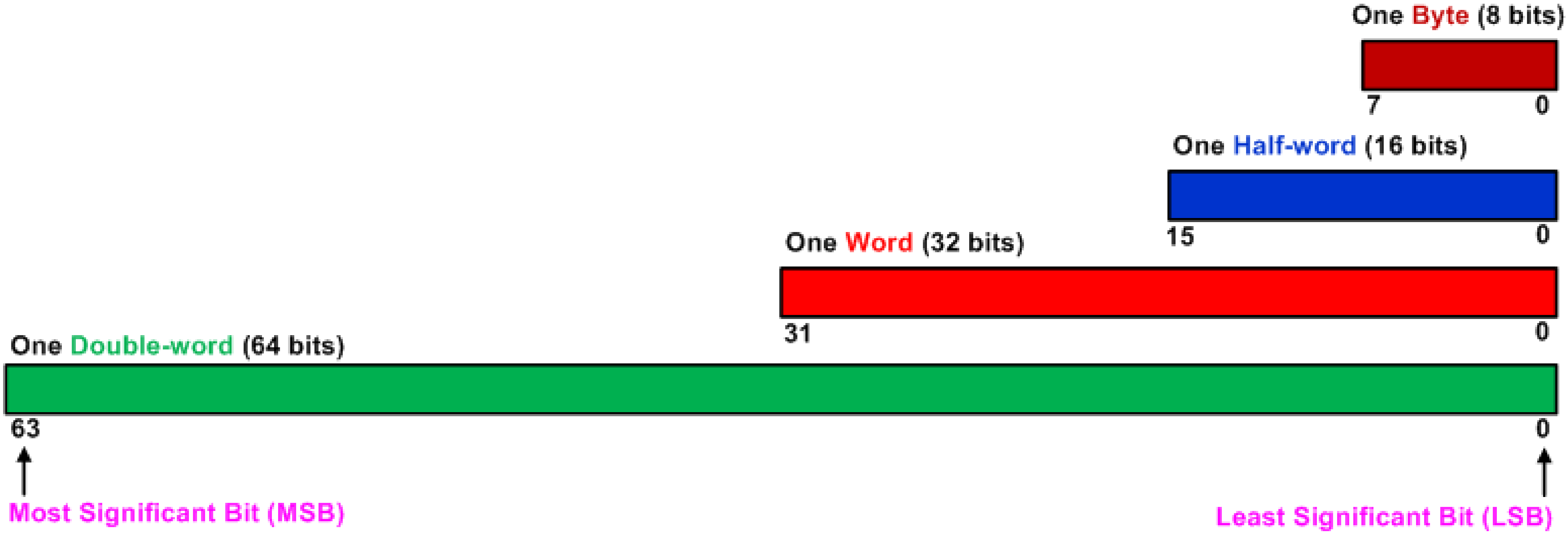
ECE3375B: Microprocessors and Microcomputers
Electrical and Computer Engineering
Western University

Unit 1b: Data Representation

Data Representation

Definitions of bit, byte, half-word, word and double-word


One Byte (8 bits)
7
0
One Half-word (16 bits)
15
0
One Word (32 bits)
31
0
One Double-word (64 bits)
63
0
Most Significant Bit (MSB)
Least Significant Bit (LSB)

Data Representation

Storage range of byte, half-word, word and double-word

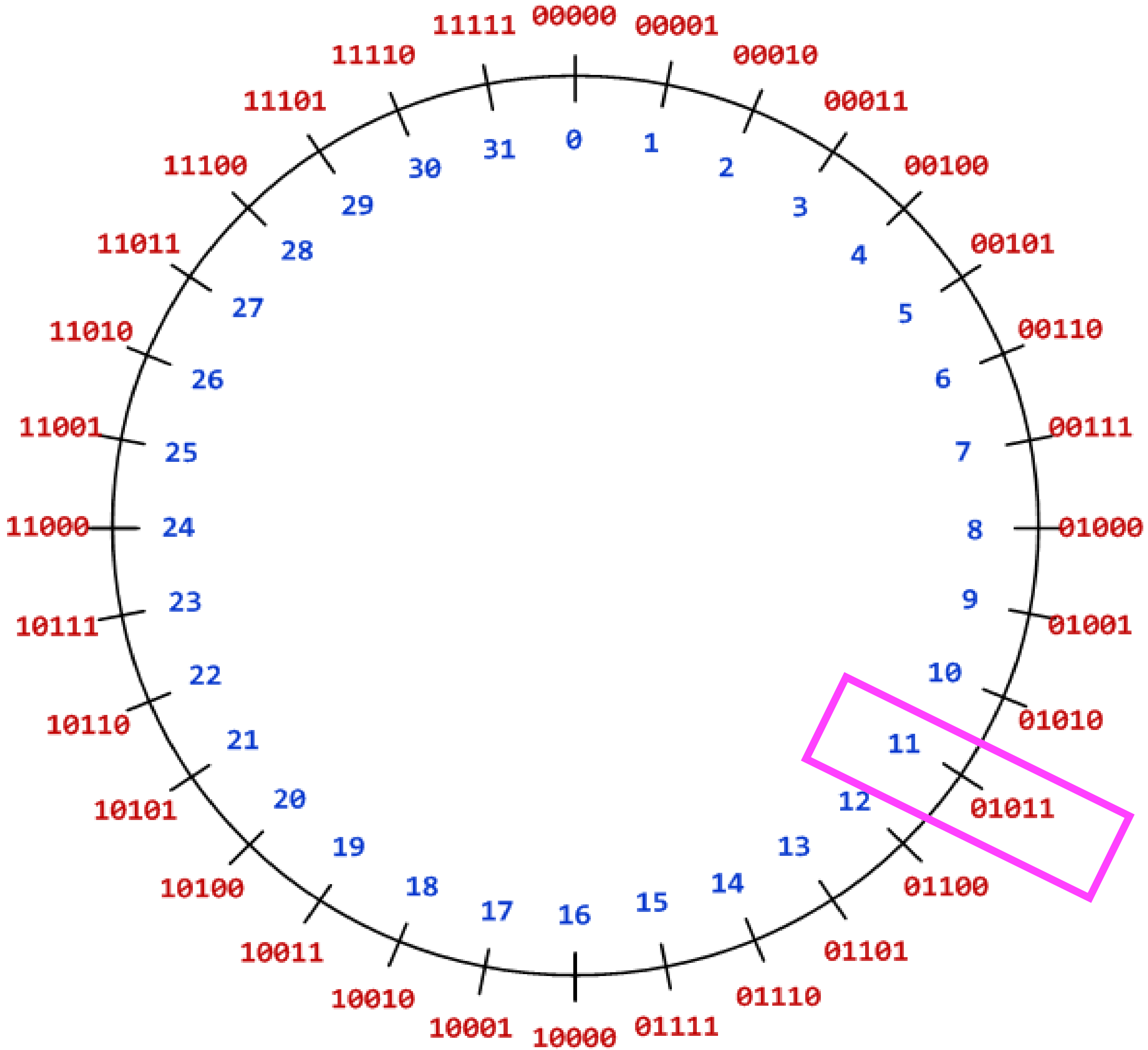
Storage Size	Range	Powers of 2
Unsigned Byte	0 to 255	0 to 2^{8}-1
Unsigned Halfword	0 to 65,535	0 to 2^{16}-1
Unsigned Word	0 to 4,294,967,295	0 to 2^{32}-1
Unsigned Double-word	0 to 18,446,744,073,709,551,615	0 to 2^{64}-1

Data Representation

Binary, Octal, Decimal and Hex

Decimal	Binary	Octal	Hex
0	0000	00	0x0
1	0001	01	0x1
2	0010	02	0x2
3	0011	03	0x3
4	0100	04	0x4
5	0101	05	0x5
6	0110	06	0x6
7	0111	07	0x7
8	1000	010	0x8
9	1001	011	0x9
10	1010	012	0xA
11	1011	013	0xB
12	1100	014	0xC
13	1101	015	0xD
14	1110	016	0xE
15	1111	017	0xF

