¿Por dónde empiezo?

Asiste al seminario del lenguaje ensamblador del microprocesador Motorola 68000 y después lee el manual de prácticas de la asignatura que está disponible en la dirección http://www.dia.eui.upm.es

¿Cuál es la estructura de un fichero con un programa en ensamblador?

Los programas están organizados en líneas. Cada línea puede ser una instrucción o una directiva del ensamblador, teniendo, en ambos casos, la estructura siguiente:

```
ETIQUETA    INSTRUCCIÓN-o-DIRECTIVA OPERANDOS
```

La ETIQUETA puede omitirse, debiendo sustituirse por un blanco o tabulador.

Un ejemplo:

```
INICIO      MOVE.L      #1,D0
<TAB>       MOVE.L      #1,D0
```

Nota: #1 es un operando con direccionamiento inmediato que indica el número 1 expresado en base decimal. Pueden utilizarse otras bases como la binaria (anteponiendo el carácter %) o la hexadecimal (carácter $)

¿Cómo indico la dirección de memoria dónde debe cargarse mi código?

Mediante la directiva ORG.

Un ejemplo:

```
            ORG         $400
INICIO      MOVE.L      #$1,D0
```

La directiva ORG $400 indica al ensamblador que coloque el código a partir de la dirección $400 de memoria que es precisamente la siguiente dirección libre trastabla de vectores de interrupción del 68000 (256 vectores de 4 octetos cada uno)

¿Cómo declaro una variable en un programa?

Las variables ocupan una o varias posiciones de memoria dependiendo del tamaño. Para declarar variables se utilizan las directivas ORG (*ORiGin*), DS (*Define Storage*) y DC (*Define Constant*) del ensamblador. Antes de declarar variables conviene tener claro en qué zona de memoria se van a colocar para que no interfieran con el código.

Un ejemplo:

Se ha decidido reservar las direcciones $1000-$1FFF para variables.

```
            ORG   $1000      * Dirección inicial de ensamblaje

* VAR
*     CAR:        CHAR;
*     INDEX:      SHORT;
*     NUM:        LONG

CAR         DS.B  1
INDEX       DS.W  1
NUM         DS.L  1
```

El ensamblador colocará las variables a partir de la dirección $1000 (directiva ORG). La variable CAR ocupa una única posición (sufijo B de la directiva DS), INDEX ocupa dos posiciones de memoria ($1002-$1003, sufijo W) y NUM ocupa 4 posiciones ($1004-$1007). La variable INDEX empieza en la dirección $1002 en lugar de la dirección $1001, ya que las variables de tipo WORD debe nalinearse en frontera par, por lo tanto, la dirección $1001 queda desaprovechada en el ejemplo anterior.

Cuando se utiliza la directiva DS las variables no se inicializan, es decir, su contenido inicial no está determinado(en el momento de arrancar el entorno de prácticas la memoria se encuentra inicializada al valor $00).

NOTA: No olvidar que las etiquetas comienzan en la primera columna. Una instrucción o directiva mal colocada puede ser tratada como una etiqueta por el ensamblador.

¿Cómo se declara una variable con un valor inicial?

Mediante la directiva DC (*Define Constant*).

Un ejemplo en el que se colocan las variables inicializadas a partir de la dirección $2000

```
* VAR
*     SALUDO:     STRING := “Buenos días”;
*     MAX_VAL:    SHORT := 50;
*     MAX_LEN:    LONG := 1000;

            ORG   $2000
SALUDO      DC.B  ‘Buenos días’,0
MAX_VAL     DC.W   50
MAX_LEN     DC.L   1000
```

La variable SALUDO ocupa las direcciones $2000-200B. MAX_VAL ocupa las direcciones $200C-200D cuyos contenidos, respectivamente, son $0 y $32 (50 en decimal). Por último, MAX_LEN ocupa las direcciones $200E-$2011 y cuyos valores iniciales son $0, $0, $03 y $E8.

¿Cómo se declara una constante?

Mediante la directiva EQU.

