Estructura de computadores

Ingeniería informática

Miguel Vidal Coll – 43225456 S vidalcollmiguel@gmail.com

Grupo 03M01

Los 4 puntos que trataremos en este informe son los siguientes:

- Declaración de variables
- Fase de fetch
- Decodificación (subrutina de librería)
- Subrutinas de usuario
- Código fuente del programa

Declaración de variables:

Registros de datos

D0: Contiene el registro de instrucción o el registro de estatus, se utiliza para las subrutinas.
D1: Registro auxiliar al mover contenidos de los registros.
D2: Se utiliza para actualizar el contador de PC de la máquina emulada
D3: Registro auxiliar, tanto en algunas instrucciones como en las subrutinas de actualización de flags.
D4: Registro de datos auxiliar en las operaciones que utilizan la ALU.
D5: Igual que D4.
D6: Registro auxiliar que se usa en SET, INC y en la subrutina FINDC.
D7: No se utiliza.

Registros de direcciones

A0: Propósito general, se utiliza tanto en instrucciones como en subrutinas como en el fetch.
A1: Se utiliza para saltar a la instrucción correspondiente dentro del JMPLIST.
A2: Se utiliza en la subrutina FINDJ para devolver la dirección del registro RJ.
A3: Se utiliza en la subrutina FINDAB y devuelve le dirección del registro fuente o destino.
A4: Registro auxiliar que utilizaremos para guardar un registro en la instrucción TRA.
A5: Registro auxiliar para guardar direcciones en las instrucciones SET e INCREASE.
A6: No se usa.
A7: Registro que se utiliza para la pila.

Fase de fetch:

La función de la fase de fetch es leer la instrucción a la que apunta el EPC dentro del vector de palabras que equivalen a las instrucciones a ejecutar.
Para ello, movemos el EPC a un registro auxiliar y multiplicamos por dos para obtener el desplazamiento real de la maquina que emulamos, ya qye el valor del PC se incrementa en 1 cada vez, pero el desplazamiento del EPC es el doble que el del PC.
Una vez hecho, movemos al registro EIR la instrucción que se encuentra en la posición donde apunta el EPC dentro del vector EPROG. Una vez que hemos guardado la instrucción, aumentamos en uno el contador de EPC así como lo haría la máquina elemental.
Lo siguiente que hacemos es reservar dos words en la pila: uno para guardar el resultado de la decodificación y otro para insertar el registro de instrucción. Una vez que dejamos la pila preparada, saltamos a la subrutina de librería donde decodificaremos la instrucción.

Decodificación (subrutina de librería)

Para la decodificación usaremos una subrutina de librería, ya que es un conjunto de operaciones iguales que se repite a lo largo de todas las instrucciones, por tanto para ahorrar código nos conviene crear una subrutina, y además será de librería para que cualquier usuario pueda usarla sin preocuparse por los valores que queden en los registros ya que usando la pila conseguimos que los valores que había en los registros antes de ejecutarse sigan intactos.

Lo primero que hacemos al llamar a la subrutina es recuperar el valor de EIR y el de la posición que habíamos guardado para el resultado. Una vez hecho esto, empezaremos a decodificar bit a bit la instrucción, encadenando operaciones de bit test desde el mas significativo (bit numero 15) hasta el que sea necesario para identificar la instrucción.
Una vez que llegamos a decodificar una cadena de bits que nos permite diferenciar la instrucción, metemos un numero del 0 al 12 en la pila, correspondiente al numero de instrucción que hemos decodificado. Después de guardar el valor de salto de la instrucción, ejecutamos una instrucción que saca el valor de D0 de la pila para dejarla como estaba antes de ejecutar la última instrucción, el retorno de subrutina.
Al llegar de la subrutina, añadimos un dos a la dirección de la pila para dejarla tal como estaba antes de ser llamada, y sacamos la dirección de salto que hemos obtenido para guardarla en el registro D1.
Una vez guardado, lo multiplicamos por 6 y lo guardamos como una dirección para saltar dentro del JMPLIST.

