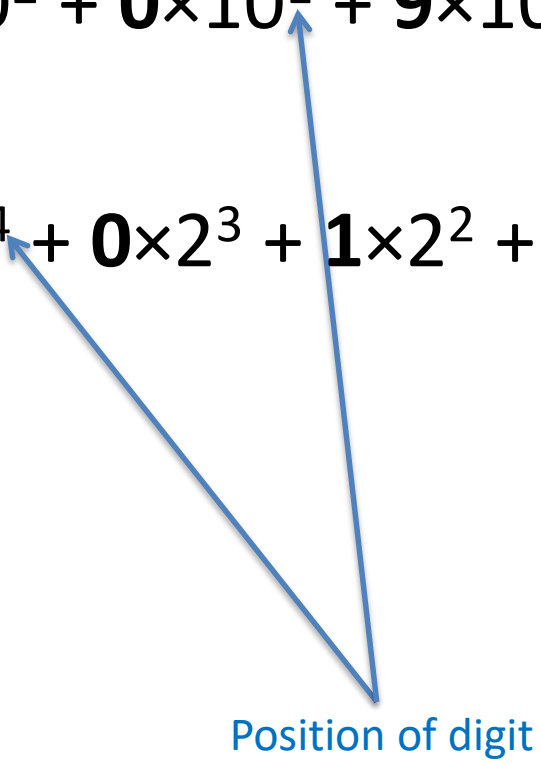
Binary!

$$\mathbf{1209}_{[10]} = \mathbf{1}\times10^3 + \mathbf{2}\times10^2 + \mathbf{0}\times10^1 + \mathbf{9}\times10^0$$

$$\mathbf{100101}_{[2]} = \mathbf{1}\times2^5 + \mathbf{0}\times2^4 + \mathbf{0}\times2^3 + \mathbf{1}\times2^2 + \mathbf{0}\times2^1 + \mathbf{1}\times2^0$$



Hexadecimal!

$0_{[16]}$	$0000_{[2]}$	$0_{[10]}$
$1_{[16]}$	$0001_{[2]}$	$1_{[10]}$
$2_{[16]}$	$0010_{[2]}$	$2_{[10]}$
$3_{[16]}$	$0011_{[2]}$	$3_{[10]}$
$4_{[16]}$	$0100_{[2]}$	$4_{[10]}$
$5_{[16]}$	$0101_{[2]}$	$5_{[10]}$
$6_{[16]}$	$0110_{[2]}$	$6_{[10]}$
$7_{[16]}$	$0111_{[2]}$	$7_{[10]}$
$8_{[16]}$	$1000_{[2]}$	$8_{[10]}$
$9_{[16]}$	$1001_{[2]}$	$9_{[10]}$
$A_{[16]}$	$1010_{[2]}$	$10_{[10]}$
$B_{[16]}$	$1011_{[2]}$	$11_{[10]}$
$C_{[16]}$	$1100_{[2]}$	$12_{[10]}$
$D_{[16]}$	$1101_{[2]}$	$13_{[10]}$
$E_{[16]}$	$1110_{[2]}$	$14_{[10]}$
$F_{[16]}$	$1111_{[2]}$	$15_{[10]}$

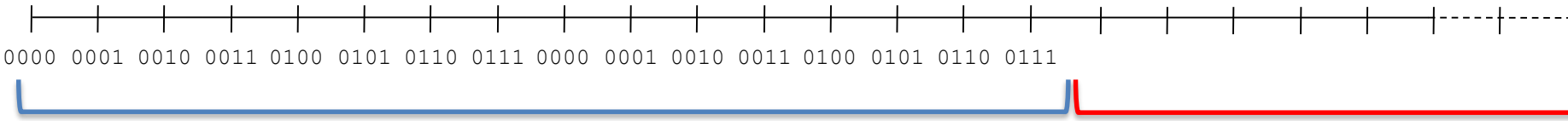
Hexadecimal!

```
--> 12648430
12648430(int)
--> 0xC0FFEE
12648430(int)
--> int2hex(12648430);
"00C0FFEE" (string)
--> int2hex(0xC0FFEE);
"00C0FFEE" (string)
--> 0xC0FFEE == 12648430;
true (bool)
```

Binary arithmetic

```
   1
  1010   (10)
+ 1010   (10)
------
 10100   (20)
```

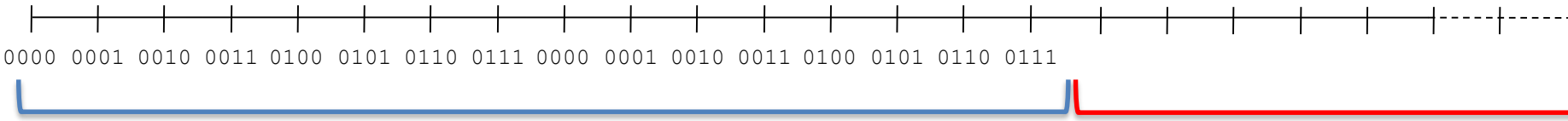
```
     0110   (6)
   x 1010   (10)
------------
     0000
    0110
   0000
+ 0110
------------
   111100   (60)
```

