Recuperación práctica BBMe: Balearic Basic Machine

Jiménez Sánchez, Pablo
pablo.jimg@gmail.com
45190686X

Palmer Pérez, Rubén
rpp776@id.uib.cat
43474448D

Curso 2018-2019

Estructura de computadores I - Grupo 1

May 2019

Índice

1 Introducción

En esta práctica se nos plantea la emulación de una máquina elemental llamada BBMe o *Balearic Basic Machine*, la cual es una **máquina elemental**. Esta emulación ha sido hecha para una máquina ***68K***.

La BBMe es una máquina de dos direcciones que trabaja con *words* de 16 *bits*. Tiene 4 registros de propósito general, 2 que sirven como interfaz con la memoria, un registro de estado, un registro de instrucción y un *program counter* (PC).

- **EIR -** Registro de instrucción donde se almacena la instrucción a ejecutar.
- **EPC -** Registro de contador de programa que apunta a la siguiente instrucción a ejecutar.
- **ER0, ER1, ER2, ER3 -** Registros de propósito general.
- **EB4 -** Registro de acceso a memoria directo por memoria.
- **EB5 -** Registro de acceso a memoria directo por registro.
- **ESR -** Registro de estado que guarda en los tres bits menos significativos los flags. *Zero*, *Negative* y *Carry* en ese orden **(0000 0000 0000 0ZNC)**.

2 Explicación general

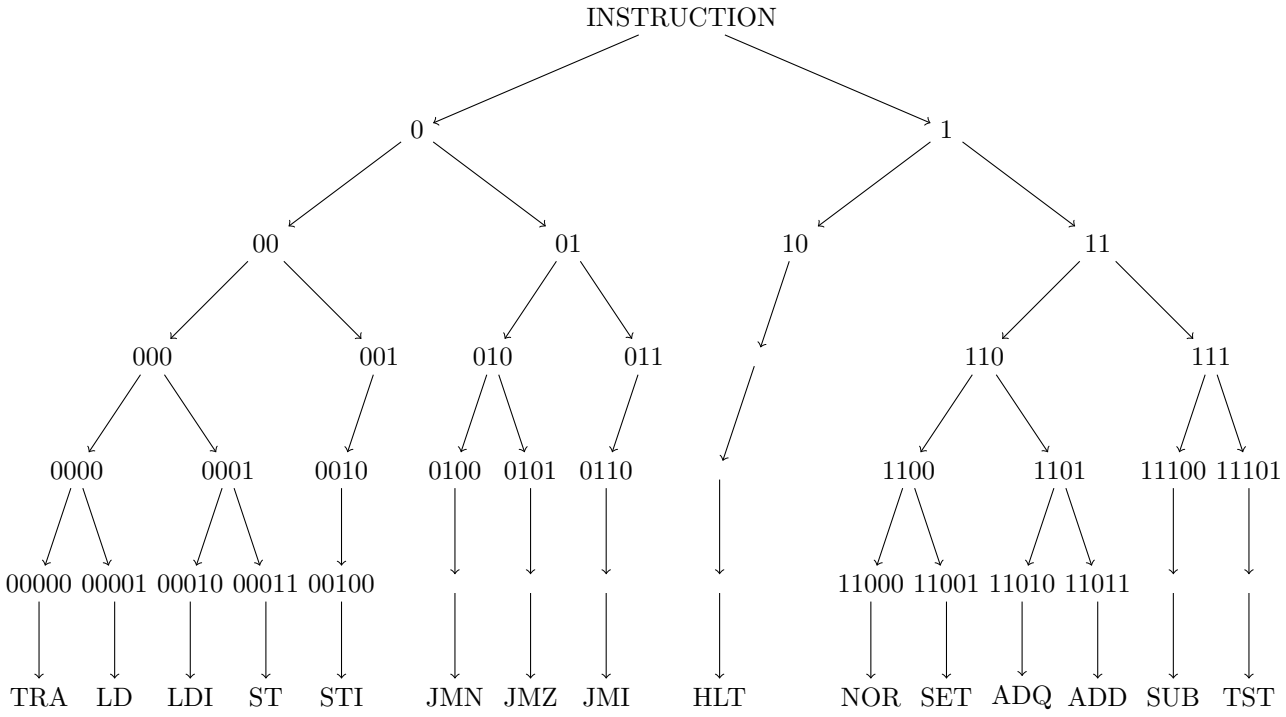
A continuación explicaremos cómo se han resuelto cada una de las distintas fases que se hallan en un procesador para poder emular nuestra máquina elemental.

