Práctica de Recuperación

Actividad evaluable: 30 % sobre la nota final de la asignatura
Fecha máxima de entrega: viernes 28 de junio de 2019 (hasta las 23:55h)

La nueva máquina: Balearic Basic Machine - Extended

El objetivo de esta práctica es, al igual que en la práctica final, escribir un programa emulador para una máquina elemental dada. En este caso, la máquina que debéis emular consiste en modificar (el mínimo necesario) el emulador que habéis escrito y verificado en la práctica final, con el objetivo de emular una nueva máquina, muy parecida a la BBM, llamada *BBMe (Balearic Basic Machine - Extended)*. Al igual que la BBM, la BBMe posee los siguientes registros:

- `R0`, `R1`, `R2` y `R3`, que son de propósito general y se utilizan en operaciones de tipo ALU, ya sea como operando fuente o como operando destino;
- `B4` y `B5`, que se utilizan como interfaz con la memoria, ya sea para acceder directamente a ella o mediante un modo de direccionamiento indirecto.

Sin embargo, presenta las siguientes **diferencias** con respecto a la BBM original:

- La BBMe sustituye las instrucciones `NOT` y `OR` por la instrucción `NOR`;
- Las instrucciones `LD` y `ST` de la BBMe permiten utilizar cualquiera de los registros `B4` y `B5` para interactuar con la memoria;
- Las instrucciones `ADD`, `SUB` y `NOR` de la BBMe son **ahora de tres direcciones**.

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 y el *hardware*. Únicamente debéis tener presente que en la BBMe 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 BBMe. Fijaos que la **codificación cambia con respecto a la BBM**. Es importante prestar atención tanto a la codificación como a la funcionalidad de cada una de las instrucciones, **especialmente en las nuevas instrucciones**. Los principales cambios **se han marcado en rojo**. 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 BBMe, 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

Id	Mnemónico	Codificación	Acción	Flags
0	TRA Xa,Xb	00000xxxxxaaabbb	Xb ← [Xa]	C = n.s.a., Z y N = s.v.Xb
1	LD M,Bj	00001xxjmmmmmmmm	Bj ← [M]	C = n.s.a., Z y N = s.v.Bj
2	LDI	00010xxxxxxxxxxx	B4 ← [[B5]]	C = n.s.a., Z y N = s.v.B4
3	ST Bj,M	00011xxjmmmmmmmm	M ← [Bj]	n.s.a.
4	STI	00100xxxxxxxxxxx	[B5] ← [B4]	n.s.a.
5	JMN M	0100xxxxmmmmmmmm	Si N = 1, PC ← M	n.s.a.
6	JMZ M	0101xxxxmmmmmmmm	Si Z = 1, PC ← M	n.s.a.
7	JMI M	0110xxxxmmmmmmmm	PC ← M	n.s.a.
8	HLT	10xxxxxxxxxxxxxx	Detiene la máquina	n.s.a.
9	NOR Rc,Ra,Rb	11000xxcccaaabbb	Rc ← [Rb] nor [Ra]	C = n.s.a., Z y N = s.v.Rc
10	SET #k,Xb	11001kkkkkkkkbbb	Xb ← k	C = n.s.a., Z y N = s.v.Xb
11	ADQ #k,Xb	11010kkkkkkkkbbb	Xb ← [Xb] + k	C, Z y N = s.v.r.
12	ADD Rc,Ra,Rb	11011xxcccaaabbb	Rc ← [Rb] + [Ra]	C, Z y N = s.v.r.
13	SUB Rc,Ra,Rb	11100xxcccaaabbb	Rc ← [Rb] - [Ra]	C, Z y N = s.v.r.
14	TST Ra,Rb	11101xxxxxaaabbb	[Rb] - [Ra]	C, Z y N = s.v.r.

LEYENDA

x: Bit no utilizado (*don't care*).
mmmmmmmm: Dirección de memoria (emulada) de 8 bits.
Xa, Xb: Cualquier registro R o B, ver aaa y bbb.
Bj: B4 o B5, ver j.
aaa, bbb, ccc: Índice del registro según: { *000* - R0, *001* - R1, *010* - R2; *011* - R3, *100* - B4, *101* - B5
j: Índice del registro B4 (j=0) o del registro B5 (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.Rc: Según el valor del registro Rc después de realizar la operación.
s.v.Bj: Según el valor del registro Bj después de realizar la operación.
s.v.B4: Según el valor del registro B4 después de realizar la operación.
s.v.Xb: Según el valor del registro Xb después de realizar la operación.

