UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
DE TELECOMUNICACIÓN

ARM ASSEMBLY LANGUAGE TUTORIAL

1. Formato general de programación en lenguaje ensamblador

La estructura genérica de una línea de código en lenguaje ensamblador se divide en campos. Estos campos son los que se enumeran en la sintaxis que se muestra:

{label} {instruction|directive|pseudo-instruction} {;comment}

Todos los campos que se listan son opcionales, lo que implica que podemos usar líneas en blanco para hace más legible el código. En caso de no utilizar el campo {*label*}, tanto las *instruction*, *directive*, o *pseudo-instruction* deben ir precedidos de, al menos, un espacio en blanco, esto es, no podrán comenzar en la primera columna del código. En cualquier caso, los campos *{label}* y *{;comment}* son siempre opcionales.

1.1.Delimitadores

En el formato anteriormente expuesto todos los campos deben estar delimitados. El delimitador más común es el espacio en blanco, si bien es posible que determinados ensambladores acepten otro tipo de delimitadores, tales como la coma, el guión, etc. Se recomienda usar siempre el espacio en blanco para su delimitación.

La programación en lenguaje ensamblador lleva asociado el uso de varios delimitadores. Los más comunes son:

- *Espacio en blanco*. Se utiliza para delimitar los campos *{label}* y *{instruction|directive|pseudo-instruction}*, los campos *{instruction|directive|pseudo-instruction}* y *{;comment}*. Dentro del campo *{instruction|directive|pseudo-instruction}* se utiliza para delimitar el *opcode* y los operandos o direcciones.

- *Coma*. La coma es el delimitador más común entre los operandos de una instrucción (ver apartado de *Instrucciones* para mayor detalle sobre operandos).

- *Asterisco o punto y coma*. El asterisco, o en algunos casos el punto y coma, se utiliza para delimitar un comentario.

- *Tabulador*. En la mayor parte de los códigos en lenguaje ensamblador, el tabulador se utiliza para alinear los campos, tal y como se muestra en el Código 1.

El Código 1 muestra algunos ejemplos de uso de los delimitadores.

```
;/* Ejemplo de línea de comentario*/
ETI     MOV     R1,#4           ;/* Línea conteniendo todos los campos */
        ADDS    R3,R0,R1        ;/* Línea sin campo label, seguida de línea sin etiqueta ni comentarios */
        NOP
        ...
```

Código 1. Uso de delimitadores en lenguaje ensamblador

1.1.1. Recomendaciones en el uso de delimitadores

A continuación se listan algunas recomendaciones en el uso de los delimitadores.

- No usar delimitadores en nombres y/o etiquetas.

- Alinear los campos para hacer más legible el código.
- No utilizar espacios en blanco después de las comas que delimitan los operandos.

1.2.Etiquetas

Las etiquetas deben colocarse en la primera columna de la línea en la que se insertan. El uso de las etiquetas facilita la escritura del código en ensamblador. Las etiquetas son especialmente útiles para:

- *Asignación de direcciones*. En este caso la etiqueta se localiza precediendo a una *instrucción* o *pseudo-instrucción*. El ensamblador asigna a la etiqueta el valor de la dirección en el punto donde está localizada la etiqueta. Su uso resulta de mucha utilidad en las instrucciones de salto y en instrucciones de direccionamiento a memoria.

  La *instrucción* `B <LABEL>`, del ARM, realiza un salto a una posición de memoria. Tomando como referencia el Código 2 que se adjunta, y suponiendo que la instrucción `MOV` está localizada en la dirección `0x50` de memoria, la instrucción `B LOOP`, debería escribirse como `B 0x50`. Con el uso de etiquetas, se deja al ensamblador la tarea de calcular las direcciones de salto. Únicamente tiene sentido etiquetar aquellas instrucciones que queramos referenciar en el código.

