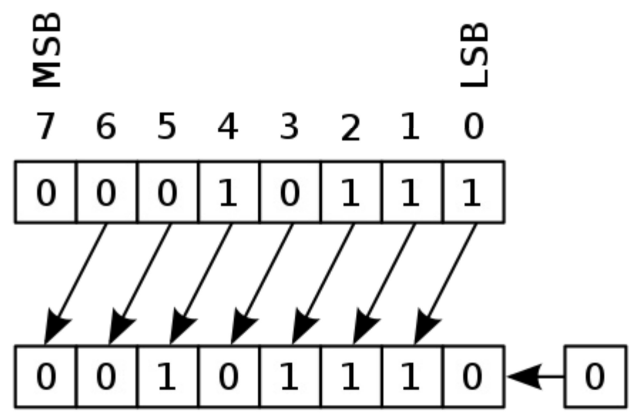
Multiplication

- Each shift to the left doubles the number, therefore each left shift multiplies the original number by 2.

- Example multiply 011 by 010 and 100

```
  011
    ×
  010
-----
  000
 011
000
-----
  110
```

```
   011
     ×
   100
------
   000
  000
 011
------
  1100
```



Signed Multiplication

- Requires special consideration for negative (2's complement) numbers

signed

3 bit multiplication	101	-3
	x 011	x 3
	101	- 9
	101	ERROR
	000	
	001111	15

Signed Multiplication

- In 2’s complement you must sign extend to the product bit width

4b x 4b

								0	1	1	1		7
							x	0	1	1	0	x	6

				0	0	0	0	0	1	1	1		7
			x	0	0	0	0	0	1	1	0	x	6
				0	0	0	0	0	0	0	0		
			0	0	0	0	0	1	1	1			
		0	0	0	0	0	1	1	1				
+	0	0	0	0	0	0	0	0					
	x	x	x	0	0	1	0	1	0	1	0		42

Must carry out all the bits

Stop after 8 bits

Signed Multiplication

- In 2's complement you must sign extend to the product bit width

```
                        4b x 4b
                        1 0 0 1          -7
                      x 1 0 1 0      x   -6

                1 1 1 1 1 0 0 1          -7
              x 1 1 1 1 1 0 1 0      x   -6
                0 0 0 0 0 0 0 0
              1 1 1 1 1 0 0 1
            0 0 0 0 0 0 0 0
          1 1 1 1 1 0 0 1
        1 1 1 1 1 0 0 1
      1 1 1 1 1 0 0 1
    1 1 1 1 1 0 0 1
+ 1 1 1 1 1 0 0 1
              x 0 0 1 0 1 0 1 0          42
```

Must carry out all the bits

Stop after 8 bits

Signed Multiplication

- In 2’s complement you must sign extend to the product bit width
- When doing it by hand - where possible multiply by the positive value

4b x 4b

```
                        0 1 1 1         7
                      x 1 0 1 0      x -6

                0 0 0 0 0 1 1 1         7
              x 1 1 1 1 1 0 1 0      x -6
                0 0 0 0 0 0 0 0
              0 0 0 0 0 1 1 1
            0 0 0 0 0 0 0 0
          0 0 0 0 0 1 1 1
        0 0 0 0 0 1 1 1
      0 0 0 0 0 1 1 1
    0 0 0 0 0 1 1 1
  + 0 0 0 1 0 1 1 1
              x 1 1 0 1 0 1 1 0       -42
```

Must carry out all the bits

Stop after 8 bits

4b x 4b

```
                1 0 1 0        -6
              x 0 1 1 1       x 7

        1 1 1 1 1 0 1 0        -6
      x 0 0 0 0 0 1 1 1       x 7
        1 1 1 1 1 0 1 0
      1 1 1 1 1 0 1 0
    1 1 1 1 1 0 1 0
  + 0 0 0 0 0 0 0 0
x x x 1 1 0 1 0 1 1 0         -42
```

Must carry out all the bits

Others are 0

Stop after 8 bits

Example

- The 6-bit integer ***L*** in the U2 code has the following form: 010011 , Calculate the number -*L**2, add 1 to it, and present the result in ***eight-bit*** format.
- Solution : -L = 101101
- -L * 2

```
 111111 101101
 000000000010
 ____________
 000000000000
111111101101
 ____________
 111111 011010
                +
 000000000001
 ____________
 1111 | 11011011
```

Stop after 8 bits

Or just shift left

Division

- Each shift to the right reduces the number by half
- $0 \longrightarrow 1000 \xrightarrow{SR} 0100 \xrightarrow{SR} 0010$

