ESTRUCTURA DE COMPUTADORES I
INGENIERÍA INFORMÁTICA
Curso: 2022-2023
GRUPO 1

Universitat de les Illes Balears

Khaoula Ikkene
khaoula.ikkene1@estudiant.uib.cat

Marc Garcia Bonet
marc.garcia4@estudiant.uib.cat

Carolina Marín Sánchez
cms213@id.uib.eu

ÍNDICE:

I. Introducción al problema

En esta práctica se nos pide emular la máquina HAL9000(Heuristically Programmed Algorithmic Computer). La emulación consta de 3 fases diferentes que se desglosarán más adelante en el presente documento.

Dicha máquina trabaja con direcciones de 16 bits, dos registros (T0,T1) de interfaz con la memoria, aunque se pueden usar como operando fuente en operaciones de tipos ALU, y 6 registros de propósito general (X2 hasta X7) empleados principalmente en las operaciones de tipo ALU.

II. Fases del emulador

1. Fetch:

En esta primera fase del ciclo de instrucción el programa almacena las einstrucciones en el *EIR* en el orden que aparecen en el vector *EMEM*. Para recorrer *EMEM* utilizaremos el *EPC* que se incrementa, justo después de leer una einstrucción, en una unidad para apuntar a la siguiente einstrucción.

En el *68k* el *PC* se incrementa en dos unidades para apuntar a la siguiente instrucción a la hora de trabajar con direcciones de 16 bits (Word), mientras en la *HAL9000* el *EPC* se incrementa de uno en uno. Para compensar esta diferencia hemos multiplicado el valor del *EPC* por dos unidades.

2. Decodificación:

En esta fase, guiándonos por el árbol de decodificación, y usando una subrutina de librería (DECOD) hemos ido identificando cada instrucción usando una serie de BTST a los 4 o 5 bits más significativos de cada instrucción. Una vez que se verifica que se trata de una determinada instrucción, guardamos en la pila su identificador para luego poder moverlo en el registro de datos D1 y usar su valor para saltar a la etiqueta correspondiente.

3. Ejecución:

En esta parte ejecutamos las instrucciones del EMEM. A contiuación se explicará el funcionamiento de cada instrucción.

LOA M, Ti

Esta instrucción debe guardar el valor contenido en la dirección M de la HAL (definida por los bits 8-1 de la codificación de la instrucción) en el registro ETi. El valor de 'i' viene designado por el último bit de la codificación de LOA, si este es 1 el valor a almacenar debe ser insertado en el registro ET1, en cambio, si es 0 el valor debe ser introducido en ET0 . Tras ejecutarla, el flag C no se va a actualizar, es decir, dicho flag debe conservar su valor anterior, y los flags Z y N se actualizarán en función del contenido del registro Ti.

STO Ti,M

Esta instrucción tiene un funcionamiento casi idéntico a la anterior, en la que movíamos el valor contenido en la dirección M a uno de los registros de interfaz. En este caso se realiza la operación inversa, es decir, se mueve el valor de uno de estos dos registros (nuevamente definido por el último bit de la codificación de la instrucción) a la dirección M de la máquina HAL. Una diferencia significativa respecto a la instrucción previa es que en STO no se actualiza ninguno de los tres flags de la HAL, mientras que en LOA se actualizan los flags Z y N.

LOIP (Xb),T

Esta instrucción guarda el valor del contenido del registro Xb (definido por el valor de b, codificado en los bits 6-4 de la codificación de la propia instrucción) en el registro ETi. Similar a las instrucciones anteriores, el valor de 'i' viene designado por el último bit de la codificación de la instrucción. Por consiguiente, si 'i' corresponde a 1, el valor se guarda en ET1 y si es 0 se guarda en ET0. Después de haber realizado lo anterior, LOIP incrementa en 1 el valor del registro Xb. Después de su ejecución, el flag C no se actualiza, mientras que los flags Z y N se actualizan en función del contenido del registro Ti después de realizar la operación.

STIP Ti,(Xb)

Esta instrucción se encarga de guardar el contenido del registro Ti, valor del cual depende de 'i' (explicada su designación anteriormente), en el contenido del registro Xb (definido por el valor de b). A continuación, al igual que LOIP, incrementa en 1 el valor del registro Xb. Sin embargo, en STIP no se actualiza ningún flag.

GOI M

Esta instrucción guarda la dirección de la posición de memoria indicada por la etiqueta M en el EPC. En otras palabras, guarda el valor de m (designado por los bits 8-1 de la codificación de la instrucción), el cual contiene dicha dirección en el EPC. Tras su ejecución, no se actualizan los flags.

GOZ M

Esta instrucción tiene el mismo funcionamiento que la anterior. Sin embargo, para que se ejecute, el valor del flag Z de ESR, es decir, su bit menos significativo ha de ser 1, es decir, Z = 1. En caso contrario (Z = 0), no se ejecuta. Tampoco se actualizan los flags en ningún caso.

GON M

Esta instrucción revisa el valor actual del flag N. Si N = 1, entonces ejecutará la misma instrucción que GOI, sin actualizar los flags bajo ningún concepto. De esta manera, podemos afirmar que GON y GOZ funcionan igual que GOI pero GON solo se ejecuta si N = 1 y GOZ si Z = 1.

EXIT

Esta instrucción detiene la máquina sin actualizar ningún flag.

COPY Rb,Rc

Esta instrucción se encarga de guardar el contenido del registro Rb (definido por el valor de b, que está codificado en los bits 6-4 de la instrucción) en el registro Rc (definido por el valor de c, que está codificado en los bits 2-0 de la instrucción). Después de su ejecución, el flag C no se actualiza, mientras que los flags Z y N se actualizan en función del contenido del registro Rc después de realizar la operación.

ADD Ra,Rb,Rc

Esta instrucción realiza la suma de los contenidos de los registros Rb y Ra, y guarda su valor en el registro Rc. El registro Rb viene definido por el valor de b (codificado en los bits 6-4 de la instrucción); el registro Ra por el valor de a (codificado en los bits 10-8 de la instrucción); y el valor de Rc por el valor de c (codificado en los bits 2-0 de la instrucción). Tras su ejecución, los flags C, Z y N se actualizan según el valor del resultado de la operación.

SUB Ra,Rb,Rc

Esta instrucción funciona de la misma forma que ADD, incluyendo los mismos registros, pero en lugar de realizar una suma hace una resta. Dicha resta se ejecuta de la siguiente forma:
A-B = A+(B`+1).
Por lo tanto, resta el contenido del registro Rb menos el contenido del registro Ra y lo guarda en el registro Rc. Los flags funcionan igual que en ADD.

SET #k,Rc

Esta instrucción extrae el valor de k de la instrucción (codificado en los bits 10-3 de la instrucción) y realiza una extensión de signo, el valor de k está en complemento a 2, para llegar a los 16 bits. Si k ≥0 añade 8 zeros en el Byte más significativo del registro que contiene a K, en caso contrario añade 8 unos a dicho Byte.
A continuación, guarda este valor extendido en el registro Rc. En cuanto a los flags, el flag C no se actualiza y los flags Z y N se actualizan en función del valor del registro Rc después de realizar la operación.