Unsigned Binary Numbers



Five-bit binary code

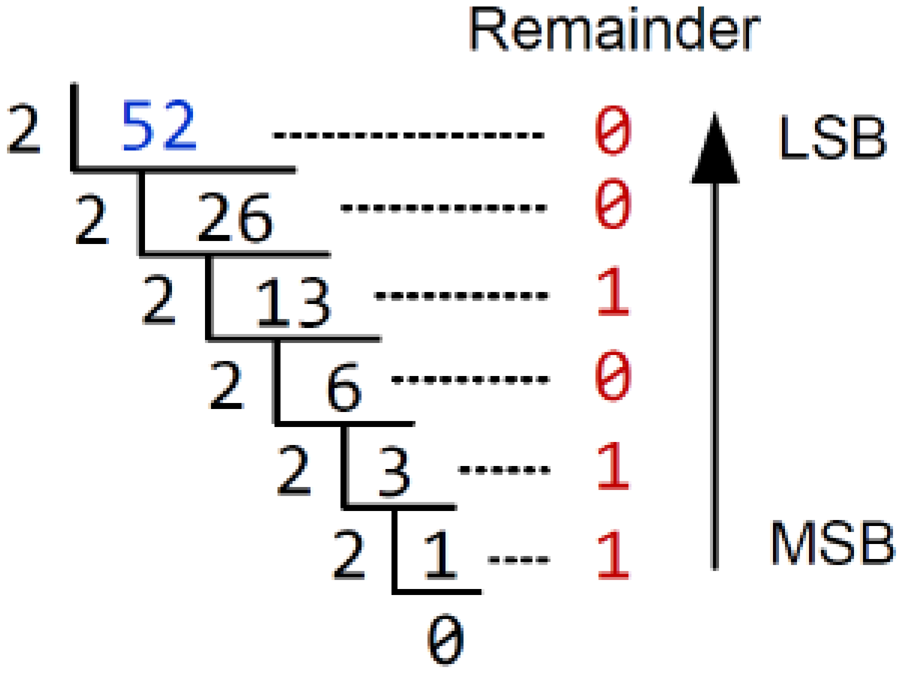
Converting Binary to Decimal

$$
\begin{aligned}
& \mathbf{01011_2} \\
&= \mathbf{0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0} \\
&\qquad = \mathbf{8 + 2 + 1} \\
&\qquad = \mathbf{11}
\end{aligned}
$$

Unsigned Binary Numbers

Converting Decimal to Binary

Convert 52 (decimal) to binary

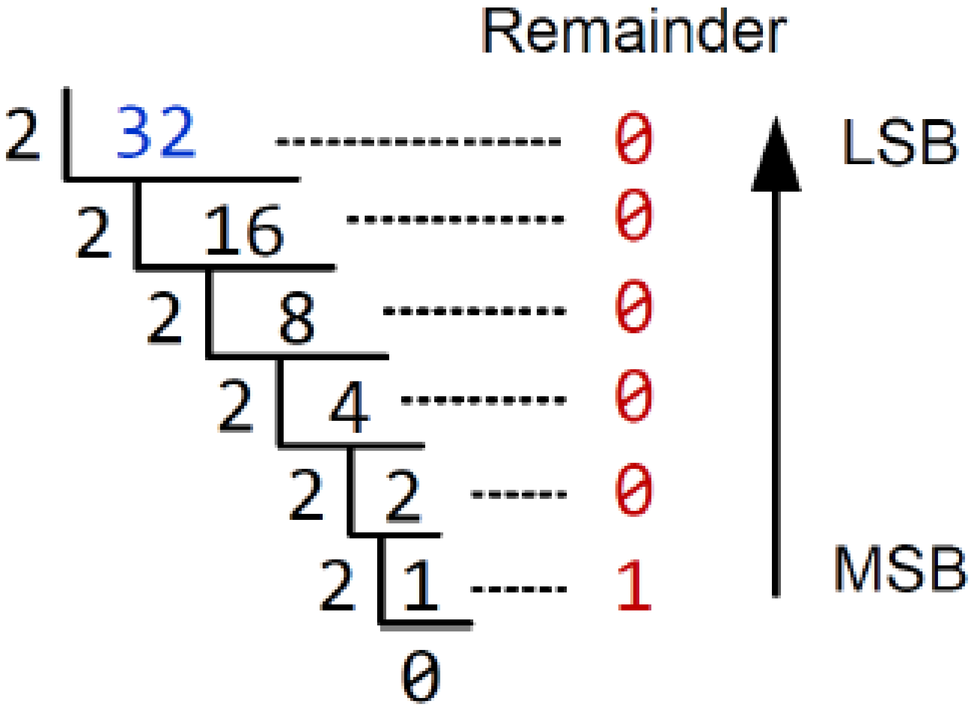


Remainder

2 | 52 ---------------- 0 LSB
2 | 26 ------------ 0
2 | 13 ---------- 1
2 | 6 ---------- 0
2 | 3 ------ 1
2 | 1 ---- 1 MSB
0

$52_{10} = 110100_2$

Convert 32 (decimal) to binary



Remainder

2 | 32 ---------------- 0 LSB
2 | 16 ------------ 0
2 | 8 ---------- 0
2 | 4 --------- 0
2 | 2 ------ 0
2 | 1 ------ 1 MSB
0

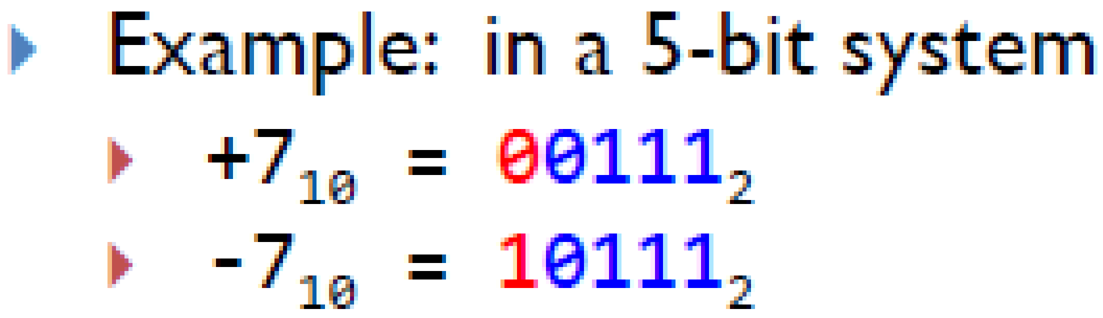
$32_{10} = 100000_2$

Signed Binary Numbers

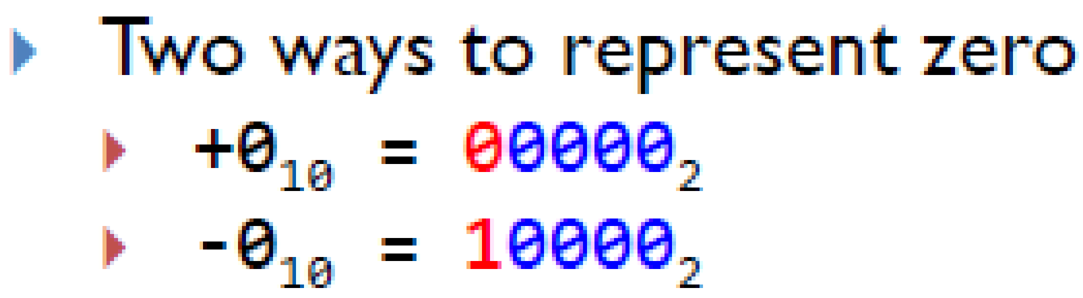
Three different approaches are described to represent signed binary