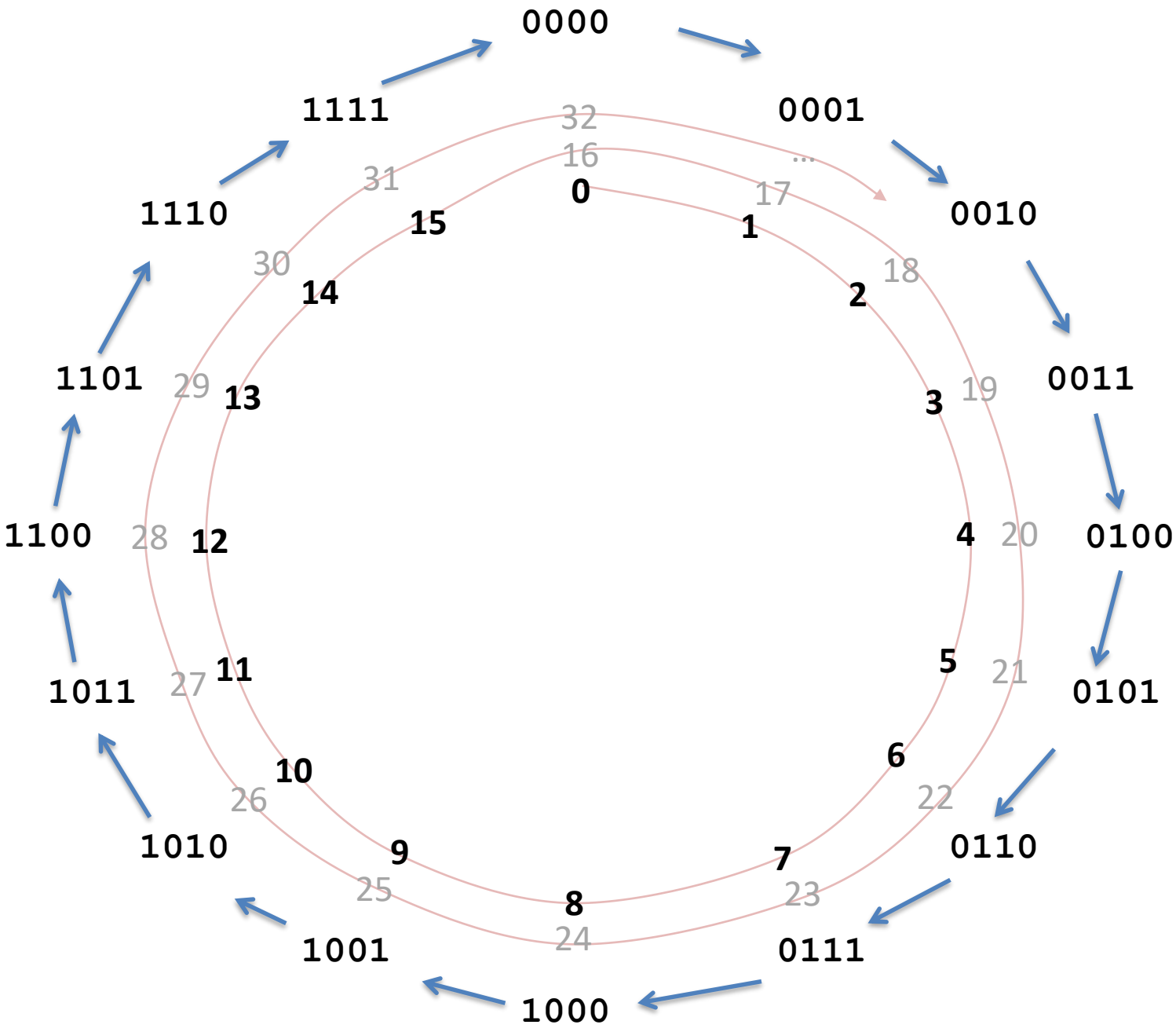


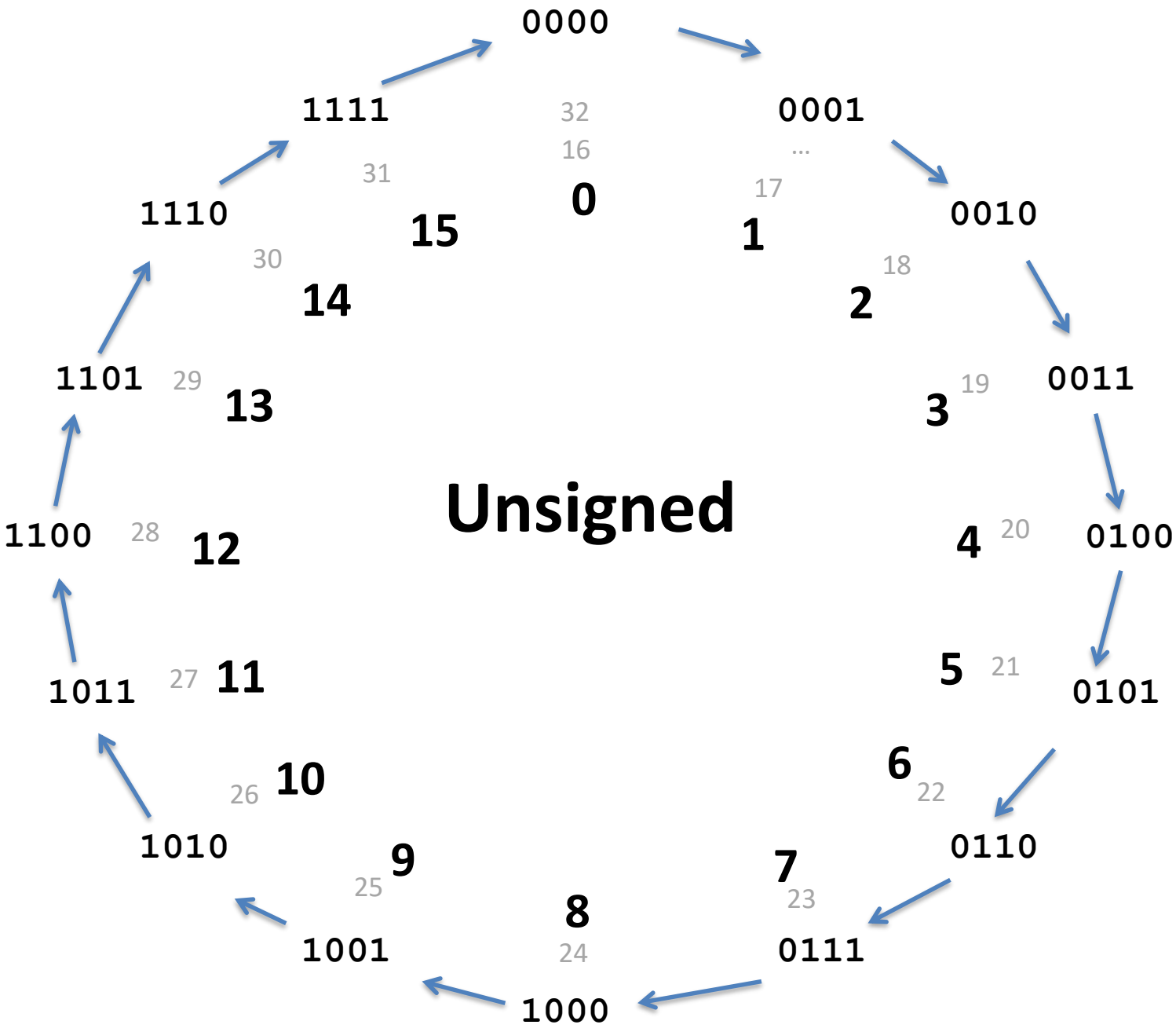
Modular arithmetic

```
   1
  1010   (10)
+ 1010   (10)
 10100   (20)

  0100   (4)
```