- Let's say we want to do $1001 \div 11$
- Step 1: Set up the problem $11\overline{)1001}$
- Step 2: Can we get a 11 out of 1? No, put a 0.

$$\begin{array}{r} 0??? \\ 11\overline{)1001} \end{array}$$

- Step 3: Can we get a 11 out of 10? No, put a 0.

$$\begin{array}{r} 00?? \\ 11\overline{)1001} \end{array}$$

Division

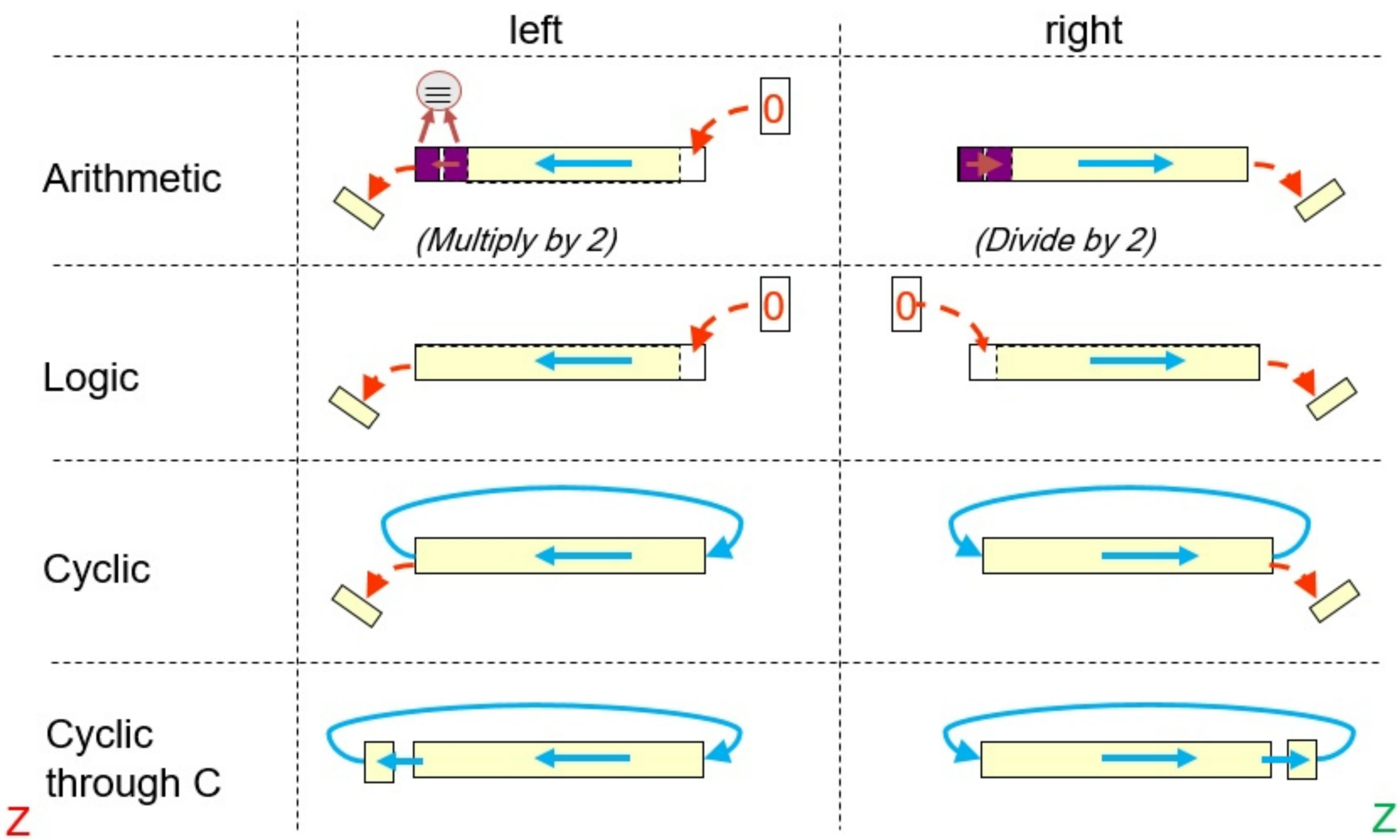
- Step 4: Can we get 11 out of 100? Yes, so it is a 1. So then we multiply 11 x 1 = 11 and we subtract that from 100 (100 – 11) and we get 1.

$$
\begin{array}{r}
001? \\
11\overline{)1001} \\
-\underline{0011} \\
00001
\end{array}
$$