1) **Sign and Magnitude**: uses the most significant bit to represent the sign and the rest of the bits for the magnitude

$$value=(-1)^{sign}\times Magnitude$$



- Example: in a 5-bit system
  - $+7_{10}$ = 00111_2
  - -7_{10} = 10111_2

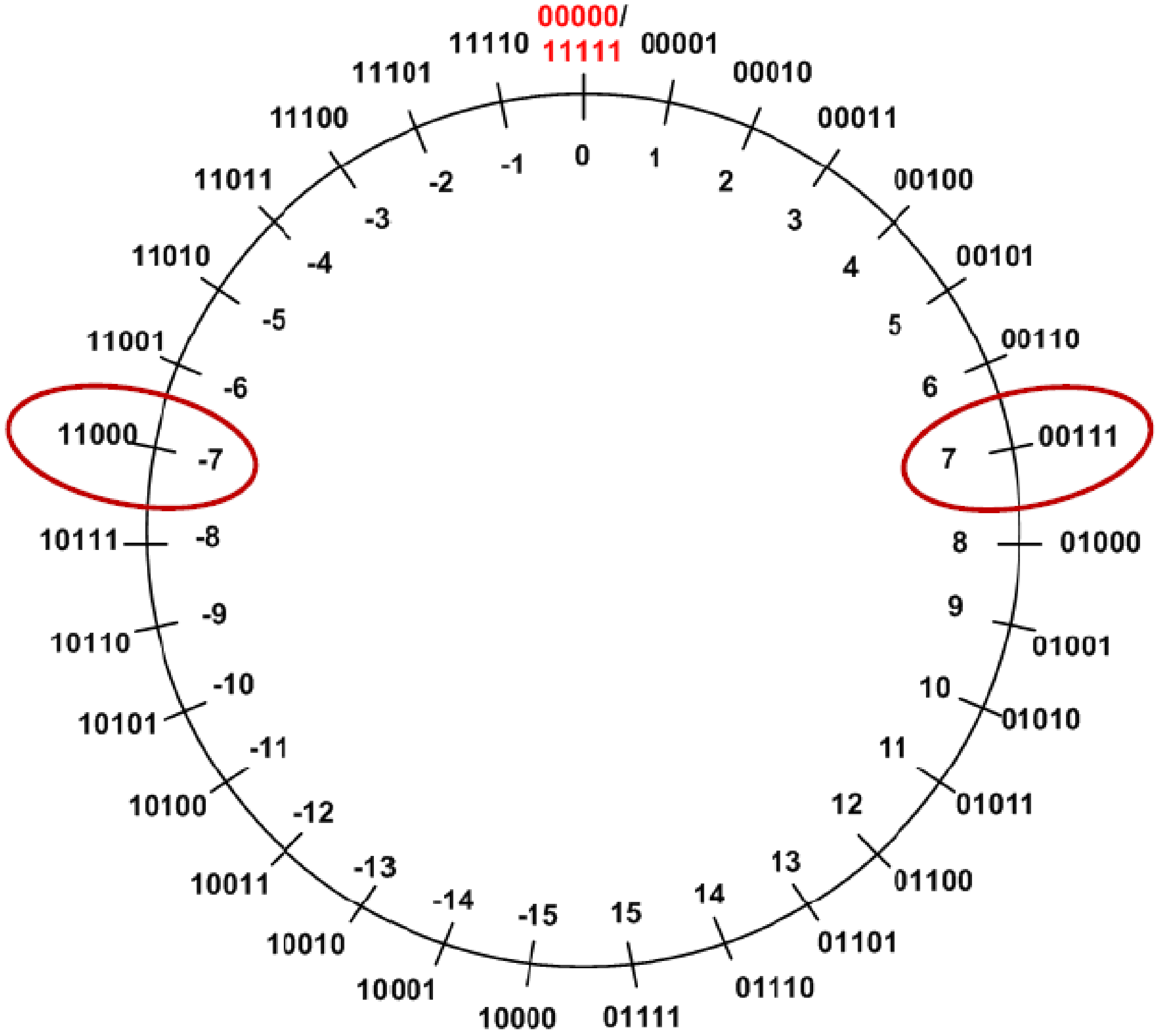


- Two ways to represent zero
  - $+0_{10}$ = 00000_2
  - -0_{10} = 10000_2

Not used is hardware due to two zeros and complexity in performing mathematical operations, (i.e. addition, subtraction, checking equality).

Signed Binary Numbers

2) **One’s compliment**: negative numbers are denoted by inverting every bit of its positive equivalent.



Example: in a 5-bit system

$$+7_{10} = 00111_2$$
$$-7_{10} = 11000_2$$

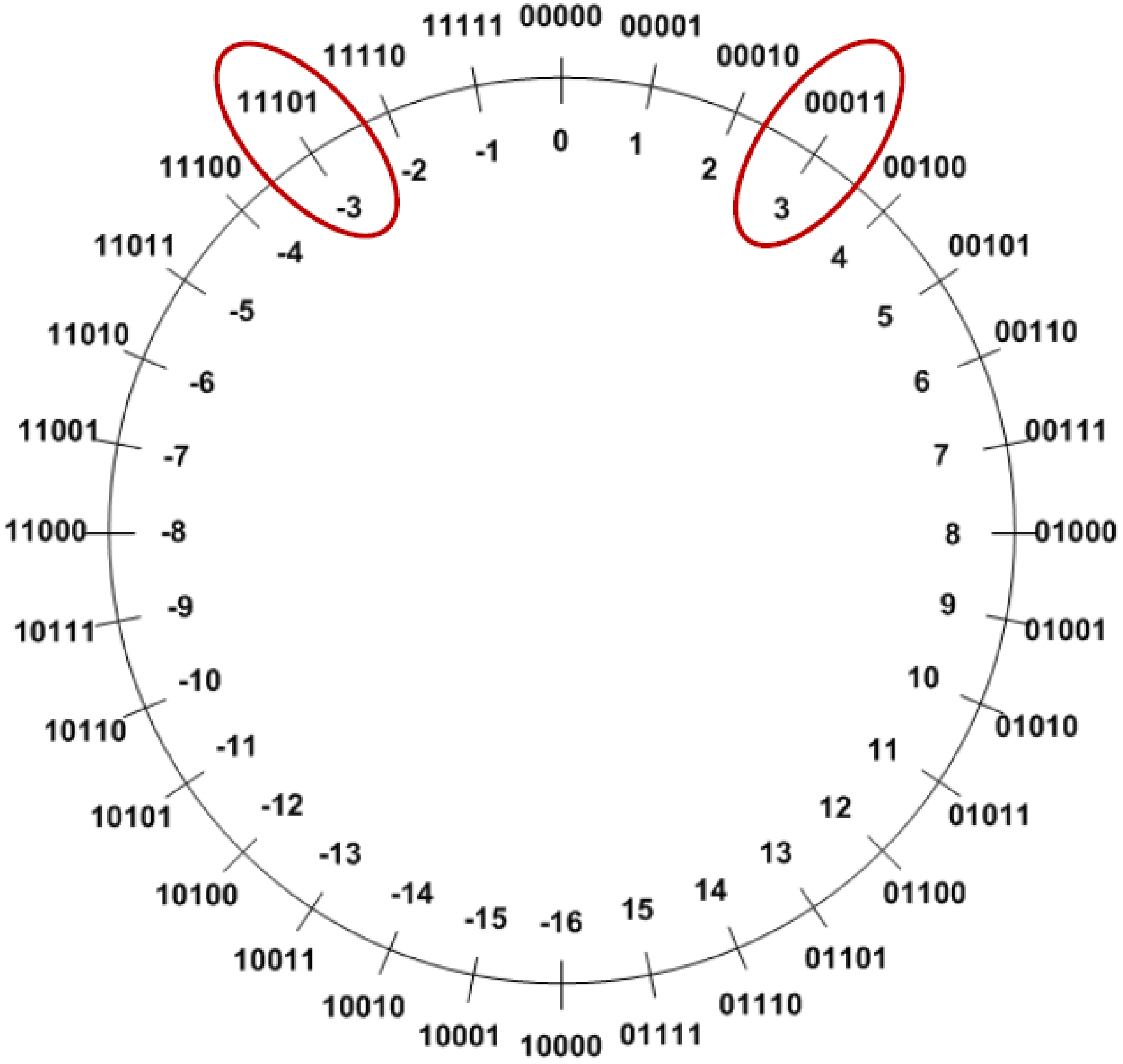
Adding +7 with -7

$$+7_{10} + (-7_{10})$$
$$= 00111_2 + 11000_2$$
$$= 11111_2$$
$$= 2^5 - 1$$

Two representations for zero which increases complexity when checking for equality

Signed Binary Numbers

3) **Two’s compliment**: negative numbers are denoted by inverting every bit of its positive equivalent and adding one.