2.1 Fase *fetch*

Para la fase *fetch* obtenemos el contador de la siguiente einstrucción desde **EPC**, lo multiplicamos por 2 debido a que la memoria nuestra máquina elemental trabaja con *words* de 16 *bits* y la de ***68K*** con *bytes*. Entonces, la buscamos en memoria (**EPROG**) y la transferimos en **EIR**.

2.2 Fase de decodificación

Para la fase de decodificación, cogemos la einstrucción contenida en **EIR** y la sometemos a una serie de tests bit a bit hasta que conseguimos identificar la instrucción a realizar. En la página siguiente se mostrará un árbol compuesto por las instrucciones dependiendo del resultado de los tests.



2.3 Fase de ejecución

Según el valor que devuelva la decodificación, se ejecutará una u otra instrucción:

- **0. TRA -** Transfiere el contenido de un registro <a> a un registro <b>.
- **1. LD -** Carga el contenido de una dirección **M** en el registro **B4** o **B5** dependiendo de un parametro **J**.
- **2. LDI -** Carga en **B4** el contenido del la dirección contenida en B5.
- **3. ST -** Guarda el conenido de **B4** o **B4** (en relación a un parámetro **J**) en un espacio de memoria indicado por un parametro **M**.
- **4. STI -** Guarda el contenido de **B4** en la dirección contenida en **B5**.
- **5. JMI -** Salto incondicional dado por un parametro **M**.
- **6. JMZ -** Salto condicional dado por un parametro **M** y el flag **Z**.
- **7. JMN -** Salto condicional dado por un parametro **M** y el flag **N**.
- **8. HLT -** Detiene la máquina.
- **9. NOR -** Hace la *NOR* lógica entre un registro <a> y un registro <b> y guarda el resultado en un registro <c>.
- **10. SET -** Mueve el valor de una constante <k> en un registro <b>.
- **11. ADQ -** Suma una constante <k> y el contenido de un registro <b> y los guarda en este último.
- **12. ADD -** Suma los contenidos de un registro <a> y un registro <b> y guarda el resultado en un registro <c>.
- **13. SUB -**Resta los contenidos de un registro <a> y un registro <b> mediante una negación del contenido del registro <b> y una suma de un 1 y guarda el resultado en un registro <c>.
- **14. TST -** Resta los contenidos de un registro <a> y un registro <b> mediante una negación del contenido del registro <b> y una suma de un 1 y **no guarda el resultado de la resta**.

3 Rutina de decodificación

En la rutina de decodificación guardamos en la primera posición disponible del *stack pointer* (SP) la instrucción a decodificar y en la segunda posición disponible reservamos un espacio para el resultado (el número de la instrucción).

4 Tabla de subrutinas

Etiqueta	**Librería**	**Entrada**	**Funcionalidad**	**Salida**
GET_ESR_C	NO	SR	Obtención flags (modifica acarreo)	D6
GET_ESR_NOTC	NO	SR	Obtención flags (no modifica acarreo)	D6
GET_A	NO	EIR	Obtención parámetro A	A0
GET_B	NO	EIR	Obtención parámetro B	A1
GET_C	NO	EIR	Obtención parámetro C	A4
GET_J	NO	EIR	Obtención del registro indicado por el parámetro J $J = 0 \implies B4$ $J = 1 \implies B5$	A3
GET_K	NO	EIR	Obtención parámetro K	D1
DREG	NO	D4	Obtención del registro mencionado en los parámetros A,B y C	A3
DCOD	SI	SUBQ.W #2,SP MOVE.W EIR,-(SP)	Decodificacion de la instrucción en EIR	MOVE.W 2(SP) ADDQ.L 4, SP

