Aspecto de un programa en Ensamblador

Veamos como es un programa completo en lenguaje ensamblador y que partes tiene. En el código hay una serie de elementos que aparecerán en todos los programas y que veremos a continuación ejemplo00.s

```
  .data
  var1 : .word 3
  var2 : .word 4
  var3 : .word 0x1234
  .text
  .global main
   main : ldr r1, puntero_var1    /* r1 <- & var1 */
          ldr r1, [r1]            /* r1 <- *r1 */
          ldr r2, puntero_var2    /* r2 <- & var2 */
          ldr r2, [r2]            /* r2 <- *r2 */
          ldr r3, puntero_var3    /* r3 <- & var3 */
          add r0, r1, r2          /* r0 <- r1 + r2 */
          str r0, [r3]            /* *r3 <- r0 */
          mov r7, #1              /* si R7=1 swi sabe que deber salir a
sistema operativo */
          swi 0           /* SWI, Software interrup */
          /* --------------------- */
          puntero_var1 : .word var1
          puntero_var2 : .word var2
          puntero_var3 : .word var3
```

La principal característica de un módulo fuente en ensamblador es que existe una clara separación entre las instrucciones y los datos. La *estructura más general de un programa es*:

- **Sección de datos**. Viene identificada por la directiva *.data*. En esta zona se definen todas las variables que utiliza el programa con el objeto de reservar memoria para contener los valores asignados. Hay que tener especial cuidado para que los datos estén alineados en palabras de 4 bytes, sobre todo después de las cadenas. Alinear significa rellenar con ceros el final de un dato para que el siguiente dato comience en una dirección múltiplo de 4 (con los dos bits menos significativos a cero). Los datos son modificables.
- **Sección de código**. Se indica con la directiva *.text*. En esta parte se escribe el programa y las instrucciones en ensamblador que operan con los datos definidos en la sección anterior.

De estas dos secciones la única que obligatoriamente debe existir es la sección .text (o sección de código). En nuestro ejemplo comprobamos que están las dos.

Un módulo fuente, como el del ejemplo, está formado por instrucciones, datos, símbolos y directivas. Estos elementos se describen a continuación.

Datos

Un dato es una entidad que aporta un valor numérico, que puede expresarse en distintas bases o incluso a través de una cadena.

Para representar números tenemos 4 bases:

- La más habitual es en su forma decimal, la cual no lleva ningún delimitador especial.

```
mov r0, #90
```

- Luego tenemos otra muy útil que es la representación hexadecimal, que indicaremos con el prefijo 0x.

```
mov r1, #0xff
```

- Otra interesante es la binaria, que emplea el prefijo 0b antes del número en binario.

```
mov r2, #0b11111111
```

- La cuarta y última base es la octal, que usaremos en raras ocasiones y se especifica con el prefijo 0. Sí, un cero a la izquierda de cualquier valor convierte en octal dicho número. Por ejemplo 015 equivale a 13 en decimal.

```
mov r3, #015
```

Todas estas bases pueden ir con un signo menos delante, codificando el valor negativo en complemento a dos. Para representar carácteres y cadenas emplearemos las comillas simples y las comillas dobles respectivamente.

Ver más información sobre los tipos de datos que podemos encontrarnos en el lenguaje ensamblador de ARM en la sección Tipos de Datos

Directivas

Las directivas son expresiones que aparecen en el módulo fuente e indican al compilador que realice determinadas tareas en el proceso de compilación, como delimitar secciones, insertar datos, crear macros, constantes simbólicas, etc… Las instrucciones se aplican en tiempo de ejecución mientras que las directivas se aplican en tiempo de ensamblado. Son fácilmente distinguibles de las instrucciones porque siempre comienzan con un punto.