Un ejemplo:

```
* CONST
*     MAX = 10
*     MIN = 1

MAX   EQU   1000  * Asigna al símbolo MAX el valor 1000 (decimal)
MIN   EQU   400   * Asigna al símbolo MIN el valor 400 (decimal)
```

Para estas definiciones el ensamblador no reserva ninguna posición de memoria, únicamente se limita a asignar al símbolo un valor. El objetivo de esta directiva es facilitar el desarrollo de los programas, perm itiendo manejar constantes simbólicas en lugar de numéricas.

La directiva EQU admite expresiones como p.e. MAX EQU 4*4

¿Qué partes forman un programa en ensamblador?

Son cinco: inicialización de los vectores de interrupción, declaración de constantes(directiva EQU) , declaración de variables sin inicializar (DS), declaración de variables inicializadas (DC), y código. Pudiendo algunos programas carecer de alguna de ellas.

Veamos un ejemplo:

```
* Inicialización de los vectores de interrupción
            ORG   $0
            DC.L  $1000   * primer vector de interrupción (SSP)
            DC.L  INICIO  * segundo vector de interrupción (PC)

* SSP es el puntero de pila de supervisor
* PC es el contador de programa

* Declaración de constantes
CTE1        EQU   0

* Declaración de variables sin inicializar
            ORG   $400
VAR1        DS.B  1

* Declaración de variables inicializadas
            ORG   $500
TEXTO       DC.B  ‘Hola’
* Código
            ORG   $600

INICIO      MOVE.L      #1,D0
            BREAK
            END
```

(Los valores numéricos de la directiva ORG que aparecenen el ejemplo, están a modo de ejemplo.)

BREAK es una instrucción especial que detiene la ejecución del procesador en el entorno BSVC, debe ser la última instrucción máquina del programaen ejecutarse.

END es una directiva del ensamblador que indica el final del programa y detiene el proceso de ensamblaje. Cualquier declaración o instrucción tras esta directiva no será tenida en cuenta por el ensamblador.

¿Puedo dividir mi programa en varios ficheros?

Si. Aunque para poder ejecutarlo en el simulador debes obtener un único fichero objeto (extensió n h68). Para conseguir esto puedes utilizar la directiva INCLUDE del ensamblador que te permitiráincorporar uno o más ficheros dentro de otro.

¿Cómo se accede a la dirección de una variable?

Por su nombre precedido del carácter almohadilla (#). Veamos unejemplo

```
            ORG   $600
SALUDO      DC.B  'Buenos días'

            ORG   $700
            MOVE.L      #SALUDO,A1
```

La instrucción MOVE almacena en el registro A1 del procesador el valor $600, que es precisamente la dirección de comienzo de la variable SALUDO. Si nos equivocamos y omitimos la almohadilla (#) en A1 se almacenará el valor $4275656E que corresponde a los valores ASCII de las cuatro primeras letras de la variable SALUDO.

¿Cómo guardo un valor en una variable?

Ejemplo para una variable de tamaño WORD

```
            ORG         $600
INDEX       DS.W        1

            ORG         $700
            MOVE.W      #43,INDEX
```

Almacena el valor 43 en la variable INDEX cuya dirección de memoria es $600.

¿Cómo utilizo los símbolos declarados mediante la directiva EQU?

El siguiente fragmento de programa:

```
MAX_VAL     EQU         8
MIN_VAL     EQU         2

            ORG         $800
            MOVE.L      #MAX_VAL,D0
```

a lmacena en el registro D0 el valor 8. Si olvidamos poner la almohadilla (#) delante del símbolo MAX_VAL, también es una instrucción válida, pero el resultado es completamente distinto, almacenándose en D0 el contenido de la dirección de memoria 8.

¿Cómo declaro un array de enteros?

Para un array de 8 enteros de 16 bits

```
            ORG   $400
NUMEROS     DS.W  8
```

El sufijo .W indica que la variable es de tipo WORD (2 octetos) y el 8 el número de palabras (elementos del array)

¿Cómo accedo a los distintos elementos de un array de caracteres?

