Práctica 2

Actividad evaluable: 25 % sobre la nota final de la asignatura
Fecha máxima de entrega: lunes 30 de mayo de 2022 (hasta las 23:59h)

Introducción

Un emulador permite que un ordenador pueda ejecutar un programa escrito para una máquina diferente. En esta práctica final debéis implementar, en lenguaje ensamblador del 68K, un programa que **emule** la ejecución de programas escritos para **una máquina elemental** dada. Estos programas deberán estar escritos usando el conjunto de instrucciones de la máquina en cuestión, y el emulador deberá funcionar para cualquier programa que respete dicho conjunto.

Para llevar a cabo esta emulación, todas las partes de la máquina elemental **se definirán en la memoria** del 68K. Por un lado, vuestro programa debe ser capaz de leer de la memoria del 68K una **secuencia de instrucciones codificadas** como *words*, de acuerdo al conjunto de instrucciones de la propia máquina elemental. Para cada una de estas instrucciones, vuestro programa aplicará un proceso de **decodificación** para determinar de qué instrucción del conjunto se trata y, a continuación, **emulará su ejecución**. Debido a que la máquina elemental dada está diseñada siguiendo una arquitectura *Von Neumann*, junto con las instrucciones que forman el programa también se almacenarán los **datos**. Por otro lado, además de la memoria para el programa y los datos, el emulador también reservará una serie de posiciones de memoria en el 68K para representar todos los **registros** de la máquina elemental a emular, así como un **registro de estado** que contendrá los *flags*.

Antes de describir en detalle el trabajo concreto a realizar en esta práctica, vamos a introducir toda la información necesaria sobre la máquina elemental que debéis emular.

JARVIS: Just A Rather Very Intelligent System

La máquina que debéis emular en esta práctica se llama *JARVIS* (*Just A Rather Very Intelligent System*). Tanto los registros como su conjunto de instrucciones son de 16 bits. La JARVIS posee los siguientes registros:

- `B0` y `B1`, registros de direcciones, que se utilizan en algunas instrucciones para acceder a memoria usando un modo de direccionamiento indexado;
- `R2`, `R3`, `R4` y `R5`, que son de propósito general y se utilizan en operaciones de tipo ALU, ya sea como operando fuente o como operando destino;
- `T6` y `T7`, que se utilizan como interfaz con la memoria, además de poder ser empleados en operaciones de tipo ALU como operando.

No vamos a entrar en los detalles del *datapath* de la máquina y asumiremos que su funcionamiento está determinado por su conjunto de instrucciones, sin preocuparnos por las conexiones

Id	Mnemónico	Codificación	Acción	Flags
0	TRA Xa,Xb	00001bbbxaaaxxxx	Xb ← [Xa]	C = n.s.a., Z y N = s.v.Xb
1	ADD Xa,Xb	00010bbbxaaaxxxx	Xb ← [Xb] + [Xa]	C, Z y N = s.v.r.
2	SUB Xa,Xb	00011bbbxaaaxxxx	Xb ← [Xb] - [Xa]	C, Z y N = s.v.r.
3	NAN Xa,Xb	00100bbbxaaaxxxx	Xb ← [Xb] nand [Xa]	C = n.s.a., Z y N = s.v.r.
4	STC #k,Xb	00101bbbkkkkkkkk	Xb ← k (Ext. signo)	C = n.s.a., Z y N = s.v.Xb
5	INC #k,Xb	00110bbbkkkkkkkk	Xb ← [Xb] + k (Ext. Signo)	C, Z y N = s.v.r.
6	LOA M	0100mmmmmmmmxxxx	T6 ← [M]	C = n.s.a., Z y N = s.v.T6
7	LOAX M(Bi),Tj	0101mmmmmmmmijxx	Tj ← [M + [Bi]]	C = n.s.a., Z y N = s.v.Tj
8	STO M	0110mmmmmmmmxxxx	M ← [T6]	n.s.a.
9	STOX Tj,M(Bi)	0111mmmmmmmmijxx	M + [Bi] ← [Tj]	n.s.a.
10	BRI M	1000mmmmmmmmxxxx	PC ← M	n.s.a.
11	BRZ M	1001mmmmmmmmxxxx	Si Z = 1, PC ← M	n.s.a.
12	BRN M	1010mmmmmmmmxxxx	Si N = 1, PC ← M	n.s.a.
13	STP	11xxxxxxxxxxxxxx	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: { *000* - B0, *001* - B1, *010* - R2; *011* - R3, *100* - R4, *101* - R5; *110* - T6, *111* - T7
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 $\in \{-128, \ldots, +127\}$.
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.

