Estructura de computadores

--Curso 2019-2020--

Práctica Final

Profesores asignatura:

Clases teóricas: ***Proenza Arenas, Julián***

Clases prácticas: ***García Fidalgo, Emilio***

Grupo: 03x01

Integrantes del grupo:

NOMBRE	DNI	CORREO ELECTRÓNICO
María Isabel Crespí Valero	45187406L	maribelcr55@gmail.com
Juan Mesquida Arenas	43477937W	juanmesqui@gmail.com

Universitat
de les Illes Balears

ÍNDICE

INTRODUCCIÓN:

A lo largo del curso se han estado enseñando ideas y conceptos que han servido para poder llevar a cabo la siguiente práctica, cuyo objetivo a realizar es crear un emulador de la máquina PS-ECI (Procesador Simple - Estructura de Computadores I).

Esta máquina a emular cuenta con 8 registros diferentes.

- T0 y T1, que se utilizan como interfaz con la memoria, además de poder ser empleados en operaciones de tipo ALU como operando;
- 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;
- B6 y B7, registros de direcciones, que se utilizan en algunas instrucciones para acceder a memoria usando un modo de direccionamiento indirecto.

Además, tiene 14 instrucciones, cada una de ellas con su respectiva función.

Id	Mnemónico	Codificación	Acción	Flags
0	EXIT	00xxxxxxxxxxxxx	Detiene la máquina	n.s.a.
1	COPY Xb,Xc	01000xxxxxbbbccc	Xc ← [Xb]	C = n.s.a., Z y N = s.v.Xc
2	ADD Xa,Xb,Xc	01001xxaaabbbccc	Xc ← [Xa] + [Xb]	C, Z y N = s.v.r.
3	SUB Xa,Xb,Xc	01010xxaaabbbccc	Xc ← [Xa] - [Xb]	C, Z y N = s.v.r.
4	AND Xa,Xb,Xc	01011xxaaabbbccc	Xc ← [Xa] and [Xb]	C = n.s.a., Z y N = s.v.r
5	NOT Xc	01100xxxxxxxxccc	Xc ← not [Xc]	C = n.s.a., Z y N = s.v.r
6	STC #k,Xc	01101kkkkkkkkccc	Xc ← k (Ext. signo)	C = n.s.a., Z y N = s.v.Xc
7	LOA M,Ti	1000ixxxmmmmmmmm	Ti ← [M]	C = n.s.a., Z y N = s.v.Ti
8	LOAI (Bj)	1001jxxxxxxxxxx	T0 ← [[Bj]]	C = n.s.a., Z y N = s.v.T0
9	STO Ti,M	1010ixxxmmmmmmmm	M ← [Ti]	n.s.a.
10	STOI (Bj)	1011jxxxxxxxxxx	[Bj] ← [T0]	n.s.a.
11	BRI M	1100xxxxmmmmmmmm	PC ← M	n.s.a.
12	BRC M	1101xxxxmmmmmmmm	Si C = 1, PC ← M	n.s.a.
13	BRZ M	1110xxxxmmmmmmmm	Si Z = 1, PC ← M	n.s.a.

Como podemos observar algunas de las funciones a implementar actualizarán los flags C, Z y N, algunas solo Z y N y otras funciones que no actualizarán ningún flag.

Todos los registros e instrucciones de esta máquina son de 16 bits.

Por lo tanto, la idea en esta práctica es poder implementar esta máquina mediante el emulador 68k usando sus instrucciones y registros.

FASES A IMPLEMENTAR:

Para poder realizar esta emulación se ha dividido en 3 fases:

La fase fetch, la fase de decodificación y por último la fase de ejecución. También utilizaremos varias secciones dadas en el fichero *PRAFIN20.X68* :

- `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.

Fase fetch:

Esta fase consiste en ir a coger una instrucción a la memoria y traerla para poder empezar a utilizarla.