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

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 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 BBMe **se incrementan de uno en uno**. Nótese que en este ejemplo los datos se almacenan después de la última instrucción del programa (`HLT`).

NOTA: para facilitar la distinción entre todo lo relativo a la BBMe 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.

Etiqueta	Ensamblador con etiquetas	@BBMe	Ensamblador sin etiquetas	Instrucciones codificadas	Hex
	LD A,B4	0:	LD 14,B4	00001 00 0 00001110	080E
	TRA B4,R0	1:	TRA B4,R0	00000 00000 100000	0020
	JMZ EXIT	2:	JMZ 11	0101 0000 00001011	500B
	LD B,B5	3:	LD 15,B5	00001 00 1 00001111	090F
	TRA B5,R1	4:	TRA B5,R1	00000 00000 101001	0029
	JMZ EXIT	5:	JMZ 11	0101 0000 00001011	500B
	SET #0,R3	6:	SET 0,R3	11001 00000000 011	C803
LOOP:	ADD R3,R0,R3	7:	ADD R3,R0,R3	11011 00 011000011	D8C3
	ADQ #-1,R1	8:	ADQ -1,R1	11010 11111111 001	D7F9
	JMZ EXIT	9:	JMZ 11	0101 0000 00001011	500B
	JMI LOOP	10:	JMI 7	0110 0000 00000111	6007
EXIT	TRA R3,B5	11:	TRA R3,B5	00000 00000 011101	001D
	ST B5,C	12:	ST B5,16	00011 00 1 00010000	1910
	HLT	13:	HLT	10 00000000000000	8000
A:	4	14:	4	00000000 00000100	0004
B:	3	15:	3	00000000 00000011	0003
C:	0	16:	0	00000000 00000000	0000

Figura 1: Ejemplo de programa para la BBMe. Observa que el programa implementa la operación $C = A \times B$, para $A \geq 0$ y $B \geq 0$. Por tanto, después de la ejecución, $C = 12\text{Dec} = 0\text{CHex}$.

Estructura del programa emulador

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

```
        ORG $1000
EPROG:  DC.W $080E,$0020,$500B,$090F,$0029,$500B,$C803
        DC.W $D8C3,$D7F9,$500B,$6007,$001D,$1910,$8000
        DC.W $0004,$0003,$0000
EIR:    DC.W 0       ;eregistro de instruccion
EPC:    DC.W 0       ;econtador de programa
ER0:    DC.W 0       ;eregistro R0
ER1:    DC.W 0       ;eregistro R1
ER2:    DC.W 0       ;eregistro R2
ER3:    DC.W 0       ;eregistro R3
EB4:    DC.W 0       ;eregistro B4
EB5:    DC.W 0       ;eregistro B5
ESR:    DC.W 0       ;eregistro de estado (00000000 00000ZNC)
```

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** `EPROG`. A modo de ejemplo, nótese que los *words* a partir de la etiqueta `EPROG` 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 `EPROG`, de acuerdo al conjunto de instrucciones indicado en la Tabla 1.

Además del vector que contiene el eprograma y las eposiciones de memoria reservadas para datos y resultados, en la cabecera se reservan una serie de *words* para emular los registros de la BBMe. 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), `ER0` (eregistro `R0`), `ER1` (eregistro `R1`), `ER2` (eregistro `R2`), `ER3` (eregistro `R3`), `EB4` (eregistro `B4`), `EB5` (eregistro `B5`) y `ESR` (eregistro de estado). Este último tendrá en sus 3 bits menos significativos los *eflags* de la BBMe con el orden indicado en la cabecera (`ZNC`).

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

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 `EPROG`.
   - 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** `HLT`. 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 BBMe**.

