ARM: Tercera Práctica

Uso de memoria y directivas

Departamento de Ingeniería en Automática, Electrónica, Arquitectura y Redes de Computadores

Universidad de Cádiz

Índice

Introducción

En esta práctica daremos el último de los tres tipos principales de instrucciones, las instrucciones de transferencia de datos. También veremos algunas directivas que necesitaréis utilizar en esta práctica.

Instrucciones de transferencia de datos

Este tipo de instrucciones sirve para mover datos entre registros y memoria principal. Hay tres tipos de instrucciones de transferencia de datos:

- **Instrucciones de carga-almacenamiento de un registro:** proveen la forma más flexible de transferir datos entre los registros y la memoria, pudiendo ser el dato una palabra (4 bytes), una media palabra (2 bytes, aunque no es posible en procesadores más antiguos), o un byte.
- **Instrucciones de carga-almacenamiento de varios registros:** menos flexibles que las anteriores (sólo funcionan con palabras), pero permiten transferir cantidades mayores de datos de forma eficiente.
- **Instrucciones de intercambio de un registro:** intercambian el valor de un registro con el de una dirección

Instrucciones de transferencia de datos

Este tipo de instrucciones sirve para mover datos entre registros y memoria principal. Hay tres tipos de instrucciones de transferencia de datos:

- **Instrucciones de carga-almacenamiento de un registro:** proveen la forma más flexible de transferir datos entre los registros y la memoria, pudiendo ser el dato una palabra (4 bytes), una media palabra (2 bytes, aunque no es posible en procesadores más antiguos), o un byte.
- **Instrucciones de carga-almacenamiento de varios registros:** menos flexibles que las anteriores (sólo funcionan con palabras), pero permiten transferir cantidades mayores de datos de forma eficiente.
- **Instrucciones de intercambio de un registro:** intercambian el valor de un registro con el de una dirección

Instrucciones de transferencia de datos

Este tipo de instrucciones sirve para mover datos entre registros y memoria principal. Hay tres tipos de instrucciones de transferencia de datos:

- **Instrucciones de carga-almacenamiento de un registro:** proveen la forma más flexible de transferir datos entre los registros y la memoria, pudiendo ser el dato una palabra (4 bytes), una media palabra (2 bytes, aunque no es posible en procesadores más antiguos), o un byte.
- **Instrucciones de carga-almacenamiento de varios registros:** menos flexibles que las anteriores (sólo funcionan con palabras), pero permiten transferir cantidades mayores de datos de forma eficiente.
- **Instrucciones de intercambio de un registro:** intercambian el valor de un registro con el de una dirección

Índice

Inicializando un puntero a dirección

Para cargar o almacenar desde una dirección de memoria, debemos **inicializar un registro en el que almacenar dicha dirección**. Contamos con la pseudoinstrucción **ADR**:

```
COPY    ADR r1, TABLA; r1 apunta a TABLA
TABLA                ; sección de código a la que ap
```

Con la instrucción ADR, hemos conseguido que el registro **r1 contenga la dirección de los datos que siguen a la etiqueta TABLA**.
Por lo general utilizaremos esta instrucción para señalar a bloques de memoria reservados por las directivas DCB, DCW y DCD, que veremos más tarde.

Índice

Instrucciones de carga/almacenamiento de un registro

Estas instrucciones calculan una dirección para la transferencia usando un registro base y un offset, y **cargan el contenido de la dirección obtenida en el registro indicado**:

```
 LDR r0, [r1] ; r0 := mem32[r1]
STR r0, [r1] ; mem32[r1] := r0
```

Si quisiéramos coger más datos de una sección de memoria y/o añadir más datos en otra sección, tendríamos que sumarle el tamaño de nuestros datos (palabras, medias palabras o bytes) en bytes al registro base.

Nota: Añadiendo una B a LDR o STR, estaremos trabajando con datos del tamaño de un byte, y no de una palabra.