ADQ #k,Rc

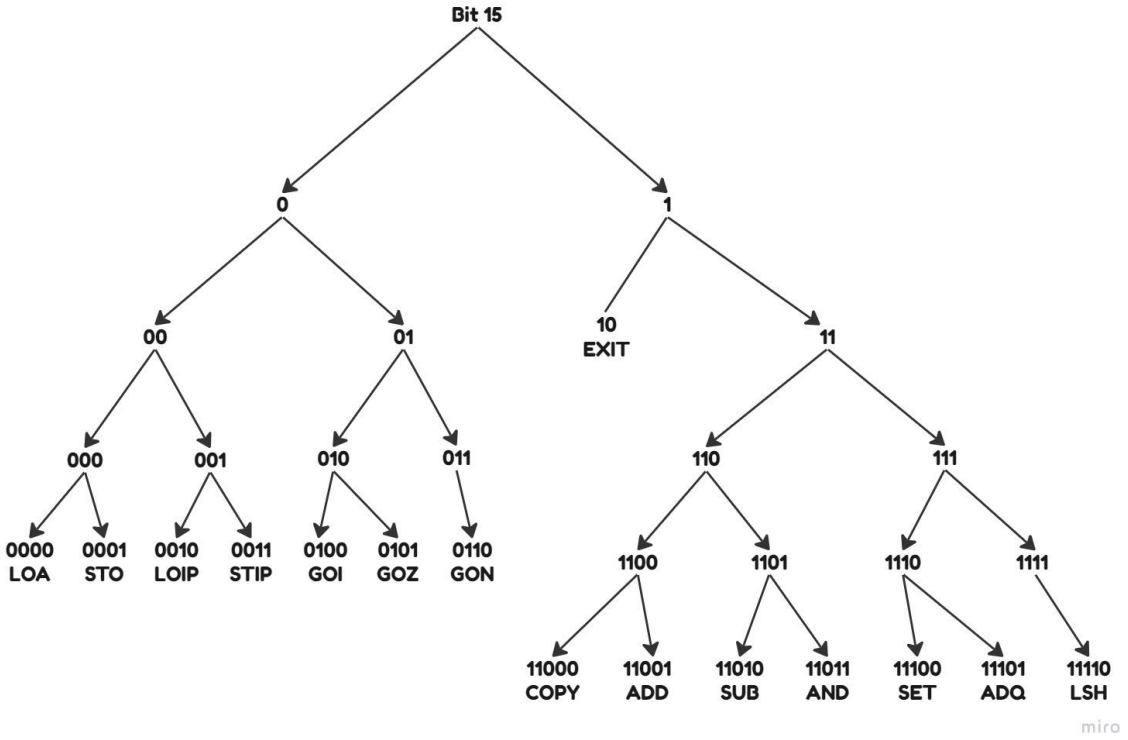
Esta instrucción también extrae el valor de k de la instrucción (codificado en los bits 10-3 de la instrucción) y realiza una extensión de signo hasta llegar a los 16 bits. A continuación, le suma el contenido del registro Rc para después guardar el resultado de la operación también en el registro Rc. Tras su ejecución, se actualizan los flags C, Z y N según el valor del resultado de la suma.

LSH #p,Rb,#n

Esta instrucción comprueba el valor de **n** (codificado en el bit menos significativo de la instrucción). Si este resulta ser 0, el contenido del registro Rb se desplaza **p** bits hacia la izquierda y guarda el resultado en el mismo registro. Mientras que si **n** = 1, el contenido del registro Rb se desplaza **p** bits hacia la derecha. Cabe destacar que el número de bits de desplazamiento, p, vienen determinados por los bits 10-8 de la instrucción. Después de su ejecución, se actualizan los flags C, Z y N según el valor del resultado de la suma.

III. Rutina de decodificación

A continuación se muestra el árbol de decodificación de las instrucciones. Para cada instrucción había que examinar como máximo sus 5 bits más significativos. El árbol se ramifica a partir del bit 15 dividiéndolo en dos partes: la primera representa las instrucciones en las cuales el bit más significativo es un 0 (bit 15 = 0) y la otra las que empiezan con 1. Para los bits (14,13, 12 y el 11) se repite el mismo proceso, siempre descartando los casos que no representan ninguna codificación de alguna instrucción.



IV. Tabla de subrutinas

Todas estas subrutinas son de usuario.

Subrutina	Parametro de entrada	Parametro de salida	Función
ValueofI	EIR	A1	Guarda la instrucción en D2. Hace un And a su bit menos significativo, donde se encuentra la i, y gaurda su valor en D2. A continuación salta a la subrutina ***ValueofR*** para tener la dirección del registro correspondiente (T0 o T1) y la guarda en el registro de direcciones A1.
ValueofM	EIR	A6	Extrae el valor de m de la instrucción guardada en el registro D2 y copia la dirección que le corresponde en el 68k en el registro de direcciones A6.

ValueofA	EIR	D2	Devuelve la dirección del registro Ra tras haber hecho un And entre el registro de datos D2 y $0F00 para obtener el valor en binario de A. A continuación, realizamos un LSR de un 1 bit para tener el valor de aaa en D2. Una vze tenemos el valor de “aaa” saltamos a la subrutina ValueofR para obtener la dirección del registro correspondente que se guarda en el registro A6
ValueofB	EIR	D2	Devuelve la dirección del registro Rb. Se repite el mismo proceso que en la anterior subrutina cambiando ahora el valor de $0F00 a $00F0 y realizando un LSR de 4 bits.
ValueofC	EIR	D2	Devuelve la dirección del registro Rc. Se repite el mismo proceso que en la subrutina ***ValueofA*** cambiando ahora el valor de $0F00 a $000F y sin realizar ningún LSR, ya que el valor binario de ccc”se encuentra en los 3 bits menos significativos de la instrucción guardada en el registro D2.
FlagZ	(A6)	ESR	Actualiza el flag Z según la instrucción ejecutada y guarda su valor en el ESR. Primero, se guarda el contenido de la dirección almacenada en el registro A6 en D4, y según el flag Z del 68k guardamos un 1 o 0 en el registro ESR en la posición que corresponde a dicho flag, en este caso en el bit 0 del ESR.
FlagN	(A6)	ESR	Actualiza el flag N según la instrucción ejecutada y guarda su valor en el ESR. Primero se guarda el contenido del registro en D4, se hace un CMP con el valor 0. Si el resultado es negativo saltamos a la etiqueta NEG (negativo) para guardar 1 en el flag N que se encuentra en el bit uno (contando de derecha a izquierda) del ESR, en caso contrario guardamos un 0 en el flag N.
FlagC	Implícito (SR)	ESR	Usando el BCC evaluamos directamente el contenido del registro de datos D6. Si el carry es cero guardamos un 0 en el flag C que se encuentra en el segundo bit del registro ESR, en caso contrario guardamos un 1 en el bit número 2 del ESR.
ValueofR	*D2*	*A6*	Devuelve la dirección del registro (T0, T1, X2 hasta X7).
ValueofK	EIR	D2	Devuelve el valor de K con la extensión de signo aplicada a dicho valor en el D2 después de haber

			aplicado una máscara con AND $07F8 a la instrucción guardada en el registro D2, un LSR de bits para sacar el valor de M y usado la instrucción EXT propia del 68k .
ValueofM Absolut	EIR	D2	Devuelve el valor de la dirección M en el registro de datos D2 después de haber hecho una máscara con una AND ($01FE) con la instrucción y un desplazamiento a la derecha de un bit para obtener el valor de M.