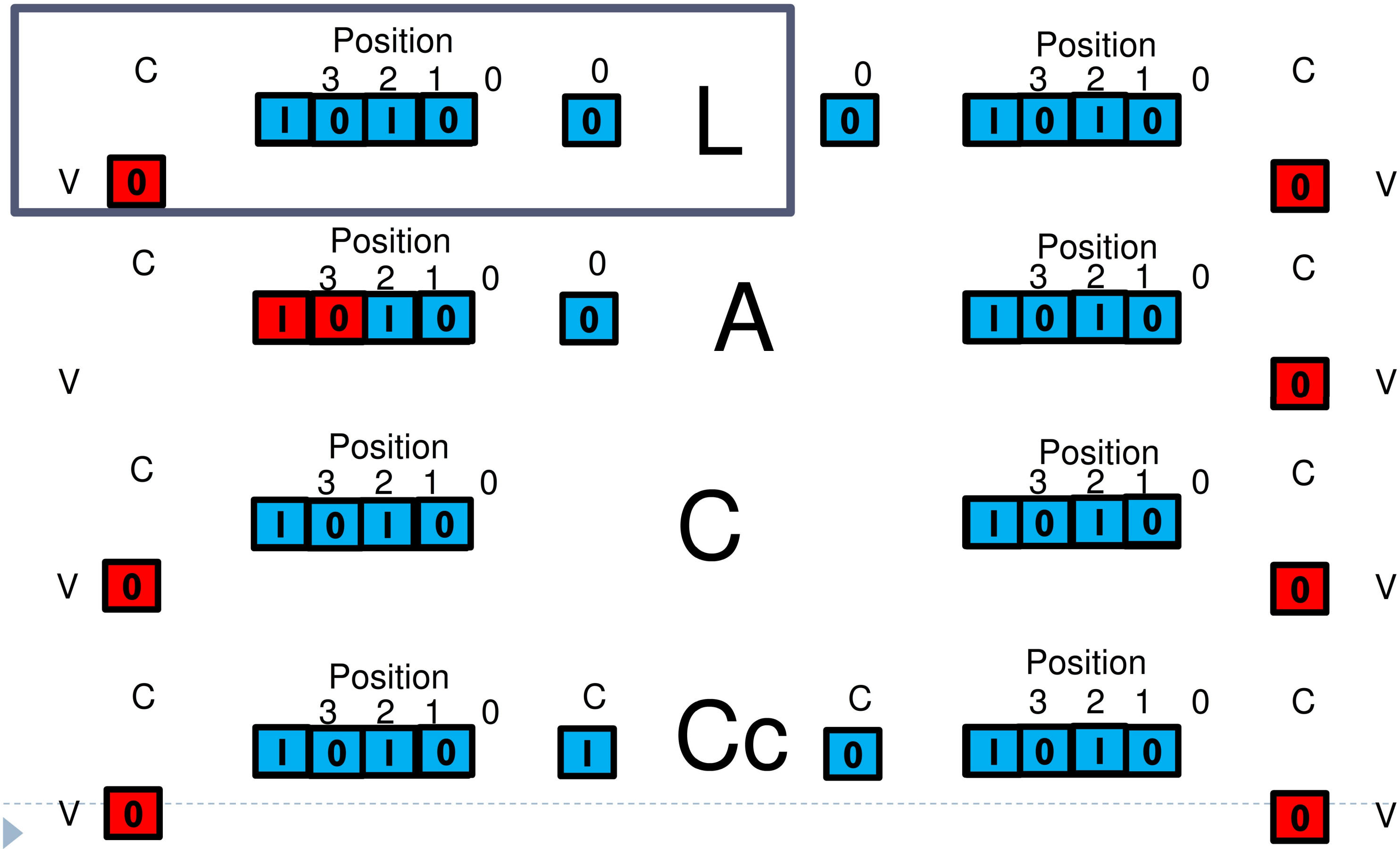
- Step 5: bring down the next 1, to make 11 and we can get one 11 out of 11.

$$
\begin{array}{r}
0011 \\
11\overline{)1001} \\
-\underline{0011} \\
000011 \\
-\underline{00011} \\
000000
\end{array}
$$

binary shift operations by 1 bit



SHIFT RIGHT LEFT



Example

- Enter the result of the LSL (logical shift left) shift of the 4-bit register: 0111
- Before shifting conditions C=1,V=1.
- Where C and V are the condition bits after shift (carry and overflow)

- Solution :
- 0111 $\xrightarrow{\text{LSL}}$ 1110 C=0 V=0