Ejemplo:

```
COPY ADR r1, TAB1  ; r1 apunta a TAB1
     ADR r2, TAB2  ; r2 apunta a TAB2
     LDR r0, [r1]  ; cargamos el 1er valor en r0
     STR r0, [r2]  ; y lo almacenamos en TAB1
     ADD r1, r1, #4; avanzamos una palabra en TAB1
     ADD r2, r2, #4; avanzamos una palabra en TAB2
     LDR r0, [r1]  ; cargamos el 2o valor en r0
     STR r0, [r2]  ; y lo almacenamos en la TAB2
TAB1               ; fuente de los datos
TAB2               ; destino
```

Direccionamiento de base + offset

Aunque la forma de usar direcciones del ejemplo anterior funciona en todos los casos, ARM cuenta con más modos de direccionamiento que permiten hacer un código más eficiente.
Si el registro base no contiene exactamente la **dirección de memoria** que necesitamos, **podemos sumarle o restarle en el momento un offset** de hasta 4 KB usando un modo de direccionamiento **pre-indexado**, de la siguiente forma:

```
LDR r0, [r1, #4] ; r0 := mem32[r1+4]
```

Auto pre-indexado

También podemos usar el direccionamiento **pre-indexado con autoindexado**, que **tras calcular la dirección de memoria** rehace el mismo cálculo para **actualizar el registro base**. Este modo se utiliza de la siguiente forma:

```
LDR r0, [r1, #4]! ; r0 := mem32[r1+4]
                  ; r1 := r1 + 4
```

Post-indexado

En el post-indexado, la dirección almacenada en el registro **se actualiza tras el acceso a memoria**:

```
LDR r0, [r1], #4   ; r0 := mem32[r1]
                   ; r1 := r1 + 4
```

Índice

Transferencia de datos de múltiples registros

Cuando tengamos que transferir cantidades considerables de datos es aconsejable utilizar estas instrucciones, que **transfieren varios registros a la vez**. Se trata de las instrucciones LDMIA y STMIA, que sólo permiten transferencias de **palabras completas**.
LDMIA y STMIA funcionan por **incremento post-indexado**. También están **LDMIB y STMIB (incremento pre-indexado)**, **LDMDA y STMDA (decremento post-indexado)** y **LDMDB y STMDB (decremento pre-indexado)**.

Ejemplo

```
LDMIA r1, {r0,r2,r5} ; r0 := mem32[r1]
                     ; r2 := mem32[r1+4]
                     ; r5 := mem32[r1+8]
STMIA r1, {r0,r2,r5} ; mem32[r1] := r0
                     ; mem32[r1+4] := r2
                     ; mem32[r1+8] := r5
```

Para poder escribir programas completos, necesitaremos usar ciertas **directivas**. A continuación veremos las directivas de uso más frecuente, muchas de ellas fundamentales para escribir programas completos.

Índice

AREA

Los **programas** en ensamblador de la arquitectura ARM suelen **empezar con la directiva AREA**, que marca el inicio de una sección de una aplicación o programa.
AREA le da un **nombre** a la sección de código e indica sus **atributos**, indicándose en primer lugar el nombre y señalando posteriormente los atributos separados por comas. A continuación veremos los atributos imprescindibles para la práctica.

- **CODE:** La sección contiene instrucciones. Por defecto este atributo implica que la sección tiene el atributo READONLY, explicado más en adelante.
- **DATA:** La sección contiene sólo datos, no instrucciones. Por defecto implica el atributo READWRITE.
- **READONLY:** Indica que de esta sección sólo se puede leer código.
- **READWRITE:** Indica que de esta sección se puede tanto leer como escribir.

Usa la directiva AREA para **subdividir los ficheros de código en secciones**. Por ejemplo, se debe subdividir un fichero para separar las secciones de datos y de código del mismo.

- **CODE:** La sección contiene instrucciones. Por defecto este atributo implica que la sección tiene el atributo READONLY, explicado más en adelante.
- **DATA:** La sección contiene sólo datos, no instrucciones. Por defecto implica el atributo READWRITE.
- **READONLY:** Indica que de esta sección sólo se puede leer código.
- **READWRITE:** Indica que de esta sección se puede tanto leer como escribir.

