	60	C0	00	44	60	C0	00	01
00001010:	C0	00	00	07	00	00	00	02	00	01	00	00	00	00	78

Este programa realiza una **carga** al registro **ER1** de la variable en la posición **7**, cuyo valor es 1. Luego realiza la **transferencia** de registros entre **R1 y R2**. A continuación, realiza un **salto incondicional** a la posición **5**, la cual cae en una operación de **suma** entre **R1 y R2**. Después de esta instrucción, la máquina se detiene por la instrucción **EHLT**.

Resultado en registros:

ER0: \$0000

ER1: \$0002

ER2: \$0002

ER3: \$0000

PRUEBA 2:

EMEM: DC.W \$A00C,\$00A0,\$7002,\$5000,\$A80D,\$00A0,\$4420,\$63FD
DC.W \$00A0,\$2060,\$800E,\$C000,\$0003,\$0004,\$0000

00	000FE0	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000FE0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
00000FF0:	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
00001000:	A0	0C	00	A0	70	02	50	00	A8	0D	00	A0	44	20	63	FD	
00001010:	00	A0	20	60	80	0E	C0	00	00	03	00	04	00	0C	C0	00	
00001020:	00	0C	00	0C	00	00	00	03	00	00	00	04	42	78	10	20	

Este programa realiza una **carga** a **R0** de la variable en la posición de memoria **12**, cuyo valor es **3**. Si este valor es **0**, salta a la etiqueta **EXIT**. Luego hace la **transferencia** entre los registros **R0** y **R2**. Luego, a **R0**, le introduce el valor de la **constante 0**. A continuación, a **R1** le introduce la variable en la posición de memoria **13**, cuyo valor es 4. Si este valor es **0**, salta a la etiqueta **EXIT**. Después de esto, se forma un **bucle**, cuyo inicio es en la posición de memoria **6**, en esa posición se encuentra una operación de **suma** entre **R2** y **R0**, a cada iteración se le **suma** la constante **-1**, resultado en un **decremento** de 1. Este bucle dura hasta que **R1** es **igual a 0**. Al acabar el bucle, salta a la etiqueta **EXIT**, donde se **guarda** el contenido de **R0**, cuyo valor es **12**, en la variable **C** cuya posición de memoria es **14**. Después de esto, se detiene la máquina mediante **EHLT**.

Resultado en registros:

ER0: \$000C

ER1: \$0000

ER2: \$0003

ER3: \$0000

PRUEBA 3:

EMEM: DC.W \$A008,\$5025,\$4040,\$0060,\$600C,\$2020,\$7003,\$C000,\$0003

```
00000FE0 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000FE0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000FF0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00001000: A0 08 50 25 40 40 00 60 60 0C 20 20 70 03 C0 00
00001010: 00 03 C0 00 00 08 00 09 00 09 00 00 00 09 00 00
```

Este programa realiza una **carga** de la variable **A** en la posición de memoria **8** al registro **R0**. Luego le introduce la **constante 9** al registro **R1**. Después entra en un **bucle** con la etiqueta **LOOP**, cuya posición de memoria es **3**, en esta posición también se encuentra una operación de **comparación** entre **R0** y **R1**, para después realizar un **salto** a la etiqueta **EXIT**, este caso se da cuando **R0** es igual a **R1**, es decir, igual a **9**. Por ello a cada iteración se le **suma** la **constante 3** a **R0**. Después de esto, se **transfiere** el contenido de **R0** a **R3**. Después de esta operación, se detiene la máquina con **EHLT**.

Resultado en registros:

ER0: \$0009

ER1: \$0009

ER2: \$0000

ER3: \$0009

VIII CONCLUSIONES

Esta práctica nos ha ayudado a entender el funcionamiento de varios conceptos de ensamblador, tales como el funcionamiento de las subrutinas de librería. Además hemos aprendido varias instrucciones del EASy68K las cuales no conocíamos y son muy útiles para casos como los que se han dado en los casos de prueba.

XI CÓDIGO FUENTE

A continuación, se encuentra el código de la máquina **SUIBC**.

*-----

* Title : PRAFIN24

* Written by : Quintana Benitez, David // Ortiz García, Alex

* Date : 26/05/2024

* Description: Emulador de la SUIBC

*-----

ORG \$1000

EMEM: DC.W \$A008,\$5025,\$4040,\$0060,\$600C,\$2020,\$7003,\$C000,\$0003

EIR: DC.W 0 ;registro de instruccion

EPC: DC.W 0 ;contador de programa

ER0: DC.W 0 ;registro R0

ER1: DC.W 0 ;registro R1

ER2: DC.W 0 ;registro R2

ER3: DC.W 0 ;registro R3

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

START:

CLR.W EPC ;ESTABLECE EL EPC A 0

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 ;MOVER VALOR DE EPC A A 0

ADD.W A0,A0 ;ADAPTAR A TAMAÑO WORD

MOVE.W EMEM(A0),EIR ;PASAR EL CONTENIDO DE EMEM EN LA POSICIÓN A0 A D0

ADDQ.W #1,EPC ;INCREMENTAR EN 1 EL 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
IBRDECOD:
    ;--- FBRDECOD: FIN SALTO A DECOD
    SUBQ.L #2,A7      ;RESERVAR ESPACIO PARA NÚMERO DE SALTO
    MOVE.W EIR,-(A7)  ;INSTRUCCIÓN A DECODIFICAR
    JSR DECOD
    ADDQ.L #2,A7
    MOVE.W (A7)+,D1   ;D1 = VALOR DE SALTO

;--- 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 ESTO
    JMP ELOA
    JMP ECMP
    JMP EADD
    JMP ESUB
    JMP ENOR
    JMP ESET
    JMP EADQ
    JMP ETRA
    JMP EJMZ
    JMP EJMN
    JMP EJMI
    JMP EHLT
;--- 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

ESTO:  ;0 - STORE
    ;STO Rj, M
    ;M <-- [Rj]
    ;INSTRUCCIÓN QUE PERMITE ALMACENAR EL CONTENIDO DE Rj EN
    ;LA POSICIÓN DE MEMORIA M
    ;CODIFICACIÓN = 100 X JXXX MMMMMMMMM
    ;NO SE ACTUALIZAN LOS FLAGS

    MOVE.W EIR,D0 ;D0 = EIR

```

```
;PREPARACIÓN DE SUBROUTINA MEM_DECOD_1
SUBQ.L #2,A7 ;ESPACIO RESERVADO PARA POSICIÓN M
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR POSICIÓN M
JSR MEM_DECOD_1
ADDQ.L #2,A7
MOVE.W (A7)+,A2 ;POSICIÓN M
```

```
ADD.L A2,A2 ;ADAPTAR A WORD
```

```
;VERIFICAR A QUE REGISTRO SE ACCEDE
BTST #11,D0
BEQ REG_01
```

```
REG_11: ;REGISTRO R1
MOVE.W ER1,EMEM(A2)
JMP SKIP_S
```

```
REG_01: ;REGISTRO R0
MOVE.W ER0,EMEM(A2)
```

```
SKIP_S:
JMP FETCH
```

```
ELOA: ;1 - LOAD
;LOA Rj,M
;Rj <-- [M]
;INSTRUCCIÓN QUE PERMITE ALMACENAR EL CONTENIDO EN LA
;POSICIÓN DE MEMORIA M EN EL EREGISTRO Rj
;CODIFICACIÓN = 101 X J XXX MMMMMMMMM
;C = NO SE ACTUALIZA, Z/N = SE ACTUALIZAN
```

```
MOVE.W EIR,D0 ;D0 = EIR
MOVE.W ESR,D2 ;D2 = ESR
```

```
;PREPARACIÓN DE SUBROUTINA MEM_DECOD_1
SUBQ.L #2,A7 ;ESPACIO RESERVADO PARA POSICIÓN M
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR POSICIÓN M
JSR MEM_DECOD_1
ADDQ.L #2,A7
MOVE.W (A7)+,A2 ;POSICIÓN M
```