Example: in a 5-bit system

$+3_{10} = 00011_2$

$-3_{10} = 11100_2 + 00001_2 = 11101_2$

Adding +3 with -3

$= 00011_2 + 11101_2$

$= 00000_2$

Most often used in modern computers. (one zero, less hardware complexity for math operations)

Signed Binary Numbers

Number ranges of signed numbers

	Sign-and-Magnitude	One's Complement	Two's Complement
Range	$[-2^{n-1}+1, 2^{n-1}-1]$	$[-2^{n-1}+1, 2^{n-1}-1]$	$[-2^{n-1}, 2^{n-1}-1]$
Zero	Two zeroes (± 0)	Two zeroes (± 0)	One zero
Unique Numbers	2^n-1	2^n-1	2^n

Storage Size	Range	Powers of 2
Signed Byte	-128 to +127	-2^{7} to 2^{7}-1
Signed Halfword	-32,768 to +32,767	-2^{15} to 2^{15}-1
Signed Word	-2,147,483,648 to +2,147,483,647	-2^{31} to 2^{31}-1
Signed Double-word	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	-2^{63} to 2^{63}-1

Carry Flag for Unsigned Addition and Subtraction

Let A and B be two unsigned binary number

The **carry flag** is set if C = A + B; C is to big to fit the n bits

$C > 2^n - 1$

The **borrow flag** is set if C = A - B; C is negative number

On ARM Cortex-M processors, the carry flag and the borrow flag are physically the same flag bit in the application program status register (APSR).

- For an unsigned subtraction, Carry = NOT Borrow

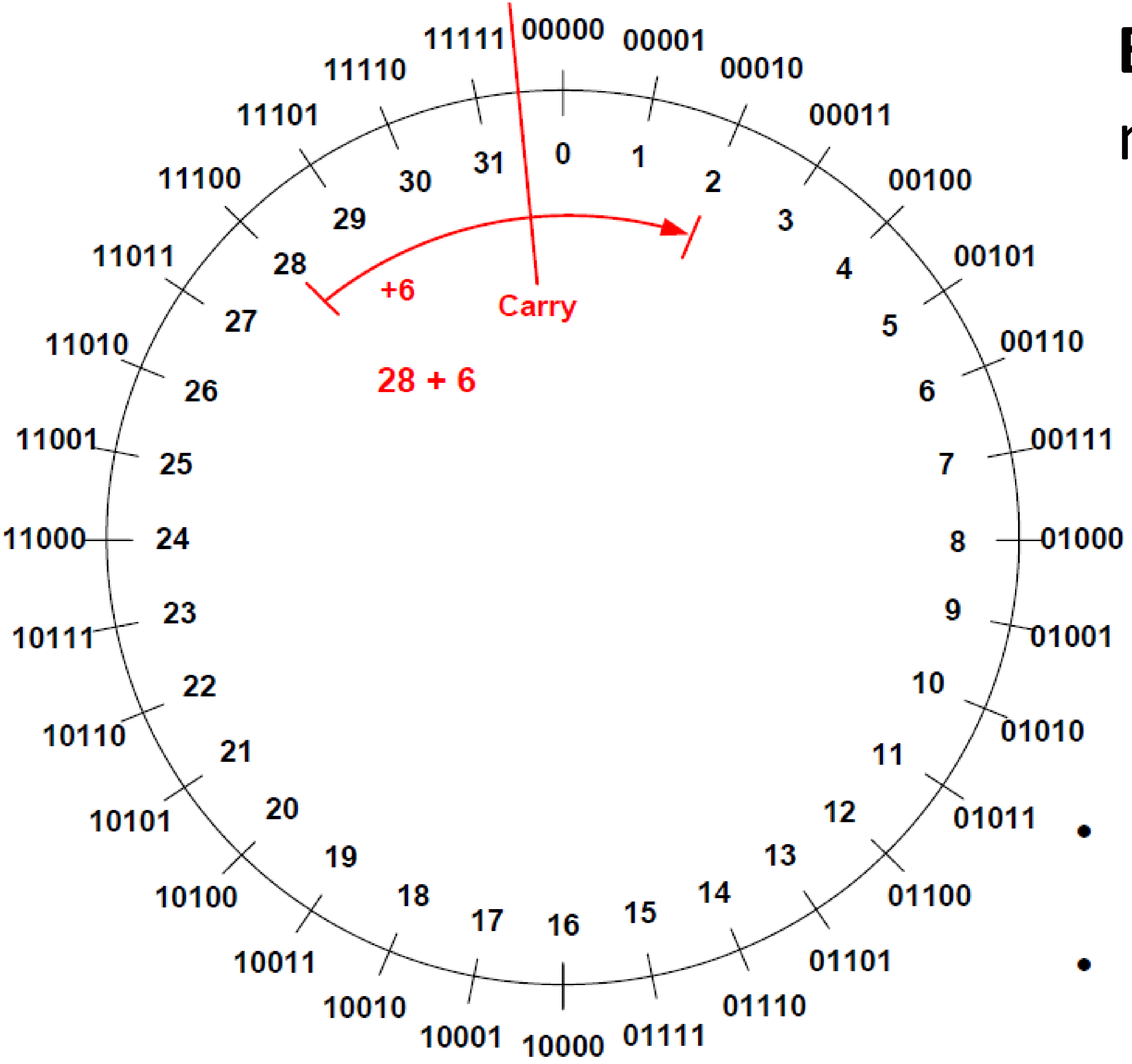
Carry Flag for Unsigned Addition and Subtraction

On ARM Cortex-M processor, the carry flag is set as follows:

- When adding two unsigned integers if the sum is too large ($\geq 2^{32}$) to be stored in a 32-bit register then the carry flag is set, otherwise it is cleared.

- When subtracting two unsigned integers the carry flag is set if no borrow occurs, implying the difference is a positive or zero. Otherwise, the carry bit is cleared.

Carry Flag for Unsigned Addition and Subtraction



Example 1: For a 5-bit unsigned number (addition)

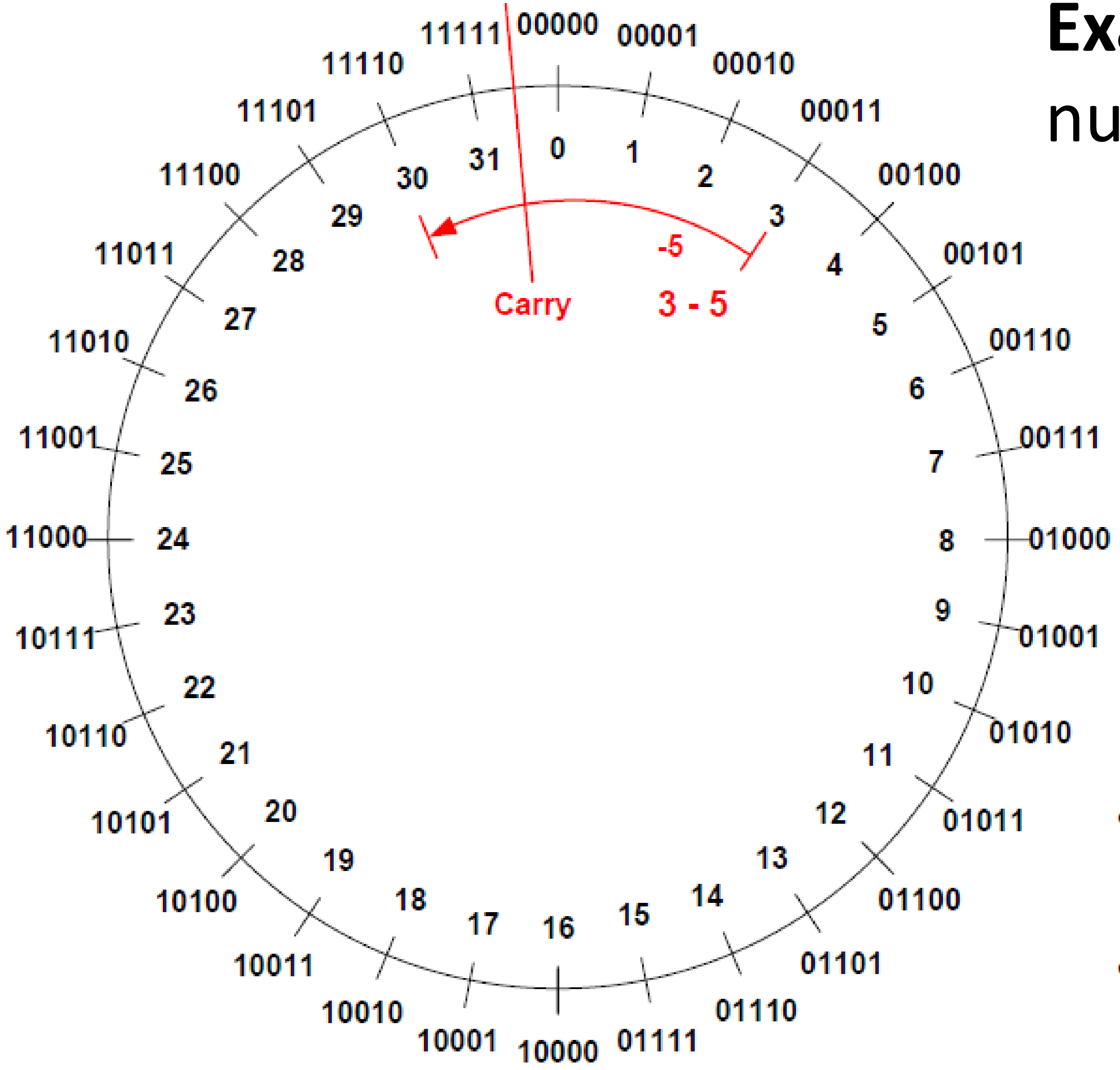
```
Carry   1 1 1 0 0
          1 1 1 0 0        28
        + 0 0 1 1 0      +  6
        -----------      ----
        1 0 0 0 1 0         2
```