Subrutinas de usuario:

En este apartado explicaré las subrutinas de usuario que he utilizado para facilitar la ejecución, ya que muchas de estas acciones se repiten en varias instrucciones. He utilizado dos tipos de subrutinas:

-Subrutinas para encontrar un valor.
-Subrutinas para actualizar los flags.

<u>Subrutinas para encontrar un valor.</u>

Son la clase de subrutinas que he utilizado para sacar el valor de una constante, una dirección de memoria o el índice de un registro. He utilizado dos técnicas diferentes: bit tests y mascaras. En los casos en los que necesitamos encontrar el indice de un registro, he utilizado bit tests de forma similar al apartado de decodificación. Esas subrutinas son las siguientes:

FINDJ: subrutina para encontrar el indice del registro ET que debemos utilizar. Como solo es un bit, miramos si es 1 o 0 y devolvemos el valor del registro ET1 o ET0 respectivamente.

FINDAB: subrutina para encontrar el indice de registro. En este caso, hacemos un bit test encadenando 3 bits y según la combinación de estos 3 devolvemos el valor del registro que corresponde según la tabla del enunciado.

Por otra parte, he utilizado dos subrutinas mas pero estas utilizan mascaras para sacar el valor de una constante o una dirección de memoria.

FINDM: subrutina para encontrar el valor de la dirección de memoria m (8 bits). Para ello aplicamos una AND entre 00FF y el valor de EIR, para asi obtener los 8 primeros bits y los demás estarán a 0

FINDC: subrutina para encontrar el valor de la constante de 8 bits. En este caso, antes de ejecutar la mascara, tenemos que mover la instrucción 3 bits a la derecha, ya que los tres primeros bits son el indice de un registro y no queremos que nos afecte en el valor de la constante. Una vez desplazados con la instrucción LSR, aplicamos la misma mascara que en la subrutina anterior. Al tener el valor de la constante, tenemos que mirar su signo: si es positiva, ejecutamos otra mascara con una OR entre 0000 y el valor que hemos obtenido para extender el signo positivo; en caso contrario, aplicamos una mascara con una OR entre FF00 y el valor que hemos obtenido para extender el signo negativo. Una vez hecho esto, devolvemos el resultado al programa principal.

Subrutinas para actualizar los flags.

Las tres subrutinas restantes son las que se utilizan para actualizar el valor de los flags Z N y C. Los tres funcionan de forma similar pero cada uno mira su condición de forma diferente:
-El flag Z compara el resultado obtenido con 0000 para saber si el resultado es 0, en caso afirmativo pone a 1 el valor del bit 2 del ESR y en caso negativo lo pone a 0.
-El flag N hace un bit test del bit numero 15 para saber si el numero es negativo, en caso afirmativo pone a 1 el valor del bit 1 del ESR y en caso negativo lo pone a 0.
-El flag C hace un bit test del bit numero 16 para saber si hay acarreo de salida, en caso afirmativo pone a 1 el valor del bit 0 del ESR y en caso negativo lo pone a 0.