Tabla 1: Conjunto de instrucciones de la JARVIS. (NOTA: recordad que debéis hacer la **extensión de signo** de las constantes **k** de 8 a 16 bits para poder operar con ellas.)

y el *hardware*. Únicamente debéis tener presente que en la JARVIS las operaciones de **resta se realizan de la siguiente forma:** $A - B = A + (\bar{B} + 1)$. En la Tabla 1 se muestra la información más relevante del conjunto de instrucciones de la JARVIS. Es importante prestar atención tanto a la codificación como a la funcionalidad de cada una de las instrucciones, así como a la actualización de los *flags* correspondientes. A la hora de codificar las instrucciones, **los bits *don't care* se sustituirán por ceros**.

A pesar de que cuando se escribe en ensamblador se utilizan siempre nombres simbólicos para denotar variables y direcciones de salto, los programas se deben traducir finalmente a lenguaje máquina. En el caso de la JARVIS, esto quiere decir que los programas que finalmente se deberán emular deberán contener direcciones numéricas absolutas tanto para especificar los saltos como para especificar los operandos de las instrucciones de interacción con la memoria. De esta forma, en la Figura 1, el programa de la izquierda no se podría ejecutar en el emulador, y se debería **transformar** en el programa de la derecha para que el emulador lo pudiera interpretar. Tal y como se hace en este ejemplo, para pasar de nombres simbólicos a direcciones absolutas

Etiqueta	Ensamblador con etiquetas	Dirección @JARVIS	Ensamblador sin etiquetas	Instrucciones codificadas	Hex
	STC #0,B0	0:	STC 0,B0	0010100000000000	2800
	STC #3,R2	1:	STC 3,R2	0010101000000011	2A03
LOOP:	LOAX A(B0),T6	2:	LOAX 14(B0),T6	0101000011100000	50E0
	TRA T6,R3	3:	TRA T6,R3	0000101101100000	0B60
	LOAX B(B0),T7	4:	LOAX 17(B0),T7	0101000100010100	5114
	TRA T7,R4	5:	TRA T7,R4	0000110001110000	0C70
	ADD R3,R4	6:	ADD R3,R4	0001010000110000	1430
	TRA R4,T6	7:	TRA R4,T6	0000111001000000	0E40
	STOX T6,C(B0)	8:	STOX T6,20(B0)	0111000101000000	7140
	INC #1,B0	9:	INC 1,B0	0011000000000001	3001
	INC #-1,R2	10:	INC -1,R2	0011001011111111	32FF
	BRZ END	11:	BRZ 13	1001000011010000	90D0
	BRI LOOP	12:	BRI 2	1000000000100000	8020
END:	STP	13:	STP	1100000000000000	C000
A:	2	14:	2	0000000000000010	0002
	3	15:	3	0000000000000011	0003
	1	16:	1	0000000000000001	0001
B:	3	17:	3	0000000000000011	0003
	2	18:	2	0000000000000010	0002
	4	19:	4	0000000000000100	0004
C:	0	20:	0	0000000000000000	0000
	0	21:	0	0000000000000000	0000
	0	22:	0	0000000000000000	0000

Figura 1: Ejemplo de programa para la JARVIS. Observa que el programa 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 `LOAX` y `STOX`). Tras la ejecución del programa, C = (0005Hex, 0005Hex, 0005Hex).

supondremos que el **programa se almacena a partir de la posición 0 de la memoria de la máquina emulada**, y que las direcciones de memoria en la JARVIS **se incrementan de uno en uno**. Nótese también que en este ejemplo los datos se almacenan después de la última instrucción del programa (`STP`).