V. Tabla de registros del 68k utilizados

Nombre del registro	Uso
A0	Contener el valor de EPC
A1	Contener la dirección de uno de los registros T0 o T1
D0	Contener el valor de EIR para poder operar con ello
D1	Contener el Id de una instrucción dada
D2	Contener el valor de EIR para poder operar con ello, el valor de M, k con la extensión de signo, el valor de aaa/bbb/ccc.
D3	Contener el valor de n de la instrucción LSH,y posteriormente el contenido del registro Rb desplazado de esta misma instrucción.
D4	Contener el valor de los flags Z, N, y C
D5	Usado temporalmente para contener el contenido de uno de los registros (Rb) y los resultados de algunas operaciones de resta AND, SET, ADQ
D7	Contener el contenido del registro Ra en algunas operaciones de la ALU.
A6	La dirección del registro (T0, T1, X2 hasta X7), dirección del contenido de M

VI. Variables adicionales

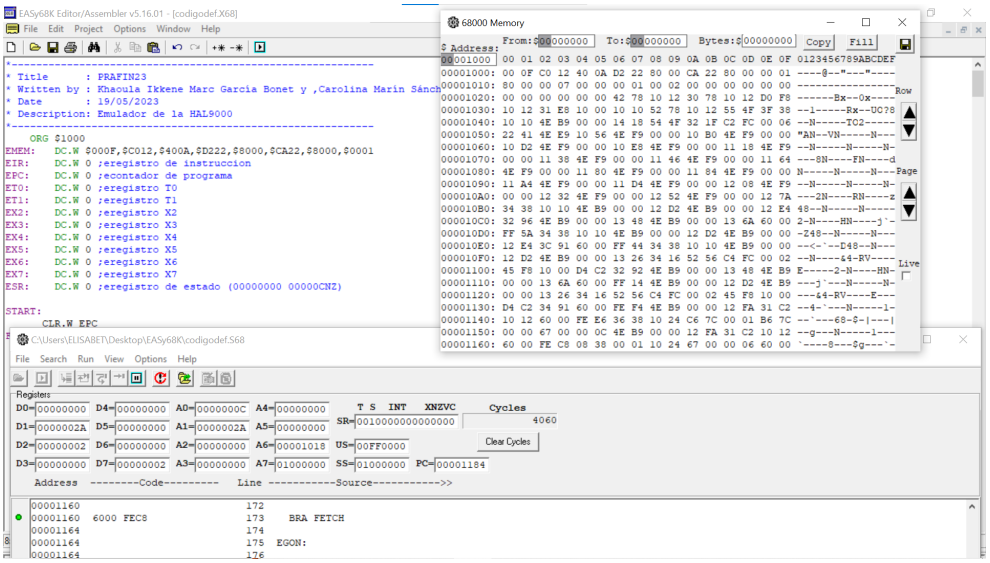
No hemos usado ninguna variable adicional.

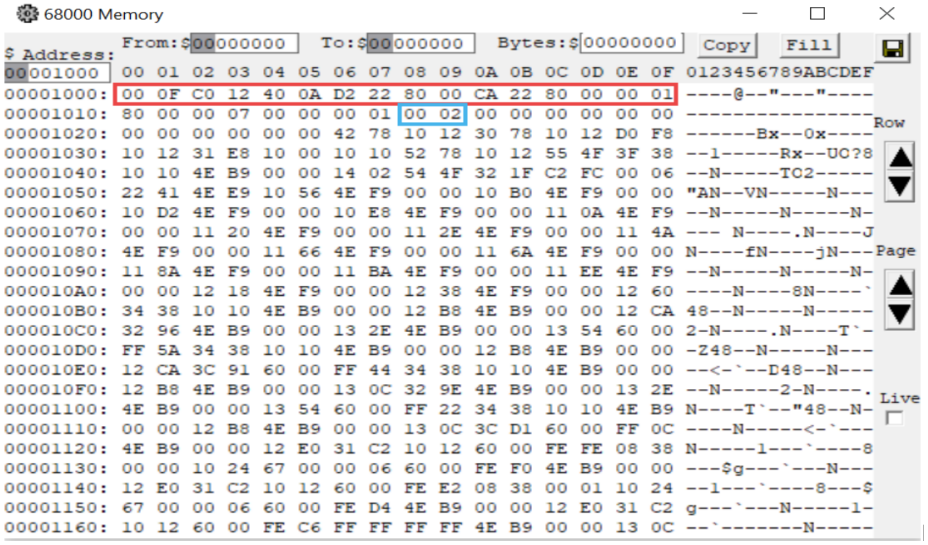
VII. Conjunto de pruebas

En este apartado hemos probado el programa principal de la práctica, el programa mínimo, y 3 pruebas adicionales.

1a Prueba: Corresponde a la prueba mínima dada en la práctica.

Dirección @HAL9000	Ensamblador sin etiquetas	Instrucciones codificadas	Hex
0:	LOA 7,T1	0000000000001111	000F
1:	COPY T1,X2	1100000000010010	C012
2:	GOI 5	0100000000001010	400A
3:	SUB X2,X2,X2	1101001000100010	D222
4:	EXIT	1000000000000000	8000
5:	ADD X2,X2,X2	1100101000100010	CA22
6:	EXIT	1000000000000000	8000
7:	1	0000000000000001	0001