Código fuente del programa:

```
*-----------------------------------------------------------
* Title      : PRAFIN17
* Written by : Miguel Vidal Coll
* Date       : 28/05/2017
* Description: Emulador de la F16M
*-----------------------------------------------------------
      ORG    $1000
EPROG: DC.W $A00E, $500B, $C002, $A20F, $500B, $C00B, $E804, $C814, $F7FB
      DC.W   $500B, $4007, $C020, $8010, $0000, $0004, $0003, $0000
EIR:   DC.W 0 ;eregistro de instrucción
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
ESR:   DC.W 0 ;eregistro de estado (00000000 00000ZNC)

START:                  ; first instruction of program


FETCH

   MOVE.W EPC,D2          ; movemos el contenido del EPC a un registro
   MULS #2,D2             ; multiplicamos por 2 para obtener el
   MOVE.L D2,A0           ; desplazamiento real
   MOVE.W EPROG(A0),EIR   ; movemos el codigo de la instruccion a EIR
   ADDQ.W #1,EPC          ; después del fetch actualizamos el EPC

   MOVE.W #0,-(A7)        ; reservamos un word para guardar el resultado
   MOVE.W EIR,-(A7)       ; de la decodificacion y otro mas para meter el
   JSR DECOD              ; EIR y a continuación saltamos a la subrutina

   ADDQ.W #2,A7           ; dejamos la pila como estaba
   MOVE.W (A7)+,D1        ; sacamos el resultado de la decodificacion

   MULU #6,D1             ; lo multiplicamos por 6 y lo guardamos como una
   MOVEA.L D1,A1          ; dirección para saltar dentro del JMPLIST
   JMP JMPLIST(A1)


JMPLIST:
   JMP EHLT
   JMP EJMI
   JMP EJMZ
   JMP EJMN
   JMP ESTO
   JMP ELOA
   JMP ETRA
   JMP EADD
   JMP ESUB
   JMP ECMP
   JMP ENAN
   JMP ESET
   JMP EINC

EHLT:
   BRA FIN                ; saltamos al final del programa

EJMI:
   JSR FINDM              ; buscamos el valor de la direccion de memoria m
   MOVE.W A0,EPC          ; metemos la dirección de salto en el EPC
   MOVE.W #0,A0           ; limpiamos el registro que hemos usado
   BRA FETCH


EJMZ:
   MOVE.W ESR,D0          ; movemos el ESR a un registro auxiliar
   BTST #2,D0             ; miramos el valor del bit Z
```

```
   BNE SISALTO          ; si Z=1 vamos a la etiqueta donde saltaremos
   BRA FETCH            ; si Z=0 volvemos a la fase de fetch


EJMN:
   MOVE.W ESR,D0        ; movemos el ESR a un registro auxiliar
   BTST #1,D0           ; miramos el valor del bit N
   BNE SISALTO          ; si N=1 vamos a la etiqueta donde saltaremos
   BRA FETCH            ; si Z=0 volvemos a la fase de fetch


SISALTO:
   JSR FINDM            ; saltamos a la subrutina donde obtendremos m
   MOVE.W A0,EPC        ; movemos la direccion de memoria al EPC
   CLR.L D0             ; limpiamos los registros que hemos usado
   MOVE.W #0,A0
   CLR.L D0
   BRA FETCH


ESTO:
   JSR FINDM            ; buscamos el valor de la direccion de memoria m
   JSR FINDJ            ; determinamos si tenemos que usar T0 o T1
   MOVE.W A0,D3         ; movemos la direccion a un registro auxiliar
   MULU #2,D3           ; calculamos el desplazamiento
   MOVE.W D3,A0         ; guardamos la nueva direccion y movemos el
   MOVE.W (A2),EPROG(A0)  ; contenido del registro ET que toca a la posicion
   BRA FETCH            ; de EPROG que hemos calculado

ELOA:
   JSR FINDM            ; obtenemos el valor de m y el del registro ET
   JSR FINDJ
   MOVE.W A0,D3         ; movemos la direccion a un registro auxiliar