```
ADD.L A2,A2 ;ADAPTAR A WORD
```

```
;PREPARACIÓN SUBROUTINA FLAGS_2
MOVE.W D2,-(A7) ;ESR
MOVE.W EMEM(A2),-(A7) ;VALOR A VERIFICAR
JSR FLAGS_2
```

ADDQ.L #2,A7
MOVE.W (A7)+,ESR ;ACTUALIZAR VALOR ESR

;VERIFICAR A QUE REGISTRO SE ALMACENA
BTST #11,D0
BEQ REG_02

REG_12: ;REGISTRO R1
MOVE.W EMEM(A2),ER1
JMP SKIP_L

REG_02: ;REGISTRO R0
MOVE.W EMEM(A2),ER0

SKIP_L:
JMP FETCH

ECMP: ;2 - COMPARE
;CMP Ra,Rj
;[Rj] - [Ra]
;INSTRUCCIÓN QUE REALIZA LA DIFERENCIA ENTRE EL
;CONTENIDO DE LOS EREGISTROS Rj y Ra, ESTA DIFERENCIA
;PERMITE MODIFICAR LOS FLAGS, NO ALMACENA EL VALOR EN
;NINGÚN REGISTRO NI ESPACIO DE MEMORIA
;CODIFICACIÓN = 010000 XXX J AA XXXX
;Z/N/C = SE ACTUALIZAN

MOVE.W EIR,D0 ;D0 = EIR
MOVE.W ESR,D2 ;D2 = ESR

;D4 = Rj
;PREPARACIÓN SUBROUTINA REG_LOA_1
SUBQ.L #2,A7 ;ESPACIO RESERVADO PARA CONTENIDO EN Rj
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO Rj
JSR REG_LOA_1
ADDQ.L #2,A7

MOVE.W (A7)+,D4

;EXTENDER EL SIGNO DE Rj
EXT.L D4

;D5 = Ra
;PREPARACIÓN SUBROUTINA REG_LOA_2
SUBQ.L #2,A7 ;ESPACIO RESERVADO PARA CONTENIDO EN Ra
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO Ra
JSR REG_LOA_2
ADDQ.L #2,A7

MOVE.W (A7)+,D5

;EXTENDER EL SIGNO DE Ra

EXT.L D5

;REALIZAR OPERACIÓN

NOT D5

ADDQ.W #1,D5

EXT.L D5 ;EXTENDER SIGNO DE Ra

ADD.L D5,D4 ;Rj + (Ra' + 1)

;PREPARACIÓN SUBROUTINA FLAGS_1

MOVE.W D2,-(A7) ;ESR

MOVE.L D4,-(A7) ;VALOR A VERIFICAR

JSR FLAGS_1

ADDQ.L #4,A7

MOVE.W (A7)+,ESR ;ACTUALIZAR VALOR ESR

JMP FETCH

EADD: ;3 - ADD

;ADD Ra,Rj

;Rj <-- [Rj] + [Ra]

;INSTRUCCIÓN QUE REALIZA LA SUMA ENTRE EL CONTENIDO DE

;LOS REGISTROS Rj Y Ra, EL RESULTADO SE ALMACENA

;EN Rj

;CODIFICACIÓN = 010001 XXX J AA XXXX

;Z/N/C = SE ACTUALIZAN

MOVE.W EIR,D0 ;D0 = EIR

MOVE.W ESR,D2 ;D2 = ESR

;D4 = Rj

;PREPARACIÓN SUBROUTINA REG_LOA_1

SUBQ.L #2,A7 ;ESPACIO RESERVADO PARA CONTENIDO DE Rj

MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO

JSR REG_LOA_1

ADDQ.L #2,A7

MOVE.W (A7)+,D4

;EXTENDER EL SIGNO DE Rj

EXT.L D4

;D5 = Ra

;PREPARACIÓN SUBROUTINA REG_LOA_2

SUBQ.L #2,A7 ;ESPACIO RESERVADO PARA CONTENIDO DE Ra
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO
JSR REG_LOA_2
ADDQ.L #2,A7

MOVE.W (A7)+,D5

;EXTENDER EL SIGNO DE Ra
EXT.L D5

;REALIZAR OPERACIÓN
ADD.L D5,D4 ;Rj <- Rj + Ra

;PREPARACIÓN SUBROUTINA FLAGS_1
MOVE.W D2,-(A7) ;ESR
MOVE.L D4,-(A7) ;VALOR A VERIFICAR
JSR FLAGS_1
ADDQ.L #4,A7
MOVE.W (A7)+,ESR ;ACTUALIZAR VALOR ESR

;PREPARACIÓN SUBROUTINA REG_STO_1
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO
MOVE.W D4,-(A7) ;VALOR A ALMACENAR
JSR REG_STO_1
ADDQ.L #4,A7

JMP FETCH

ESUB: ;4 - SUBSTRACT

;SUB Ra,Rj
;Rj <-- [Rj] - [Ra]
;INSTRUCCIÓN QUE REALIZA LA DIFERENCIA ENTRE EL CONTENIDO DE
;LOS EREGISTRO Rj Y Ra, EL RESULTADO SE ALMACENA
;EN Rj
;CODIFICACIÓN = 010010 XXX J AA XXXX
;Z/N/C = SE ACTUALIZAN

MOVE.W EIR,D0 ;D0 = EIR
MOVE.W ESR,D2 ;D2 = ESR

;D4 = Rj
;PREPARACIÓN SUBROUTINA REG_LOA_1
SUBQ.L #2,A7 ;ESPACIO RESERVADO PARA CONTENIDO DE Rj
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO
JSR REG_LOA_1
ADDQ.L #2,A7

MOVE.W (A7)+,D4

;EXTENDER EL SIGNO DE Rj
EXT.L D4

;D5 = Ra
;PREPARACIÓN SUBROUTINA REG_LOA_2
SUBQ.L #2,A7 ;ESPACIO RESERVADO PARA CONTENIDO DE Ra
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO
JSR REG_LOA_2
ADDQ.L #2,A7

MOVE.W (A7)+,D5

;EXTENDER EL SIGNO DE Ra
EXT.L D5

;REALIZAR OPERACIÓN
NOT.W D5
ADDQ.W #1,D5
ADD.L D5,D4 ;Rj <- Rj + (Rb' + 1)

;PREPARACIÓN SUBROUTINA FLAGS_1
MOVE.W D2,-(A7) ;ESR
MOVE.L D4,-(A7) ;VALOR A VERIFICAR
JSR FLAGS_1
ADDQ.L #4,A7
MOVE.W (A7)+,ESR ;ACTUALIZAR VALOR ESR

;PREPARACIÓN SUBROUTINA REG_STO_1
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO
MOVE.W D4,-(A7) ;VALOR A ALMACENAR
JSR REG_STO_1
ADDQ.L #4,A7

JMP FETCH

ENOR: ;5 - NOT OR
;NOR Ra,Rj
;Rj <-- [Rj] nor [Ra]
;INSTRUCCIÓN QUE REALIZA LA OPERACIÓN LÓGICA NOR ENTRE
;EL CONTENIDO DE LOS REGISTROS Rj Y Ra, EL RESULTADO
;SE ALMACENA EN Rj
;CODIFICACIÓN = 010011 XXX J AA XXXX
;C = NO SE ACTUALIZA, Z/N = SE ACTUALIZAN

MOVE.W EIR,D0 ;D0 = EIR
MOVE.W ESR,D2 ;D2 = ESR