El uso de directivas es aplicable sólo al entorno del compilador, por tanto varían de un compilador a otro y para diferentes versiones de un mismo compilador. Las directivas más frecuentes en el **as** son:

- *Directivas de asignación:* Se utilizan para dar valores a las constantes o reservar posiciones de memoria para las variables (con un posible valor inicial). **.byte, .hword, .word, .ascii, .asciz, .zero y .space** son directivas que indican al compilador que reserve memoria para las variables del tipo indicado. Por ejemplo:

```
a1:    .byte 1             /* tipo byte, inicializada a 1 */
var2 : .byte 'A'           /* tipo byte, al caracter 'A' */
var3 : .hword 25000        /* tipo hword (16 bits ) a 25000 */
var4 : .word 0x12345678    /* tipo word de 32 bits */
b1:    .ascii " hola "     /* define cadena normal */
```

```
b2:    .asciz " chau "     /* define cadena acabada en NUL */
dat1 : .zero 300           /* 300 bytes de valor cero */
dat2 : .space 200, 4       /* 200 bytes de valor 4 */
```

La directiva **.equ** (ó .set) es utilizada para asignar un valor a una constante simbólica:

```
.equ N, -3 /* en adelante N se sustituye por -3 */
```

- *Directivas de control:*

.text y .data sirven para delimitar las distintas secciones de nuestro módulo.

.align alineamiento es para alinear el siguiente dato, rellenando con ceros, de tal forma que comience en una dirección múltiplos del número que especifiquemos en alineamiento, normalmente potencia de 2. Si no especificamos alineamiento por defecto toma el valor de 4 (alineamiento a palabra):

```
a1: .byte 25   /* definimos un byte con el valor 25 */
    .align    /* directiva que rellena con 3 bytes */
a2: .word 4   /* variable alineada a tama ño palabra */
```

.include para incluir un archivo fuente dentro del actual. **.global** hace visible al enlazador el símbolo que hemos definido con la etiqueta del mismo nombre.

- *Directivas de operando:* Se aplican a los datos en tiempo de compilación. En general, incluyen las operaciones lógicas &, |, □, aritméticas +, -, *, /, % y de desplazamiento <, >, «, »:

```
.equ pies, 9            /* definimos a 9 la constante pies */
.equ yardas, pies /3    /* calculamos las yardas = 3 */
.equ pulgadas, pies *12 /* calculamos pulgadas = 108 */
```

- *Directivas de Macros*: Una *.macro* es un conjunto de sentencias en ensamblador (directivas e instrucciones) que pueden aparecer varias veces repetidas en un programa con algunas modificaciones (opcionales). Por ejemplo, supongamos que a lo largo de un programa realizamos varias veces la operación n^2+1 donde n y el resultado son registros. Para acortar el código a escribir podríamos usar una macro como la siguiente:

```
.macro CuadM1 input, aux, output
       mul aux, input, input
       add output, aux, #1
.endm
```

Esta macro se llama CuadM1 y tiene tres parámetros (input, aux y output). Si posteriormente usamos la macro de la siguiente forma:

```
CuadM1 r1, r8, r0
```

el ensamblador se encargará de expandir la macro, es decir, en lugar de la macro coloca:

```
mul r8, r1, r1
add r0, r8, #1
```

No hay que confundir las macros con los procedimientos. Por un lado, el código de un procedimiento es único, todas las llamadas usan el mismo, mientras que el de una macro aparece (se expande) cada vez que se referencia, por lo que ocuparán más memoria. Las macros serán más rápidas en su ejecución, pues es secuencial, frente a los procedimientos, ya que implican un salto cuando aparece la llamada y un retorno cuando se termina. La decisión de usar una macro o un procedimiento dependerá de cada situación en concreto, aunque las macros son muy flexibles (ofrecen muchísimas más posibilidades de las comentadas aquí).

Símbolos

Los símbolos son representaciones abstractas que el ensamblador maneja en tiempo de ensamblado pero que en el código binario resultante tendrá un valor numérico concreto. Hay tres tipos de símbolos: las etiquetas, las macros y las constantes simbólicas.

Como las etiquetas se pueden ubicar tanto en la sección de datos como en la de código, la versatilidad que nos dan las mismas es enorme. En la zona de datos, las etiquetas pueden representar variables, constantes y cadenas. En la zona de código podemos usar etiquetas de salto, funciones y punteros a zona de datos.