   MULU #2,D3           ; calculamos el desplazamiento
   MOVE.W D3,A0         ; guardamos la nueva direccion y movemos el
   MOVE.W EPROG(A0),(A2)  ; contenido de la posicion de EPROG al registro
                        ; ET0 o ET1 segun la instruccion
   JSR FLAG_Z           ; actualizamos los flags mediante subrutinas
   JSR FLAG_N
   MOVE.L #0,A0         ; limpiamos los registros que hemos usado
   MOVE.L #0,A2
   CLR.L D0
   CLR.L D3
   BRA FETCH


ETRA:
   MOVE.W EIR,D0        ; movemos el EIR a un reigstro auxiliar
   JSR FINDAB           ; encontramos el registro que debemos usar mirando
   MOVE.W A3,A4         ; los 3 ultimos bits, guardamos el resultado en
   LSR #3,D0            ; otro registro y desplazamos 3 bits para buscar
   JSR FINDAB           ; el segundo registro que debemos usar
   MOVE.W (A3),D1       ; movemos el contenido del segundo registro dentro
   MOVE.W D1,(A4)       ; del primer registro
```

```
   MOVE.W #0,A3          ; limpiamos los registros que hemos utilizado
   MOVE.W #0,A4
   CLR.L D0
   BRA FETCH


EADD:
   MOVE.W EIR,D0
   JSR FINDAB            ; buscamos el primer registro a usar
   MOVE.W (A3),D4        ; lo guardamos en un registro
   LSR #3,D0             ; desplazamos 3 bits para leer el siguiente
   JSR FINDAB
   MOVE.W (A3),D5        ; lo guardamos en otro registro
   ADD.W D4,D5           ; sumamos el contenido de los dos registros
   MOVE.W D5, (A3)       ; guardamos el resultado en el registro que toca
   MOVE.W D5,D0          ; actualizamos los flags mediante el registro
   JSR FLAG_C            ; correspondiente
   JSR FLAG_N
   JSR FLAG_Z
   MOVE.L #0,A3          ; limpiamos los registros que hemos usado
   CLR.L D0
   CLR.L D4
   CLR.L D5
   BRA FETCH


ESUB:
   MOVE.W EIR,D0         ; ejecutamos los mismos pasos que en la
   JSR FINDAB            ; instruccion EADD pero en este caso restamos
   MOVE.W (A3),D4        ; el contenido de los dos registros
   LSR #3,D0
   JSR FINDAB
   MOVE.W (A3),D5
   SUB.W D4,D5
   MOVE.W D5,(A3)
   MOVE.W D5,D0
   JSR FLAG_C
   JSR FLAG_N
   JSR FLAG_Z
   MOVE.L #0,A3          ; limpiamos los registros que hemos usado
   CLR.L D0
   CLR.L D4
   CLR.L D5
   BRA FETCH


ECMP:
   MOVE.W EIR,D0         ; hacemos lo mismo que en la instrucción ESUB
   JSR FINDAB            ; pero en este caso no guardamos el resultado
   MOVE.W (A3),D4        ; solo actualizamos los flags
   LSR #3,D0
   JSR FINDAB
   MOVE.W (A3),D5
   SUB.W D4,D5
```

```
   MOVE.W D5,D0
   JSR FLAG_C
   JSR FLAG_N
   JSR FLAG_Z
   MOVE.L #0,A3          ; limpiamos los registros que hemos usado
   CLR.L D0
   CLR.L D4
   CLR.L D5
   BRA FETCH


ENAN:
   MOVE.W EIR,D0         ; obtenemos los registros mediante las mismas
   JSR FINDAB            ; instrucciones que las tres anteriores
   MOVE.W (A3),D4
   LSR #3,D0
   JSR FINDAB
   MOVE.W (A3),D5
   AND.W D4,D5           ; ejecutamos una AND bit a bit
   NOT.W D5              ; negamos el resultado para obtener una NAND
   MOVE.W D5,(A3)
   MOVE.W D5,D0          ; actualizamos los flags mediante el registro
   JSR FLAG_Z            ; correspondiente
   JSR FLAG_N
   MOVE.W #0,A3          ; limpiamos los registros que hemos usado
   CLR.L D0
   CLR.L D4
   CLR.L D5
   BRA FETCH


ESET:
   MOVE.W EIR,D0
   JSR FINDAB            ; buscamos el valor del registro a usar
   JSR FINDC             ; buscamos el valor de la cosntante c
