

Práctica PS-ECI: Procesador Simple - Estructura de Computadores I

Seguí Vives, Mateu <i>mateufassers6b@gmail.com</i>	Palmer Pérez, Rubén <i>ruben.palmer1@estudiant.uib.cat</i>
20549548R	43474448D

Curso 2019-2020
Estructura de Computadores I - Grupo (numero de grupo)

17 de mayo de 2020

Índice

1. Introduction	3
1.1. Registros e Instrucciones	3
1.2. Ejecución de instrucciones	3
1.2.1. <i>fetch</i>	3
1.2.2. <i>decodificación</i>	4
1.2.3. <i>ejecución</i>	4
2. Rutina de decodificación	5
3. Tabla de subrutinas	6
4. Tabla de registros del 68K	7
5. Conjunto de pruebas	8
6. Conclusiones	8
7. Código fuente	8

1. Introduction

En esta práctica se nos presenta con el reto de emular una máquina llamada *PS-ECI (Procesador Simple - Estructura de Computadores I)* en el **68K** usando los conocimientos que hemos ido adquiriendo a lo largo de nuestro curso.

Las instrucciones y registros de la *PS-ECI* son de 16bits. A lo largo de esta documentación, a la hora de hablar sobre registros emulados, direcciones emuladas, etc, se usara el pronombre **e**.

1.1. Registros e Instrucciones

A continuacion se mostraran los registros que tiene esta máquina y sus respectivas funciones:

- **T0, T1** - Propio de operaciones de tipo **ALU** e interfaz con a memoria principal.
- **R2, R3, R4, R5** - De uso general ademas de usarse en algunas operaciones de tipo **ALU**.
- **B6, B7** - Registros de direcciones usados en algunas instrucciones mediante el direccionamiento indirecto.
- **EIR** - Registro de instrucción.
- **EPC** - Contador de programa.
- **ESR** - Registro de estado que guarda los flags *Zero*, *Negative* y *Carry* tal que: (00000000 0000ZNC).

1.2. Ejecución de instrucciones

Debemos simular cada uno de los pasos que una maquina debe hacer a la hora de ejecutar cualquier instrucción. Será necesario, pues, emular la fase de fetch, decodificación y la propia ejecución

1.2.1. *fetch*

Empezando con la fase de *fetch*, tenemos el contador de programa **EPC** que apunta a la siguiente instrucción. Sin embargo hay que tener en cuenta que la *SP-ECI* trabaja con registros de 16bits mientras que la maquina del *68K* trabaja con bytes. Para arreglar este problema, **multiplicaremos por 2 nuestro EPC**. Una vez tenemos la dirección de la siguiente instrucción, podemos buscarla en la memoria emulada **EPROG** y la transferimos a **EIR**

1.2.2. *decodificación*

En la decodificación de la instrucción se mira el valor de la instrucción bit a bit de izquierda a derecha y según esos valores va saltando a la instrucción correspondiente.