En el caso de la emulación de la máquina, lo que se ha hecho es ir mirando la dirección de memoria del vector EPROG a través de la instrucción LEA que a cada nueva iteración que se haga en el fetch, irá aumentando para recorrer todo el vector. A su vez, el contador del programa EPC irá aumentando de uno en uno ya que es así cómo funcionarán las posiciones de memoria del programa en el emulador.

```
MOVE.W EPC,D0
MULS.W #2,D0      ; Multiplicamos por 2 debido a que es un vector de Words
LEA EPROG,A0      ; Cargamos la dirección de memoria de EPROG en A0
ADD.W D0,A0       ;Siguiente einstrucción = EPROG + 2*EPC
MOVE.W (A0),D0
MOVE.W D0,EIR     ; y meteremos la instrucción pertinente en EIR
ADDQ.W #1,EPC     ; A medida que avancemos en el programa emulado,
                  ; el contador EPC irá aumentando de 1 en 1
```

A partir de este punto, se prepara la pila del 68k para poder ejecutar la subrutina de librería para la decodificación de las instrucciones. Una vez de vuelta tras la subrutina, se dejará la pila como estaba y ya se tendrá el índice de la instrucción elegida.

```
; ESCRIBID VUESTRO CODIGO AQUI

SUBQ.W #2,SP            ; Reservamos un espacio vacío donde meteremos el índice de instrucción
MOVE.W EIR,-(SP)        ; También reservaremos un espacio para el EIR
JSR DECOD               ; Salto a la subrutina
MOVE.W 2(SP),D1           ; Metemos el resultado en el registro D1
ADDQ.W #4,SP            ; Vaciamos la pila
```

Fase decodificación:

En esta fase se ha tenido que utilizar una subrutina de librería. La pila ha sido muy útil ya que fue un apoyo a la hora de guardar registros sin que se perjudicase la libertad a la hora de trabajar dentro del programa principal. Por este motivo, se pudo volver a utilizar el registro utilizado dentro de la subrutina DECOD sin miedo a perder ningún dato importante.

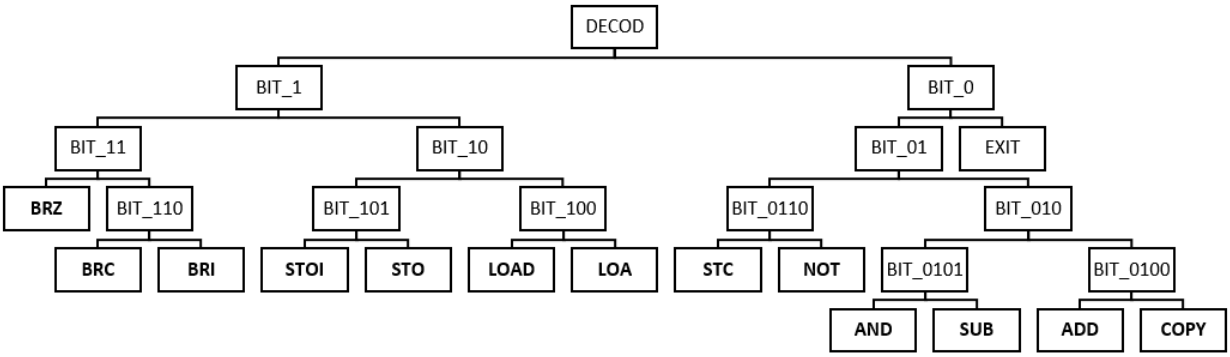
La decodificación consiste en coger la instrucción guardada en el vector EPROG y a partir de ahí ir mirando qué bits son los que componen la parte más significativa para poder deducir cuál es la instrucción.

```
DECOD:
    ; ESCRIBID VUESTRO CODIGO AQUI

    MOVE.W D0,-(SP)   ; Hacemos un PUSH del registro que vamos a usar en la subrutina
    MOVE.W 6(SP),D0   ; Estamos metiendo en D0 el contenido del EIR
                      ; Empezamos a mirar qué instrucción es empezando por el bit número 15
    BTST.L #15,D0      ; Método de la máscara para comprobar si el bit 15
    BNE BIT_1         ; es 1 o 0
    BRA BIT_0
```

