**DESARROLLO TALLER 5 PROCESADORES Y ENSAMBLADOR**

**DIEGO ARMANDO CALERO GALVIS**

**DANIEL BONILLA ECHEVERRY**

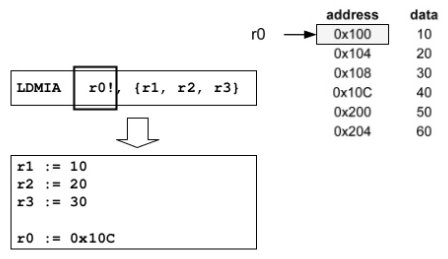
1. Las instrucciones con las cuales se transfiere bloques de registros hacia la memoria o se leen bloques de datos hacia los registros son LDM (Carga múltiples registros) y STM (Almacena múltiples registros) los cuales sirven para la transferencia de múltiples registros en una sola instrucción y las cuales tienen el siguiente formato.



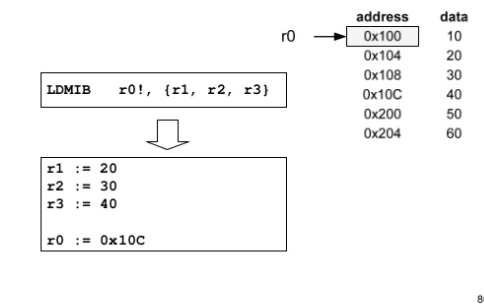
El simbolo “!” en la instrucción denota que existe write back en el registro base , el cual es r0 en los ejemplos anteriores.

Los modos de direccionamiento para ldm y stm son los siguientes:

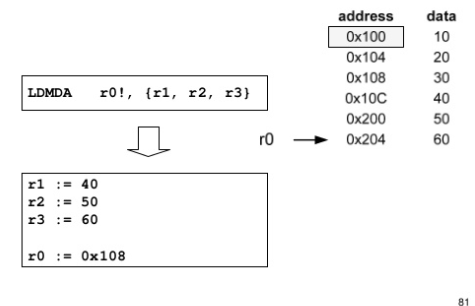
1. **ia**: Incrementa dirección después (increment after) de cada transferencia. Es el modo por defecto en caso de omitirlo.



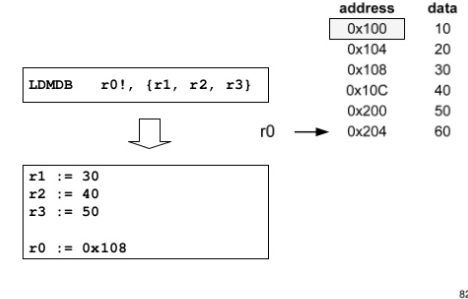
1. **ib**: Incrementa dirección antes (increment before) de cada transferencia.



1. **da**: Decrementa después (decrement after) de cada transferencia.



1. **db**: Decrementa antes (decrement before) de cada transferencia.



1. La pila : Se denomina pila de programa a aquella zona de memoria, organizada de forma LIFO (Last In, First Out), que el programa emplea principalmente para el almacenamiento temporal de datos. Esta pila, definida en memoria, es fundamental para el funcionamiento de las rutinas.

El puntero de pila es r13 aunque por convención nunca se emplea esa nomeclatura, sino que lo llamamos sp (stack pointer o puntero de pila). Dicho registro apunta siempre a la palabra de memoria que corresponde a la cima de la pila (última palabra introducida en ella).

1. Se puede definir el sentido de la pila en ARM con las instrucciones push (orden ascendente) y pop (orden descendente).

La pila tiene asociadas dos operaciones: push (meter un elemento en la pila) y pop (sacar un elemento de la pila). En la operación push primero decrementamos en 4 (una palabra son 4 bytes) el registro sp y luego escribimos dicho elemento en la posición de memoria apuntada por sp. Decimos que la pila crece hacia abajo, ya que cada vez que insertamos un dato el sp se decrementa en 4.

De esta forma, la instrucción push realmente implementa las dos siguientes instrucciones:



Para sacar elementos de la pila tenemos la operación pop, que primero extrae el elemento de la pila y luego incrementa el puntero (la pila decrece hacia arriba). Por tanto, la instrucción pop es equivalente a:



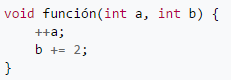
Un uso muy común de la pila es salvaguardar una serie de registros, que queremos usar para hacer las operaciones que necesitemos pero que al final tenemos que restaurar a sus valores originales.

