Instrucciones de multiplicación, división

Las instrucciones de multiplicación admiten muchas posibilidades, debido a que es una operación en la cual el resultado tiene el doble de bits que cada operando. En la siguiente tabla vemos las 5 instrucciones de multiplicación que existen.

Instrucción	Bits	Nombre
mul	32=32×32	Multiplicación truncada
umull	64=32×32	Multiplicación sin signo de 32 bits
smull	64=32×32	Multiplicación con signo de 32 bits
smulw<y>	32=32×16	Multiplicación con signo de 32×16 bits
smul<x><y>	32=16×16	Multiplicación con signo de 16×16 bits

mul

La instrucción **mul** realiza una multiplicación truncada, es decir, nos quedamos con los 32 bits inferiores del resultado. Como el signo del resultado es el bit más significativo el cual no obtenemos, esta multiplicación es válida tanto para operandos naturales (sin signo) como para enteros (con signo).

Sintaxis

```
MUL{<cond>}{S}  <Rd>, <Rm>, <Rs>
```

En el ejemplo de abajo **r0 = parte_baja(r1*r2)**:

```
mul r0, r1, r2
```

umull y smull

Las dos siguientes multiplicaciones (umull y smull) son largas, por eso la **l** del final, donde el resultado es de 64 bits. Si los operandos son naturales escogemos la multiplicación sin signo (unsigned) umull. Por el contrario, si tenemos dos enteros como factores hablamos de multiplicación con signo (signed) smull.

Sintaxis

```
UMULL{<cond>}{S}  <RdLo>, <RdHi>, <Rm>, <Rs>
```

Donde:

<RdLo> Almacena los 32 bits más bajos del resultado.

<RdHi> Almacena los 32 bits más altos del resultado.

En ambos ejemplos la parte baja del resultado se almacena en **r0**, y la parte alta en **r1**.

Para hacer que **r1:r0 = r2*r3:**

```
umull r0, r1, r2, r3
smull r0, r1, r2, r3
```

smulw<y>

Ahora veamos **smulw<y>** . Es con signo, y el sufijo <y> puede ser una b para seleccionar la parte baja del registro del segundo factor, o una t para seleccionar la alta.

Sintaxis:

```
SMULW<y>{<cond>}  <Rd>, <Rm>, <Rs>
```

Donde

<y> Especifica que mitad del registro <Rs> es usada como operando para multiplicar.

Si <y> es B, la parte más baja (bits[15:0]) de <Rs> es usada.

Si <y> is T, la parte más alta (bits[31:16]) de <Rs> es usada.

Según el ejemplo **r0 = r1*parte_baja(r2)**.

```
smulwb r0, r1, r2
```

smul<x><y>

Por último tenemos smul<x><y> también con signo, donde se seleccionan partes alta o baja en los dos factores, puesto que ambos son de 16 bits.

Sintaxis

```
SMUL<x><y>{<cond>}  <Rd>, <Rm>, <Rs>
```

Donde

<x> Especifica que mitad del registro <Rm> es usada como primer operando de la multiplicación.

Si <x> es B, la parte más baja (bits[15:0]) de <Rm> es usada.

Si <x> es T, la parte más alta (bits[31:16]) de <Rm> es usada.

<y> Especifica que mitad del registro <Rs> es usada como segundo operando para la multiplicación.

Si <y> es B, la parte más baja (bits[15:0]) de <Rs> es usada.

Si <y> es T, la parte más alta (bits[31:16]) de <Rs> es usada.

En el ejemplo **r0 =parte_alta(r1)*parte_baja(r2)**.

```
   smultb r0, r1, r2
```

En los dos últimos tipos smulw<y> y smul<x><y> no se permite el sufijo s para actualizar los flags.

Ejercicio: Completá los recuadros en blanco con los resultados en hexadecimal. Luego compila el código de abajo y comprobá con gdb que los cálculos anteriores son correctos.

Producto	Factor1	Factor2
mul		
umul		
smul		
smulwb		
smultt		

```
      .data
 var1 : .word 0x12341111
 var2 : .word 0x87652222
 @
 .text
 .global main
  main :
       ldr r0, = var1   /* r0 <- & var1 */
       ldr r1, = var2   /* r1 <- & var2 */
       //
       ldr r3, [r0]           /* r3 <- var1 */
       ldr r4, [r1]           /* r4 <- var2 */
       smulbb r5, r3, r4      /* r5 <- baja(var1)*baja(var2)*/
                              /* r5=0x2468642 */
       //
       ldrh r3, [r0]    /* r3 <- baja(var1) */
       ldrh r4, [r1]    /* r4 <- baja(var2) */
       muls r5, r3, r4  /* r5 <- baja(var1)*baja (var2) */
                        /* r5=0x2468642 */
       ldr r3, [r0]           /* r3 <- var1 */
       ldr r4, [r1]           /* r4 <- var2 */
       umull r5, r6, r3, r4   /* r6:r5 <- var1*var2  sin signo*/
       smull r5, r6, r3, r4   /* r6:r5 <- var1*var2  con signo */
  fin:
       mov r7, #1
       swi 0
```

división

No existe la operación/instrucción división en ensamblador arm v6, deberemos programarla .

Desarrollaremos un programa en código ensamblador ARM que divida dos números tamaño **word A y B** y escribiremos el resultado en el número **C** mediante restas utilizando el algoritmo de abajo.

Pseudocódigo para realizar la división A/B con restas sucesivas.

```
C = 0
while( A >= B )
{
A = A - B
C = C + 1
}
```

La ausencia de una instrucción de división se debe a que ARMv6 es una arquitectura RISC, y en este tipo de arquitecturas las intrucciones deben ser simples de tal forma de poder ejecutarse en un solo ciclo de instrucción.

De todos modos, la arquitectura ARM se puede extender con el coprocesador de punto flotante llamado VFP (Vector Floating-Point). La arquitectura de este coprocesador admite instrucciones de división en punto flotante, pero nosotros no estudiaremos esta extensión.

Volver

(189)