Extra bit is discarded.

5-bit result

- Carry flag = 1, indicating carry has occurred on unsigned addition.
- Carry flag is 1 because the result crosses the boundary between 31 and 0.

Carry Flag for Unsigned Addition and Subtraction



Example 2: For a 5-bit unsigned number (subtraction)

```
Borrow   1 1 1 0 0

         0 0 0 1 1        3
       - 0 0 1 0 1     -  5

         1 1 1 1 0       30
```

- Carry flag = 0, indicating borrow has occurred on unsigned subtraction.
- For subtraction, carry = NOT borrow.

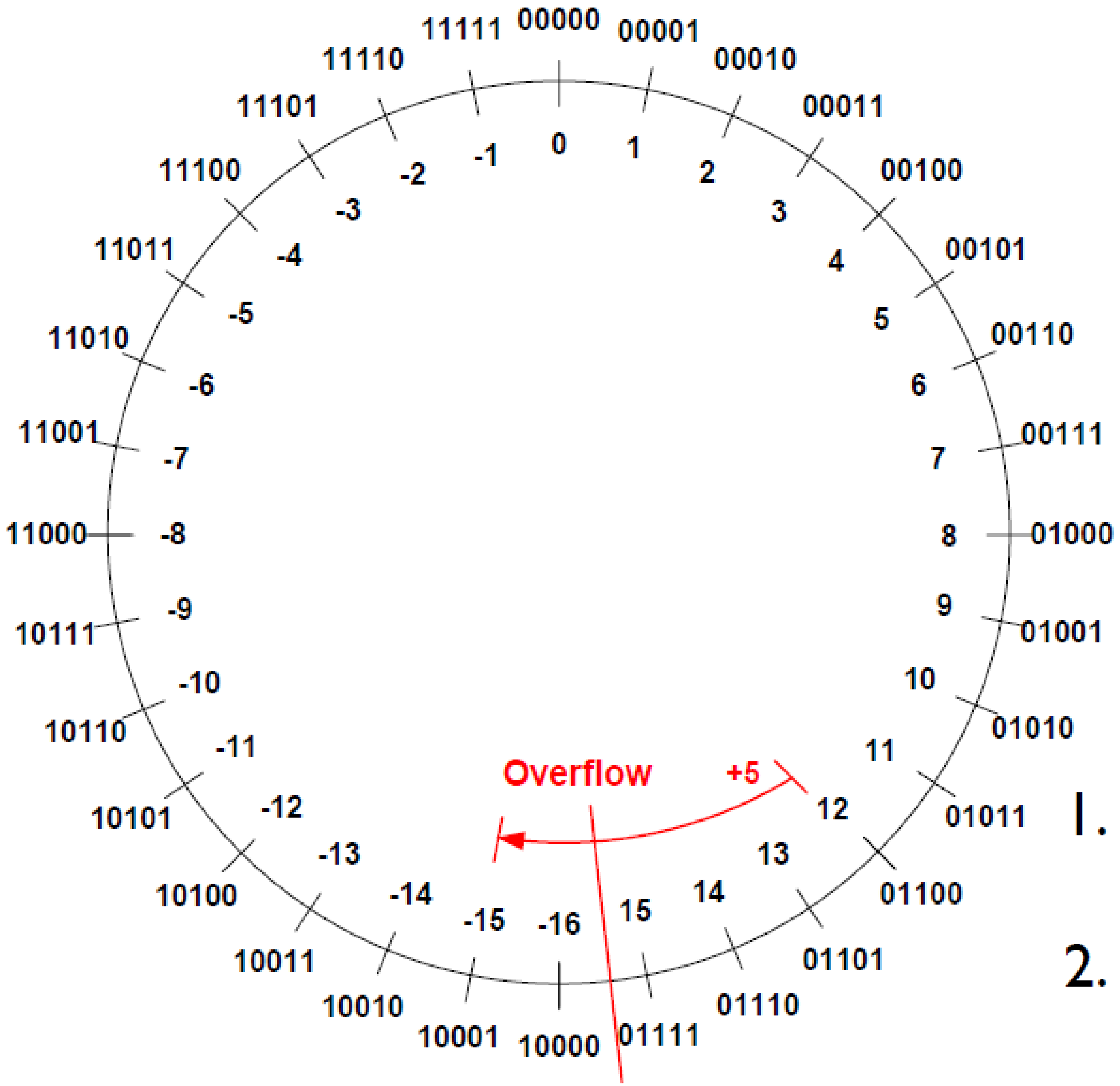
Overflow for Signed Add/Sub

- When adding signed numbers represented in two's complement, overflow occurs only in two scenarios:
  1. adding two positive numbers but getting a non-positive result, or
  2. adding two negative numbers but yielding a non-negative result.

- Similarly, when subtracting signed numbers, overflow occurs in two scenarios:
  1. subtracting a positive number from a negative number but getting a positive result, or
  2. subtracting a negative number from a positive number but producing a negative result.

Overflow cannot occur when adding operands with different signs or when subtracting operands with the same signs.