NOTA: para facilitar la distinción entre todo lo relativo a la JARVIS y lo relativo al 68K, se añadirá a partir de ahora el prefijo "e" a todo lo que pertenezca a la primera. Así, el registro `Ri` de la máquina emulada lo denotaremos por `ERi`, los programas de la máquina emulada los denotaremos por eprogramas, etc.

Estructura del programa emulador

El programa que debéis diseñar y escribir comenzará con una cabecera como la siguiente:

```
        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        ;eregistro de instruccion
EPC:    DC.W 0        ;econtador de programa
EB0:    DC.W 0        ;eregistro B0
EB1:    DC.W 0        ;eregistro B1
ER2:    DC.W 0        ;eregistro R2
ER3:    DC.W 0        ;eregistro R3
ER4:    DC.W 0        ;eregistro R4
ER5:    DC.W 0        ;eregistro R5
ET6:    DC.W 0        ;eregistro T6
ET7:    DC.W 0        ;eregistro T7
ESR:    DC.W 0        ;eregistro de estado (00000000 00000ZCN)
```

El **eprograma** que se quiera emular en cada caso se introducirá como un vector de *words* (16 bits) del 68K **a partir de la etiqueta** `EMEM`. A modo de ejemplo, nótese que los *words* a partir de la etiqueta `EMEM` de la cabecera anterior **se corresponden con la codificación de las einstrucciones del eprograma que se muestra en la Figura 1**. Como bien se ha dicho anteriormente, vuestro programa debe ser capaz de emular la ejecución de **cualquier eprograma** que se introduzca codificado dentro del vector `EMEM`, de acuerdo al conjunto de instrucciones indicado en la Tabla 1. Nótese que las palabras de `EMEM` indicadas en rojo **son los datos del programa** y no se corresponden, por tanto, con ninguna instrucción codificada de la JARVIS.

Además del vector que contiene el eprograma y las eposiciones de memoria reservadas para datos, en la cabecera se reservan una serie de *words* para emular los registros de la JARVIS. Al final de la ejecución de cada una de las einstrucciones del eprograma, estos *words* deberán contener el valor correcto del eregistro. Así pues, estas posiciones se llamarán `EIR` (eregistro de instrucción), `EPC` (econtador de programa), `EB0` (eregistro `B0`), `EB1` (eregistro `B1`), `ER2` (eregistro `R2`), `ER3` (eregistro `R3`), `ER4` (eregistro `R4`), `ER5` (eregistro `R5`), `ET6` (eregistro `T6`), `ET7` (eregistro `T7`) y `ESR` (eregistro de estado). Este último tendrá en sus 3 bits menos significativos los *eflags* de la JARVIS con el orden indicado en la cabecera (`ZCN`).

El programa emulador que debéis escribir será un bucle que llevará a cabo los siguientes pasos para cada einstrucción del eprograma indicado en `EMEM`:

1. **Realizar el *fetch*** de la siguiente einstrucción a ejecutar.
   - Utilizar el valor contenido en el eregistro `EPC` para acceder a la siguiente einstrucción a ejecutar del vector `EMEM`.
   - Almacenar la einstrucción en el eregistro `EIR`.
   - Incrementar el eregistro `EPC` en uno para apuntar a la siguiente einstrucción a ejecutar.
2. **Decodificar** la einstrucción para determinar de cuál se trata.

- Llamar a una **subrutina de librería** que, a partir de la codificación del conjunto de instrucciones, analizará de qué einstrucción se trata y devolverá un valor númerico que la identifique de forma unívoca (columna `Id` de la Tabla 1).

3. **Emular la ejecución** de la einstrucción.

   - Con el valor devuelto por la subrutina de decodificación, saltar a una posición del programa donde se lleven a término las operaciones sobre los eregistros y/o eposiciones de memoria correspondientes a la fase de ejecución de la einstrucción (columna `Acción` de la Tabla 1).
   - Volver al inicio del programa para realizar el *fetch* de la siguiente instrucción, tras haber actualizado el valor de los eregistros y/o posiciones de memoria pertinentes, así como los *eflags* (eregistro `ESR`), si fuera necesario.

El emulador debe ejecutar el bucle hasta encontrar la **einstrucción** `STP`. Para hacer la emulación de forma correcta, al final de cada ciclo completo de una einstrucción (pasos 1, 2, 3), todos los eregistros, las eposiciones de memoria y los *eflags* **deben actualizarse con el valor correcto que obtendrían en la JARVIS**.