Supongamos un array de 8 caracteres

```
            ORG         $400
TIRA        DS.B        8

            ORG         $600
            MOVE.L      #TIRA,A0
            MOVE.L      #0,D0
            MOVE.B      #'H',0(A0,D0)           * TIRA[0] := 'H';
            MOVE.L      #1,D0
            MOVE.B      #'O',0(A0,D0)           * TIRA[1] := 'O';
            MOVE.L      #2,D0
            MOVE.B      #'L',0(A0,D0)           * TIRA[2] := 'L';
            MOVE.L      #3,D0
            MOVE.B      #'A',0(A0,D0)           * TIRA[3] := 'A';
```

Inicializa los cuatro primeros elementos del array TIRA con los valores 'H','O','L','A'.
En la primera instrucción se almacena en el registro A0 la dirección de comienzo del array. Para acceder a cada elemento del array se utiliza un desplazamiento desde el primer elemento, que en este caso se va almacenando en el registro D0. El tipo de direccionamiento utilizado es el indirecto indexado.
El resto de elementos del array (5..7) quedan sin inicializar.

Una alternativa al código anterior es:

```
            MOVE.B      #'H',TIRA+0
            MOVE.B      #'O',TIRA+1
            .........
```

La primera solución será más útil cuando tengamos que realizar un acceso secuencial a todos los elementos del array mediante un bucle. Para ello utilizaremos el registro D0 como índice para acceder a cada uno de los elementos y como condición de finalización del bucle.

¿Cómo declaro un array de registros?

Parta mos de la declaración Pascal:

```
VAR TABLA: ARRAY [0..MAX_LEN-1] OF RECORD
                                        ALTURA:     SHORT;  // WORD
                                        PESO:       SHORT;  // WORD
                                    END;
```

En ensamblador quedaría:

```
      * Declaración de constantes
      MAX_LEN           EQU   8
      TAM_RECORD        EQU   4     * tamaño (ALTURA) + tamaño (PESO)
      OFFSET_ALTURA     EQU   0
      OFFSET_PESO       EQU   2

      * Declaración de variables sin inicializar
                        ORG   $400
      TABLA             DS.B  MAX_LEN*TAM_RECORD      * 32 octetos
```

Otra forma:

```
      * Declaración de variables sin inicializar
                        ORG   $400
      TABLA             DS.W  MAX_LEN*2         * 2 WORDs
```

¿Cómo accedo a los distintos elementos de un array de registros?

Partiendo de la declaración de la pregunta anterior, vamos aprogramar los accesos siguientes

```
* TABLA[0].ALTURA := 100;
      MOVE.L      #$0,D0            * índice en D0
      MULU        #TAM_RECORD,D0    * Comienzo del elemento en el array
      MOVE.L      #TABLA,A0         * Dir. Inicio de la tabla
      MOVE.W      #100,OFFSET_ALTURA(A0,D0)

* TABLA[0].PESO := 100;
      MOVE.L      #$0,D0
      MULU        #TAM_RECORD,D0
      MOVE.L      #TABLA,A0
      MOVE.W      #100,OFFSET_PESO(A0,D0)

* TABLA[1].ALTURA := 50;
      MOVE.L      #$1,D0
      MULU        #TAM_RECORD,D0
      MOVE.L      #TABLA,A0
      MOVE.W      #50,OFFSET_ALTURA(A0,D0)
```

¿Cómo programo una sentencia IF en ensamblador?

Supongamos que se quiere programar la sentencia Pascal siguiente:

```
IF X > 12 THEN X := 2*X+4 ELSE X:= X+Y
```

En primer lugar hay que declarar y dar un valor inicial a las variables X e Y.

El programa quedaría:

```
* Vectores de interrupción
            ORG   $0
            DC.L  INICIO * CP inicial
            DC.L  $8000  * Puntero de pila de supervisor inicial


* Declaración de variables inicializadas
            ORG   $400
X           DC.W  $20
Y           DC.W  $4

            ORG         $500
INICIO      MOVE.W      X,D0  * Movemos el valor de X al reg. D0
            CMP.W       #12,D0      * IF X > 12
            BGT         MAYOR
            JMP         MENOR
MAYOR       MOVE.W      #2,D1       * THEN
            MULU        D1,D0       *     X := 2 * X
            ADDQ.W      #4,D0       *     X := X + 4
            JMP         FIN
MENOR       ADD.W       Y,D0        * ELSE X := X + Y

FIN         MOVE.W      D0,X        *
            BREAK                   *
            END
```

Para programar este tipo de sentencias conviene repasar las instrucciones de comparación (CMP y TST) y salto condicional (Bcc) disponibles en el lenguaje ensamblador del 68000.