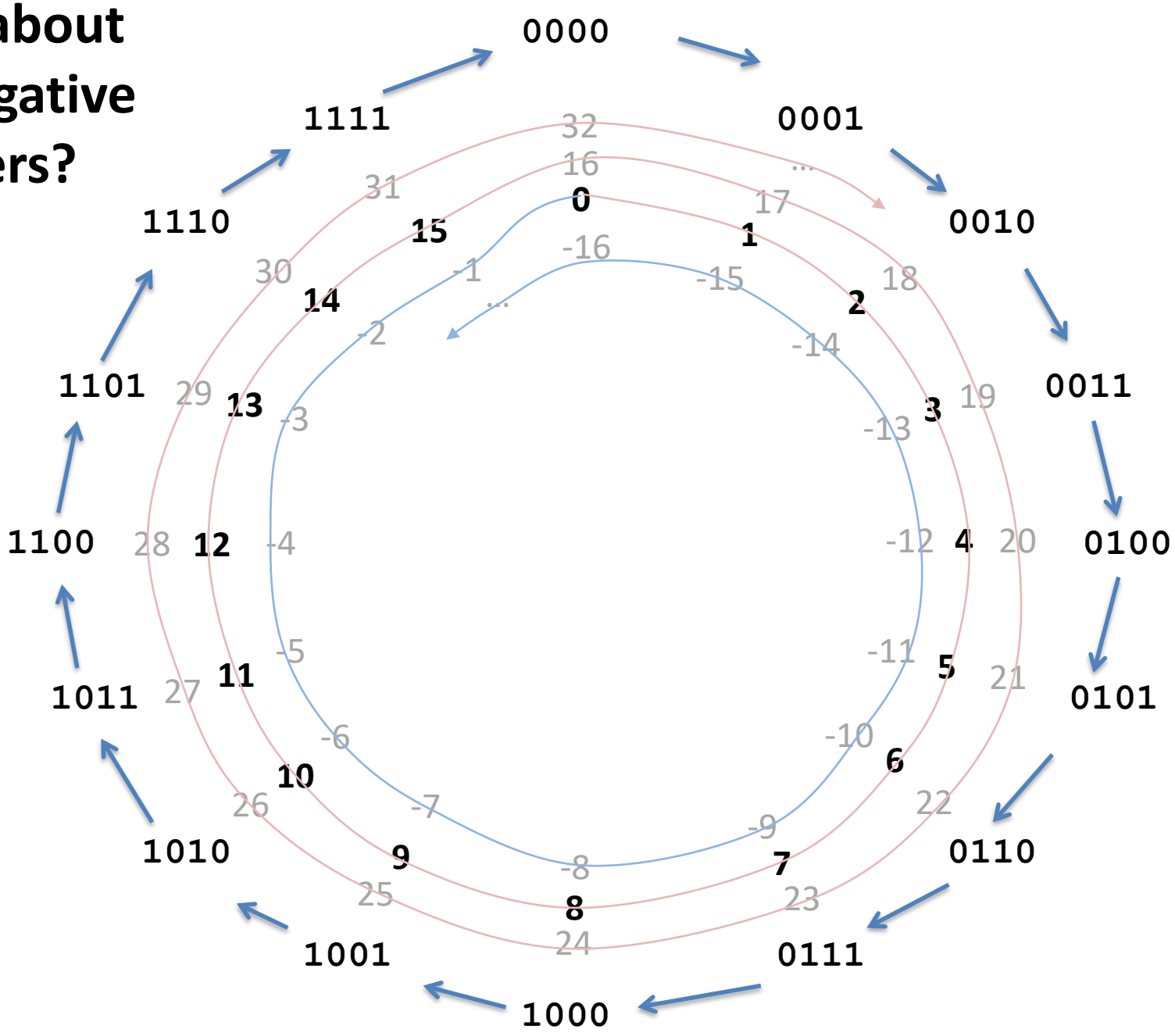
```
    0110   (6)
  x 1010   (10)
  ------
    0000
   0110
  0000
+ 0110
  ------
  1100   (60)

  1100   (12)
```