Subrutina de decodificación

La subrutina de decodificación **debe cumplir todos los requisitos de una subrutina de librería** indicados tanto en clase de teoría como en las sesiones prácticas. El paso de parámetros se debe realizar de la siguiente forma:

- En primer lugar, el programa principal debe reservar un *word* (16 bits) en la pila para que la subrutina pueda dejar el resultado de la decodificación (columna `Id` de la Tabla 1).
- A continuación, el programa principal insertará en la pila el contenido del eregistro `EIR` (16 bits), que servirá como parámetro de entrada a la subrutina.

Tras esto, se invocará a la subrutina. Ésta no debe utilizar ninguna otra información externa a parte del parámetro de entrada recibido. En caso de necesitar posiciones de memoria extra para cálculos, éstas **deben reservarse en la propia pila**.

Recordad que por el hecho de ser de librería, **la subrutina debe salvar, antes de su ejecución, los registros del 68K que utilice** con vistas a poder recuperar su valor antes de retornar el control al programa principal. Después, debe analizar la codificación de la einstrucción y retornar un valor **comprendido entre 0 y 13**, de acuerdo con la columna `Id` de la Tabla 1: 0 en el caso de un `TRA`, 1 en el caso de un `ADD`, y así sucesivamente.

Al terminar la decodificación, la subrutina debe recuperar el valor de los registros modificados, dejar la pila tal y como estaba al principio de la ejecución de la subrutina, poner el resultado de la decodificación en el lugar correspondiente en la pila y, finalmente, retornar el control al programa principal. Desde el programa principal se debe eliminar de la pila el `EIR`

introducido anteriormente como parámetro, y se debe **recuperar el resultado de la decodificación**. El *stack pointer* **debe quedar como estaba** antes de iniciar el proceso de llamada a la subrutina. **Se recomienda que todo lo que metáis en la pila sean *words* (16 bits) o *long words* (32 bits)**, y no *bytes*.

Para iniciar la fase de ejecución de la einstrucción decodificada se utilizará el valor numérico devuelto por la subrutina. Este valor **se debe introducir dentro de un registro del 68k**, por ejemplo `D1`, y se debe modificar para servir como **índice en la tabla de saltos** que se muestra a continuación[1]:

```
        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
```

En este listado, las etiquetas indican las direcciones donde se inicia la fase de ejecución de las correspondientes einstrucciones. Así pues, lo único que queda es programar **a partir de cada una de estas etiquetas** todo lo necesario para emular la ejecución de cada einstrucción (columna `Acción` de la Tabla 1). Observad que cada fase de ejecución debe terminar con un salto al principio del programa para continuar con el *fetch* de la siguiente einstrucción, excepto cuando se ejecuta la einstrucción `STP`. La fase de ejecución de esta einstrucción **debe detener la máquina**, lo cual es equivalente a finalizar el programa emulador. El código anterior **lo podéis utilizar directamente** en vuestra práctica para saltar a la fase de ejecución correspondiente de cada einstrucción.

Especificación del trabajo a realizar

- Debéis implementar la decodificación de las einstrucciones mediante una subrutina de librería **que haga el paso de parámetros de la forma indicada y que se llame** `DECOD`. Naturalmente, vuestra subrutina debe poder ser utilizada por cualquier usuario si

[1] El hecho de multiplicar por 6 se debe a que la codificación de cada instrucción `JMP` requiere 6 bytes.

sabe cómo se llama la subrutina, cómo pasarle los parámetros y cómo obtener el resultado. Además, **la subrutina debe figurar al final del fichero donde entreguéis el programa emulador**.