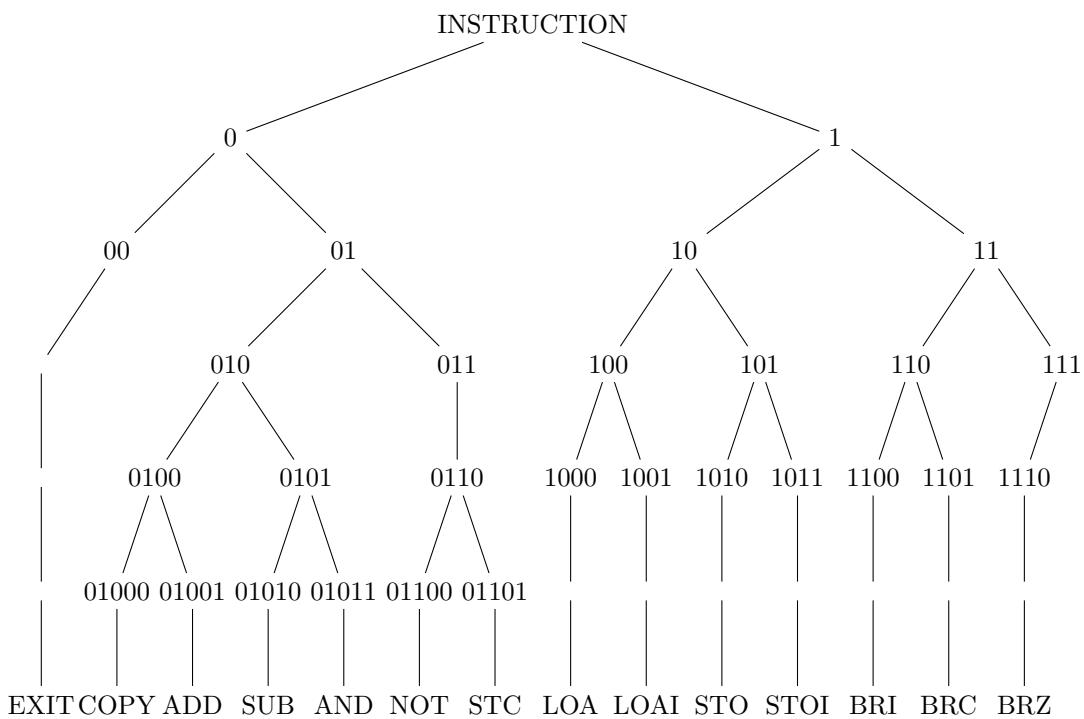
1.2.3. *ejecución*

Según el valor que nos devuelva la fase de *decodificación*, se ejecutara una de las siguientes instrucciones:

- **0. EXIT:** Detiene la máquina
- **1. COPY:** Carga el contenido de cualquier registro $< b >$ a otro registro $< c >$
- **2. ADD:** Suma los contenidos de dos registros $< a >$ y $< b >$ y guarda el resultado en el registro $< c >$
- **3. SUB:** Resta los contenidos de dos registros $< a >$ y $< b >$ y guarda el resultado en el registro $< c >$
- **4. AND:** Realiza la *AND* bit a bit lógica entre los registros $< b >$ y $< c >$ y el resultado es guardado en el registro $< c >$
- **5. NOT:** Realiza la *NOT* bit a bit lógica del registro $< c >$ y el resultado es guardado en si mismo.
- **6. STC:** Guarda una constante **k** con extensión de signo en un registro $< c >$
- **7. LOA:** Guarda el contenido de una dirección **M** en un registro **T0** o **T1** (dependiendo de una variable **i** codificada en la instrucción)
- **8. LOAI:** Guarda el contenido del contenido del registro **B6** o **B7** (dependiendo de una variable **j** codificada en la instrucción) en el registro **T0**
- **9. STO:** Guarda el contenido de un registro **T0** o **T1** (dependiendo de una variable **i** codificada en la instrucción) en la dirección **M**
- **10. STOI:** Guarda el contenido del registro **T0** en el contenido de **B6** o **B7** (dependiendo de una variable **j** codificada en la instrucción)
- **11. BRI:** Branch incondicional dado por una dirección **M**
- **12. BRC:** Branch condicional dado por una dirección **M** y el eflag **C**
- **13. BRZ:** Branch condicional dado por una dirección **M** y el eflag **Z**

2. Rutina de decodificación

Para decodificar las instrucciones, usamos una subrutina de librería **DECOD** que usa la instrucción **BTST** para mirar el valor del bit actual en orden del más significativo al menos significativo. Dependiendo del valor del bit, se hace el salto correspondiente a la siguiente etiqueta hasta llegar a la que coincide con el valor de la instrucción. Previamente, se reservará un espacio en la pila para que la subrutina nos devuelva el valor asociado de la instrucción y haremos un push del registro **EIR**. A continuación se presenta el arbol de decodificación



