06 Corrimientos, multiplicación y división

Arquitectura de Computadoras y Ensambladores 1
M.Sc. Luis Fernando Espino Barrios
2024

Instrucciones de corrimiento

Instrucción	Descripción
asr Rd, Rn, Rm	Shift aritmético por la derecha. Rd = sign(Rn>>Rm)
lsr Rd, Rn, Rm	Shift lógico por la derecha. Rd = Rn>>Rm
lsl Rd, Rn, Rm	Shift lógico por la izquierda. Rd = Rn<<Rm
ror Rd, Rn, Rm	Rotación por la derecha

Shift lógico por la izquierda

lsl w0, w0, 7

w0: 0000 0000 0000 0000 0000 0000 0000 0111

w0: 0000 0000 0000 0000 0000 0011 1000 0000

Shift lógico por la derecha

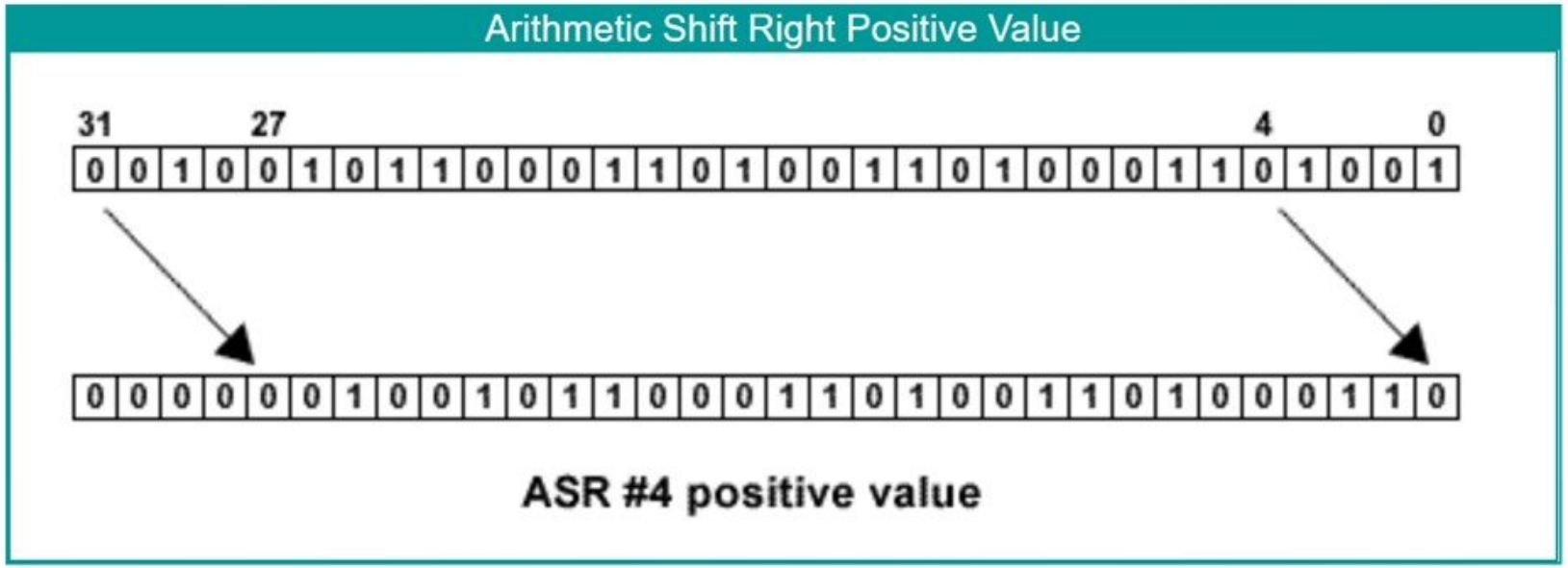
lsr w0, w0, 2

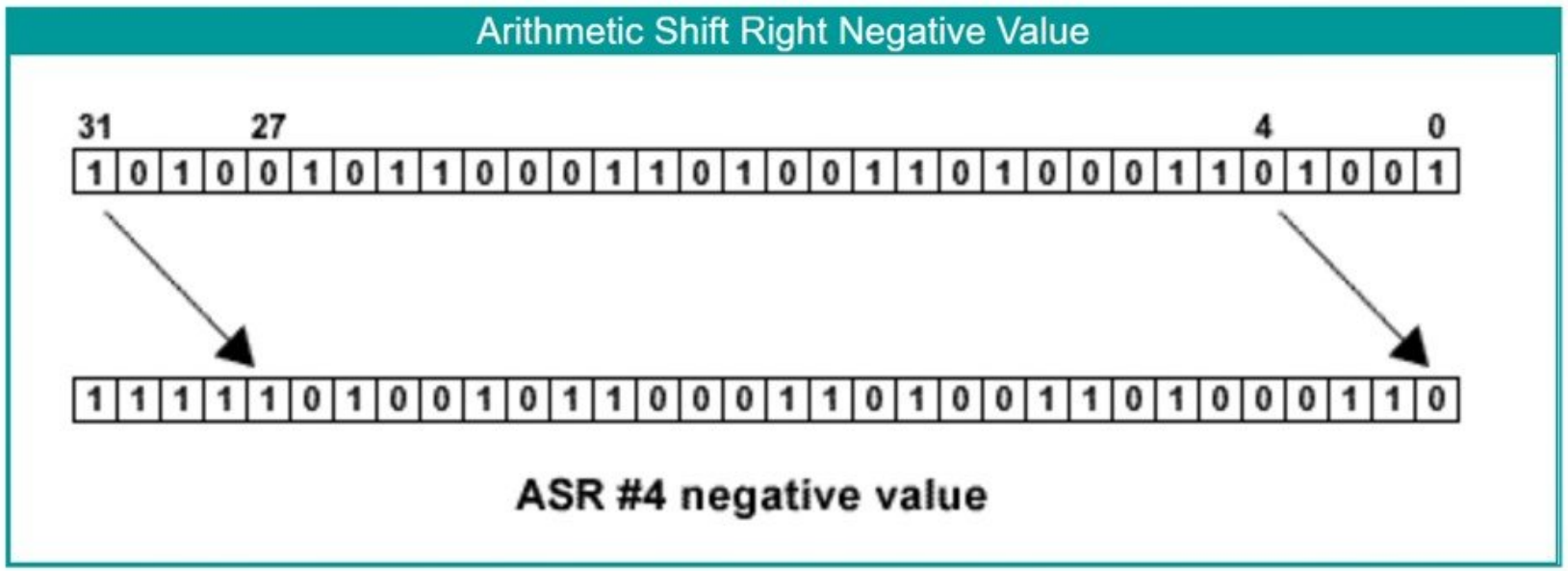
w0: 0000 0000 0000 0000 0000 0100 0000 0000

w0: 0000 0000 0000 0000 0000 0001 0000 0000

Shift aritmético por la derecha

- Se comporta similar que la instrucción de shift lógico, la diferencia radica en el uso con valor sin signo o con signo, es decir, positivos o negativos.
- La idea es preservar la cualidad de positivo o negativo.


Arithmetic Shift Right Positive Value
31
27
4
0
0 0 1 0 0 1 0 1 1 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 1
0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0 1 1 0
ASR #4 positive value


Arithmetic Shift Right Negative Value
31
27
4
0
1 0 1 0 0 1 0 1 1 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 1
1 1 1 1 1 0 1 0 0 1 0 1 1 0 0 0 1 1 0 1 0 0 1 1 0 1 0 0 0 1 1 0