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 14**, de acuerdo con la columna `Id` de la Tabla 1: 0 en el caso de un `TRA`, 1 en el caso de un `LD`, 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:
        ETRA
        ELD
        ELDI
        EST
        ESTI
```

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

```
EJMN
EJMZ
EJMI
EHLT
ENOR
ESET
EADQ
EADD
ESUB
ETST
```

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 `HLT`. 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 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 BBMe (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 **PRAREC19.X68**, que se distribuye **junto con este enunciado**.
- El fichero (**PRAREC19.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 **PRAREC19.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:**

```
*-------------------------------------------------------------
* Title      : PRAREC19
* Written by : <nombres completos de los autores>
* Date       : 28/06/2019
* Description: Emulador de la BBMe
*-------------------------------------------------------------
```

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

```
       ORG $1000
EPROG: DC.W $080E,$0020,$500B,$090F,$0029,$500B,$C803
       DC.W $D8C3,$D7F9,$500B,$6007,$001D,$1910,$8000
       DC.W $0004,$0003,$0000
EIR:   DC.W 0       ;eregistro de instruccion
EPC:   DC.W 0       ;econtador de programa
ER0:   DC.W 0       ;eregistro R0
ER1:   DC.W 0       ;eregistro R1
ER2:   DC.W 0       ;eregistro R2
ER3:   DC.W 0       ;eregistro R3
EB4:   DC.W 0       ;eregistro B4
EB5:   DC.W 0       ;eregistro B5
ESR:   DC.W 0       ;eregistro de estado (00000000 00000ZNC)

START:
       CLR.W EPC

FETCH:
```

- Evidentemente, el programa debe funcionar correctamente a pesar de que se modifiquen los valores contenidos dentro del vector `EPROG` (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 `EPROG`.
- 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 punto 6 de 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 tanto el programa como un informe del trabajo realizado, el nombre del cual deberá ser **PRAREC19.pdf**; los dos ficheros se deben enviar dentro de un fichero comprimido llamado **PRAREC19.zip**. **Una copia impresa del informe se debe dejar en el casillero del profesor correspondiente al grupo** (Emilio García, Conserjería Anselm Turmeda), mientras que **el fichero comprimido**

(.zip), que contiene el informe (.pdf) y el programa ejecutable (.X68), **se debe entregar a través de Aula Digital**.

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 BBMe 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, por ejemplo, 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, suficientemente comentado.
3. La copia impresa se debe presentar **grapada**, sin páginas en blanco; no se debe encuadernar ni meter dentro de fundas, carpetas ni ningún tipo de sobre.
4. Las **indicaciones sobre el estilo de programación y documentación** incluidas en la P3 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.
5. **La fecha límite de entrega de la práctica será el viernes 28 de junio de 2019 (hasta las 23:55h)**. Las prácticas entregadas **después** del día 28 de junio **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**. La **copia física** del informe se puede dejar en el casillero al **día siguiente** de la entrega por Aula Digital si ese día es fin de semana o festivo.
6. Como paso previo a la corrección, se ejecutará sobre vuestro emulador el siguiente eprograma:

@BBMe	Ensamblador sin etiquetas	Instrucciones codificadas	Hex
0:	LD 7,B5	00001 00 1 00000111	0907
1:	TRA B5,R2	00000 00000 101010	002A
2:	JMI 5	0110 0000 00000101	6005
3:	SUB R2,R2,R2	11100 00 010010010	E092
4:	HLT	10 00000000000000	8000
5:	ADD R2,R2,R2	11011 00 010010010	D892
6:	HLT	10 00000000000000	8000
7:	1	00000000 00000001	0001

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

```
        ORG $1000
EPROG:  DC.W $0907,$002A,$6005,$E092,$8000,$D892,$8000,$0001
EIR:    DC.W 0       ;eregistro de instruccion
EPC:    DC.W 0       ;econtador de programa
ER0:    DC.W 0       ;eregistro R0
ER1:    DC.W 0       ;eregistro R1
ER2:    DC.W 0       ;eregistro R2
ER3:    DC.W 0       ;eregistro R3
EB4:    DC.W 0       ;eregistro B4
EB5:    DC.W 0       ;eregistro B5
ESR:    DC.W 0       ;eregistro de estado (00000000 00000ZNC)
```

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**.

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

8. 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.