3. Tabla de subrutinas

Etiqueta	Librería	Entrada	Funcionalidad	Salida
FLAGSNC	NO	SR	Obtención flags (No modifica acarreo)	D6
FLAGS	NO	SR	Obtención flags (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	A2
GET_K	NO	EIR	Obtención de la constante k	D2
GET_M	NO	EIR	Obtención de la constante M	D2
GET_I	NO	EIR	Obtención del registro indicado por el parámetro I $I = 0 \implies T0$ $I = 1 \implies T1$	A4
GET_J	NO	EIR	Obtención del registro indicado por el parámetro J $J = 0 \implies B6$ $J = 1 \implies B7$	A4
REGDEC	NO	D4	Obtención del registro mencionado en los parámetros A,B y C	A4
DECOD	SI	SUBQ.W #2,SP MOVE.W EIR,-(SP)	Decodificación de la instrucción en EIR	ADDQ.W #2,SP MOVE.W (SP)+,D1

4. Tabla de registros del 68K

Registros	Función	Utilización
A0	Cálculo del PC Dirección del registro <a>	Fase de Fetch Como parámetro <a>
A1	Cálculo de índice de instrucción Dirección del registro 	Salto a la fase de ejecución Como parámetro
A2	Dirección del registro <c>	Como parámetro <c>
A3	Edirección M	Parámetro M
A4	Masking	Cálculo de registros
D0	Contenido del registro <a>	Como parametro <a>
D1	Indice del salto a instrucción Contenido del registro 	Salto a la fase de ejecución Como parámetro
D2	Almacenar el parámetro K Almacenar el parámetro M	Ejecución de instrucciones
D3	Calculo de saltos	BRC, BRZ
D4	Masking	GET_A, GET_B, GET_C GET_K, REGDEC
D5	Flag Z	Cálculo de Flags
D6	Flag N	
D7	Flag C	

En este apartado cabe comentar un par de puntos que creemos importantes. Aunque gran parte de las subrutinas se pudiesen hacer de libreria (usando la pila) creemos que la molestia de trabajar supera al beneficio de tener más registros para el usuario. De esta manera, hemos intenado usar la pila lo menos posible usando, por ejemplo, los registros **D5**, **D6** & **D7** únicamente para los flags. Haciendolo de esta manera hacemos que el programa sea mucho más simple de comprender y tenemos el efecto secundario de no tener que preocuparnos de saber si ese registro aya esta siendo usado por otra rutina .

5. Conjunto de pruebas

Para comprobar que nuestra emulación se ejecuta correctamente, hemos decidido probar con un código que utilice el resto de instrucciones que quedan por ejecutar en las dos pruebas proporcionadas en la práctica.

@PS-ECI	Ensamblador	Codificación	Hex
0:	LOA C,T0	1000 0000 0000 1100	800C
1:	LOA D,T1	1000 1000 0000 1101	880D
2:	AND T0,T1,T0	0101 1000 0000 1000	5808
3:	NOT T0	0110 0000 0000 0000	6000
4:	SUB T0,T0,T0	0101 0000 0000 0000	5000
5:	BRC 7	1101 0000 0000 0111	D007
6:	EXIT	0000 0000 0000 0000	0000
7:	STC #4,T0	0110 1000 0010 0000	6820
8:	ADD T0,T1,T1	0100 1000 0000 1001	4809
9:	COPY T1,B6	0100 0000 0000 1110	400E
A:	STOI (B6)	1011 0000 0000 0000	B000
B:	EXIT	0000 0000 0000 0000	0000
C:	000A	0000 0000 0000 1010	000A
D:	0002	0000 0000 0000 0002	0002

Este programa no tiene 'sentido' ya que solo queremos probar que las instrucciones que no se habían tratado antes funcionen. Tras la ejecución de este programa, en la dirección @\\$100C debería haber un 4

6. Conclusiones

Como conclusión podemos decir que esta práctica nos ha ayudado a terminar de asimilar conceptos previamente adquiridos en la asignatura, ya que a la hora de la práctica se ve la idea con mayor claridad. Además, la práctica nos ha servido de ayuda para estudiar de cara a los exámenes.