Overflow for Signed Add/Sub

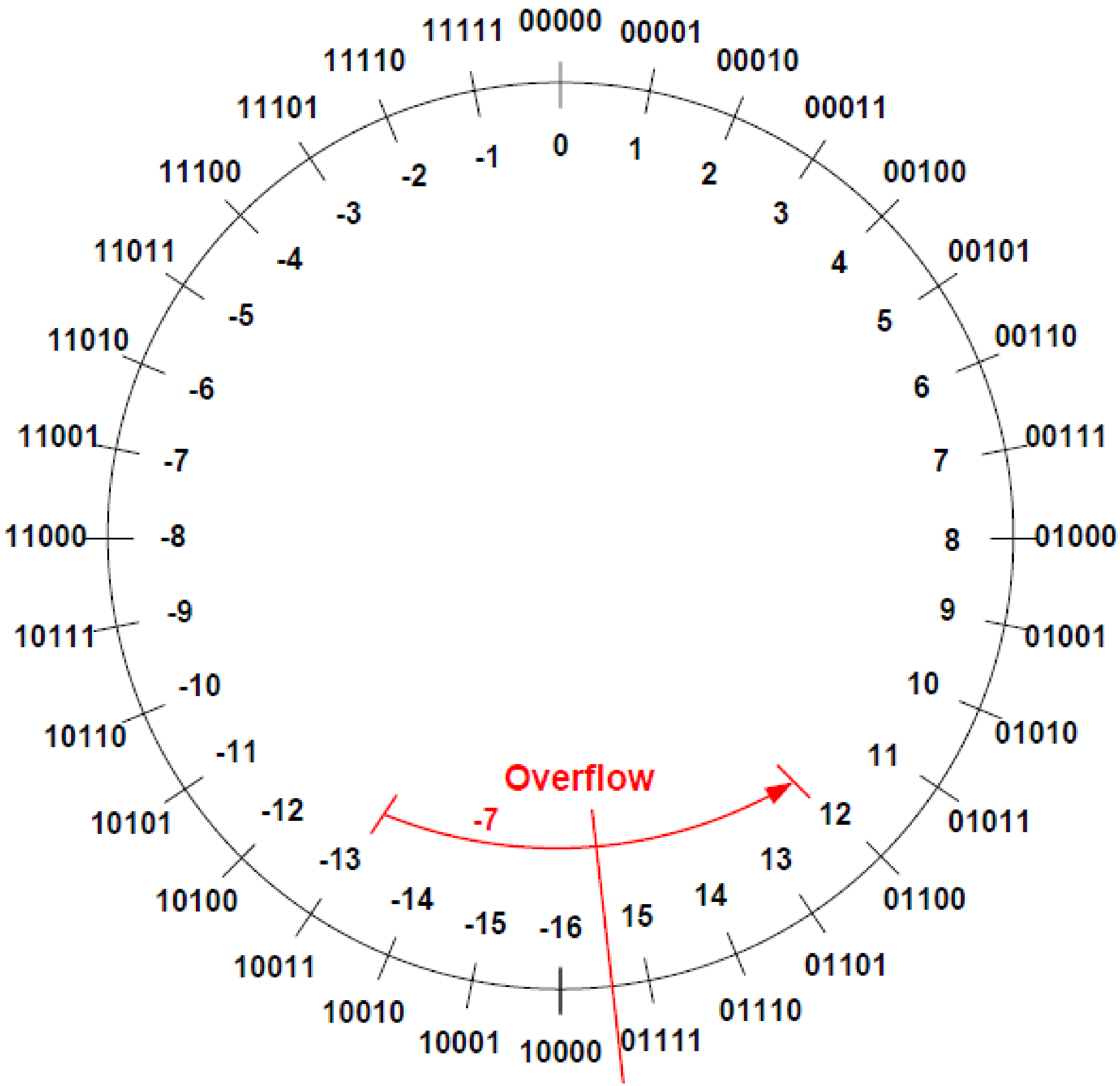


Example 3:For a 5-bit signed number (addition)

$$
\begin{array}{rccccc}
 & 0 & 1 & 1 & 0 & 0 \\
+ & 0 & 0 & 1 & 0 & 1 \\
\hline
 & 1 & 0 & 0 & 0 & 1
\end{array}
\qquad
\begin{array}{rr}
 & 12 \\
+ & 5 \\
\hline
 & -15
\end{array}
$$

1. On addition, overflow occurs if $\mathit{sum} \geq 2^4$ when adding two positives.
2. Overflow never occurs when adding two numbers with different signs.

Overflow for Signed Add/Sub



Example 4: For a 5-bit signed number (subtraction)

$$\begin{array}{rccccc} & 1 & 0 & 0 & 1 & 1 \\ + & 1 & 1 & 0 & 0 & 1 \\ \hline \boxed{1} & 0 & 1 & 1 & 0 & 0 \end{array} \qquad \begin{array}{r} -13 \\ + \;\; -7 \\ \hline 12 \end{array}$$

Extra bit is discarded.

5-bit result

On addition, overflow occurs if $sum < -2^4$ when adding two negatives.

Signed or Unsigned Numbers

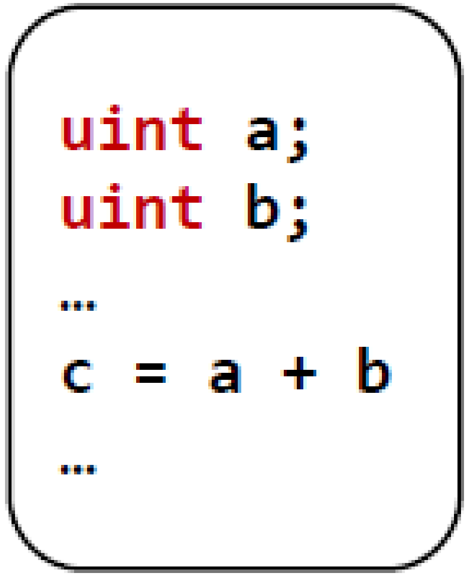
- Are *a* and *b* signed or unsigned numbers?

```
a = 0b10000
b = 0b10000
c = a + b
```

- CPU does not know the answer at all.
- Therefore the hardware sets up both the carry flag and the overflow flag.
- It is software's (programmers'/compilers') responsibility to interpret the flags.

Signed or Unsigned Numbers

In C programming language **uint** is used for unsigned integer declaration and **int** is used for signed int declaration.

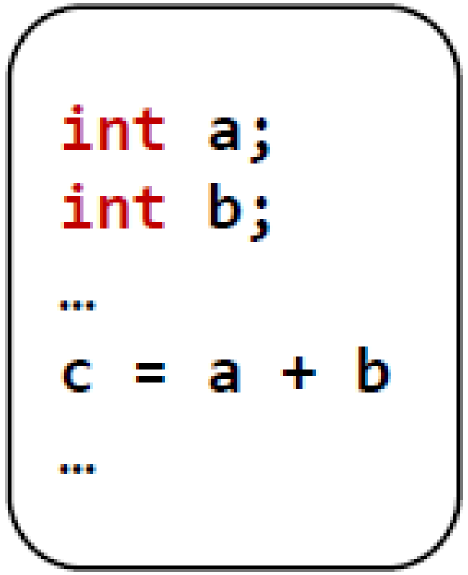


```
uint a;
uint b;
…
c = a + b
…
```

C Program

Unsigned integer declaration

Assembly code checks carry flag



```
int a;
int b;
…
c = a + b
…
```

C Program

Signed integer declaration

Assembly code checks overflow flag

Signed or Unsigned Numbers

Example 5:

For unsigned integer declaration (5-bit example)

```
a = 0b10000
b = 0b10000
```

a = 16 (decimal)

b = 16 (decimal)

Thus $a + b = 32 > 2^5 - 1$ (carry has occurred)

If a and b are signed integer declarations

```
a = 0b10000
b = 0b10000
```

a = -16 (decimal)

b = -16 (decimal)

Thus $a + b = -32 < -2^4$ (overflow has occurred)

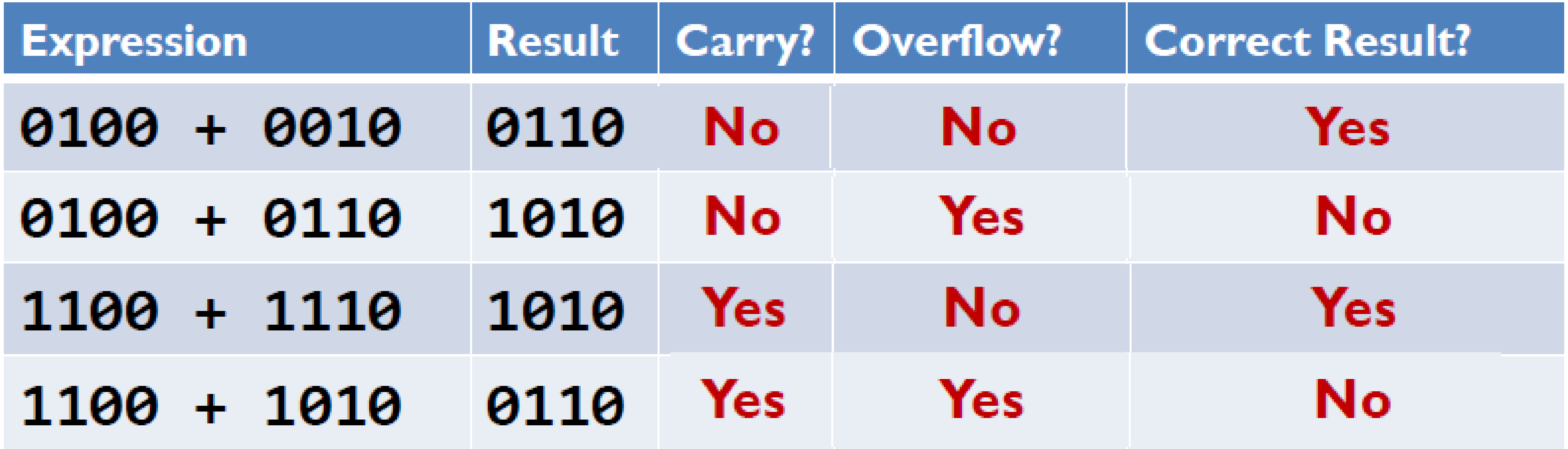
Signed or Unsigned Numbers

Example 6: Two’s compliment 4-bit example (complete the table)