```
;D4 = Rj
;PREPARACIÓN SUBROUTINA REG_LOA_1
SUBQ.L #2,A7 ;ESPACIO RESERVADO PARA CONTENIDO DE Rj
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO
JSR REG_LOA_1
ADDQ.L #2,A7
```

```
MOVE.W (A7)+,D4
```

```
;EXTENDER EL SIGNO DE Rj
EXT.L D4
```

```
;D5 = Ra
;PREPARACIÓN SUBROUTINA REG_LOA_2
SUBQ.L #2,A7 ;ESPACIO RESERVADO PARA CONTENIDO DE Ra
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO
JSR REG_LOA_2
ADDQ.L #2,A7
```

```
MOVE.W (A7)+,D5
```

```
;EXTENDER EL SIGNO DE Ra
EXT.L D5
```

```
;REALIZAR OPERACIÓN
OR D5,D4
NOT D4 ;Rj <- (Rj ^ Ra)'
```

```
;PREPARACIÓN SUBROUTINA FLAGS_2
MOVE.W D2,-(A7) ;ESR
MOVE.W D4,-(A7) ;VALOR A VERIFICAR
JSR FLAGS_2
ADDQ.L #2,A7
MOVE.W (A7)+,ESR ;ACTUALIZAR VALOR ESR
```

```
;PREPARACIÓN SUBROUTINA REG_STO_1
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO
MOVE.W D4,-(A7) ;VALOR A ALMACENAR
JSR REG_STO_1
ADDQ.L #4,A7
```

```
JMP FETCH
```

```
ESET: ;6 - SET
;SET #c,Rb
;Rb <-- c
;INSTRUCCIÓN QUE PERMITE ALMACENAR UNA CONSTANTE DENTRO
;DEL REGISTRO Rb
```

;CODIFICACIÓN = 0101 XX CCCCCCCC BB
;C = NO SE ACTUALIZA, Z/N = SE ACTUALIZAN

MOVE.W EIR,D0 ;D0 = EIR
MOVE.W ESR,D2 ;D2 = ESR

;D4 = c
;PREPARACIÓN SUBROUTINA CONSTANT_DECOD
SUBQ.L #2,A7 ;ESPACIO RESERVADO PARA CONSTANTE CODIFICADA
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR CONSTANTE
JSR CONSTANT_DECOD
ADDQ.L #2,A7

MOVE.W (A7)+,D4

;EXTENDER EL SIGNO DE LA CONSTANTE
EXT.L D4

;PREPARACIÓN SUBROUTINA FLAGS_2
MOVE.W D2,-(A7) ;ESR
MOVE.W D4,-(A7) ;VALOR A VERIFICAR
JSR FLAGS_2
ADDQ.L #2,A7
MOVE.W (A7)+,ESR ;ACTUALIZAR VALOR ESR

;PREPARACIÓN SUBROUTINA REG_STO_2
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO
MOVE.W D4,-(A7) ;VALOR A ALMACENAR
JSR REG_STO_2
ADDQ.L #4,A7

JMP FETCH

EADQ: ;7 - ADD QUICK

;ADQ #c,Rb
;Rb <-- [Rb] + c
;INSTRUCCIÓN QUE REALIZA LA SUMA ENTRE EL CONTENIDO
;DEL REGISTRO Rb Y UNA CONSTANTE CODIFICADA EN LA
;MISMA INSTRUCCIÓN
;CODIFICACIÓN = 0110 XX CCCCCCCC BB
;Z/N/C = SE ACTUALIZAN

MOVE.W EIR,D0 ;D0 = EIR
MOVE.W ESR,D2 ;D2 = ESR

;D4 = Rb
;PREPARACIÓN SUBROUTINA REG_LOA_3
SUBQ.L #2,A7 ;ESPACIO RESERVADO PARA CONTENIDO DE Rb

```
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO
JSR REG_LOA_3
ADDQ.L #2,A7
```