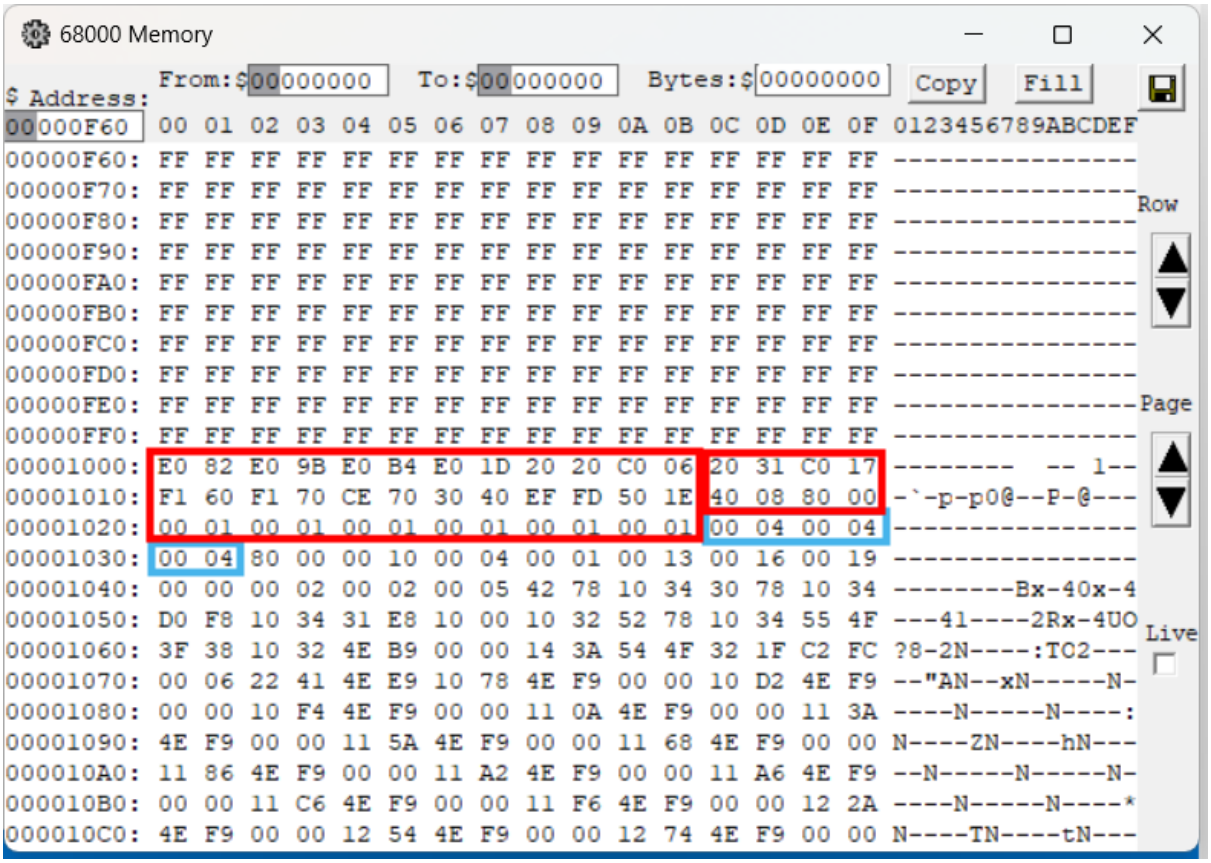


En la primera imagen se puede observar la cabecera del programa, el contenido de la memoria hasta la posición 116F (hex y el valor de todos los registros tras la ejecución del programa.

En la segunda imagen se puede apreciar con más claridad el contenido de la memoria, marcado en rojo se encuentra el contenido del vector EMEM, mientras que la posición de memoria 1018(hex, marcado con color azul, contiene, como dice el enunciado, el valor 0002(hex.

2a Prueba: Ejemplo base de la práctica (enunciado)

ET:	Ensamblador con etiquetas	Dirección @HAL9000	Ensamblador sin etiquetas	Instrucciones codificadas	Hex
	SET #16,X2	0	SET 16,X2	1110000010000010	E082
	SET #19,X3	1	SET 19,X3	1110000010011011	E09B
	SET #22,X4	2	SET 22,X4	1110000010110100	E0B4
	SET #3,X5	3	SET 3,X5	1110000010011101	E01D
LOOP:	LOIP (X2),T0	4	LOIP (X2),T0	0010000000100000	2020
	COPY T0,X6	5	COPY T0,X6	1100000000000110	C006
	LOIP (X3),T1	6	LOIP (X3),T1	0010000000110001	2031
	COPY T1,X7	7	COPY T1,X7	1100000000010111	C017
	LSH #1,X6,#0	8	LSH 1,X6,0	1111000101100000	F160
	LSH #1,X7,#0	9	LSH 1,X7,0	1111000101110000	F170
	ADD X6,X7,T0	10	ADD X6,X7,T0	1100111001110000	CE70
	STIP T0,(X4)	11	STIP T0,(X4)	0011000001000000	3040
	ADQ #-1,X5	12	ADQ -1,X5	1110111111111101	EFFD
	GOZ END	13	GOZ 15	0101000000011110	501E
	GOI LOOP	14	GOI 4	0100000000001000	4008
END:	EXIT	15	EXIT	1000000000000000	8000
A:	1	16	1	0000000000000001	0001
	1	17	1	0000000000000001	0001
	1	18	1	0000000000000001	0001
B:	1	19	1	0000000000000001	0001
	1	20	1	0000000000000001	0001
	1	21	1	0000000000000001	0001
C	0	22	0	0000000000000000	0000
	0	23	0	0000000000000000	0000
	0	24	0	0000000000000000	0000



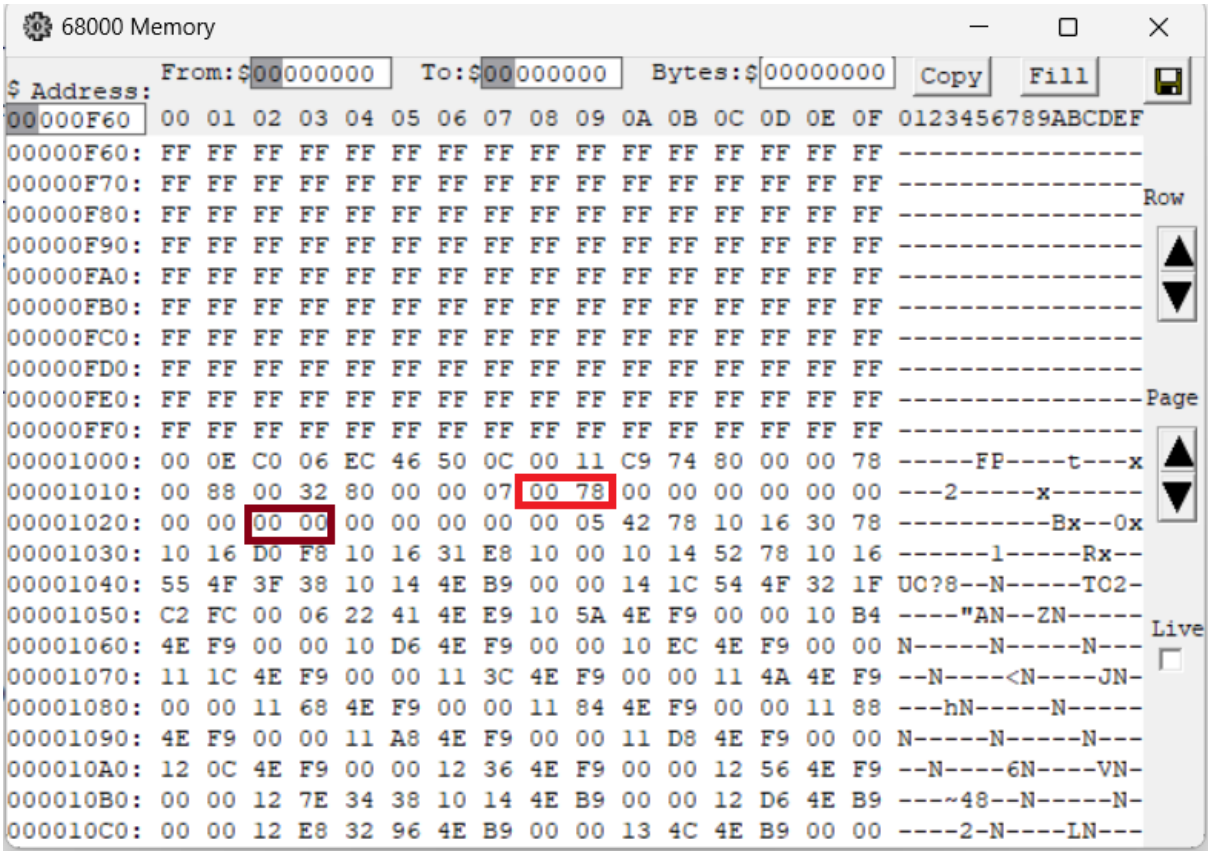
Tras la ejecución se observa que los 3 words correspondientes al vector C (marcados en azul) contienen el valor hexadecimal 04. Mientras que marcado en rojo se ve el vector EMEM.