Como un plus, esta práctica nos ha permitido practicar LATEX y algo de programación. La documentación como se ha mencionado previamente, está hecha por completo en LATEX, además Rubén Palmer Pérez programó un pequeño proyecto en java para poder hacer programas de prueba de la máquina. El código fuente del programa y del documento de LATEX se puede encontrar [AQUI](#)

7. Código fuente

*

```
* Title      : PRAFIN20
* Written by : Mateu Segui Vives , Ruben Palmer Perez
* Date       : 20/05/2020
* Description: Emulador de la PS-ECI
```

```
ORG $1000
EPROG: DC.W $8810 ,$400A ,$E00D ,$688E ,$9000 ,$4003 ,$E00D ,$6804
        DC.W $6FFD,$48A4,$495B,$E00D,$C009,$4020,$A012,$0000
        DC.W $0004,$0003,$0000
EIR:   DC.W 0 ; registro de instrucción
EPC:   DC.W 0 ; econtrolador de programa
ET0:   DC.W 0 ; registro T0
ET1:   DC.W 0 ; registro T1
ER2:   DC.W 0 ; registro R2
ER3:   DC.W 0 ; registro R3
ER4:   DC.W 0 ; registro R4
ER5:   DC.W 0 ; registro R5
EB6:   DC.W 0 ; registro B6
EB7:   DC.W 0 ; registro B7
ESR:   DC.W 0 ; registro de estado (00000000 00000ZNC)
```

START:

```
CLR.W EPC
```

FETCH:

```
;--- IFETCH: INICIO FETCH
;*** En esta sección debéis introducir el código necesario para cargar
;*** en el EIR la siguiente instrucción a ejecutar, indicada por el EPC
;*** y dejar listo el EPC para que apunte a la siguiente instrucción
```

```
MOVE.W EPC, A0
ADD.W A0,A0
MOVE.W EPROG(A0) ,EIR
ADDQ.W #1,EPC
```

```
;--- FFETCH: FIN FETCH
```

```
;--- IBRDECOD: INICIO SALTO A DECOD
;*** En esta sección debéis 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
MOVE.W EIR,-(SP)
JSR DECOD
ADDQ.W #2,SP
MOVE.W (SP)+,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 EEXIT
JMP ECOPY
JMP EADD
JMP ESUB
JMP EAND
JMP ENOT
JMP ESTC
JMP ELOA
JMP ELOAI
JMP ESTO
JMP ESTOI
JMP EBRI
JMP EBRC
JMP EBRZ

;--- FBREXEC: FIN SALTO A FASE DE EJECUCION

;--- IEXEC: INICIO EJECUCION
;*** En esta seccion debeis implementar la ejecucion de cada instr.
```

EEXIT:

SIMHALT

ECOPY:

```
JSR GET.B  
JSR GET.C  
MOVE.W (A1),D1  
MOVE.W D1,(A2)  
JSR FLAGSNC  
BRA FETCH
```

EADD:

```
JSR GET.A  
JSR GET.B  
JSR GET.C  
MOVE.W (A0),D0  
MOVE.W (A1),D1  
ADD.W D0,D1  
JSR FLAGS  
MOVE.W D1,(A2)  
BRA FETCH
```

ESUB:

```
JSR GET.A  
JSR GET.B  
JSR GET.C  
MOVE.W (A0),D0  
MOVE.W (A1),D1  
NEG.W D1  
ADD.W D0,D1  
JSR FLAGS  
MOVE.W D1,(A2)  
BRA FETCH
```

EAND:

```
JSR GET.A  
JSR GET.B  
JSR GET.C  
MOVE.W (A0),D0  
MOVE.W (A1),D1  
AND.W D0,D1
```

```
JSR FLAGSN  
MOVE.W D1,( A2)  
BRA FETCH
```