Expression	Result	Carry?	Overflow?	Correct Result?
0100 + 0010	0110			
0100 + 0110	1010			
1100 + 1110	1010			
1100 + 1010	0110			

Signed or Unsigned Numbers

In two’s compliment, 4-bit numbers range from +7 to -8



	Expression	Result	Carry?	Overflow?	Correct Result?
4+2=6	0100 + 0010	0110	No	No	Yes
4+6=-6	0100 + 0110	1010	No	Yes	No
-4+(-2)=-6	1100 + 1110	1010	Yes	No	Yes
-4+(-6)=-10	1100 + 1010	0110	Yes	Yes	No

Two’s Compliment

Two’s complement simplifies hardware

Operation	Are signed and unsigned operations the same?
Addition	Yes
Subtraction	Yes
Multiplication	Yes if the product is required to keep the same number of bits as operands
Division	No

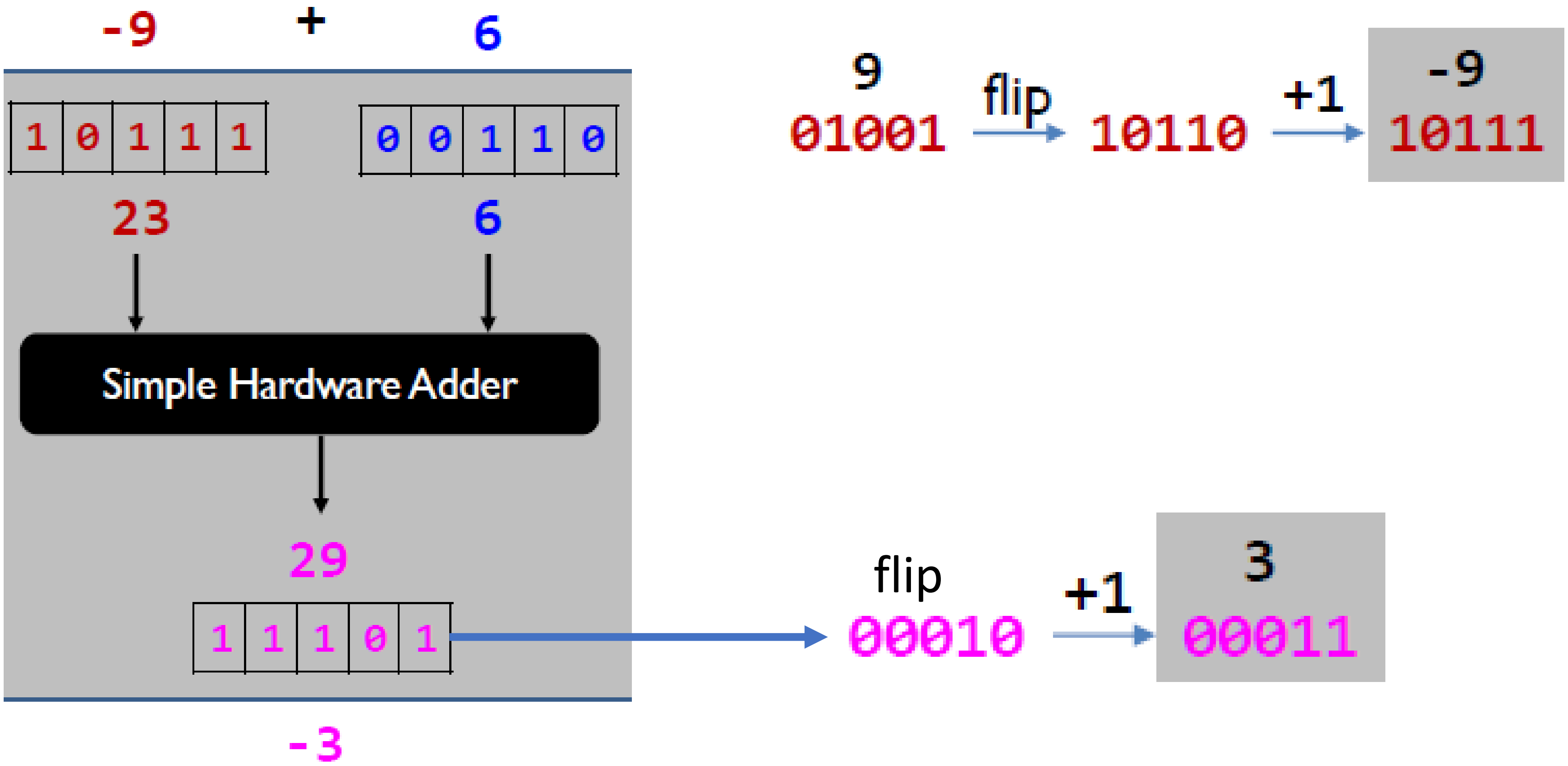
In two’s complement, the same hardware works correctly for both signed and unsigned addition/subtraction.

If the product is required to keep the same number of bits as operands, unsigned multiplication hardware works correctly for signed numbers.

However, this is not true for division.

Signed or Unsigned Numbers

Adding two signed numbers is the same as adding two unsigned numbers (5-bit example shown)