```
MOVE.W (A7)+,D4
```

```
;D5 = c
;PREPARACIÓN SUBROUTINA CONSTANT_DECOD
SUBQ.L #2,A7 ;ESPACIO RESERVADO PARA CONSTANTE CODIFICADA
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR CONSTANTE
JSR CONSTANT_DECOD
ADDQ.L #2,A7
```

```
MOVE.W (A7)+,D5
```

```
;EXTENDER EL SIGNO DE LA CONSTANTE
EXT.L D5
```

```
;REALIZAR OPERACIÓN
ADD.L D5,D4 ;Rb <- Rb + c
```

```
;PREPARACIÓN SUBROUTINA FLAGS_1
MOVE.W D2,-(A7) ;ESR
MOVE.L D4,-(A7) ;VALOR A VERIFICAR
JSR FLAGS_1
ADDQ.L #4,A7
MOVE.W (A7)+,ESR ;ACTUALIZAR VALOR DE ESR
```

```
;PREPARACIÓN SUBROUTINA REG_STO_2
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO
MOVE.W D4,-(A7) ;VALOR A ALMACENAR
JSR REG_STO_2
ADDQ.L #4,A7
```

```
JMP FETCH
```

ETRA: ;8 - TRANSFER

```
;TRA Ra,Rb
;Rb <-- [Ra]
;INSTRUCCIÓN QUE PERMITE MOVER EL CONTENIDO DEL EREGISTRO
;Ra AL EREGISTRO Rb
;CODIFICACIÓN = 0111 XXXXXX AAXX BB
;C = NO SE ACTUALIZA, Z/N = SE ACTUALIZA
```

```
MOVE.W EIR,D0 ;D0 = EIR
MOVE.W ESR,D2 ;D2 = ESR
```

```
;D5 = Ra
```

```
;PREPARACIÓN SUBROUTINA REG_LOA_2
SUBQ.L #2,A7 ;ESPACIO RESERVADO PARA CONTENIDO DE Ra
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO
JSR REG_LOA_2
ADDQ.L #2,A7
```

```
MOVE.W (A7)+,D5
```

```
;EXTENDER EL SIGNO DE Rq
EXT.L D5
```

```
;D4 = Rb
;OPERACIÓN A REALIZAR
MOVE.L D5,D4 ;Rb <- Ra
```

```
;PREPARACIÓN SUBROUTINA FLAGS_2
MOVE.W D2,-(A7) ;ESR
MOVE.W D4,-(A7) ;VALOR A VERIFICAR
JSR FLAGS_2
ADDQ.L #2,A7
MOVE.W (A7)+,ESR ;ACTUALIZAR VALOR ESR
```

```
;PREPARACIÓN REG_STO_2
MOVE.W D0,-(A7) ;INSTRUCCIÓN A DECODIFICAR REGISTRO
MOVE.W D4,-(A7) ;VALOR A ALMACENAR
JSR REG_STO_2
ADDQ.L #4,A7
```

```
JMP FETCH
```

```
EJMZ: ;9 - JUMP IF Z
;JMZ M
;If Z = 1, EPC <-- M
;INSTRUCCIÓN QUE REALIZA UN SALTO A LA POSICIÓN DE MEMORIA M
;SI EL FLAG Z ESTÁ ACTIVADO (Z = 1)
;CODIFICACIÓN = 0000 MMMMMMMMM XXXX
;NO SE ACTUALIZAN LOS FLAGS
```

```
MOVE.W EIR,D0 ;D0 = EIR
MOVE.W ESR,D2 ;D2 = ESR
```

```
;D3 = M
;PREPARACIÓN SUBROUTINA MEM_DECOD_2
SUBQ.L #2,A7
MOVE.W D0,-(A7)
JSR MEM_DECOD_2
ADDQ.L #2,A7
MOVE.W (A7)+,D3
```

```
;VERIFICAR VALOR DE FLAG Z
BTST #2,D2
BEQ SKIP2
```

```
MOVE.W D3,EPC ;ACTUALIZAR EPC CON VALOR EN D3
```

```
SKIP2:
    JMP FETCH
```

```
EJMN: ;10 - JUMP IF N
    ;JMZ N
    ;If N = 1, EPC <-- M
    ;INSTRUCCIÓN QUE RELIZA UN SALTO A LA POSICIÓN DE MEMORIA M
    ;SI EL FLAG N ESTÁ ACTIVADO (N = 1)
    ;CODIFICACIÓN = 0001 MMMMMMMM XXXX
    ;NO SE ACTUALIZAN LOS FLAGS
```

```
MOVE.W EIR,D0 ;D0 = INSTRUCCIÓN
MOVE.W ESR,D2 ;D2 = ESR
```

```
;D3 = M
;PREPARACIÓN SUBROUTINA MEM_DECOD_2
SUBQ.L #2,A7
MOVE.W D0,-(A7)
JSR MEM_DECOD_2
ADDQ.L #2,A7
MOVE.W (A7)+,D3
```