El planteamiento inicial se ha realizado a través de un árbol de decodificación mediante etiquetas donde se ha ido mirando bit a bit desde el más significativo hasta que se obtuviese la instrucción correspondiente ya que de esta manera era mucho más fácil visualizar y entender cómo se hacía esta parte del programa. Para ello hemos usado la instrucción BTST y mediante BNE y BEQ , se saltó a la etiqueta correspondiente de nuestro árbol.

Así se ha sacado paso a paso cómo llegar a la instrucción pertinente:



Finalmente, al encontrar la instrucción que se correspondía con la decodificación de sus bits más significativos, la subrutina se ha encargado de cargar el número representativo de dicha instrucción en el espacio reservado previamente, para que después al volver de la subrutina se pudiese recuperar ese valor y guardarlo.

Fase ejecución:

Una vez hecho el fetch , lo que quedaba por hacer era programar el funcionamiento de las instrucciones del PS-ECI y la actualización de los flags.

Id	Mnemónico	Codificación	Acción	Flags
0	EXIT	00xxxxxxxxxxxxx	Detiene la máquina	n.s.a.
1	COPY Xb,Xc	01000xxxxxbbbccc	Xc ← [Xb]	C = n.s.a., Z y N = s.v.Xc
2	ADD Xa,Xb,Xc	01001xxaaabbbccc	Xc ← [Xa] + [Xb]	C, Z y N = s.v.r.
3	SUB Xa,Xb,Xc	01010xxaaabbbccc	Xc ← [Xa] - [Xb]	C, Z y N = s.v.r.
4	AND Xa,Xb,Xc	01011xxaaabbbccc	Xc ← [Xa] and [Xb]	C = n.s.a., Z y N = s.v.r
5	NOT Xc	01100xxxxxxxxccc	Xc ← not [Xc]	C = n.s.a., Z y N = s.v.r
6	STC #k,Xc	01101kkkkkkkkccc	Xc ← k (Ext. signo)	C = n.s.a., Z y N = s.v.Xc
7	LOA M,Ti	1000ixxxmmmmmmmm	Ti ← [M]	C = n.s.a., Z y N = s.v.Ti
8	LOAI (Bj)	1001jxxxxxxxxxxx	T0 ← [[Bj]]	C = n.s.a., Z y N = s.v.T0
9	STO Ti,M	1010ixxxmmmmmmmm	M ← [Ti]	n.s.a.
10	STOI (Bj)	1011jxxxxxxxxxxx	[Bj] ← [T0]	n.s.a.
11	BRI M	1100xxxxmmmmmmmm	PC ← M	n.s.a.
12	BRC M	1101xxxxmmmmmmmm	Si C = 1, PC ← M	n.s.a.
13	BRZ M	1110xxxxmmmmmmmm	Si Z = 1, PC ← M	n.s.a.

Muchas de las instrucciones nuevas ya tenían una similar en el 68k, así que en ese aspecto no era complicado pensar cómo llevar a cabo esta parte. Sin embargo, también se tenía que tener en cuenta que los registros eran diferentes y por lo tanto se tenían que codificar los operandos destino y fuente de forma particular. Además de los registros Ti y Bj, que tenían su particularidad.

Había que tener en cuenta el apartado de los flags ya que no tenían por qué coincidir con las posiciones que se nos había pedido que guardásemos los nuevos flags del emulador. Había que ir con cuidado de guardarlos correctamente, ya que si no lo hacíamos justo después de la acción correspondiente, se estarían guardando unos flags probablemente erróneos.