ENOT:

```
JSR GET.C  
MOVE.W (A2),D0  
NOT.W D0  
JSR FLAGSN  
MOVE.W D0,( A2)  
BRA FETCH
```

ESTC:

```
JSR GET.K  
JSR GET.C  
MOVE.W D2,( A2)  
JSR FLAGSN  
BRA FETCH
```

ELOA:

```
JSR GET_I  
MOVE.W EIR, D2  
AND.W #$00FF, D2 ;Sacar M  
MOVE.W D2,A3  
ADD.W A3,A3  
MOVE.W EPROG(A3),( A4)  
JSR FLAGSN  
BRA FETCH
```

ELOAI:

```
JSR GET_J  
MOVE.W (A4),A4  
ADD.W A4,A4  
MOVE.W EPROG(A4),ET0  
JSR FLAGSN  
BRA FETCH
```

ESTO:

```
JSR GET_I  
MOVE.W EIR, D2
```

```
AND.W #$00FF , D2
MOVE.W D2,A3
ADD.W A3,A3
MOVE.W (A4) ,EPROG(A3)
BRA FETCH
```

ESTOI:

```
JSR GET_J
MOVE.W (A4) ,A4
ADD.W A4,A4
MOVE.W (ET0) ,EPROG(A4)
BRA FETCH
```

EBRI:

```
MOVE.W EIR , D2
AND.W #$00FF , D2
MOVE.W D2,EPC
BRA FETCH
```

EBRC:

```
MOVE.W ESR, D3
BTST #0, D3 ; Test flag C
BEQ FETCH
MOVE.W EIR, D2
AND.W #$00FF , D2
MOVE.W D2,EPC
BRA FETCH
```

EBRZ:

```
MOVE.W ESR, D3
BTST #2, D3 ; Test flag C
BEQ FETCH
MOVE.W EIR, D2
AND.W #$00FF , D2
MOVE.W D2,EPC
BRA FETCH
```

;--- FEXEC: FIN EJECUCION

;--- ISUBR: INICIO SUBRUTINAS

```
;*** Aqui debéis incluir las subrutinas que necesite vuestra solución
;*** SALVO DECOD, que va en la siguiente sección
```

```
;Con respecto a los flags hemos decidido hacer la siguiente acción:
;Una subrutina para calcular los flags Z & N y otra únicamente para
;el flag C. De esta manera para actualizar todos los flags actualizaremos
;zaremos Z&N y luego actualizaremos Z
```

FLAGSNC:

```
MOVE.W SR, D5
MOVE.W D5, D6
AND.W #4,D5 ;Flag Z
AND.W #8,D6 ;Flag N
LSR.W #2,D6
OR.W D5,D6
MOVE.W D6,ESR
RTS
```

FLAGS:

```
MOVE.W SR, D7
JSR FLAGSNC
AND.W #1,D7 ;Flag C
OR.W D7,D6 ;D6 <- ESR con Z & N
MOVE.W D6,ESR
RTS
```

GET_A: ;Pone en el registro A0 la dirección del operando a

```
MOVE.W #$1C0, D4
AND.W EIR, D4
LSR #6,D4
JSR REGDEC
MOVE.W A4,A0
RTS
```

GET_B: ;Pone en el registro A1 la dirección del operando b

```
MOVE.W #$38, D4
AND.W EIR, D4
LSR #3, D4
```

```
JSR REGDEC  
MOVE.W A4,A1  
RTS
```

GET_C: ;Pone en el registro A2 la dirección del operando c

```
MOVE.W #$7, D4  
AND.W EIR, D4  
JSR REGDEC  
MOVE.W A4,A2  
RTS
```

GET_K: ;Pone en el registro D2 el valor k con extensión

```
MOVE.W #$7F8, D4  
AND.W EIR, D4  
LSR #3, D4  
EXT.W D4  
MOVE.W D4,D2  
RTS
```

GET_M: ;Pone en el registro D2 el valor M

```
MOVE.W EIR,D2  
AND.W #$00FF,D2  
RTS
```

GET_I: ;Pone en el registro A4 la dirección de T1 o T0

```
BTST #11, EIR ;Mirar si es T1 o T0  
BEQ RT0
```