ASR #4 negative value

Rotación por la derecha

ror x0, x0, 2

x0: 0000 .. (56 bits 0’s) .. 0111

x0: 1100 .. (56 bits 0’s) .. 0001

```
.global _start

.data
out: .ascii " - - - \n"

.text
_start:
    ldr x1, =out          // load output

    mov w0, 0b1001        // 9 = 1001
    asr w0, w0, 2         // w0 >> 2
    add w0, w0, 48        // adjust ascii
    strb w0, [x1]         // store 2 = 0010

    mov w0, 0b10000       // 16 = 10000
    lsr w0, w0, 1         // w0 >> 1
    add w0, w0, 48        // adjust ascii
    strb w0, [x1, 2]      // store 8 = 01000
```

```
mov w0, 0b10000     // 16 = 10000
lsr w0, w0, 1       // w0 >> 1
add w0, w0, 48      // adjust ascii
strb w0, [x1, 2]    // store 8 = 01000

mov w0, 0b010       // 2 = 010
lsl w0, w0, 1       // w0 << 1
add w0, w0, 48      // adjust ascii
strb w0, [x1, 4]    // store 4 = 100

mov x0, 0b0001      // 1
ror x0, x0, 61      // ror 61
add w0, w0, 48      // adjust ascii
strb w0, [x1, 6]    // store 8 = 1000

mov x0, 1           // stdout
mov x2, 8           // size
mov x8, 64          // write
svc 0               // syscall

mov x0, x1          // return value
mov x8, 93          // exit
svc 0               // syscall
```

Pregunta

ARM no implementa la instrucción de rotación por izquierda (ROL). ¿Por qué no hay instrucción ROL y cómo se escribe con una operación equivalente?