Por último, la codificación de las constantes podía ser un poco liosa a causa de la extensión de signo al ser un número negativo, pero eso tenía fácil solución con la instrucción EXT y la máscara que se hacía con AND.

Se han utilizado varias subrutinas ya que éstas facilitaban el trabajo al utilizar etiquetas más representativas de algunos conjuntos de instrucciones; aumentaban la legibilidad, en resumen. Además de ahorrar en código, pues de esta forma se evita la redundancia. Otro motivo más es que con el uso de subrutinas queda un programa más limpio y estructurado.

TABLA SUBRUTINAS

Nombre	Tipo	Entrada	Salida	Función
DECOD	**Librería**	**EIR**	**D0**	**Decodificar las instrucciones**
CZN	**Usuario**	**D4**	**D7**	**Actualizar flag C,Z y N**
ZN	**Usuario**	**D4**	**D7**	**Actualizar flags N y Z**
REGISTRO	**Usuario**	**D2, D0**	**D1**	**Obtener operando fuente**
REGISTRO_DESTINO	**Usuario**	**D0**	**---**	**Introducir en operando destino**
M	**Usuario**	**D0**	**D5**	**Obtener M,s**
T	**Usuario**	**D0**	**D2**	**Obtener operando fuente T**
T_DEST	**Usuario**	**D0**	**D6**	**Obtener operando destino T**
B	**Usuario**	**D0**	**D2**	**Obtener operando fuente B**
CAMBIO SIGNO	**Usuario**	**D1**	**D1**	**Cambiar signo en C2 a un número**
BRANCH_PC	**Usuario**	**D5**	**---**	**Hacer un branch dependiendo de M**

Como se puede ver, se ha utilizado una subrutina de librería (DECOD) para la parte de decodificación. Para el resto de subrutinas se ha optado por realizarlas de usuario ya que de esta manera no se tenía que tener en cuenta constantemente la pila y era más sencillo. Estas últimas han llevado a cabo la parte de ejecución (CZN, ZN, REGISTRO, REGISTRO_DESTINO, M, T, T_DEST, B CAMBIO SIGNO Y BRANCH_PC).

TABLA REGISTROS 68K

Nombre	Función
A0	**-Guardar dirección de memoria de EPROG**
A1	**- Apoyo para las instrucciones STOI y LOAI**
A2	**- Apoyo para las instrucciones STO y LOA**
SP	**-Stack pointer de la pila utilizado para la subrutina DECOD**
D0	**-Guardar instrucción correspondiente**
D1	**- Guardar registros fuente**
D2	**- Registro utilizado como parámetro para las subrutinas de decodificación de los registros**
D3	**- Registro destino**
D4	**- Guardar flags 68k**
D5	**- Guardar constantes M y K**
D6	**- Registros Ti y Bj**
D7	**- Flags del emulador**

Al mirar la tabla vemos que solo se han utilizado 4 registros de direcciones (A0, A1, A2 y SP) y los 8 registros de datos(D0-D7), algunos incluso se han reutilizado para varias acciones. Los registros de datos se han utilizado para la parte de ejecución. Los registros de direcciones han variado entre fetch y ejecución.