3a prueba:

Tras la ejecución del programa en la memoria debe quedar 120 en decimal que es equivalente a 78 en Hexadecimal en la posición que corresponde al registro T0 (@1018) y 0 en la posición que ocupa el X6 (@1022). Además, el contenido del registro X4 no debe modificarse, es decir, debe quedar 0, su contenido inicial, ya que la condición de salto se cumple (instrucción 3) y se debe detener la máquina sin ejecutar las instrucciones 4 y 5.

EMEM: $000E,$C006,$EC46,$500C,$0011,$C974,$8000,$0078,$0088, $0032

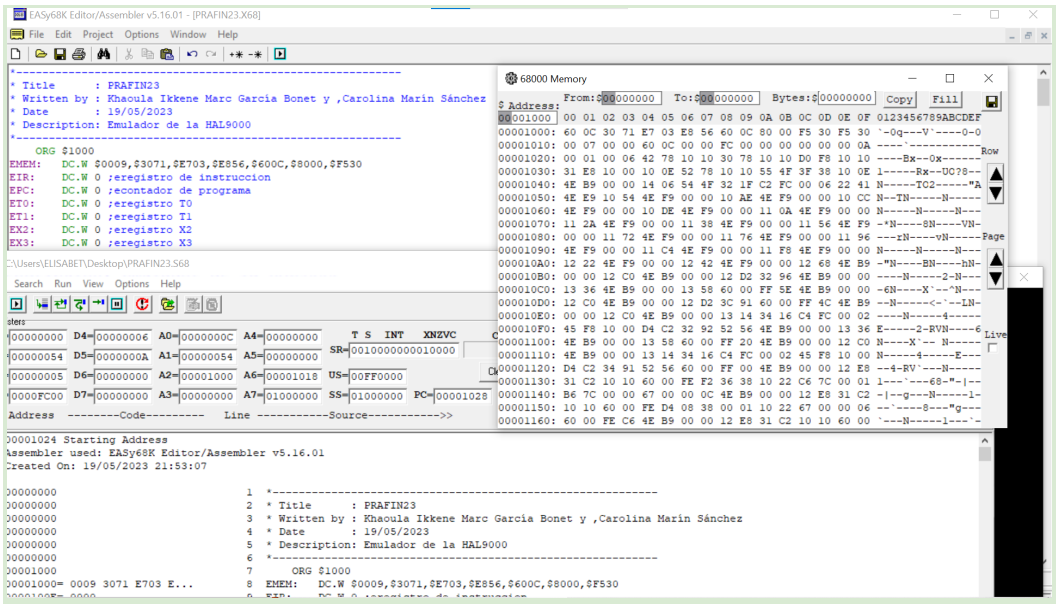
Dirección @HAL9000	***Ensamblador***	***Instrucciones codificadas***	***HEX***
0:	LOA 7,T0	0000000000001110	000E
1:	COPY T0, X6	1100000000000110	C006
2:	ADQ -120, X6	1110110001000110	EC46
3:	GOZ 6	0101000000001100	500C
4:	LOA 9, T1	0000000000010001	0011
5:	ADD T1, X7, X4	1100100101110100	C974
6:	EXIT	1000000000000000	8000
7:	78 Hex	0000000001111000	0078
8:	88 Hex	0000000010001000	0088
9:	32 Hex	0000000000110010	0032

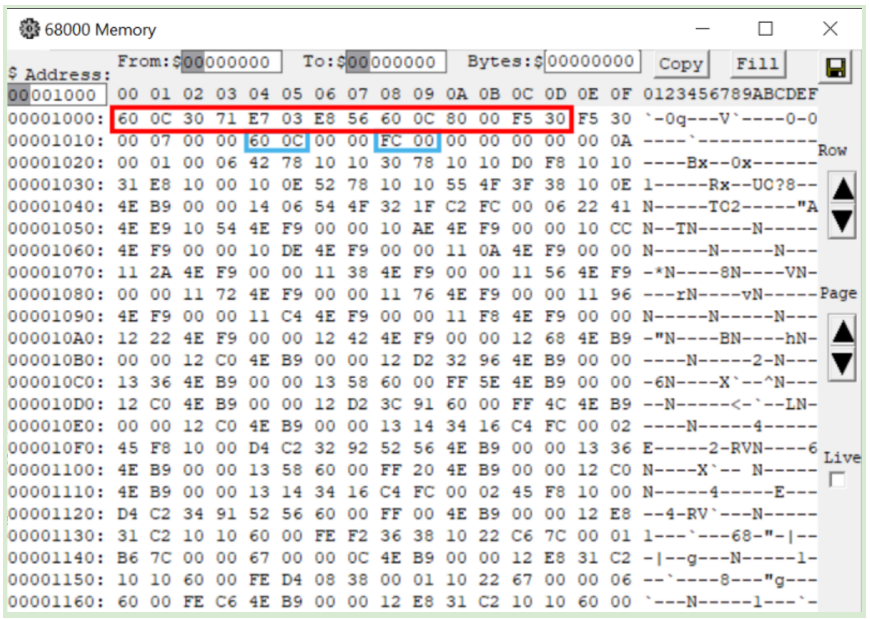


4a prueba:

Dirección @HAL9000	***Ensamblador***	***Instrucciones codificadas***	***HEX***
0:	LOA 4,T1	0000000000001001	0009
1:	STIP T1,X7	0011000001110001	3071
2:	SET #-32,X3	1110011100000011	E703
3:	ADQ #10,X3	1110100001010110	E856
4:	GON 6	0110000000001000	600C
5:	EXIT	1000000000000000	8000
6:	LSH #5,X3,#0	1111010100110000	F530

Este programa empieza con un LOA de la dirección 4 de la HAL, o 1008 del 68k, en ET1, tras ella ET1 contiene 600C. Acto seguido se ejecuta una instrucción del tipo STIP en la que se mueve el contenido de ET1 a la posición de memoria cuya dirección se encuentra en EX7, en este caso la 0 de la HAL, entonces, en 1000(hex se coloca el valor 600C, mientras que el contenido de EX7 aumenta en 1. Después se ejecuta un SET sobre EX3 del valor -32 decimal, para después sumarle el valor 10 decimal con un ADQ. Con este valor se ejecuta un salto condicional GON a la dirección 6 de la HAL, la cual se ejecuta ya que se hace un BMI con -22 decimal, evitando así el EXIT. Finalmente, se ejecuta una instrucción de desplazamiento de 5 bits hacia la izquierda sobre el eregistro EX3, el cual resulta en el valor FC00(hex (marcado en azul en la segunda fotografía) ya que se tenía 1111 1111 1110 0000 y tras el desplazamiento 1111 1100 0000 0000.