   MOVE.W D6,(A5)        ; movemos el valor de la constante al registro
   MOVE.W (A5),D0        ; actualizamos los flags mediante el registro
   JSR FLAG_Z            ; correspondinte
   JSR FLAG_N
   MOVE.W #0,A5          ; limpiamos los registros que hemos usado
   CLR.L D6
   CLR.L D0
   BRA FETCH


EINC:
   MOVE.W EIR,D0
   JSR FINDAB            ; buscamos el valor del registro a usar
   JSR FINDC             ; buscamos el valor de la constante c
   ADD.W (A3),D6         ; le sumamos la constante al valor del registro
   MOVE.W D6,(A3)
   MOVE.W (A3),D0        ; actualizamos los flags mediante el registro
   JSR FLAG_C            ; correspondiente
   JSR FLAG_Z
```

```
   JSR FLAG_N
   MOVE.W #0,A5          ; limpiamos los registros que hemos usado
   CLR.L D6
   CLR.L D0
   BRA FETCH


************************** SUBRUTINAS USU **************************

FINDM:
   MOVE.W EIR,D0          ; movemos el EIR a un registro auxiliar
   AND #$00FF,D0          ; aplicamos una mascara para sacar la direccion
   MOVE.W D0,A0           ; la movemos a un registro de direcciones
   CLR.L D0             ; limpiamos el registro que hemos utilizado
   RTS


FINDJ:
   MOVE.W EIR,D0          ; movemos el EIR a un registro auxiliar
   BTST #9,D0            ; miramos el valor del bit 9
   BEQ RT0              ; si es 0 devolveremos la direccion de T0
   BNE RT1              ; si es 1 devolveremos la direccion de T1
RT0
   MOVE.W #ET0, A2         ; movemos el valor de ET0 a un registro de
   CLR.L D0             ; direcciones y limpiamos el registro
   RTS
RT1
   MOVE.W #ET1, A2         ; hacemos lo mismo que en la etiqueta de RT0
   CLR.L D0             ; pero en este caso con el valor de ET1
   RTS


FLAG_Z:
   MOVE.W ESR,D3          ; movemos ESR a un registro auxiliar
   CMP #$0000,D0          ; miramos si todos los bits son 0
   BEQ Z1               ; si todos son 0 saltamos a la etiqueta de Z=1
   BRA Z0
Z1
   BSET #2,D3            ; ponemos a 1 el bit 2, que equivale al flag Z
   MOVE.W D3,ESR          ; guardamos el resultado en el status register
   CLR.L D3             ; limpiamos el registro que hemos utilizado
   RTS
Z0
   BCLR #2,D3            ; ponemos a 0 el bit que corresponde al flag Z
   MOVE.W D3,ESR          ; guardamos el resultado en el status register
   CLR.L D3             ; limpiamos el registro que hemos utilizado
   RTS


FLAG_N:
   MOVE.W ESR,D3          ; movemos ESR a un registro auxiliar
   BTST #15,D0           ; miramos el valor del bit correspondiente al
   BNE N1               ; flag N y saltamos a la etiqueta correspondiente
   BRA N0
```

```
N1
   BSET #1,D3            ; realizamos lo mismo que en el flag anterior pero
   MOVE.W D3,ESR         ; esta vez con el bit de la posicion 1
   CLR.L D3
   RTS
N0
   BCLR #1,D3
   MOVE.W D3,ESR
   CLR.L D3
   RTS


FLAG_C:
   MOVE.W ESR,D3         ; movemos ESR a un registro auxiliar
   BTST #16,D0           ; miramos el ultimo bit para saber si hay carry
   BNE C1                ; saltamos a la etiqueta correspondiente
   BRA C0
C1
   BSET #0,D3            ; realizamos lo mismo que en el flag anterior pero
   MOVE.W D3,ESR         ; esta vez con el bit de la posicion 0
   CLR.L D3
   RTS
C0
   BCLR #0,D3
   MOVE.W D3,ESR
   CLR.L D3
   RTS


FINDC:
   LSR #3,D0             ; desplazamos 3 bits para tener la c al inicio