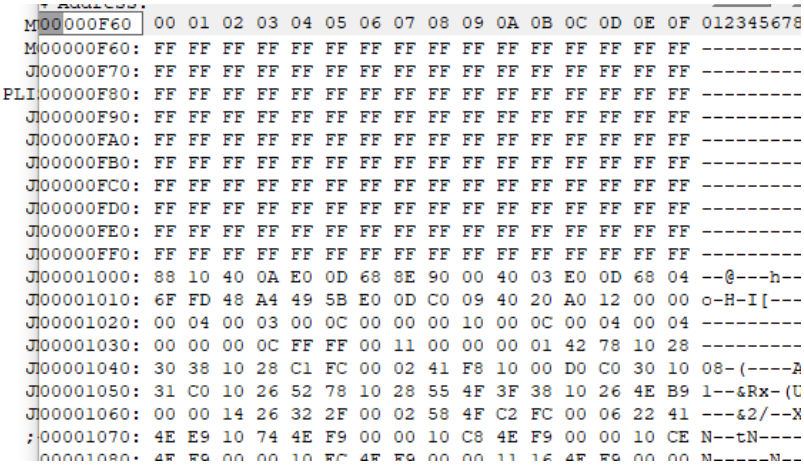
CONJUNTO DE PRUEBAS

```
    ORG $1000
EPROG: DC.W $8007,$4002,$C005,$5092,$0000,$4892,$0000,$0001
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
ER2:   DC.W 0 ;eregistro R2
ER3:   DC.W 0 ;eregistro R3
ER4:   DC.W 0 ;eregistro R4
ER5:   DC.W 0 ;eregistro R5
EB6:   DC.W 0 ;eregistro B6
EB7:   DC.W 0 ;eregistro B7
ESR:   DC.W 0 ;eregistro de estado (00000000 00000ZNC)
```

```
00000FA0: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E
00000FA0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000FB0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000FC0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000FD0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000FE0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000FF0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00001000: 80 07 40 02 C0 05 50 92 00 00 48 92 00 00 00
00001010: 00 00 00 07 00 01 00 00 00 02 00 00 00 00 00
00001020: 00 00 00 00 00 02 42 78 10 12 30 38 10 12 C1
00001030: 00 02 41 F8 10 00 D0 C0 30 10 31 C0 10 10 52
00001040: 10 12 55 4F 3F 38 10 10 4E B9 00 00 13 FE 32
00001050: 00 02 58 4F C2 FC 00 06 22 41 4E E9 10 5E 4E
00001060: 00 00 10 B2 4E F9 00 00 10 B8 4E F9 00 00 10
00001070: 4E F9 00 00 11 04 4E F9 00 00 11 34 4E F9 00
```

Como se puede ver en @1018 tenemos 0002hex (2 dec), que es efectivamente el resultado que tenía que dar al ejecutar dichas instrucciones.

```
    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 ;eregistro de instruccion
EPC:   DC.W 0 ;econtador de programa
ET0:   DC.W 0 ;eregistro T0
ET1:   DC.W 0 ;eregistro T1
ER2:   DC.W 0 ;eregistro R2
ER3:   DC.W 0 ;eregistro R3
ER4:   DC.W 0 ;eregistro R4
ER5:   DC.W 0 ;eregistro R5
EB6:   DC.W 0 ;eregistro B6
EB7:   DC.W 0 ;eregistro B7
ESR:   DC.W 0 ;eregistro de estado (00000000 00000ZNC)
```



Como se puede observar en @1025 aparece el valor 0C, por lo que nuestro emulador ha realizado las instrucciones satisfactoriamente.

CONCLUSIONES

Esta práctica nos ha ayudado a entender en gran medida muchos de los conceptos de la asignatura ya que a la hora de realizar la emulación de la máquina propuesta y sus funciones hemos tenido que poner a prueba nuestros conocimientos sobre el tema y hemos desarrollado un programa que utilizaba gran cantidad de dichos conceptos explicados estos últimos meses de curso de la asignatura.

Además de eso estamos muy satisfechos ya que su elaboración nos ha permitido entenderlos con mayor precisión mediante los videos y pdf explicativos y el trabajo puesto de nuestra parte, ya que muchos de ellos no habían quedado tan claros.

También hemos podido organizarnos y ordenarnos mucho mejor en cuanto a la forma de realizar el código en el emulador 68k para poder resolver el programa de una manera más eficiente, eficaz y que facilite la comprensión del usuario que vaya a interactuar con dicho programa.

Por último, queríamos agradecer a los profesores de la asignatura por su gran trabajo y esfuerzo durante el periodo de excepcionalidad y su gran adaptación a esta situación mediante videos informativos de la asignatura y su gran comunicación y atención.

Juan y Maribel.