  La *pseudo-instrucción* `LDR Rd,ADDR`, del ARM, carga en el registro destino (`Rd`) el dato localizado en una posición de memoria de dirección `ADDR`. Tomando como referencia el Código 2 que se adjunta, y suponiendo que el dato está localizado en la dirección `0x150` de memoria, la instrucción `LDR R1,DATO`, debería escribirse como `LDR R1,0x150`. Con el uso de etiquetas, se deja al ensamblador la tarea de calcular las direcciones de los datos.

- *Asignación de constantes*. En este caso la etiqueta se localiza antes de una *directiva*. El ensamblador asigna a la etiqueta el valor definido por la directiva. Este uso resulta de mucha utilidad cuando se desea parametrizar un código.

  La *pseudo-instrucción* `LDR Rd,ADDR`, del ARM, carga en el registro destino (`Rd`) el dato localizado en una posición de memoria de dirección `ADDR`. Tomando como referencia el Código 2 que se adjunta, y suponiendo que el dato está localizado en la dirección `0x150` de memoria, la instrucción `LDR R1,DATO`, debería escribirse como `LDR R1,0x150`. Con el uso de etiquetas, se deja al ensamblador la tarea de calcular las direcciones de los datos.

```
        ...
MULT    EQU     4
LOOP    MOV     R0,#MULT        ;/* Comienzo del bucle */
        LDR     R1,DATO
        ...
        B       LOOP            ;/* Seguimos en el buble */
DATO    DCD     6
        ...
```

Código 2. Ejemplo de uso de las etiquetas

1.2.1. Recomendaciones en el uso de etiquetas

A continuación se listan una serie de reglas básicas en el uso de etiquetas que nos permitirán evitar problemas a la hora de escribir en ensamblador.

- No usar nombres reservados, tales como aquellos usados por las *instrucciones* (MOV, LDR, ADD, etc.) y *directivas* (DCD, EQU, DCB, etc.).
- Usar etiquetas con significado. Esto nos ayudará a recordar más fácilmente su uso.
- No usar caracteres especiales en el nombre de una etiqueta.
- La etiqueta debe comenzar con un carácter y no un número.

1.3. Instrucciones

El campo *{instruction|directive|pseudo-instruction}* se divide, a su vez, en los campos *{mnemonic}* y *{operand or address}*.

Las instrucciones en sí mismo se referencian mediante lo que se denominan *mnemonics* (*mnemónicos*) y son directamente interpretables por el procesador. Estos *mnemónicos* corresponden con las instrucciones que son capaces de interpretar el procesador (ver el documento con el listado de instrucciones del ARM). Dependiendo de la instrucción que se trate, estos *mnemónicos* van acompañado de un número variable de operandos o direcciones, que hacen referencia a los operandos o las direcciones con las que trabaja la instrucción.

El Código 3 muestra un ejemplo de varias instrucciones del ARM y sus operandos. En estos ejemplos observamos como hay instrucciones con dos operandos, fuente y destino, caso de la instrucción MOV, instrucciones con tres operandos, dos operandos fuentes y un operando destino, caso de la primera instrucción ADDS, etc. Para una misma instrucción, el campo operando será de longitud variable, dependiendo del modo en que se especifiquen los operandos, contendrá más o menos sub-operandos, tal y como se puede observar en las instrucciones

```
;/* Ejemplo de instrucciones y sus operandos*/
        MOV     R1,#4           ;/* R1 se carga con el dato inmediato 4 */
        ADDS    R3,R0,R1        ;/* R3 <- R0 + R1*/
        ADDS    R2,R0,R1 ,LSL #2 ;/* R2 <- R0 + 4 x R1*/
        BEQ     SALIR           ;/* Si r2 es cero salimos */
        LDM     SP!,{R0,LR}     ;/* Recuperamos valores de la pila */
        ...
```

Código 3. Código conteniendo instrucciones del ARM

Como regla básica se recomienda no usar espacios después de los delimitadores de los operandos (usar MOV R1,#4 en lugar de MOV R1, #4, evitando introducir espacios después del delimitador ",").