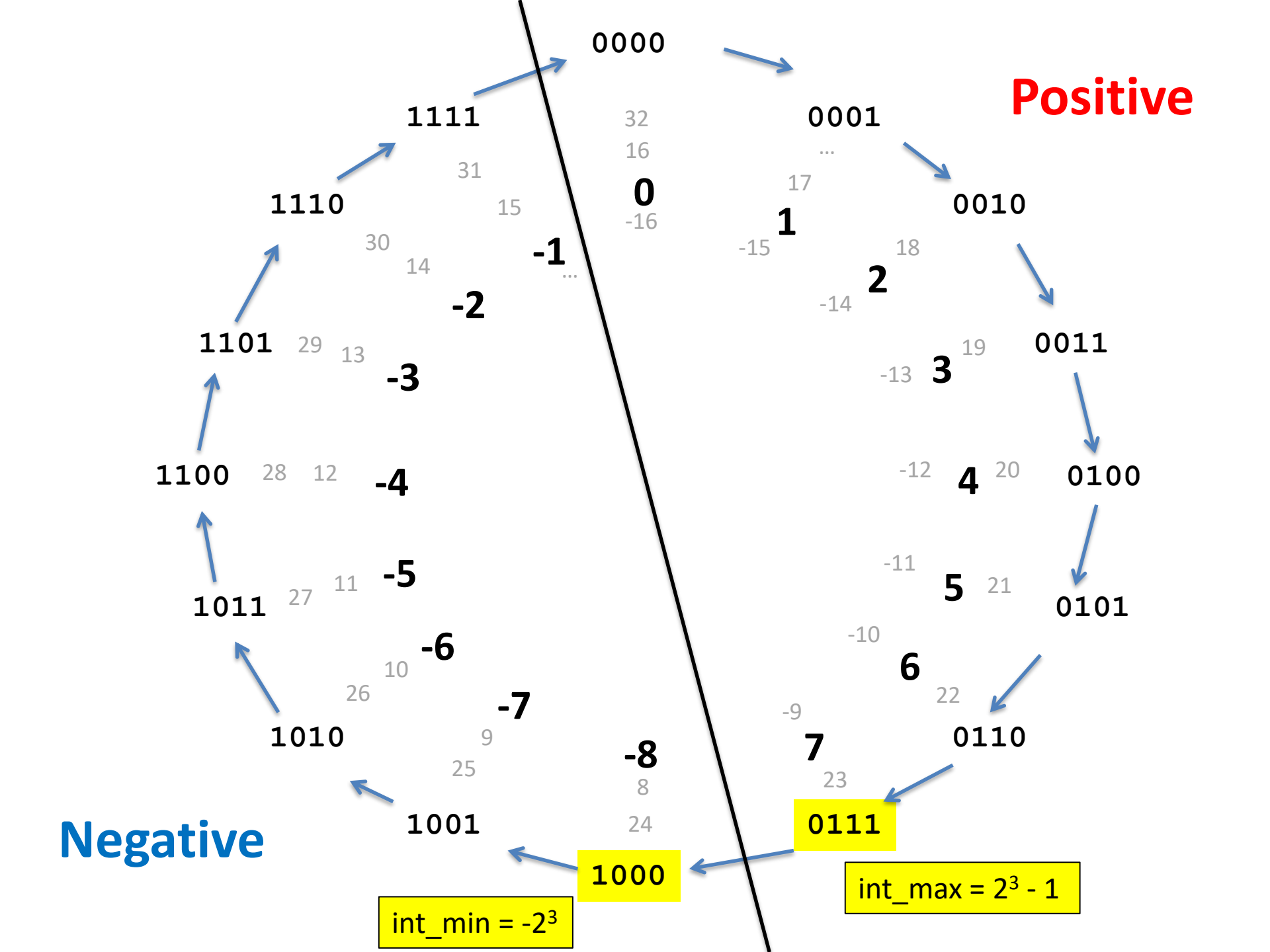


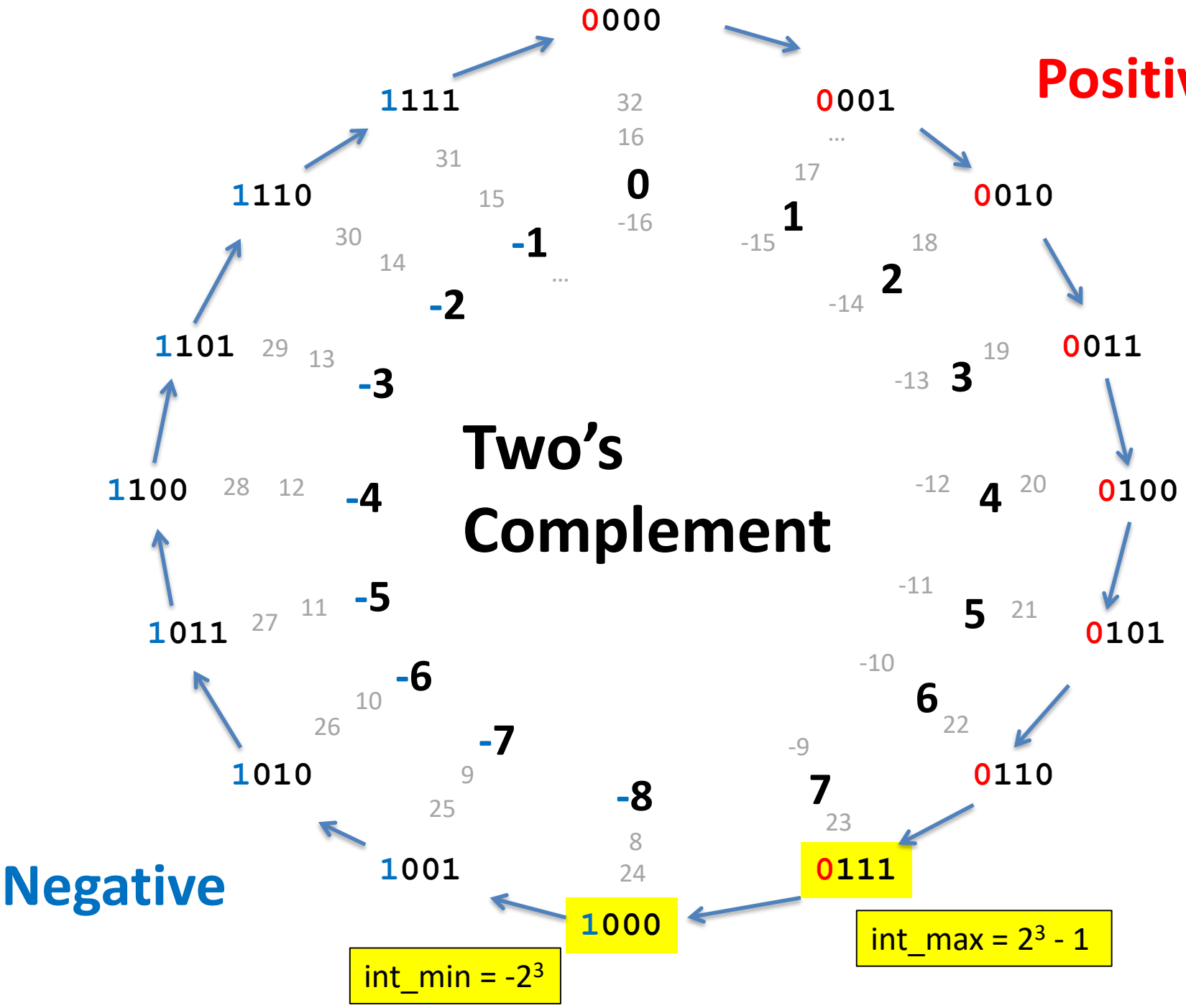

0000
0001
...
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32


Unsigned
0000
0
0001
1
0010
2
0011
3
0100
4
0101
5
0110
6
0111
7
1000
8
1001
9
1010
10
1011
11
1100
12
1101
13
1110
14
1111
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
...

What about the negative numbers?




0000
1111
0001
32
16
31
0
17
1110
0010
15
-16
1
-1
30
-15
18
14
2
-2
-14
1101
29
0011
19
13
3
-3
-13
1100
28
12
-4
4
-12
20
0100
-5
-11
1011
27
11
5
21
0101
-6
-10
10
6
26
22
-7
-9
1010
9
7
0110
-8
25
23
8
1001
24
0111
1000
int_min = -2^3
int_max = 2^3 - 1
Positive
Negative