1. Para el procesador ARM las instrucciones push y pop no existen. Sin embargo tenemos las instrucciones stm y ldm que son mucho más potentes y el ensamblador permite las pseudoinstrucciones push y pop que de forma transparente traducirá a stm y ldm.
2. Las subrutinas (funciones o procedimientos) son abstracciones que se usan en los lenguajes de alto nivel para simplificar el código y poder reusarlo. A la subrutina se la llama (invoca) desde el programa invocante mediante una instrucción particular. Pero antes de llamar a la subrutina tenemos que haber colocado los datos de entrada a ésta en un lugar accesible (registros reservados o pila). En general al trabajar con subrutinas se divide el trabajo a realizar entre el programa invocante y la subrutina.

**Ventajas de usar subrutinas**

* Evitan la repetición de código, puesto que una subrutina puede ser invocada varias veces desde el mismo programa.
* Favorecen la reutilización de código.
* Facilitan la lectura y comprensión de código fuente al ocultar los detalles menos importantes, siempre y cuando los nombres de las subrutinas sean informativos de su función.
* Ayudan a corregir errores, puesto que un error en una subrutina sólo se corrige una vez aunque esa subrutina sea llamada en diferentes puntos del programa principal.
  1. Una de las formas más sencillas de transferir los parámetros entre el programa llamante y la subrutina es utilizar los registros de propósito general del computador. Para ello el programa que realiza la llamada debe cargar en un conjunto de registros previamente “acordados” los datos sobre los que va a procesar la subrutina y posteriormente hacer la llamada a la subrutina. La subrutina que se ha llamado realizará los cálculos utilizando los valores que el llamante ha cargado en los registros de entrada y almacenará los resultados de la subrutina en el banco de registros del computador.

Paso de parámetros (argumentos) por valor: cuando es por **valor**, la información de la variable se almacenan en una dirección de memoria diferente al recibirla en la función, por lo tanto si el valor de esa variable cambia no afecta la variable original, solo se modifica dentro del contexto de la función.



Paso de parámetros (argumentos) por referencia : cuando es por **referencia**, la variable que se recibe como parámetro en la función apunta exactamente a la misma dirección de memoria que la variable original por lo que si dentro de la función se modifica su valor también se modifica la variable original.



* 1. Convención AAPCS:

Podemos seguir nuestras propias reglas, pero si queremos interactuar con las librerías del sistema, tanto para llamar a funciones como para crear nuestras propias funciones y que éstas sean invocadas desde un lenguaje de alto nivel, tenemos que seguir una serie de pautas, lo que se denominamos AAPCS (Procedure Call Standard for the ARM Architecture).

1. Podemos usar hasta cuatro registros (desde r0 hasta r3) para pasar parámetros y hasta dos (r0 y r1) para devolver el resultado.

2. No estamos obligados a usarlos todos, si por ejemplo la función sólo usa dos parámetros de tipo int con r0 y r1 nos basta. Lo mismo pasa con el resultado, podemos no devolver nada (tipo void), devolver sólo r0 (tipo int ó un puntero a una estructura más compleja), o bien devolver r1:r0 cuando necesitemos enteros de 64 bits (tipo long long).

3. Los valores están alineados a 32 bits (tamaño de un registro), salvo en el caso de que algún parámetro sea más grande, en cuyo caso alinearemos a 64 bits. Un ejemplo de esto lo hemos visto en el Ejercicio 2.5, donde necesitábamos pasar dos parámetros: una cadena (puntero de 32 bits) y un entero tipo long long. El puntero a cadena lo almacenábamos en r0 y el entero de 64 bits debe empezar en un registro par (r1 no vale) para que esté alineado a 64 bits, serían los registros r2 y r3. En estos casos se emplea little endian, la parte menos significativa sería r2 y la de mayor peso, por tanto, r3.

4. El resto de parámetros se pasan por pila. En la pila se aplican las mismas reglas de alineamiento que en los registros. La unidad mínima son 32 bits, por ejemplo si queremos pasar un char por valor, extendemos de byte a word rellenando con ceros los 3 bytes más significativos. Lo mismo ocurre con los enteros de 64 bits, pero en el momento en que haya un sólo parámetro de este tipo, todos los demás se alinean a 64 bits.

5. Es muy importante preservar el resto de registros (de r4 en adelante incluyendo lr). La única excepción es el registro r12 que podemos cambiar a nuestro antojo. Normalmente se emplea la pila para almacenarlos al comienzo de la función y restaurarlos a la salida de ésta. Puedes usar como registros temporales (no necesitan ser preservados) los registros desde r0 hasta r3 que no se hayan empleado para pasar parámetros.

6. La pila debe estar alineada a 8 bytes, esto quiere decir que de usarla para preservar registros, debemos reservar un número par de ellos. Si sólo necesitamos preservar un número impar de ellos, añadimos un registro más a la lista dentro del push, aunque no necesite ser preservado.

7. Aparte de para pasar parámetros y preservar registros, también podemos usar la pila para almacenar variables locales, siempre y cuando cumplamos la regla de alinear a 8 bytes y equilibremos la pila antes de salir de la función.