- Una vez implementada la subrutina, debéis implementar el programa emulador de acuerdo a las especificaciones de la JARVIS (Tabla 1). Debéis dedicar una atención especial a la obtención de los *eflags* correctos generados por la fase de ejecución de las einstrucciones.
- Debéis programar vuestra práctica en un fichero, ejecutable sobre el emulador del 68k, llamado **PRAFIN22.X68**, que se distribuye **junto con este enunciado**.
- El fichero (**PRAFIN22.X68**) incluye una **serie de comentarios para delimitar las diferentes secciones** que debe contener el emulador, junto con las pertinentes explicaciones para cada una de ellas. Estos comentarios **NO DEBEN eliminarse o modificarse**, y vuestro código para cada una de las secciones deberá ir justo después de los comentarios explicativos de la sección en cuestión.
- Las secciones incluidas en el fichero **PRAFIN22.X68** y que por tanto debe contener vuestro emulador son:
  - `FETCH`, en la que debéis introducir el código necesario para llevar a cabo el *fetch* de la siguiente einstrucción a ejecutar, tal y como se ha indicado anteriormente en este documento;
  - `BRDECOD`, dónde se debe preparar la pila para la llamada a la subrutina `DECOD`, realizar la llamada a dicha subrutina (`JSR`) y, tras esto, vaciar la pila de forma correcta, almacenando el resultado de la decodificación en un registro del 68k;
  - `BREXEC`, que incluye una sección de código destinada a saltar a la fase de ejecución de la einstrucción decodificada por `DECOD`. Esta sección la proporciona el propio enunciado y, si se almacena el resultado de la decodificación en el registro `D1`, **no es necesario modificarla**;
  - `EXEC`, en la que debéis programar las fases de ejecución de cada una de las einstrucciones de la máquina. Tras finalizar la fase de ejecución pertinente, **no debéis olvidar volver a la fase de *fetch*** para procesar la siguiente einstrucción del eprograma;
  - `SUBR`, dónde deben ir todas las subrutinas, de usuario o de librería, que implementéis en vuestro emulador, **a excepción de la subrutina de decodificación** `DECOD`; y, finalmente,
  - `DECOD`, en la que debéis implementar la subrutina de decodificacion, que deberá ser de librería, siguiendo la interfaz especificada en este enunciado.
- **Las primeras líneas del fichero deberán ser inexcusablemente[2] las siguientes (sustituyendo el nombre de los autores), respetando incluso el hecho de que las etiquetas están en mayúsculas:**

[2] Esto no implica que no tengáis que hacer pruebas con más casos.

```
*-----------------------------------------------------------
* Title      : PRAFIN22
* Written by : <nombres completos de los autores>
* Date       : 30/05/2022
* 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       ;eregistro de instruccion
EPC:  DC.W 0       ;econtador de programa
EB0:  DC.W 0       ;eregistro B0
EB1:  DC.W 0       ;eregistro B1
ER2:  DC.W 0       ;eregistro R2
ER3:  DC.W 0       ;eregistro R3
ER4:  DC.W 0       ;eregistro R4
ER5:  DC.W 0       ;eregistro R5
ET6:  DC.W 0       ;eregistro T6
ET7:  DC.W 0       ;eregistro T7
ESR:  DC.W 0       ;eregistro de estado (00000000 00000ZCN)

START:
      CLR.W EPC

FETCH:
```

- Evidentemente, el programa debe funcionar correctamente a pesar de que se modifiquen los valores contenidos dentro del vector `EMEM` (debe funcionar para cualquier otro eprograma) y sin que sea necesario modificar ningún otro dato introducido por vosotros. Cualquier referencia al eprograma deberá realizarse siempre mediante la etiqueta `EMEM`.
- El programa tampoco debe depender de las direcciones absolutas de los eregistros: siempre debéis hacer referencia a cada eregistro por su etiqueta. Esto quiere decir que no debéis acceder a un eregistro mediante la etiqueta de otra variable. En consecuencia, en la cabecera, **se debe poder modificar el orden en el que aparecen los eregistros**, o introducir directivas del tipo `DS.W 0`, y el emulador debe continuar funcionando correctamente.
- Se recomienda verificar que vuestro emulador **puede ejecutar correctamente, al menos, el programa de ejemplo** que se proporciona en este enunciado (Figura 1).
- Os debéis asegurar también de que la ejecución del **programa considerado como mínimo para poder evaluar vuestra práctica** es correcta (ver la siguiente sección).
- Es recomendable revisar los guiones de las prácticas **realizadas durante el curso** para conocer todos los detalles de implementación relacionados con la práctica final.