Rotación por la derecha

ror x0, x0, 2

x0: 0000 .. (56 bits 0’s) .. 0111
x0: 1100 .. (56 bits 0’s) .. 0001

R0 se rotó a la derecha 2 bits.
R0 se rotó a la izquierda (64-2) bits

Instrucciones de multiplicación con overflow de 32 a 32 y 64 a 64 bits

Instrucción	Efecto	Descripción
mult Rd, Rn, Rm	Rd=Rn*Rm	Multiplicación
madd Rd, Rn, Rm, Ra	Rd=Ra+Rn*Rm	Multiplicación y suma
msub Rd, Rn, Rm, Ra	Rd=Ra-Rn*Rm	Multiplicación y resta
mneg Rd, Rn, Rm	Rd=-(Rn*Rm)	Multiplicación y negación

Instrucciones de multiplicación de 32 a 64 bits

Name	Effect	Description
`smull`	$\texttt{Xd} \leftarrow signExtend(\texttt{Wn}) \times signExtend(\texttt{Wm})$	Signed multiply long.
`smaddl`	$\texttt{Xd} \leftarrow \texttt{Xa} + signExtend(\texttt{Wn}) \times signExtend(\texttt{Wm})$	Signed multiply add long.
`smsubl`	$\texttt{Xd} \leftarrow \texttt{Xa} - signExtend(\texttt{Wn}) \times signExtend(\texttt{Wm})$	Signed multiply subtract long.
`smnegl`	$\texttt{Xd} \leftarrow -(signExtend(\texttt{Wn}) \times signExtend(\texttt{Wm}))$	Signed multiply negate long.
`umull`	$\texttt{Xd} \leftarrow \texttt{Wn} \times \texttt{Wm}$	Unsigned multiply long.
`umaddl`	$\texttt{Xd} \leftarrow \texttt{Ra} + \texttt{Wn} \times \texttt{Wm}$	Unsigned multiply add long.
`umsubl`	$\texttt{Xd} \leftarrow \texttt{Ra} - \texttt{Wn} \times \texttt{Wm}$	Unsigned multiply subtract long.
`umnegl`	$\texttt{Xd} \leftarrow -(\texttt{Wn} \times \texttt{Wm})$	Unsigned multiply negate long.

Instrucciones de multiplicación de 64 a 128 bits

Name	Effect	Description
`smulh`	$Xd \leftarrow (signExtend(Xn) \times signExtend(Xm))[127:64]$	Signed multiply high.
`umulh`	$Xd \leftarrow (Xn \times Xm)[127:64]$	Unsigned multiply high.

Instrucciones de división entera

Instrucción	Efecto	Descripción
sdiv Rd, Rn, Rm	Rd=Rn/Rm	División con signo
udiv Rd, Rn, Rm	Rd=Rn/Rm	División sin signo

Ejercicio

- Devuelva en el valor de retorno la pendiente de la recta de 2 puntos, por ejemplo, (1,2) y (3,4)

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

```
.global _start

_start:
    mov x0, 1               // x1
    mov x1, 2               // y1
    mov x3, 3               // x2
    mov x4, 4               // y2

    sub x5, x4, x1          // y2 - y1
    sub x6, x3, x0          // x2 - x1

    sdiv x0, x5, x6         // slope

    mov x8, 93              // exit
    svc 0                   // syscall
```

Instrucciones de comparación

Name	Effect	Description
`cmp`	Rn − Operand2	Compare and set PSTATE flags.
`cmn`	Rn + Operand2	Compare negative and set PSTATE flags.
`tst`	Rn ∧ Operand2	Test bits and set PSTATE flags.
`teq`	Rn ⊕ Operand2	Test equivalence and set PSTATE flags.

Instrucciones varias

Name	Effect	Description
`clz`	Rd ← *CountLeadingZeros*(Rn)	Count leading zeros in Rn.
`cls`	Rd ← *CountLeadingSignBits*(Rn)	Count leading ones or zeros in Rn.

Name	Effect	Description
`mrs`	`Xt` ← `PSTATE`	Move from Process State.
`msr`	`PSTATE` ← `Xt`	Move to Process State.

Name	Effect	Description
`nop`	No effects	No Operation.

Name	Effect	Description
`svc`	Request Operating System Service	Perform software interrupt.

Bibliografía

Elahi, A. (2022). Fundamentals of Computer Architecture and ARM Assembly Language. Springer.

Harris, S. & Harris, D. (2016). Digital Design and Computer Architecture: ARM Edition. Elsiever Inc.

Mazidi, M. & Otros. (2013). ARM Assembly Language: Programming & Architecture.

Smith, S. (2019). Raspberry Pi Assembly Language Programming: ARM Processor Coding. Apress.