Usa la directiva AREA para **subdividir los ficheros de código en secciones**. Por ejemplo, se debe subdividir un fichero para separar las secciones de datos y de código del mismo.

- **CODE:** La sección contiene instrucciones. Por defecto este atributo implica que la sección tiene el atributo READONLY, explicado más en adelante.
- **DATA:** La sección contiene sólo datos, no instrucciones. Por defecto implica el atributo READWRITE.
- **READONLY:** Indica que de esta sección sólo se puede leer código.
- **READWRITE:** Indica que de esta sección se puede tanto leer como escribir.

Usa la directiva AREA para **subdividir los ficheros de código en secciones**. Por ejemplo, se debe subdividir un fichero para separar las secciones de datos y de código del mismo.

- **CODE:** La sección contiene instrucciones. Por defecto este atributo implica que la sección tiene el atributo READONLY, explicado más en adelante.
- **DATA:** La sección contiene sólo datos, no instrucciones. Por defecto implica el atributo READWRITE.
- **READONLY:** Indica que de esta sección sólo se puede leer código.
- **READWRITE:** Indica que de esta sección se puede tanto leer como escribir.

Usa la directiva AREA para **subdividir los ficheros de código en secciones**. Por ejemplo, se debe subdividir un fichero para separar las secciones de datos y de código del mismo.

Índice

RN

La directiva RN es utilizada para **definir un nombre para un determinado registro**, y así recordar mejor la función que se le va a dar. El formato de la directiva es:

```
nombre RN expresión
```

donde `expresión` es un valor entre 0 y 15 correspondientes a los registros r0 a r15. Ejemplo:

```
contPrograma RN 15
```

Índice

EQU

Otorga un nombre simbólico a una constante numérica o a un valor relativo a un registro o un programa. La sintaxis es la siguiente:

```
nombre EQU expresión, tipo
```

donde `expresión` es una dirección relativa a un registro, a un programa, una dirección absoluta, o una constante entera de 32 bits. El parámetro `tipo` es opcional y puede ser una de las siguientes opciones: CODE16, CODE32, DATA, pero no nos será de mucha utilidad por el momento.

Índice

ENTRY

La directiva ENTRY **declara un punto de entrada a un programa**. Debe especificarse **un punto de entrada para cada fichero de código**. Se declara un punto de entrada a un programa escribiendo simplemente `ENTRY`.

Índice

DCB

La directiva DCB **asigna uno o más bytes de memoria y define el contenido inicial de la memoria**. La sintaxis es:

```
etiqueta DCB expresión, expresión...
```

donde `expresión` es una expresión numérica que devuelve un entero entre -128 y 255 o una cadena de caracteres especificada entre comillas.
Si se va a utilizar una instrucción justo después de esta directiva, se debe utilizar una directiva ALIGN para asegurar que la instrucción sea alineada.

DCW

DCW realiza la misma tarea de DCB, pero asignando medias palabras (halfwords-2 bytes). Si escribimos DCWU, el alineamiento de memoria será arbitrario. La sintaxis es la siguiente:

```
etiqueta DCWU expresión, expresión...
```

donde `expresión` puede tomar valores entre -32768 y 65535.

DCD

DCD, en cambio, **trabaja con palabras completas** (4 bytes). `expresión` puede ser una expresión numérica o relativa al programa. DCD inserta hasta 3 bytes de relleno antes de la primera palabra definida, si es necesario, para alcanzar el alineamiento de 4 bytes. Usa DCDU si no necesitas alineamiento. El formato aparece a continuación:

```
etiqueta DCDU expresión, expresión
```

Índice

END

La directiva END informa al ensamblador de que se ha alcanzado el **final de un fichero**. **Todo fichero en lenguaje ensamblador debe terminar con la directiva END** escrita en una línea por separado.