Entrega, presentación y evaluación de la práctica

1. Cada grupo de prácticas debe entregar, **a través de Aula Digital**, tanto el programa (**PRAFIN22.X68**) como un informe del trabajo realizado, el nombre del cual deberá ser **PRAFIN22.pdf**. Los dos ficheros (el informe en formato PDF y el programa ejecutable .X68) se deben enviar dentro de un fichero comprimido llamado **PRAFIN22.zip**.

2. El **informe deberá contener**:

   - Una **portada** con el nombre de la asignatura y el curso académico, y los nombres, DNIs, grupo de la asignatura y direcciones de correo electrónico de los integrantes del grupo.
   - Una breve **introducción** a la JARVIS y al problema propuesto.
   - Una explicación general al **trabajo** realizado para solventar la práctica, resumiendo cómo se ha implementado cada una de las fases del emulador (*fetch*, decodificación y ejecución).
   - Una descripción de la **rutina de decodificación** mediante un árbol para indicar la secuencia de bits analizados durante el proceso.
   - Una **tabla de subrutinas** utilizadas en la solución, indicando si son de librería o de usuario y sus interfaces de entrada y de salida.
   - Una **tabla de registros del 68k** siempre utilizados para el mismo propósito y su función, en caso de que existan.
   - Una **tabla de variables adicionales** definidas y su función, en caso de que existan.
   - Un **conjunto de pruebas** hechas sobre vuestra máquina elemental (máximo 5), especificando el eprograma y el resultado obtenido mediante vuestro emulador. Los eprogramas incluidos en este documento se pueden añadir a dicho conjunto de pruebas.
   - Una sección de **conclusiones** acerca del trabajo realizado, los conocimientos adquiridos y una valoración personal sobre la práctica.
   - El **código fuente** del emulador.

3. Las **indicaciones sobre el estilo de programación y documentación** incluidas en la práctica P1 se deberían respetar igualmente en el caso de esta práctica: clarificar el código fuente con las correspondientes indentaciones, incluir comentarios útiles (y no excesivos), utilizar nombres de etiquetas apropiados, evitar líneas de código y comentarios de más de 80 carácteres, etc.

4. **La fecha límite de entrega de la práctica será el lunes 30 de mayo de 2022 (hasta las 23:59h)**. Las prácticas entregadas **después** del día 30 de mayo **sufrirán una penalización en su nota global** de 1 punto por cada día de retraso. Por tanto, **el máximo retraso admisible es de 5 días naturales**.

5. Como paso previo a la corrección, se ejecutará sobre vuestro emulador el siguiente eprograma:

Dirección @JARVIS	Ensamblador sin etiquetas	Instrucciones codificadas	Hex
0:	LOA 7	0100000001110000	4070
1:	TRA T6,R2	0000101001100000	0A60
2:	BRI 5	1000000001010000	8050
3:	SUB R2,R2	0001101000100000	1A20
4:	STP	1100000000000000	C000
5:	ADD R2,R2	0001001000100000	1220
6:	STP	1100000000000000	C000
7:	1	0000000000000001	0001

que, usando la codificación del conjunto de instrucciones, da lugar a la siguiente cabecera:

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

Tras la ejecución del mismo, **la posición de memoria del 68K correspondiente a ER2 (en este caso, @1018Hex) debería contener el valor 2Dec = 0002Hex**. La correcta ejecución del programa anterior se considera un **requisito mínimo para que vuestra práctica sea evaluada**. En caso de que dicho programa no funcione como se indica, la práctica **no se corregirá** y obtendrá una calificación de **suspenso**. En ningún caso se considerarán como válidas aquellas soluciones que **escriban directamente en la memoria del 68K el resultado esperado**.

6. La práctica quedará automáticamente suspendida en el caso de que no se observen las restricciones anteriores.

7. De acuerdo con la guía docente, la excesiva similitud entre prácticas, o partes de prácticas (p.e. la subrutina de decodificación) a criterio del profesor, será considerada copia, y las dos prácticas quedarán automáticamente suspendidas, así como, al menos, la presente convocatoria de la asignatura.

NOTA: Cualquier modificación sobre la práctica se publicará en la página web de la asignatura en Aula Digital.