```
;VERIFICAR VALOR DE FLAG N
BTST #1,D2
BEQ SKIP3
```

```
MOVE.W D3,EPC ;ACTUALIZAR EPC CON VALOR EN D3
```

```
SKIP3:
    JMP FETCH
```

```
EJMI: ;11 - UNCONDITIONAL JUMP
    ;JMI M
    ;EPC <-- M
    ;INSTRUCCIÓN QUE REALIZA UN SALTO A LA POSICIÓN DE MEMORIA M
    ;CODIFICACIÓN = 0010 MMMMMMMM XXXX
    ;NO SE ACTUALIZAN LOS FLAGS
```

```
MOVE.W EIR,D0 ;D0 = EIR
```

```
;D3 = M
```


;PREPARACIÓN SUBROUTINA MEM_DECOD_2

SUBQ.L #2,A7

MOVE.W D0,-(A7)

JSR MEM_DECOD_2

ADDQ.L #2,A7

MOVE.W (A7)+,D3

MOVE.W D3,EPC ;ACTUALIZAR EPC CON VALOR EN D3

JMP FETCH

EHLT: ;12 - HALT

;HLT

;INSTRUCCIÓN QUE DETIENE LA MÁQUINA

;CODIFICACIÓN = 11 XXXXXXXXXXXXXXXX

SIMHALT

;--- 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

MEM_DECOD_1: ;SUBROUTINA DE LIBRERÍA QUE PERMITE OBTENER

;LA DIRECCIÓN A CARGAR/GUARDAR (LOA/STO)

MOVE.L D0,-(A7) ;D0 = INSTRUCCIÓN

MOVE.W 8(A7),D0

AND.W #\$00FF,D0

MOVE.W D0,10(A7)

MOVE.L (A7)+,D0

RTS

MEM_DECOD_2: ;SUBROUTINA DE LIBRERÍA QUE PERMITE OBTENER

;LA DIRECCIÓN A ALMACENAR EN EL EPC (JMZ, JMN, JMI)

MOVE.L D0,-(A7)

MOVE.W 8(A7),D0 ;D0 = INSTRUCCIÓN

AND.W #\$0FF0,D0

LSR.W #4,D0

MOVE.W D0,10(A7)

MOVE.L (A7)+,D0
RTS

REG_LOA_1: ;SUBROUTINA DE LIBRERÍA QUE PERMITE OBTENER
;EL VALOR DEL REGISTRO CODIFICADO J(CMP, ADD, SUB, NOR)

MOVE.L D0,-(A7)
MOVE.W 8(A7),D0 ;D0 = INSTRUCCIÓN

BTST #6,D0
BEQ J_0

J_1: ;REGISTRO ER1
MOVE.W ER1,10(A7)
JMP SKIP_SR1

J_0: ;REGISTRO ER0
MOVE.W ER0,10(A7)

SKIP_SR1:
MOVE.L (A7)+,D0
RTS

REG_LOA_2: ;SUBROUTINA DE LIBRERÍA QUE PERMITE OBTENER
;EL VALOR DEL REGISTRO CODIFICADO AA(CMP, ADD, SUB, NOR, TRA)

MOVE.L D0,-(A7)
MOVE.W 8(A7),D0 ;D0 = INSTRUCCIÓN

BTST #5,D0
BEQ A_101

;ÁRBOL DE CODIFICACIÓN PARA Ra

A_321:
BTST #4,D0
BEQ A_21

A_31: ;REGISTRO ER3
MOVE.W ER3,10(A7)
JMP SKIP_SR2

A_21: ;REGISTRO ER2
MOVE.W ER2,10(A7)
JMP SKIP_SR2

A_101:

```
BTST #4,D0
BEQ A_01
```

```
A_11: ;REGISTRO ER1
      MOVE.W ER1,10(A7)
      JMP SKIP_SR2
```

```
A_01: ;REGISTRO ER0
      MOVE.W ER0,10(A7)
```

```
SKIP_SR2:
      MOVE.L (A7)+,D0
      RTS
```

```
REG_LOA_3: ;SUBROUTINA DE LIBRERÍA QUE PERMITE OBTENER
           ;EL VALOR DEL REGISTRO CODIFICADO BB (SET, ADQ, TRA)
```

```
MOVE.L D0,-(A7)
MOVE.W 8(A7),D0 ;D0 = INSTRUCCIÓN
```

```
BTST #1,D0
BEQ B_101
```

```
;ÁRBOL DE CODIFICACIÓN PARA Rb
B_321:
```

```
      BTST #0,D0
      BEQ B_21
```

```
B_31: ;REGISTRO ER3
      MOVE.W ER3,10(A7)
      JMP SKIP_SR3
```

```
B_21: ;REGISTRO ER2
      MOVE.W ER2,10(A7)
      JMP SKIP_SR3
```

```
B_101:
      BTST #4,D0
      BEQ B_01
```

```
B_11: ;REGISTRO ER1
      MOVE.W ER1,10(A7)
      JMP SKIP_SR3
```

```
B_01: ;REGISTRO ER0
      MOVE.W ER0,10(A7)
```

```
SKIP_SR3:
```

```
MOVE.L (A7)+,D0
RTS
```

REG_STO_1: ;SUBROUTINA DE LIBRERÍA QUE PERMITE ALMACENAR
;EL VALOR AL REGISTRO CODIFICADO J(ADD, SUB, NOR)