1.4. Directivas

Las directivas corresponden a las instrucciones en lenguaje ensamblador que no son directamente interpretadas por el procesador. Las directivas son instrucciones que serán interpretadas por el ensamblador. Se utilizan, entre otras funciones, para definir símbolos, designar áreas de memoria, definir tablas, delimitar código etc.

Las directivas varían dependiendo del ensamblador que se utilice, si bien en su mayoría su interpretación resulta obvia. A continuación se listan las directivas más comunes

- *EQU (Equate)*. Corresponde a una directiva de igualdad. El mnemónico más común para esta directiva es EQU. Su sintaxis es *{label} EQU {operand or address}*. Mediante esta directiva se puede asignar el valor del campo *{operand or address}* a la etiqueta localizada en el campo *{label}*. La

asignación puede realizarse bien con datos o direcciones. El Código 4 muestra dos ejemplos de su definición y uso.

La directiva EQU no ocupa espacio en memoria, por lo que puede estar localizada en cualquier parte del código.

```
;/* Ejemplos de directiva EQU y su uso */
PI      EQU     314             ;/* Definimos el valor de PI */
STADDR  EQ      0x40000000      ;/* Definimos dirección de comienzo de la memoria RAM */
        MOV     R1,#PI          ;/* Cargamos R1 con el valor de PI */
        ...
```

Código 4. Ejemplos de la directiva EQU y su uso

- *DEFINE CONSTANT*. Esta directiva permite definir datos fijos en posiciones de memoria de programa. Estos datos pueden ser de múltiple naturaleza, tales como numéricos, caracteres, mensajes (cadenas de caracteres), etc. Comúnmente los ensambladores suelen ofrecer varios tipos de directivas de definición de constantes, dependiendo del tipo/tamaño de datos que se quiera definir, *byte* (***D**efine **C**onstant **B**yte* o DCB), *palabra* (***D**efine **C**onstant **W**ord* o DCW). En el caso del ARM se proporciona la directiva ***D**efine **C**onstant **D**ata*, o DCD, para definir datos de 32 bits, equivalente a la directiva DCW.

  El uso del campo *{label}* es opcional, si bien en la mayor parte de los casos es recomendable. La etiqueta en el campo *{label}* toma la dirección de memoria donde se ha definido la constante.

  Una misma directiva puede utilizarse para definir varios datos, en cuyo caso se hará uso del delimitador coma para separar las diferentes constantes. En este caso, la etiqueta en el campo *{label}* toma la dirección de la primera constante definida.

  La definición de constantes tipo carácter o mensajes hace uso de los delimitadores ‘ y “, tal y como se muestra en los ejemplos del Código 5. En este caso, en memoria se almacena el código ASCII del carácter o caracteres.

```
;/* Ejemplos de directivas DC */
DATAB   DCB     0x54            ;/* Constante tipo byte en memoria con valor 54 en hexadecimal */
        ALIGN                   ;/* Alineamiento de palabras */
DATAW   DCW     6,1,2,3         ;/* Definición de cuatro constantes de tamaño palabra, inicializadas */
                                ;/* con los valores 6, 1, 2 y 3 respectivamente */
ERRMES  DCB     "ERROR"         ;/* Almacenamiento en memoria de los códigos ASCII correspondientes a los */
                                ;/* caracteres E, R, R, O y R. Cada código ASCII ocupa un byte en memoria */
        ...
```

Código 5. Ejemplo de directivas DC

- *ALIGN*. La directiva ALIGN se utiliza para alinear datos y/o instrucciones en memoria. Al tratarse de un procesador de 32 bits, las palabras deben estar alineadas, es decir, deben estar almacenadas en direcciones múltiplos de palabras. En el Código 5, y suponiendo que la dirección de almacenamiento de DATAB es la dirección 0x012, al introducir la directiva ALIGN, la palabra etiquetada como DATAW se alinea en la posición 0x014.