   MOVE.W #$00FF, D2     ; obtenemos los ultimos 8 bits con una mascara
   AND D0,D2
   BTST #7,D2            ; miramos el msb para determinar si el numero es
   BEQ POS               ; positivo o negativo y saltamos a la etiqueta
   BRA NEG               ; correspondiente
POS
   OR.W #$0000,D2        ; realizamos la extension de signo con bits a 0
   MOVE.W D2,D6          ; guardamos el resultado en un nuevo registro
   CLR.L D2              ; limpiamos los registros que hemos usado
   CLR.L D0
   RTS
NEG
   OR.W #$FF00,D2        ; realizamos la extension de signo con bits a 1
   MOVE.W D2,D6          ; guardamos el resultado en un nuevo registro
   CLR.L D2              ; limpiamos los registros que hemos usado
   CLR.L D0
   RTS


FINDAB:
   BTST #2,D0            ; decodificamos los ultimos 3 bits para saber
   BNE DR1               ; que registro utilizar, de manera similar a la
   BRA DR0               ; decodificacion de la instruccion
```

```
DR1
   BTST #1,D0
   BNE DR11
   BRA DR10
DR0
   BTST #1,D0
   BNE DR01
   BRA DR00
DR11
   BTST #0,D0
   BNE DR111
   BRA DR110
DR10
   BTST #0,D0
   BNE DR101
   BRA DR100
DR01
   BTST #0,D0
   BNE DR011
   BRA DR010
DR00
   BTST #0,D0
   BNE DR001
   BRA DR000
DR000
   MOVE.W #ET0,A3
   RTS
DR001
   MOVE.W #ET1,A3
   RTS
DR010
   MOVE.W #ER2,A3
   RTS
DR011
   MOVE.W #ER3,A3
   RTS
DR100
   MOVE.W #ER4,A3
   RTS
DR101
   MOVE.W #ER5,A3
   RTS
DR110
   RTS                  ; estas combinaciones no se usan
DR111
   RTS


************************** DECODIFICACION **************************

DECOD:

   MOVE.W D0,-(A7)
   MOVE.W 6(A7),D0         ; obtenemos el EIR de la pila
```

```
   BTST.L #15,D0          ; decodificamos bit a bit mediante BTST
   BNE DX1                ; hasta conocer el tipo de instruccion
   BRA DX0

DX0:
   BTST.L #14,D0
   BNE DX01
   BRA DX00
DX00:
   MOVE.W #0,8(A7)        ; una vez encontrada la instruccion, metemos en
   MOVE.W (A7)+,D0        ; la pila la posicion de esa instruccion en el
   RTS                    ; JMPLIST (en este caso HALT es la posicion 0)
DX01:
   BTST.L #13,D0
   BNE DX011
   BRA DX010
DX011:
   MOVE.W #3,8(A7)
   MOVE.W (A7)+,D0
   RTS
DX010:
   BTST.L #12,D0
   BNE DX0101
   BRA DX0100
DX0101:
   MOVE.W #2,8(A7)
   MOVE.W (A7)+,D0
   RTS
DX0100:
   MOVE.W #1,8(A7)
   MOVE.W (A7)+,D0
   RTS
DX1:
   BTST.L #14,D0
   BNE DX11
   BRA DX10
DX11:
   BTST.L #13,D0
   BNE DX111
   BRA DX110
DX111:
   BTST.L #12,D0
   BNE DX1111
   BRA DX1110
DX1111:
   MOVE.W #12,8(A7)
   MOVE.W (A7)+,D0
   RTS
DX1110:
   BTST.L #11,D0
   BNE DX11101
   BRA DX11100
DX11101:
   MOVE.W #11,8(A7)
```

```
    MOVE.W (A7)+,D0
    RTS
DX11100:
    MOVE.W #10,8(A7)
    MOVE.W (A7)+,D0
    RTS
DX110:
    BTST.L #12,D0
    BNE DX1101
    BRA DX1100
DX1101:
    BTST.L #11,D0
    BNE DX11011
    BRA DX11010
DX11011:
    MOVE.W #9,8(A7)
    MOVE.W (A7)+,D0
    RTS
DX11010:
    MOVE.W #8,8(A7)
    MOVE.W (A7)+,D0
    RTS
DX1100:
    BTST.L #11,D0
    BNE DX11001
    BRA DX11000
DX11001:
    MOVE.W #7,8(A7)
    MOVE.W (A7)+,D0
    RTS
DX11000:
    MOVE.W #6,8(A7)
    MOVE.W (A7)+,D0
    RTS
DX10:
    BTST.L #13,D0
    BNE DX101
    BRA DX100
DX101:
    MOVE.W #5,8(A7)
    MOVE.W (A7)+,D0
    RTS
DX100:
    MOVE.W #4,8(A7)
    MOVE.W (A7)+,D0
    RTS

FIN:
    SIMHALT             ; finaliza el programa
    END    START         ; last line of source
```