¿Cómo programo una sentencia FOR en ensamblador?

Supongamos que se quiere programar en ensamblador el siguiente fragmento de Pascal:

```
FOR I : = 1 TO MAX_LEN DO
     TABLA[I].ALTURA := 0;
     TABLA[I].PESO := 0;
END;
```

En lenguaje ensamblador quedaría:

```
* Vectores de interrupción
                ORG   $0
                DC.L  INICIO      * CP inicial
                DC.L  $8000       * Puntero de pila inicial
                                  * en modo supervisor


* Declaración de constantes
MAX_LEN         EQU         8
TAM_RECORD      EQU         4     *tam(ALTURA)+tam (PESO)
OFFSET_ALTURA   EQU         0
OFFSET_PESO     EQU         2

* Declaración de variables sin inicializar
                ORG         $400
TABLA           DS.B        MAX_LEN*TAM_RECORD      *32 octetos
I               DS.W        1

* Código
                ORG         $500
INICIO          MOVE.L      #TABLA,A0
                MOVE.W      #$1,I             * I := 1

FOR             CMP.W       #MAX_LEN,I        * I < MAX_LEN ?
                BEQ         FIN
                MOVE.W      I,D1
                MULU        #TAM_RECORD,D1
                MOVE.W      #$0,OFFSET_ALTURA(A0,D1)
                MOVE.W      #$0,OFFSET_PESO(A0,D1)
                ADD.W       #$1,I             * I := I + 1
                JMP         FOR

FIN             BREAK
                END
```

NOTA: Podría haberse utilizado un registro de datos como índice del bucle FOR

¿Cómo programo una sentencia WHILE en ensamblador?

Partimos del fragmento siguiente de código Pascal

```
WHILE X = Y DO
BEGIN
     X := X + 1
END
```

En lenguaje ensamblador quedaría (se han omitido las declaraciones iniciales):

```
            ORG         $400
BUCLE       CMP.W       D0,D1       * D0 = X, D1 = Y
            BNE         FIN         * WHILE X=Y DO
            ADDQ        #1,D0       *     X = X + 1
            BRA         BUCLE       *
FIN         BREAK
            END
```

¿Cómo programo una sentencia REPEAT en ensamblador?

Partimos del fragmento siguiente de código Pascal:

```
Y := 20;
X := 0;
REPEAT
      X := X + 1
UNTIL X = Y
```

En lenguaje ensamblador quedaría:

```
      ORG         $400

      MOVE.W      #20,D1      * Y := 20
      MOVE.W      #0,D0       * X := 0
                                    * REPEAT
BUCLE ADDQ        #1,D0       *     X := X + 1
      CMP.W       D0,D1
      BNE         BUCLE       * UNTIL X = Y
      BREAK
      END
```

¿Cómo paso parámetros a una subrutina?

Mediante la pila. El programa que llama a una subrutina es el encargado de colocar los parámetros en la pila y, tras completarse la llamada, retirarlos.

¿Cuántas formas existen de pasar parámetros a una subrutina?

Básicamente dos, por valor y por referencia. El paso por valor se utiliza para parámetros de entrada y por referencia, para parámetros de entrada/salida (los de tipo VAR en Pascal).
En el método por valor, el programa llamante mete en la pila el valor de la variable que pasa como parámetro. En el caso de la referencia mete la dirección de memoria de la variable, lo que permite a la subrutina modificar el contenido de la variable al disponer de su dirección de memoria.

Veamos un ejemplo del paso de parámetros por valor:

```
CAR   DS.B        1

      ORG         $400
* Paso por valor de la variable CAR a la subrutina WRITE
      MOVE.B      CAR,-(SP)
      BSR         WRITE
      ADDQ        #2,SP
      … … … … … … … … … …
```

La instrucción MOVE mete en la pila el valor de la variable CAR. La pila se maneja mediante el registro A7 o SP (*Stack Pointer*) que guarda la dirección de memoria de la cima de la pila. La pila crece de las direcciones altas a las bajas, de ahí el predecremento del segundo operando. Tras apilar el parámetro se llama a la subrutina WRITE, que recibirá el parámetro en la pila. Una vez finalizada la subrutina es necesario eliminar de la pila el parámetro. Para ello basta con incrementar el puntero de pila (instrucción ADDQ).