- *SPACE*. La directiva SPACE se utiliza para reservar un bloque de memoria inicializado a cero. Su sintaxis es *{label} SPACE expr*. El campo *expr* evalúa el número de bytes a reservar. Al especificar el tamaño en bytes, se recomienda usar la directiva ALIGN para alinear cualquier código o datos de tamaño palabra que se declare posteriormente. El Código 6 muestra un ejemplo de uso de la directiva SPACE.

```
;/* Ejemplo de directiva SPACE */
LONG    EQU     256*256         ;/* Definición de long */
DATOS   SPACE   LONG            ;/* Reserva una zona de 256*256 bytes */
        ALIGN                   ;/* Alineamiento del código localizado a partir de esta posición */
                                ;/* Sólo es necesario para alinear código o palabras, pero no bytes */
        ...
```

Código 6. Ejemplo de directiva SPACE

1.4.1. Recomendaciones en el uso de directivas

A continuación se listan una serie de reglas básicas en el uso de las directivas:

- Localizar las directivas EQU al comienzo de la sección de código donde vayan a ser utilizadas.
- La localización de las directivas DC depende del tipo de constantes que se vayan a definir. Si las constantes van a ser fijas, éstas estarán localizadas en la sección de código donde se vayan a referenciar. Si las constantes van a ser posiciones de memoria de escritura, únicamente es posible ubicarlas en secciones de datos.
- Usar la directiva ALIGN cada vez que se localicen instrucciones o palabras después de definir o reservar bytes.

1.5.Pseudo-instrucciones

Las pseudo-instrucciones son instrucciones que, si bien no pueden ser traducidas directamente a lenguaje máquina, pueden traducirse a este lenguaje mediante una instrucción o combinación de instrucciones alternativas en tiempo de ensamblado.

A continuación se listan las pseudo-instrucciones más comunes reconocidas por los ensambladores del ARM.

- *ADR*. La pseudo-instrucción ADR carga una dirección relativa a contador de programa en un registro. Su sintaxis es *ADR Rd,label*. En la sintaxis, *Rd* representa el registro destino donde se almacenará la dirección calculada. La etiqueta *label* debe estar localizada dentro de la misma sección (ver definición de sección mediante directiva AREA).

  La pseudo-instrucción ADR permite usar un modo de direccionamiento no disponible en el *Instruction Set Architecture* del ARM. Se recomienda su uso para inicializar punteros. En el Código 7, donde se ha añadido una primera columna que representa la dirección de memoria, tras la ejecución del código R0 valdrá 0x00000028 y R1 valdrá 0x00000003.

```
;/* Ejemplos de definición y uso de la pseudo-instrucción ADR */
                ADR     R0,TABLA        ;/* Cargamos R0 con puntero a TABLA */
                LDR     R1, [R0]        ;/* Cargamos en R1 la primera palabra de la TABLA */
                ...
0x0028  TABLA   DCD     3,5,7,0xf       ;/* Definición de la TABLA */
        ...
```

Código 7. Ejemplo de definición y uso de la pseudo-instrucción ADR

- *LDR*. El mnemónico LDR puede corresponder a una instrucción lenguaje ensamblador del ARM o a una pseudo-instrucción del mismo. Su uso como pseudo-instrucción depende del modo de direccionamiento que se especifique. Hay que tener en cuenta que el único modo de direccionamiento implementado por el ARM es indirecto por registro. La sintaxis de la pseudo-

instrucción LDR es *LDR Rt,{=}{expr | label-expr}*. Existen dos posibles formas de especificar el campo *{expr | label-expr}*, utilizando el signo de igualdad o sin él.

El uso de la pseudo-instrucción con la sintaxis *LDR Rt,= {expr | label-expr}* permite cargar una constante de 32 bits o una dirección en el registro codificado en el campo *Rt*. Su uso es equivalente al modo de direccionamiento inmediato.