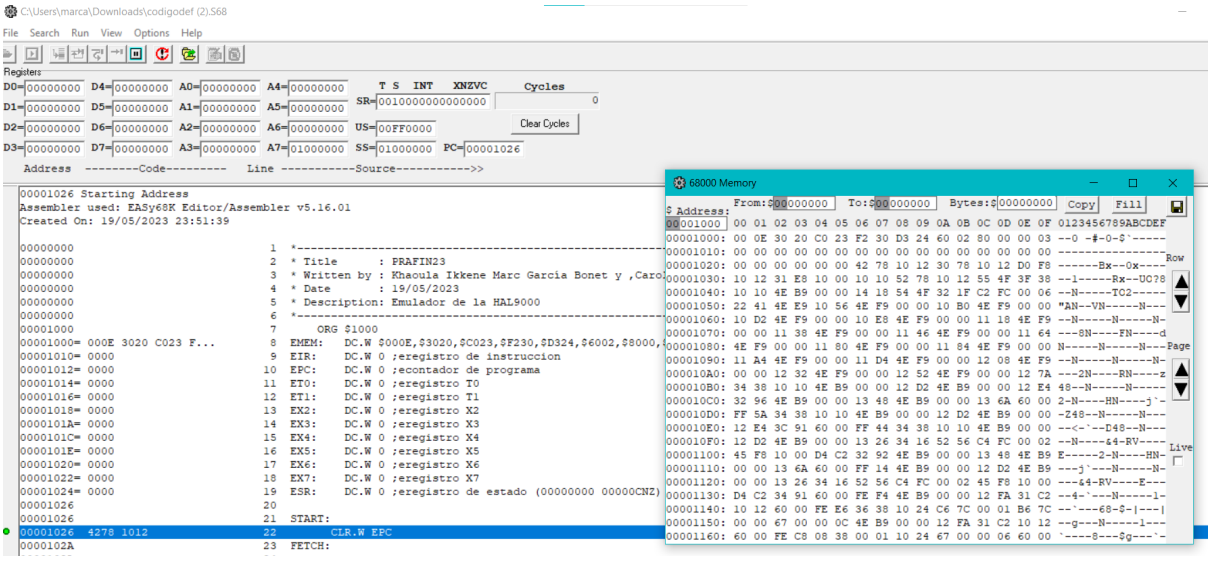


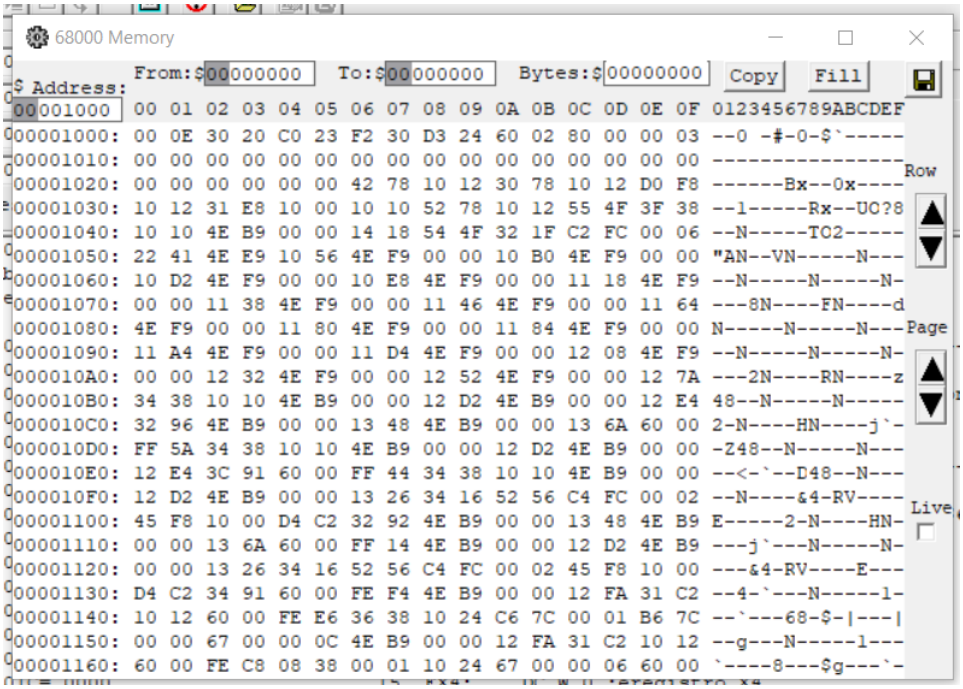


5a prueba:

Dirección @HAL9000	*Ensamblador*	*Instrucciones codificadas*	*HEX*
0:	LOA 7,T0	0000000000001110	000E
1:	STIP T0,X2	0011000000100000	3020
2:	COPY X2,X3	1100000000100011	C023
3:	LSH #2,X3,#0	1111001000110000	F230
4:	SUB X3,X2,X4	1101001100100100	D324
5:	GON 1	0110000000000010	6002
6:	EXIT	1000000000000000	8000
7:	3	0000000000000011	0003

Tras la ejecución del programa en la memoria debe quedar 3 en decimal que es equivalente a 3 en Hexadecimal en la posición que corresponde al registro T0 (@1014) y, también, debe quedar 2 en decimal que equivale a 2 en Hexadecimal en la posición que corresponde al registro X2 (@1018). De igual forma, 8 en decimal que es equivalente a 8 en Hexadecimal en la posición que ocupa X3 (@101A). Además, en X4, que ocupa la posición (@101C) 6 en decimal que equivale a FFFA en Hexadecimal.





VIII. Conclusiones

La primera vez que se nos presentó la práctica, parecía extremadamente desafiante, prolongada y compleja. Sin embargo, a lo largo de las tres sesiones dedicadas a explicar minuciosamente los aspectos de la práctica, las consideraciones a tener en cuenta y el proceso a seguir, gradualmente fuimos modificando nuestra impresión inicial.

La práctica, como se conoce su objetivo, nos resultó sumamente útil para profundizar en los conceptos teóricos relacionados con el ensamblador Motorola del 68k, así como para identificar nuestras posibles debilidades. Es decir, al enfrentarnos a la práctica, encontramos ciertos apartados más difíciles que otros y nos esforzamos por superar esos desafíos.

En este contexto, algunos aspectos, como la ejecución, demostraron ser más complicados que, por ejemplo, la etapa de búsqueda de instrucciones (Fetch). Principalmente, dedicamos más tiempo a programar las instrucciones de LOIP, STIP y las instrucciones de salto. Asimismo, la fase de pruebas que realizamos nos llevó cierto tiempo al codificar las instrucciones y verificar sus resultados.

Como una ventaja adicional, esta práctica nos ha permitido prepararnos para el examen y mejorar nuestra comprensión del funcionamiento del ensamblador, la decodificación de instrucciones y su ejecución.