Las macros y las constantes simbólicas son símbolos cuyo ámbito pertenece al preprocesador, a diferencia de las etiquetas que pertenecen al del ensamblador. Se especifican con las directivas .macro y .equ respectivamente y permiten que el código sea más legible y menos repetitivo.

Instrucciones

Las instrucciones del as (a partir de ahora usamos as para referirnos al ensamblador) responden al formato general:

```
   Etiqueta : Nemónico Operando/s      /* Comentario */
```

De estos campos, sólo el nemónico (nombre de la instrucción) es obligatorio. En la sintaxis del **as** cada instrucción ocupa una línea terminando preferiblemente con el ASCII 10 (LF), aunque son aceptadas las 4 combinaciones: CR, LF, CR LF y LF CR. Los campos se separan entre sí por al menos un carácter espacio (ASCII 32) o un tabulador y no existe distinción entre mayúsculas y minúsculas.

```
   main : ldr r1, puntero_var1 /* r1 <- & var1 */
```

El Campo *etiqueta*, si aparece, debe estar formado por una cadena alfanumérica. La cadena no debe comenzar con un dígito y no se puede utilizar como cadena alguna palabra reservada del **as** ni nombre de registro del microprocesador. En el ejemplo, la etiqueta es **main:**.

El campo *Nemónico* (ldr en el ejemplo) es una forma abreviada de nombrar la instrucción del procesador. Está formado por caracteres alfabéticos (entre 1 y 11 caracteres).

El campo *Operando/s* indica dónde se encuentran los datos. Puede haber 0, 1 ó más operandos en una instrucción. Si hay más de uno normalmente al primero se le denomina destino (salvo excepciones como str) y a los demás fuentes, y deben ir separados por una coma. Los operandos pueden ser registros, etiquetas, valores inmediatos o incluso elementos más complejos como desplazadores/rotadores o indicadores de pre/post-incrementos. En cualquiera de los casos el tamaño

debe ser una palabra (32 bits), salvo contadas excepciones como ldr y str donde puede ser media palabra (16 bits) o un byte (8 bits). En el ejemplo r1 es el operando destino, de tipo registro, y puntero_var1 es el operando fuente, una etiqueta. Tanto r1 como puntero_var1 hacen referencia a un valor de tamaño palabra (32 bits).

Para más detalles sobre el nemónico y los operandos ver Formato de las instrucciones de Ensamblador ARM)

El campo *Comentario* es opcional (r1 ← &var1, en el ejemplo) y debe comenzar con la secuencia /* y acabar con */ al igual que los comentarios multilínea en C. No es obligatorio que estén a la derecha de las instrucciones, aunque es lo habitual. También es común verlos al comienzo de una función (ocupando varias líneas) para explicar los parámetros y funcionalidad de la misma.

Cada instrucción del **as** se refiere a una operación que puede realizar el microprocesador. También hay pseudoinstrucciones que son tratadas por el preprocesador como si fueran macros y codifican otras instrucciones, como lsl rn, #x que codifica mov rn, rn, lsl #x, o bien push/pop que se traducen instrucciones stm/ldm más complejas y difíciles de recordar para el programador. Podemos agrupar el conjunto de instrucciones del **as**, según el tipo de función que realice el microprocesador, en las siguientes categorías:

- *Instrucciones de transferencia de datos* Mueven información entre registros y posiciones de memoria. En la arquitectura ARMv6 no existen puertos ya que la E/S está mapeada en memoria. Pertenecen a este grupo las siguientes instrucciones: **mov, ldr, str, ldm, stm, push, pop.**
- *Instrucciones aritméticas.* Realizan operaciones aritméticas sobre números binarios o BCD. Son instrucciones de este grupo **add, cmp, adc, sbc, mul.**
- *Instrucciones de manejo de bits.* Realizan operaciones de desplazamiento, rotación y lógicas sobre registros o posiciones de memoria. Están en este grupo las instrucciones: **and, tst, eor, orr, LSL, LSR, ASR, ROR, RRX.**
- *Instrucciones de transferencia de control.* Se utilizan para controlar el flujo de ejecución de las instrucciones del programa. Tales como **b, bl, bx, blx** y sus variantes condicionales.

(3)