```
MOVE.L D0,-(A7)
MOVE.W 10(A7),D0 ;D0 = INSTRUCCIÓN
```

```
BTST #6,D0
BEQ J_01
```

```
J_11: ;REGISTRO ER1
MOVE.W 8(A7),ER1
JMP SKIP_SR4
```

```
J_01: ;REGISTRO ER0
MOVE.W 8(A7),ER0
```

```
SKIP_SR4:
MOVE.L (A7)+,D0
RTS
```

REG_STO_2: ;SUBROUTINA DE LIBRERÍA QUE PERMITE ALMACENAR
;EL VALOR AL REGISTRO CODIFICADO B(SET, ADQ, TRA)

```
MOVE.L D0,-(A7)
MOVE.W 10(A7),D0 ;D0 = INSTRUCCIÓN
```

```
BTST #1,D0
BEQ B_102
```

;ÁRBOL DE CODIFICACIÓN PARA Rb

```
B_322: ;REGISTROS ER3 Y ER2
BTST #0,D0
BEQ B_22
```

```
B_32: ;REGISTRO ER3
MOVE.W 8(A7),ER3
JMP SKIP_SR5
```

```
B_22: ;REGISTRO ER2
MOVE.W 8(A7),ER2
JMP SKIP_SR5
```

```
B_102: ;REGISTROS ER1 Y ER0
BTST #0,D0
BEQ B_02
```

```
B_12: ;REGISTROS ER1
      MOVE.W 8(A7),ER1
      JMP SKIP_SR5
```

```
B_02: ;REGISTROS ER0
      MOVE.W 8(A7),ER0
```

```
SKIP_SR5:
      MOVE.L (A7)+,D0
      RTS
```

```
CONSTANT_DECOD: ;SUBROUTINA DE LIBRERÍA QUE DECODIFICA LA CONSTANTE
                ;CODIFICADA DENTRO DE LA INSTRUCCIÓN
```

```
MOVE.L D0,-(A7)
MOVE.W 8(A7),D0 ;D0 = INSTRUCCIÓN
```

```
AND.W #$03FC,D0 ;#$03FC = 0000 0011 1111 1100
LSR.W #2,D0 ;DESPLAZAR LA MÁSCARA DOS BITS A LA DERECHA
EXT.W D0 ;EXTENDER SIGNO DE CONSTANTE
MOVE.W D0,10(A7)
```

```
MOVE.L (A7)+,D0
RTS
```

```
FLAGS_1: ;SUBROUTINA DE LIBRERÍA QUE VERIFICA SI SE TIENE
          ;QUE ACTIVAR ALGÚN FLAG (ZNC)(CMP, ADD, SUB, ADQ)
```

```
MOVE.L D0,-(A7)
MOVE.L D2,-(A7)
MOVE.L 12(A7),D0 ;D0 = VALOR A VERIFICAR
MOVE.W 16(A7),D2 ;D2 = ESR
```

```
FLAGZ: ;VERIFICACIÓN FLAG Z
      CMP.W #0,D0 ;COMPARAR VALOR CON 0
      BEQ UPDZ
```

```
BCLR #2,D2 ;Z = 0
JMP FLAGN
```

```
UPDZ:
      BSET #2,D2 ;Z = 1
```

```
FLAGN: ;VERIFICACIÓN FLAG N
      BTST #15,D0 ;VERIFICAR EL VALOR DEL BIT 15 (BIT MÁS SIGNIFICATIVO)
      BNE UPDN
```

```
BCLR #1,D2 ;N = 0
JMP FLAGC
```

```
UPDN:
    BSET #1,D2 ;N = 1
```

```
FLAGC: ;VERIFICACIÓN FLAG C
    BTST #16,D0 ;VERIFICAR EL VALOR DEL BIT 16
            ;(1 BIT MAYOR AL TAMAÑO WORD)
    BNE UPDC
```

```
BCLR #0,D2 ;C = 0
JMP SKIP_SR6
```

```
UPDC:
    BSET #0,D2 ;C = 1
```

```
SKIP_SR6:
    MOVE.W D2,16(A7)
    MOVE.L (A7)+,D2
    MOVE.L (A7)+,D0
    RTS
```

```
FLAGS_2: ;SUBROUTINA DE LIBRERÍA QUE VERIFICA SI SE TIENE
            ;QUE ACTIVAR ALGÚN FLAG (ZN)(LOA, NOR, SET, TRA)
```