5 Tabla de registros del 68K

Registros	Función	Utilización
A0	Búsqueda de instrucciónes Como parámetro <a>	Fase de Fetch Dirección del registro <a>
A1	Cálculo de instrucciónes Como parámetro <b>	Cálculo de instrucciónes Como parámetro <b>
A4	Cálculo de instrucciónes Como parámetro <c>	Cálculo de instrucciónes Como parámetro <c>
A3	Cálculo de instrucciónes Como parámetro <j>	Cálculo de instrucciónes Como parámetro <j>
A2	Auxiliar	Cálculo de los parametros <a>, <b> & <c>
D0	Auxiliar	Cálculo de los parametros <a>, <b>, <c> & <k> Masking Incialización de la pila
D1	Auxiliar	Almacenar decodificaciónes Como parámetro <k> Cálculo de Flags
D2	Almacenar el contenido de <a>	Ejecución de instrucciones
D3	Almacenar el contenido de <b>	Ejecución de instrucciones
D4	Masking	GET_A, GET_B, GET_C GET_K, DREG
D5	Flag Z	Cálculo de Flags
D6	Flag N	
D7	Flag C	

6 Conjunto de pruebas

A continuación, para comprobar que nuestra emulación se ejecuta correctamente, hemos decidido probar con un código que probase el resto de instrucciones que quedan por ejecutar en ambos programas suministrados en el enunciado de la práctica.

@BBMe	**Ensamblador**	**Codificación**	**Hex**
0:	SET 9,B5	1100 1000 0100 1101	C84D
1:	LDI	0001 0000 0000 0000	1000
2:	SET 2,R0	1100 1000 0001 0000	C810
3:	TST R0,B4	1110 1000 0000 0100	E804
4:	JMZ 6	0101 0000 0000 0110	5006
5:	HLT	1000 0000 0000 0000	8000
6:	NOR B4,R0,B4	1100 0001 0000 0100	C104
7:	STI	0010 0000 0000 0000	2000
8:	HLT	1000 0000 0000 0000	8000
9:	2	0000 0000 0000 0002	0002

Este código convierte el contenido de la posición **9** en sun negación mediante una *NOR* consigo misma si el contenido de **R0** (el cual actualizamos con un *SET*) es igual a la posición **9**. Si no son iguales, el programa detiene la máquina. Por lo tanto, el contenido en la posición **9** de la **BBMe** equivale a **#$FFFD** (equivalente a la posición **$1012**) en ***68K***.

7 Conclusiones

Para realizar esta práctica ha sido necesario primero comprender el funcionamiento del ***68K*** (pila, acceso a memoria, operaciones con registros, etc). Entonces, entendiendo como debería operar la *BBMe*, cómo funciona la decodificación de cada instrucción y cómo funcionan los accesos a memoria, hemos sido capaces de simular su funcionalidad recurriendo a las capacidades del propio ***68K***.