El uso de la pseudo-instrucción con la sintaxis *LDR Rt,{label-expr}* permite cargar el contenido de una dirección en el registro codificado en el campo *Rt*. Su uso es equivalente al modo de direccionamiento directo.

```
;/* Ejemplos de definición y uso de la pseudo-instrucción LDR */
                LDR     R0,=0x3FFA      ;/* Carga R0 con el dato inmediato 0x3FFA */
                LDR     R1,=TABLA       ;/* Carga R1 con la dirección de TABLA (0x0028) */
                LDR     R2,TABLA        ;/* Carga R2 con el dato almacenado en la dirección TABLA (3) */
                ...
0x0028  TABLA   DCD     3,5,7,0xf       ;/* Definición de la TABLA */
        ...
```

Código 8. Ejemplo de definición y uso de la pseudo-instrucción LDR

2. Estructura de un módulo en ensamblador del ARM

El Código 9 ejemplifica la estructura de un módulo escrito en lenguaje ensamblador del ARM.

```
;/*****************************************************************************/
;/* main.s: Main code                                                         */
;/*****************************************************************************/
;/* <<< Programa de ejemplo para mis estudiantes >>>                          */
;/*****************************************************************************/

        AREA    MAIN,CODE,READONLY
        EXPORT  __main          ;/* Hace accesible el modulo __main desde otros modulos */
        ENTRY                   ;/* Marca la primera instrucción a ejecutar */

__main  LDR     R0,=TABLA       ;/* r0 puntero a TABLA */
        LDR     R1,[R0,#0]      ;/* r1 se carga con el primer dato */
        LDR     R2,[R0,#4]      ;/* r2 se carga con el segundo dato */
        ADD     R3,R2,R1        ;/* Sumamos y depositamos en r3 */
        LDR     R0,=SUMA        ;/* r0 puntero a SUMA */
        STR     R3,[R0]         ;/* Almacenamos resultado en SUMA */
        MOVS    PC,LR           ;/* Retornamos */

TABLA   DCD     8,9,5,6,4,1,2,3

        AREA    DATOS,DATA,READWRITE
SUMA    SPACE   4

        END
```

Código 9. Estructura de un módulo en lenguaje ensamblador del ARM

2.1. Cabecera

Se recomienda el uso de una cabecera significativa. Esta cabecera deberá contener, al menos, el propósito del programa.

2.2. La directiva AREA

La directiva `AREA` indica al ensamblador que comienza una nueva sección de código o datos. Las secciones son independientes, con nombre propio y albergan instrucciones y/o datos que son manipulados por el *linker*.

Su sintaxis es *`AREA section_name, {attr}{,attr}...`*, donde el primer campo indica el nombre de la sección que, en todo caso, debe ser único. No existe ninguna restricción en el nombre de la sección, salvo el caso particular en que éste comience por un número, en cuyo caso el nombre de la sección se debe introducir entre barras verticales, por ejemplo, *`|1_Data|`*.

En el ejemplo que se muestra en el Código 9 se han insertado dos áreas, denominadas `MAIN` y `DATOS`.

2.3. La directiva END

La directiva `END` indica al ensamblador que se ha alcanzado el final del fichero fuente. Es de destacar que no está asociado a una sección sino que, por el contrario, está asociado al fichero fuente, debiendo localizarse al final del mismo.

3. Bibliografía

En este apartado se muestran algunas referencias bibliográficas utilizadas para la realización del presente documento, así como para ampliar los conceptos que aquí se explican.

[1] *RealView Assembler User's Guide*, disponible en http://www.keil.com/support/man/docs/armasm/, y visitado por última vez el 7 de octubre de 2008. En este enlace se puede consultar, de forma *online*, el manual de usuario del ensamblador de RealView.

[2] *ARM Assembler Workbook*, disponible en http://www.cse.psu.edu/~anand/497b/labs/**arm**.doc, *version 1.0, January 14th, 1997*, visitado por última vez el 7 de octubre de 2008.