```
MOVE.L D0,-(A7)
MOVE.L D2,-(A7)
MOVE.W 12(A7),D0 ;D0 = VALOR A VERIFICAR
MOVE.W 14(A7),D2 ;D2 = ESR
```

```
FLAGZ1: ;VERIFICACIÓN FLAG Z
    CMP.W #0,D0 ;COMPARAR VALOR CON 0
    BEQ UPDZ1
```

```
BCLR #2,D2 ;Z = 0
JMP FLAGN1
```

```
UPDZ1:
    BSET #2,D2 ;Z = 1
```

```
FLAGN1: ;VERIFICACIÓN FLAG N
    BTST #15,D0 ;VERIFICAR EL VALOR DEL BIT 15 (BIT MÁS SIGNIFICATIVO)
    BNE UPDN1
```

```
BCLR #1,D2 ;N = 0
JMP SKIP_SR7
```

```
UPDN1:
    BSET    #1,D2    ;N = 1
```

```
SKIP_SR7:
    MOVE.W  D2,14(A7)
    MOVE.L  (A7)+,D2
    MOVE.L  (A7)+,D0
    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  D0,-(A7)
MOVE.L  D2,-(A7)
SUBQ.L  #2,A7    ;ESPACIO RESERVADO PARA NÚMERO DE SALTO
MOVE.W  14(A7),D0 ;D0 = INSTRUCCIÓN
MOVE.W  #15,D2   ;D2 = NÚMERO DE BIT A TESTEAR (MSB BIT = 15)
BTST.L  D2,D0    ;BTST.L #15,D0
BEQ B0
```

```
B1:
    SUBQ.L  #1,D2 ;D2 = 14
    BTST.L  D2,D0 ;BTST.L #14,D0
    BEQ B10
```

```
B11: ;EHLT
    MOVE.W  #12,0(A7)
    JMP ENDROUT
```

```
B10:
    SUBQ.L  #1,D2 ;D2 = 13
    BTST.L  D2,D0 ;BTST.L #13,D0
    BEQ B100
```

```
B101: ;ELOA
    MOVE.W  #1,0(A7)
    JMP ENDROUT
```

```
B100: ;ESTO
    MOVE.W  #0,0(A7)
    JMP ENDROUT
```

B0:

```
SUBQ.L #1,D2 ;D2 = 14
BTST.L D2,D0 ;BTST.L #14,D0
BEQ B00
```

B01:

```
SUBQ.L #1,D2 ;D2 = 13
BTST.L D2,D0 ;BTST.L #13,D0
BEQ B010
```

B011:

```
SUBQ.L #1,D2 ;D2 = 12
BTST.L D2,D0 ;BTST.L #12,D0
BEQ B0110
```

B0111: ;ETRA

```
MOVE.W #8,0(A7)
JMP ENDROUT
```

B0110: ;EADQ

```
MOVE.W #7,0(A7)
JMP ENDROUT
```

B00:

```
SUBQ.L #1,D2 ;D2 = 13
BTST.L D2,D0 ;BTST.L #13,D0
BEQ B000
```

B001: ;EJMI

```
MOVE.W #11,0(A7)
JMP ENDROUT
```

B010:

```
SUBQ.L #1,D2 ;D2 = 12
BTST.L D2,D0 ;BTST.L #12,D0
BEQ B0100
```

B0101: ;ESET

```
MOVE.W #6,0(A7)
JMP ENDROUT
```

B000:

```
SUBQ.L #1,D2 ;D2 = 12
BTST.L D2,D0 ;BTST.L #12,D0
BEQ B0000
```

B0001: ;EJMN


```

    MOVE.W #10,0(A7)
    JMP ENDROUT

B0000: ;EJMZ
    MOVE.W #9,0(A7)
    JMP ENDROUT

B0100:
    SUBQ.L #1,D2 ;D2 = 11
    BTST.L D2,D0 ;BTST.L #11,D0
    BEQ B01000

B01001:
    SUBQ.L #1,D2 ;D2 = 10
    BTST.L D2,D0 ;BTST.L #10,D0
    BEQ B010010

B010011: ;ENOR
    MOVE.W #5,0(A7)
    JMP ENDROUT

B010010: ;ESUB
    MOVE.W #4,0(A7)
    JMP ENDROUT

B01000:
    SUBQ.L #1,D2 ;D2 = 10
    BTST.L D2,D0 ;BTST.L #10,D0
    BEQ B010000

B010001: ;EADD
    MOVE.W #3,0(A7)
    JMP ENDROUT

B010000: ;ECMP
    MOVE.W #2,0(A7)

ENDROUT: ;ENDROUT
    MOVE.W 0(A7),16(A7)

    ADDQ.L #2,A7
    MOVE.L (A7)+,D2
    MOVE.L (A7)+,D0
    RTS
;--- FDECOD: FIN DECOD

END    START

```