8 Código fuente *68K*

```
*-----------------------------------------------------------
* Title      : PRAREC19
* Written by : Pablo Jimenez Sanchez, Ruben Palmer Perez
* Date       : 28/06/2019
* Description: Emulador de la BBMe
*-----------------------------------------------------------
    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:
    ;--- 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
```

```
        MOVE.W EPC,A0
        ADD.W A0,A0             ;multiplicacion x2 del valor para que al ser
        MOVE.W EPROG(A0),EIR    ;sumado coincida con el valor real en 68K
        ADDQ.W #1,EPC

    ;--- FFETCH: FIN FETCH

    ;--- IBRDECOD: INICIO SALTO A DECOD
        ;*** En esta seccion debeis preparar la pila para llamar a la subrutina
        ;*** DECOD, llamar a la subrutina, y vaciar la pila correctamente,
        ;*** almacenando el resultado de la decodificacion en D1

        SUBQ.W #2,SP            ;preparacion de la pila
        MOVE.W EIR,-(SP)        ;paso de par metros por pila
        JSR DECOD
        MOVE.W 2(SP),D1         ;obtencion del resultado
        ADDQ.L #4,SP    ;limpieza de la pila

    ;--- FBRDECOD: FIN SALTO A DECOD

    ;--- IBREXEC: INICIO SALTO A FASE DE EJECUCION
        ;*** Esta seccion se usa para saltar a la fase de ejecucion
        ;*** NO HACE FALTA MODIFICARLA
    MULU #6,D1
    MOVEA.L D1,A1
    JMP JMPLIST(A1)
JMPLIST:
    JMP ETRA
    JMP ELD
    JMP ELDI
    JMP EST
    JMP ESTI
```

```
    JMP EJMN
    JMP EJMZ
    JMP EJMI
    JMP EHLT
    JMP ENOR
    JMP ESET
    JMP EADQ
    JMP EADD
    JMP ESUB
    JMP ETST
    ;--- FBREXEC: FIN SALTO A FASE DE EJECUCION

    ;--- IEXEC: INICIO EJECUCION
        ;*** En esta seccion debeis implementar la ejecucion de cada einstr.
ETRA:
    JSR GET_A
    JSR GET_B
    MOVE.W (A0),D2
    MOVE.W D2,(A1)
    JSR GET_ESR_NOTC
    BRA FETCH
ELD:
    MOVE.W EIR, D0
    AND.W #$00FF,D0
    MOVE.W D0,A0
    ADD.W A0,A0
    JSR GET_J
    MOVE.W EPROG(A0),(A3)
    JSR GET_ESR_NOTC
    BRA FETCH
ELDI:
    MOVE.W EB5,A0
```

```
        ADD.W A0,A0
        MOVE.W EPROG(A0),EB4
        JSR GET_ESR_NOTC
        BRA FETCH
EST:
        MOVE.W EIR, D0
        AND.W #$00FF,D0
        MOVE.W D0,A0
        ADD.W A0,A0
        JSR GET_J
        MOVE.W (A3),EPROG(A0)
        BRA FETCH
ESTI:
        MOVE.W EB5,A0
        ADD.W A0,A0
        MOVE.W EB4,EPROG(A0)
        BRA FETCH
EJMN:
        MOVE.W ESR,D1
        BTST.L #1,D1
        BEQ FETCH
        MOVE.W EIR,D0
        AND.W #$00FF,D0
        MOVE.W D0,EPC
        BRA FETCH
EJMZ:
        MOVE.W ESR,D1
        BTST.L #2,D1
        BEQ FETCH
        MOVE.W EIR,D0
        AND.W #$00FF,D0
        MOVE.W D0,EPC
```

```
        BRA  FETCH
EJMI:
        MOVE.W  EIR,D0
        AND.W #$00FF,D0
        MOVE.W  D0,EPC
        BRA  FETCH
EHLT:
        SIMHALT
ENOR:
        JSR  GET_A
        JSR  GET_B
        JSR  GET_C
        MOVE.W  (A0),D2
        MOVE.W  (A1),D3
        OR.W D2,D3
        NOT.W D3
        MOVE.W D3,(A4)
        JSR  GET_ESR_NOTC
        BRA  FETCH
ESET:
        JSR  GET_B
        JSR  GET_K
        MOVE.W D1,(A1)
        JSR  GET_ESR_NOTC
        BRA  FETCH
EADQ:
        JSR  GET_B
        JSR  GET_K
        MOVE.W  (A1),D3
        ADD.W D3,D1
        JSR  GET_ESR_C
        MOVE.W D1,(A1)
```

```
        BRA  FETCH
EADD:
        JSR  GET_A
        JSR  GET_B
        JSR  GET_C
        MOVE.W  (A0),D2
        MOVE.W  (A1),D3