Para el caso de los parámetros por referencia

```
CAR   DS.B        1

      * Paso por referencia de la variable
      * CAR a la subrutina READ
      ORG $400
      MOVE.L      #CAR,-(SP)
      BSR         READ
      ADDQ        #4,SP
      … … … … … … … … … …
```

Como puede verse las diferencias son dos entre ambas formas de apilar los parámetros. La primera diferencia está en la instrucción (MOVE) que pasa la dirección de la variable CAR y no su contenido como ocurría en el paso por valor. Al pasar la dirección se almacenan en la pila 4 octetos (sufijo L de la instrucción). La segunda diferencia, consecuencia de la primera, está en la instrucción ADDQ que debe eliminar de la pila 4 octetos en lugar de 2.

¿Qué ocurre cuando hay más de un parámetro?

Se apilan uno tras otro. La subrutina que recibe los parámetros debe conocer el tamaño y el orden de cada uno de ellos para poder acceder a su contenido.

¿Cómo se accede a los parámetros en una subrutina?

La subrutina utilizará el registro A7 (SP) para acceder a los parámetros. Hay que tener en cuenta que tras la llamada a la subrutina (instrucción BSR) el procesador inserta en la pila la dirección de retorno (4 octetos), por lo que a la hora de acceder a los parámetros deberá aplicar un desplazamiento positivo de 4 octetos para salvar la dirección de retorno que se encuentra en la cima de la pila.

Veamos un ejemplo para el caso de la subrutina WRITE (paso por valor)

```
WRITE: MOVE.B     4(SP),D0
```

Accede al contenido del parámetro salvando la dirección de retorno que ocupa 4 octetos. Obsérvese que el tipo de acceso de la instrucción (BYTE) coincide con el de la instrucción que lo insertó en la pila (sufijo .B).

Para el caso de la subrutina READ, donde el parámetro es pasado por referencia, el acceso es similar, pero ahora en la pila está la dirección de la variable y no su contenido.

```
READ: MOVE.L      4(SP),A0
      MOVE.B      #'A',(A0)
```

La primera instrucción recupera el parámetro, guardándolo en el registro A0, y la segunda instrucción almacena el carácter 'A' en la dirección pasada como parámetro.

¿Qué modos de direccionamiento utilizo para manejar los parámetros?

Indexado con predecremento para apilar los parámetros e indirecto con desplazamiento para accederlos.

¿Qué ocurre con los parámetros de tamaño BYTE?

El puntero de pila (registro A7 o SP) siempre guarda direcciones pares, por lo tanto, al insertar un octeto en la pila utilizando el direccionamiento indexado con predecremento, el puntero de pila se decrementa en dos en lugar de uno solamente. Este hecho debe ser tenido en cuenta en el desapilado de los parámetros de tipo BYTE.

¿Dónde se almacenan las variables locales?

Las variables locales se almacenan en la pila. La subrutina que necesite utilizar variables locales reservará el espacio necesario en la pila, y antes de devolver el control de la ejecución al punto desde donde fue llamada, deberá liberar el espacio reservado para las variables locales.

Si se crean variables locales hay que tener en cuenta que estarán situadas en la pila justo encima de la dirección de retorno (4 octetos), para acceder a las variables locales también se utilizará el direccionamiento indexado con desplazamiento.

Al utilizar las variables locales hay que tener cuidado para no modificar la dirección de retorno de la subrutina. Antes de ejecutar la instrucción RTS (retorno de subrutina) el puntero de pila deberá apuntar a la dirección de retorno (almacenada en la pila por las instrucciones BSR o JSR de llamada a subrutina).

¿A qué se debe el error de ensamblaje "Symbol value differs between first and second pass"?

En las etiquetas sólo son significativos los 8 primeros caracteres. Si en un programa se utilizan dos o más etiquetas cuyos 8 primeros caracteres son iguales se provocará la aparición de este error en la fase de ensamblaje.

¿A qué se debe el error de ensamblaje "MOVEQ instruction constant out of range"?

La instrucción ADDQ sólo admite como direccionamiento inmediato una constante entre 1 y 8. Y la instrucción MOVEQ solo admite constantes en el rango de valores que van desde -128 a 127.