RT1:

```
LEA.L ET1,A4  
RTS
```

RT0:

```
LEA.L ET0,A4  
RTS
```

GET_J: ;Pone en el registro A4 la dirección de B6 o B7

```
BTST #11, EIR ;Mirar si es B6 o B7  
BEQ RB6
```

RB7:

```
LEA.L EB7,A4  
RTS
```

RB6:

```
LEA.L EB6,A4  
RTS
```

REGDEC: ; Decodifica los 3 bits menos significativos
; del registro D4 para determinar que
; registro emulado es

```
BTST.L #2,D4  
BEQ R0
```

R1:

```
BTST.L #1,D4  
BEQ R10
```

R11:

```
BTST.L #0,D4  
BEQ R110
```

R111: ;EB7

```
LEA.L EB7,A4  
RTS
```

R110: ;EB6

```
LEA.L EB6,A4  
RTS
```

R10:

```
BTST.L #0,D4  
BEQ R100
```

R101: ;ER5

```
LEA.L ER5,A4  
RTS
```

R100: ;ER4

```
LEA.L ER4,A4  
RTS
```

R0:

```
BTST.L #1,D4  
BEQ R00
```

R01:

```

        BTST.L #0,D4
        BEQ R010
R011: ;ER3
        LEA.L ER3,A4
        RTS
R010: ;ER2
        LEA.L ER2,A4
        RTS
R00: 
        BTST.L #0,D4
        BEQ R000
R001: ;ET1
        LEA.L ET1,A4
        RTS
R000: ;ET0
        LEA.L ET0,A4
        RTS

;---- FSUBR: FIN SUBRUTINAS

;---- IDECOD: INICIO DECOD
;*** Tras la etiqueta DECOD, debéis implementar la subrutina de
;*** decodificacion , que deberá ser de librería , siguiendo la interfaz
;*** especificada en el enunciado

DECOD:
        BTST.B #7,4(SP)
        BEQ IN0
IN1:
        BTST.B #6,4(SP)
        BEQ IN10
IN11:
        BTST.B #5,4(SP)
        BEQ IN110
IN110: ;BRZ
        MOVE.W #13,6(SP)
        RTS
IN110:
        BTST.B #4,4(SP)
        BEQ IN1100

```

```
IN1101: ;BRC
        MOVE.W #12,6(SP)
        RTS
IN1100: ;BRI
        MOVE.W #11,6(SP)
        RTS
IN10:
        BTST.B #5,4(SP)
        BEQ IN100
IN101:
        BTST.B #4,4(SP)
        BEQ IN1010
IN1011: ;STOI
        MOVE.W #10,6(SP)
        RTS
IN1010: ;STO
        MOVE.W #9,6(SP)
        RTS
IN100:
        BTST.B #4,4(SP)
        BEQ IN1000
IN1001: ;LOAI
        MOVE.W #8,6(SP)
        RTS
IN1000: ;LOA
        MOVE.W #7,6(SP)
        RTS
IN0:
        BTST.B #6,4(SP)
        BEQ IN00
IN01:
        BTST.B #5,4(SP)
        BEQ IN010
IN0110:
        BTST.B #3,4(SP)
        BEQ IN01100
IN01101: ;STC
        MOVE.W #6,6(SP)
        RTS
IN01100: ;NOT
```

```
MOVE.W #5,6(SP)
RTS
IN010:
BTST.B #4,4(SP)
BEQ IN0100
IN0101:
BTST.B #3,4(SP)
BEQ IN01010
IN01011: ;AND
MOVE.W #4,6(SP)
RTS
IN01010: ;SUB
MOVE.W #3,6(SP)
RTS
IN0100:
BTST.B #3,4(SP)
BEQ IN01000
IN01001: ;ADD
MOVE.W #2,6(SP)
RTS
IN01000: ;COPY
MOVE.W #1,6(SP)
RTS
IN00: ;EXIT
MOVE.W #0,6(SP)
RTS

;--- FDECOD: FIN DECOD

END START
```