IX. Código fuente

```
ORG $1000
EMEM:  DC.W $E082,$E09B,$E0B4,$E01D,$2020,$C006,$2031,$C017,$F160
     DC.W $F170,$CE70,$3040,$EFFD,$501E,$4008,$8000,$0001
     DC.W $0001,$0001,$0001,$0001,$0001,$0000,$0000,$0000
EIR:   DC.W 0 ;eregistro de instruccion
EPC:   DC.W 0 ;econtador de programa
ET0:   DC.W 0 ;eregistro T0
ET1:   DC.W 0 ;eregistro T1
EX2:   DC.W 0 ;eregistro X2
EX3:   DC.W 0 ;eregistro X3
EX4:   DC.W 0 ;eregistro X4
EX5:   DC.W 0 ;eregistro X5
EX6:   DC.W 0 ;eregistro X6
EX7:   DC.W 3 ;eregistro X7
ESR:   DC.W 0 ;eregistro de estado (00000000 00000CNZ)

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  ;Para  compensar la diferencia con el PC del 68k
    MOVE.W EMEM(A0),EIR ; relativo al PC
    ADDQ.W #1,EPC

  ;--- FFETCH: FIN FETCH



  ;--- IBRDECOD: INICIO SALTO A DECOD
    ;*** En esta sección debeis preparar la pila para llamar a la subrutina
    ;*** DECOD, llamar a la subrutina, y vaciar la pila correctamente,
    ;*** almacenando el resultado de la decodificación en



          ; ESCRIBID VUESTRO CODIGO AQUI
      ; en la subrutina usaremos D0 para copiar el EIR

            SUBA.W #2,A7 ; guardar espacio para el parametro de salida D1
```

```
            MOVE.W EIR, -(A7) ; guardar espacio para el parametro de entrada EIR


            JSR DECOD
            ADDQ.W #2,A7 ; Eliminar el EIR de la pila
            MOVE.W (A7)+,D1 ; guardar el id en 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 ELOA
  JMP ESTO
  JMP ELOIP
  JMP ESTIP
  JMP EGOI
  JMP EGOZ
  JMP EGON
  JMP EEXIT
  JMP ECOPY
  JMP EADD
  JMP ESUB
  JMP EAND
  JMP ESET
  JMP EADQ
  JMP ELSH
  ;--- FBREXEC: FIN SALTO A FASE DE EJECUCION



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

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

  JSR ValueofI
;A1 contiene la @ del registro destino.
  JSR ValueofM
;A6 contiene la @ del registro fuente.
  MOVE.W (A6),(A1)

  ;Actualizar los flags
```

```
   JSR FlagZ
   JSR FlagN

   BRA FETCH

ESTO:

   JSR ValueofI
  ;A1 contiene la @ del registro fuente.
   JSR ValueofM
   ;A6 contiene la @ de la posición destino.
   MOVE.W (A1),(A6)

   ;No se actualizan los flags

   BRA FETCH

ELOIP:

   JSR ValueofI
   ;A1 contiene la @ del registro destino.
   JSR ValueofB
   ;A6 tiene el registro cuya dirección contiene la posición de memoria fuente

   MOVE.W (A6),D2
   MULU.W #2,D2
   LEA.L EMEM,A2
   ADDA.W D2,A2  ;A2 contiene la posición de memoria fuente.
   MOVE.W (A2),(A1)
   ADD.W #1,(A6) ;Xb <- [Xb]+1

    ;Actualizar los flags
   JSR FlagZ
   JSR FlagN

   BRA FETCH



ESTIP:

   JSR ValueofI
   ;A1 contiene la dirección del registro definido por 'i'.
   JSR ValueofB
   ;A6 contiene la dirección de Rb.
   MOVE.W (A6),D2
   MULU.W #2,D2
   LEA.L EMEM,A2
   ADD.W D2,A2 ;A2 contiene la dirección destino.
   MOVE.W (A1),(A2)
```

```
   ADDQ.W #1,(A6) ;Xb <- [Xb] + 1

  ;No se actualizan los flags
   BRA FETCH

EGOI:


   JSR ValueofMAbsolut
   ;D2 contiene el valor de M
   MOVE.W D2,EPC ; cargar M en el EPC

   ;No se actualizan los flags

   BRA FETCH

EGOZ:

   MOVE.W ESR,D3
   AND.W #1,D3
   CMP #0,D3  ;Se usa D3 para comprobar si el bit 0 de ESR (Flag Z) es 1 o 0
   BEQ Z0 ; salta si Z = 0
   JSR ValueofMAbsolut
   ;D2 contiene el valor de M
   MOVE.W D2,EPC  ; cargar M en el EPC


Z0:
  ;no se cumple la condición
  BRA FETCH

EGON:

   BTST #1,ESR
   BEQ N1 ;salta si N = 1
   BRA FETCH
N1:
   JSR ValueofMAbsolut
   ;D2 contiene el valor de M
   MOVE.W D2,EPC  ; cargar M en el EPC
   BRA FETCH

EEXIT:


  SIMHALT ; detener la máquina
ECOPY:

   JSR ValueofB
   ;A6 contiene la dirección de Rb.
```

```
   MOVE.W (A6),D7 ;D7=[Rb]

   JSR ValueofC
   ;A6 contiene la dirección de Rc.
   MOVE.W D7,(A6) ;Rc <- [Rb]

 ;Actualizar los flags
   JSR FlagZ
   JSR FlagN


   BRA FETCH

EADD:

   JSR ValueofA
   ;A6 contiene la dirección de Ra.
   MOVE.W (A6),D7 ;D7=[Ra]

   JSR ValueofB
   ;A6 contiene la dirección de Rb.
   ADD.W (A6),D7 ;D7=[Ra]+[Rb]
   JSR FlagC ; actualizar el FlagC justo después de ejecutar la instrucción

   JSR ValueofC
   ;A6 contiene la dirección de Rc.
   MOVE.W D7,(A6) ;Rc <- [Ra]+[Rb]
 ;Actualizar los flags
   JSR FlagZ
   JSR FlagN

   BRA FETCH

ESUB:

   JSR ValueofA
   ;A6 contiene la dirección de Ra.
   MOVE.W (A6),D7
   NOT.W D7
   ADDQ.B #1,D7 ;D7=[Ra]'


   JSR ValueofB
   ;A6 contiene la dirección de Rb.
   MOVE.W (A6),D5 ;D5=[Rb]
   ADD.W D7,D5 ;D5=[Rb]-[Ra]
   JSR FlagC  ; actualizar el FlagC justo después de ejecutar la instrucción

   JSR ValueofC
   ;A6 contiene la dirección de Rc.
```

```
   MOVE.W D5,(A6) ;Rc=[Rb]-[Ra]

   JSR FlagZ
   JSR FlagN

   BRA FETCH

EAND:

   JSR ValueofA
   ;A6 contiene la dirección de Ra.
   MOVE.W (A6),D7 ;D7=[Ra]

   JSR ValueofB
   ;A6 contiene la dirección de Rb.
   MOVE.W (A6),D5 ;D5=[Rb]
   AND.W D7,D5

   JSR ValueofC
   ;A6 contiene la dirección de Rc.
   MOVE.W D5,(A6) Rc <- [Ra] ^ [Rb]

   JSR FlagZ
   JSR FlagN

   BRA FETCH

ESET:

   JSR ValueofK
   ;D2 contiene el valor de K.
   MOVE.W D2,D5 ;Se guarda el valor de D2 porque D2 se usará en ValueofC
   JSR ValueofC
   MOVE.W D5,(A6)

   JSR FlagZ
   JSR FlagN

   BRA FETCH

EADQ:

   JSR ValueofK
   ;D2 contiene el valor de K.
   MOVE.W D2,D5 ;Se guarda el valor de D2 porque D2 se usará en ValueofC