        ADD.W D2,D3
        JSR  GET_ESR_C
        MOVE.W D3,(A4)
        BRA  FETCH
ESUB:
        JSR  GET_A
        JSR  GET_B
        JSR  GET_C
        MOVE.W  (A0),D2
        MOVE.W  (A1),D3
        NOT.W D3
        ADDQ.W  #1,D3
        ADD.W D2,D3
        JSR  GET_ESR_C
        MOVE.W D3,(A4)
        BRA  FETCH
ETST:
        JSR  GET_A
        JSR  GET_B
        MOVE.W  (A0),D2
        MOVE.W  (A1),D3
        NOT.W D3
        ADDQ.W  #1,D3
        ADD.W D2,D3
        JSR  GET_ESR_C
```

```
    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

GET_ESR_C:          ;actualiza el valor de los flags (incluyendo carry)
    JSR GET_ESR_NOTC
    AND.W #1,D7
    OR.W D7,D6
    MOVE.W D6,ESR
    RTS
GET_ESR_NOTC:       ;actualiza el valor de los flags (sin incluir carry)
    MOVE.W SR,D6
    MOVE.W D6,D5
    AND.W #8,D5
    LSR.W #2,D5
    AND.W #4,D6
    OR.W D5,D6
    MOVE.W D6,ESR
    RTS
GET_A:              ;identifica el registro del parametro A
    MOVE.W #56,D4
    AND.W EIR,D4
    LSR #3,D4
    JSR DREG
    MOVE.W A2,A0
    RTS
GET_B:              ;identifica el registro del parametro B
    MOVE.W #7,D4
    AND.W EIR,D4
```

```
    JSR DREG
    MOVE.W A2,A1
    RTS
GET_C:              ;identifica el registro del parametro C
    MOVE.W #448,D4
    AND.W EIR,D4
    LSR #6,D4
    JSR DREG
    MOVE.W A2,A4
    RTS
GET_J:
    BTST.B #8, EIR
    BEQ JB4
JB5:
    LEA.L EB5, A3
    RTS
JB4:
    LEA.L EB4, A3
    RTS
GET_K:              ;extrae la constante K de la instruccion
    MOVE.W #2040,D4
    AND.W EIR,D4
    LSR #3,D4
    EXT.W D4
    MOVE.W D4,D1
    RTS
DREG:               ;decodificador similar al usado para identificar
    BTST.L #2,D4    ;la instruccion pero para los registros
    BNE R10
R0:
    BTST.L #1,D4
    BNE R01
```

```
R00:
    BTST.L #0,D4
    BNE R001
R000:
    LEA.L ER0,A2
    RTS
R001:
    LEA.L ER1,A2
    RTS
R01:
    BTST.L #0,D4
    BNE R011
R010:
    LEA.L ER2,A2
    RTS
R011:
    LEA.L ER3,A2
    RTS
R10:
    BTST.L #0,D4
    BNE R101
R100:
    LEA.L EB4,A2
    RTS
R101:
    LEA.L EB5,A2
    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:
    BTST.B #7, 4(SP)
    BEQ I0
I1:
    BTST.B #6, 4(SP)
    BEQ I10
I11:
    BTST.B #5, 4(SP)
    BEQ I110
I111:
    BTST.B #3, 4(SP)
    BEQ I11100
I11101:
    MOVE.W #14, 6(SP)    ;TST
    RTS
I11100:
    MOVE.W #13, 6(SP)    ;SUB
    RTS
I110:
    BTST.B #4, 4(SP)
    BEQ I1100
I1101:
    BTST.B #3, 4(SP)
    BEQ I11010
I11011:
    MOVE.W #12, 6(SP)    ;ADD
    RTS
I11010:
    MOVE.W #11, 6(SP)    ;ADQ
    RTS
I1100:
```

```
    BTST.B #3, 4(SP)
    BEQ I11000
I11001:
    MOVE.W #10, 6(SP)    ;SET
    RTS
I11000:
    MOVE.W #9, 6(SP)     ;NOR
    RTS
I10:
    MOVE.W #8, 6(SP)     ;HLT
    RTS
I0:
    BTST.B #6, 4(SP)
    BEQ I00
I01:
    BTST.B #5, 4(SP)
    BEQ I010
I011:
    MOVE.W #7, 6(SP)     ;JMI
    RTS
I010:
    BTST.B #4, 4(SP)
    BEQ I0100
I0101:
    MOVE.W #6, 6(SP)     ;JMZ
    RTS
I0100:
    MOVE.W #5, 6(SP)     ;JMN
    RTS
I00:
    BTST.B #5, 4(SP)
    BEQ I000
```

```
I001:
    MOVE.W #4, 6(SP)      ;STI
    RTS
I000:
    BTST.B #4, 4(SP)
    BEQ I0000
I0001:
    BTST.B #3, 4(SP)
    BEQ I00010
I00011:
    MOVE.W #3, 6(SP)      ;ST
    RTS
I00010:
    MOVE.W #2, 6(SP)      ;LDI
    RTS
I0000:
    BTST.B #3, 4(SP)
    BEQ I00000
I00001:
    MOVE.W #1, 6(SP)      ;LD
    RTS
I00000:
    MOVE.W #0, 6(SP)      ;TRA
    RTS
    ;--- FDECOD: FIN DECOD
    END     START
```