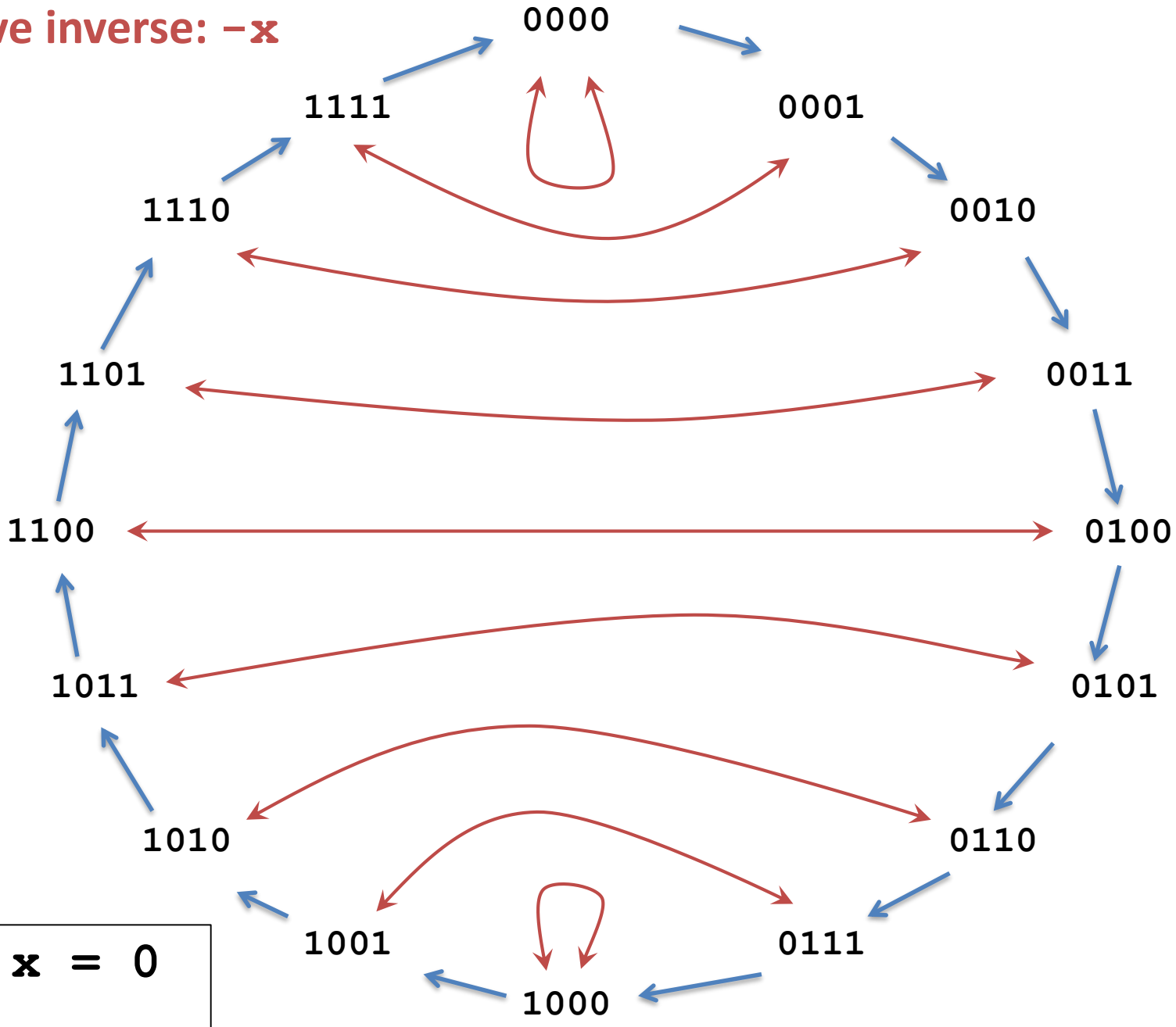

Positive
Negative
Two's Complement
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
0
1
2
3
4
5
6
7
-8
-7
-6
-5
-4
-3
-2
-1
32
16
31
15
17
...
-16
-15
18
-14
19
-13
-12
20
-11
21
-10
22
-9
23
8
24
25
9
26
10
27
11
28
12
29
13
30
14
...
int_min = -2^3
int_max = 2^3 - 1

Overflow

```
L_M_BV_32 := TBD.T_ENTIER_32S ((1.0/C_M_LSB_
if L_M_BV_32 > 32767 then
   P_M_DERIVE(T_ALG.E_BV) := 16#7FFF#;
elsif L_M_BV_32 < -32768 then
   P_M_DERIVE(T_ALG.E_BV) := 16#8000#;
else
   P_M_DERIVE(T_ALG.E_BV) := UC_16S_EN_16NS(
end if;
P_M_DERIVE(T_ALG.E_BH) :=
    UC_16S_EN_16NS (TDB.T_ENTIER_16S ((1.0/C_M
```