   JSR ValueofC
   ADD.W D5,(A6) Rc <- [Rc]+K

   JSR FlagC
   JSR FlagZ
```

```
   JSR FlagN

   BRA FETCH

ELSH:

   JSR ValueofB

   MOVE.W EIR,D2
   AND.W #$0700,D2 ;0700 para que los unos coincidan con la posición de ppp.
   LSR.W #8,D2 ;D2 contiene el valor de ppp

   MOVE.W EIR,D3
   AND.W #1,D3 ;D3 contiene el valor de n
   CMP.W #0,D3
   BEQ N0

   MOVE.W (A6),D3 ;Si n=1 se produce un desplazamiento a la derecha de ppp bits
   LSR.W D2,D3
   JSR FlagC
   MOVE.W D3,(A6) ;Rb <- [Rb] right shift p

   JSR FlagZ
   JSR FlagN


   BRA FETCH

N0:

   MOVE.W (A6),D3 ;Si n=0 se produce un desplazamiento a la izquierda
   LSL.W D2,D3
   JSR FlagC
   MOVE.W D3,(A6) ;Rb <- [Rb] left shift p

   JSR FlagZ
   JSR FlagN

   BRA FETCH

   ;--- FEXEC: FIN EJECUCION



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

                  ; ESCRIBID VUESTRO CODIGO AQUI
```

```
ValueofI: ; obtener el valor del i

   MOVE.W EIR,D2 ;D2 contiene la codificación de la instrucción
   AND.W #1,D2

   JSR ValueofR

   MOVE.W A6,A1 ;Se guarda el valor de A6 en A1 por si se vuelve a usar A6.
   RTS

 ValueofM: ; para sacar m de la instrucción
   MOVE.W EIR,D2
   AND.W #$01FE,D2 ;01FE para que los unos coincidan con M
   LSR.W #1,D2
   MULU.W #2,D2 ;se multiplica por 2 porque una posición de la HAL son dos del 68k
   LEA.L EMEM,A6
   ADD.W D2,A6
   RTS

 ValueofMAbsolut: ;Obtener M sin cargarlo en A6
   MOVE.W EIR, D2
   AND.W #$01FE, D2
   LSR.W #1,D2
   RTS

ValueofK:

 MOVE.W EIR,D2
 AND.W #$07F8,D2 ;07F8 para que los unos coincidan con la posición de K
 LSR.W #3,D2 ;Despl. 3 a la derecha porque el último bit de K es el #3 de D2
 EXT.W D2 ;Extensión de signo de K
 RTS


ValueofA: ;obtener el valor de aaa
   MOVE.W EIR,D2
   AND.W #$0700,D2 ;0700 para que los unos coincidan con la posición de aaa
   LSR.W #8,D2 ; D2 contiene el valor de aaa

   JSR ValueofR
   RTS


ValueofB: ;obtener el valor de bbb

   MOVE.W EIR,D2
   AND.W #$0070,D2 ;0070 para que los unos coincidan con la posición de bbb
   LSR.W #4,D2 ; D2 contiene el valor de bbb

   JSR ValueofR
```

```
    RTS

ValueofC:  ;obtener el valor de ccc
    MOVE.W EIR,D2
    AND.W #$0007, D2 ;D2 contiene el valor de ccc

    JSR ValueofR
    RTS

FlagZ:

 ;El registro D4 será utilizado para las subrutinas de actualización de flags

 MOVE.W (A6),D4
 BNE CERO ;salta Si Z=0

 MOVE.W ESR, D4
 BSET.L #0,D4
 MOVE.W D4, ESR ;Flag Z HAL = 1
 RTS
CERO:
 MOVE.W ESR, D4
 BCLR.L #0, D4
  MOVE.W D4, ESR ;Flag Z HAL = 0

 RTS

FlagN:
   MOVE.W (A6),D4
   CMP.W #0,D4
   BMI NEG ;salta Si N=1
   ;Si N = 0
   MOVE.W ESR, D4
   BCLR.L #1, D4
   MOVE.W D4, ESR ;Flag N HAL = 0

   RTS
NEG:
   MOVE.W ESR, D4
   BSET.L #1,D4
   MOVE.W D4, ESR ;Flag Z HAL = 1

   RTS

FlagC:

  BCS C1  ;Si el carry del 68k está a 1 se salta a C1
  MOVE.W ESR, D4
  BCLR.L #2,D4 ;Si no, el bit correspondiente a Carry de ESR se pone a 0
  MOVE.W D4, ESR
```

```
 RTS
C1:
 MOVE.W ESR, D4 ;Se mueve ESR a D4 y se pone el bit de carry a 1.
 BSET.L #2,D4
 MOVE.W D4, ESR


 RTS

ValueofR:

 ;Se va mirando el valor de D2 para saber qué registro se está tratando.

 CMP.B #7,D2
 BEQ R7
 CMP.B #6,D2
 BEQ R6
 CMP.B #5,D2
 BEQ R5
 CMP.B #4,D2
 BEQ R4
 CMP.B #3,D2
 BEQ R3
 CMP.B #2,D2
 BEQ R2
 CMP.B #1,D2
 BEQ R1
 LEA.L ET0,A6
 RTS

 ;Se carga A6 con la dirección del registro obtenido con los CMP.

 R1:
 LEA.L ET1,A6
 RTS
 R2:
 LEA.L EX2,A6
 RTS
 R3:
 LEA.L EX3,A6
 RTS
 R4:
 LEA.L EX4,A6
 RTS
 R5:
 LEA.L EX5,A6
 RTS
 R6:
 LEA.L EX6,A6
```

```
 RTS
 R7:
 LEA.L EX7,A6
 RTS




  ;--- FSUBR: FIN SUBRUTINAS



  ;--- 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.W D0, -(A7)  ; guardar el registro de datos a usar
  MOVE.W 6(A7),D0      ; mover EIR en D0
  BTST.L #15, D0
  BNE B1  ; bit = 1; Z = 0




  ;Se miran los 4 o 5 primeros bits para saber de qué instrucción se trata.
B0:

 BTST.L #14, D0
  BNE B01

B00:
 BTST.L #13, D0
  BNE B001
B000:
   BTST.L #12, D0
  BNE B0001
B0000:
 MOVE.W #0, 8(A7)  ; LOA
    JMP ET3
B0001:
  MOVE.W #1,8(A7) ;STO

   JMP ET3

B001:
  BTST.L #12,D0
  BNE B0011
```

```
B0010:
   MOVE.W #2,8(A7) ; LOIP
    JMP ET3
B0011:
   MOVE.W #3, 8(A7) ; STIP

    JMP ET3
B01:
   BTST.L #13,D0
   BNE B011
B010:
   BTST.L #12,D0
   BNE B0101
B0100:
   MOVE.W #4,8(A7) ; GOI
   ;RTS
    JMP ET3
B011:
   BTST.L #12,D0
   BNE B0111
B0110:
 MOVE.W #6,8(A7) ; GON
   JMP ET3
B0101:
   MOVE.W #5,8(A7) ; GOZ
   JMP ET3

B0111:
   ; no existe esta instrucción
      JMP ET3

B1:
   BTST.L #14, D0
   BNE B11
B10:
   MOVE.W #7,8(A7) ; EXIT
    JMP ET3

B11:
   BTST.L #13, D0
   BNE B111
B110:
   BTST.L #12,D0
   BNE B1101
B1100:
   BTST.L #11,D0
   BNE B11001
B11000:
   MOVE.W #8,8(A7) ; COPY
   JMP ET3
```

```
B11001:
  MOVE.W #9,8(A7) ; ADD
  JMP ET3

B1101:
  BTST.L #11,D0
  BNE B11011
B11010:
  MOVE.W #10,8(A7) ; SUB
  JMP ET3
B11011:
  MOVE.W #11,8(A7) ; AND
   JMP ET3
B111:
  BTST.L  #12,D0
  BNE B1111
B1110:
  BTST.L #11, D0
  BNE B11101
B11100:
  MOVE.W #12, 8(A7) ; SET
  JMP ET3
B1111:
  BTST.L #11, D0
  BEQ B11110

B11110:
  MOVE.W #14,8(A7) ; LSH
  JMP ET3
B11101:
  MOVE.W #13, 8(A7) ; ADQ

 ET3:
  MOVE.W  (A7)+, D0 ; recuperar el valor de D0
  RTS


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