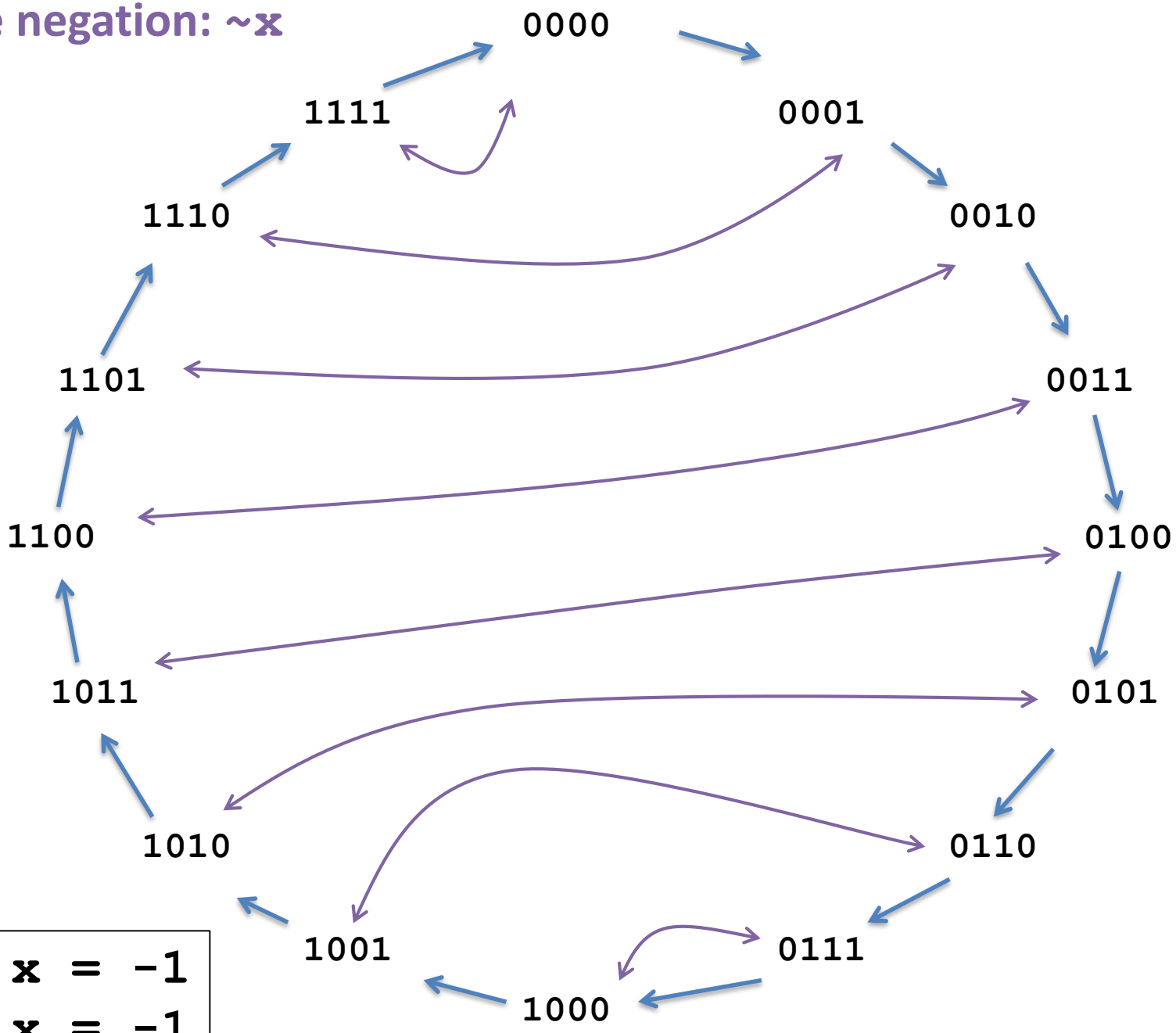
Ariane 5

Additive inverse: −x



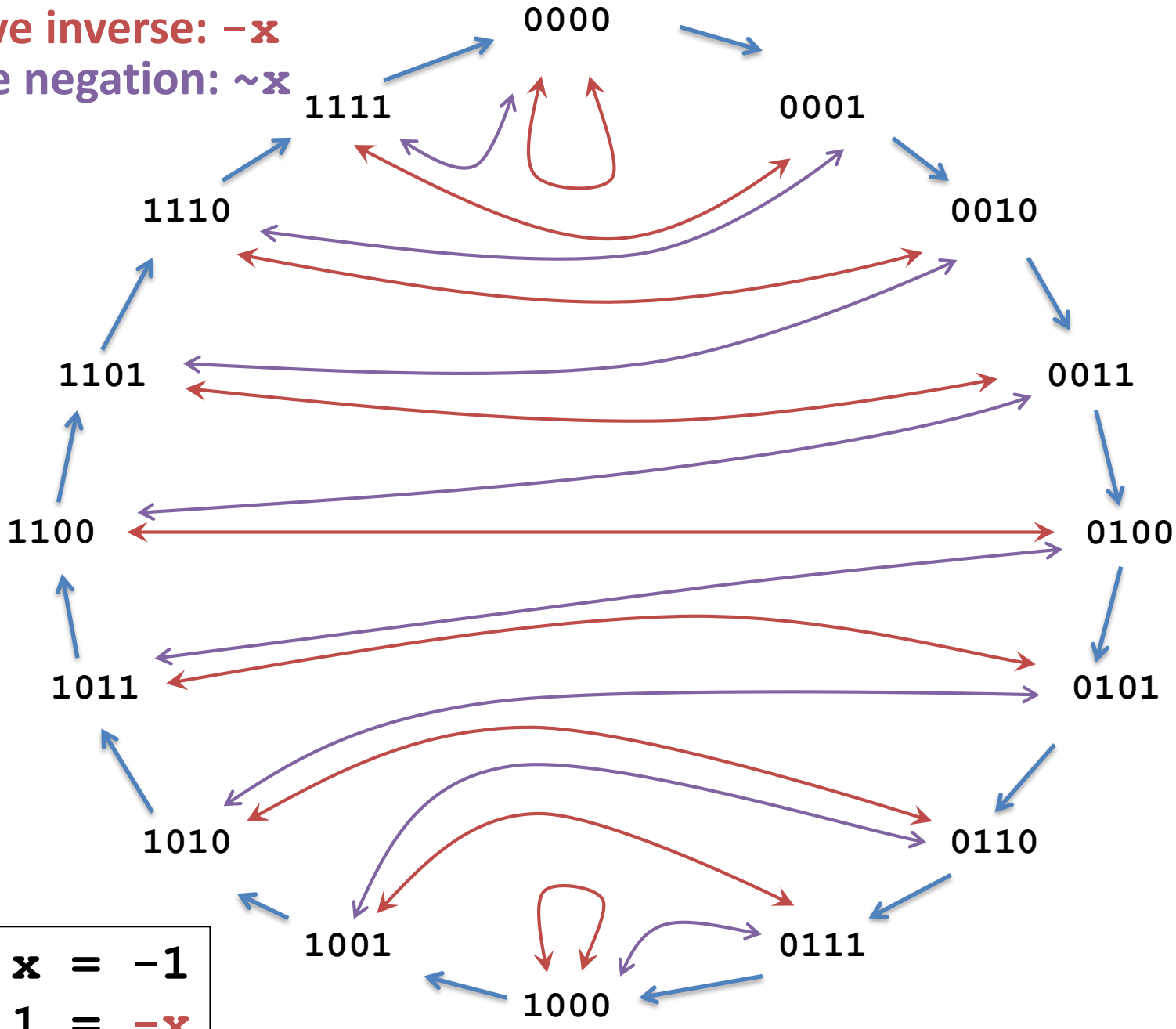
$-x + x = 0$

Bitwise negation: ~x




```
~x + x = -1
~x | x = -1
```

Additive inverse: −x
Bitwise negation: ~x




```
~x + x = -1
~x + 1 = -x
```