-9 + 6

1	0	1	1	1

23

0	0	1	1	0

6

Simple Hardware Adder

29

1	1	1	0	1

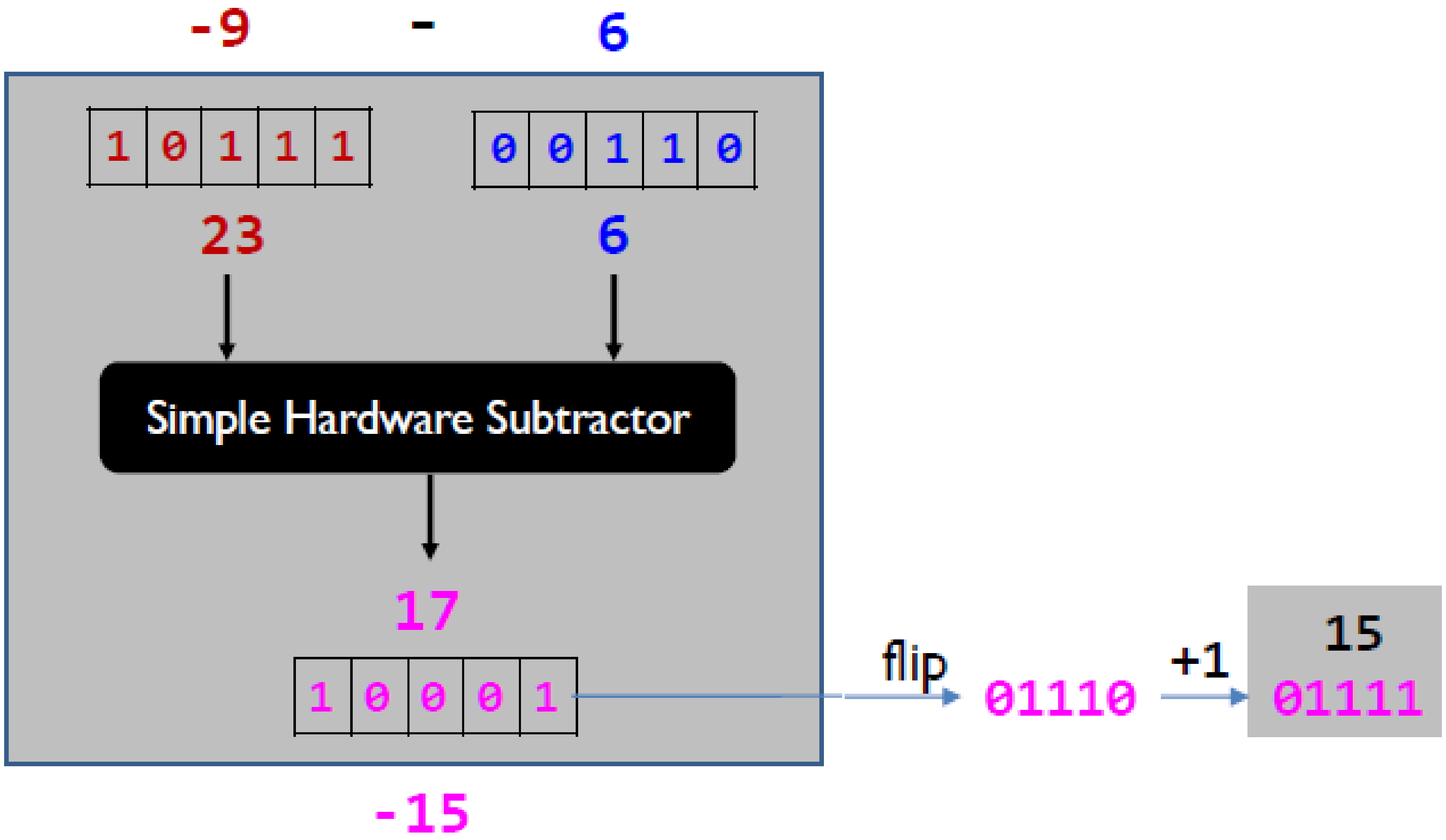
-3

9 $01001 \xrightarrow{\text{flip}} 10110 \xrightarrow{+1}$ -9 10111

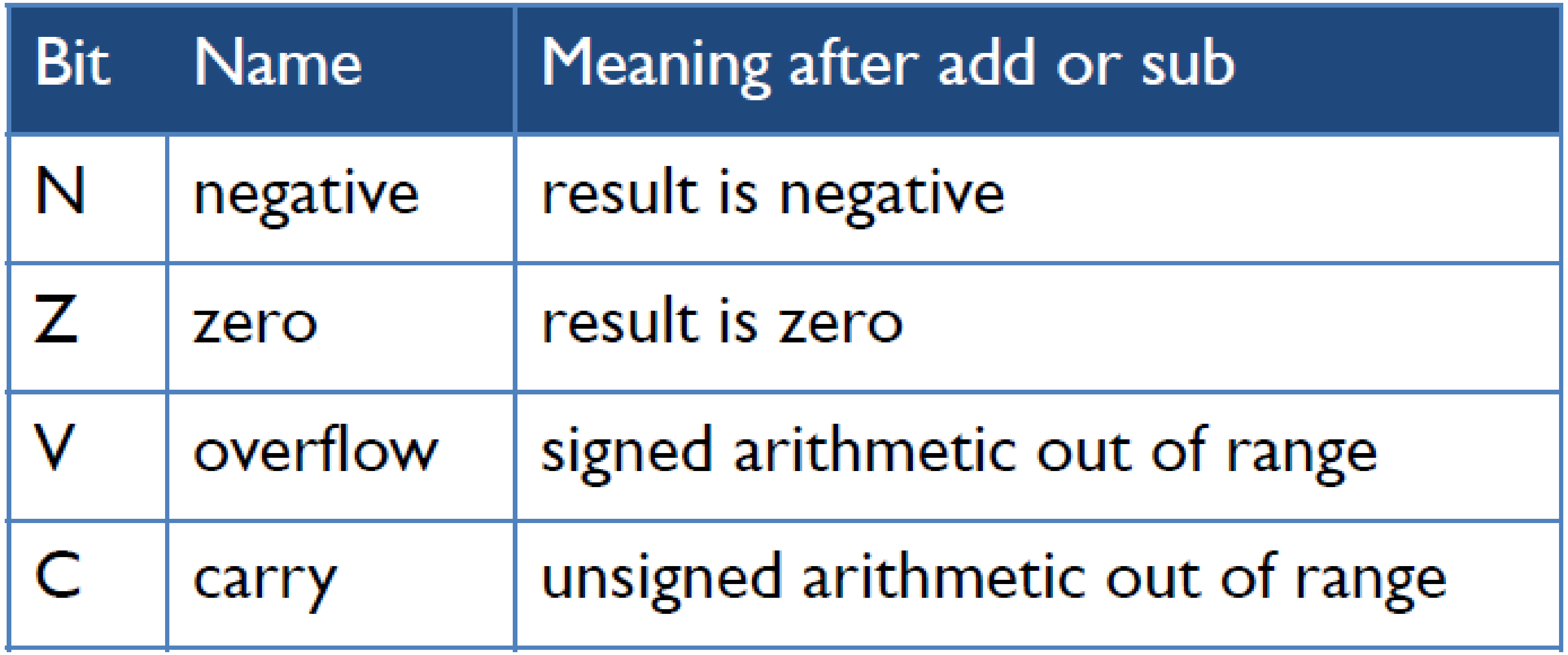
$\xrightarrow{\text{flip}} 00010 \xrightarrow{+1}$ 3 00011

Signed or Unsigned Numbers

Subtracting two signed numbers is the same as subtracting two unsigned numbers (5-bit example shown)



Condition Codes After Adding/Subtracting



Bit	Name	Meaning after add or sub
N	negative	result is negative
Z	zero	result is zero
V	overflow	signed arithmetic out of range
C	carry	unsigned arithmetic out of range

- C is set upon an **unsigned** addition if the answer is wrong
- C is cleared upon an **unsigned** subtract if the answer is wrong
- V is set upon a **signed** addition or subtraction if the answer is wrong

Condition Codes After Adding/Subtracting

$$c = a \pm b$$

	Carry (for unsigned)	Overflow (for signed)
Add	$Carry = 1$ if c is too large to fit in.	$Overflow = 1$ if c is too large or too small to fit in
Subtract	$Borrow = 1, i.e.$ $Carry = 0$ if $a < b$.	

- ARM Cortex-M/A has no dedicated borrow flag, carry flag is reused.
- For unsigned subtract, $Borrow = \overline{Carry}$

- Signed Subtraction is converted to sign addition
- $a - b = a + (-b)$

Additional Examples

Example 7: Complete the following arithmetic operations in two’s complement representation. What are the value of the carry flag and overflow flag? (Assume a six-bit system)

a) -6 + (-30)
b) 25 + 11
c) 16 – 20

Example 8: What are the value of the carry flag and overflow flag after each operation? (Assume a six-bit system)

a) -7 × (3)
b) -7 × (-6)
c) 21 ÷ 3
d) 21 ÷ (-3)

Additional Examples

Example 9: Given the following two 32-bit binary numbers *A* and *B*, find the logic expressions for the NZCV flags when *A* and *B* are added. The result is R.

$$A = a_{31}a_{30}a_{29}\cdots a_2a_1a_0$$
$$B = b_{31}b_{30}b_{29}\cdots b_2b_1b_0$$
$$R = r_{31}r_{30}r_{29}\cdots r_2r_1r_0$$

Example 10: Given the following two 32-bit binary numbers *A* and *B*, find the logic expressions for the NZCV flags when B is subtracted from *A*. The result is R.

$$A = a_{31}a_{30}a_{29}\cdots a_2a_1a_0$$
$$B = b_{31}b_{30}b_{29}\cdots b_2b_1b_0$$
$$R = r_{31}r_{30}r_{29}\cdots r_2r_1r_0$$

Formal Representation After Addition

$$R = X + Y$$

Bit	Name	Meaning after add or sub
N	negative	result is negative
Z	zero	result is zero
V	overflow	signed arithmetic out of range
C	carry	unsigned arithmetic out of range

When adding two 32-bit integers X and Y, the flags are

- $N = R_{31}$ (sign bit is one if negative and zero if positive)
- Z is set if R is zero.
- C is set if the result is incorrect for an unsigned addition

$$C = X_{31} \& Y_{31} \;\|\; X_{31} \& \overline{R_{31}} \;\|\; Y_{31} \& \overline{R_{31}}$$

$\& = \text{AND}$, $\|$=OR

- V is set if the result is incorrect for a signed addition.

$$V = X_{31} \& Y_{31} \& \overline{R_{31}} \;\|\; \overline{X_{31}} \& \overline{Y_{31}} \& R_{31}$$

Formal Representation After Subtraction

$$R = X - Y$$

Bit	Name	Meaning after add or sub
N	negative	result is negative
Z	zero	result is zero
V	overflow	signed arithmetic out of range
C	carry	unsigned arithmetic out of range

When subtracting two 32-bit integers X and Y, the flags are

- $N = R_{31}$ (sign bit is one if negative and zero if positive)
- Z is set if R is zero.
- C is ***clear*** if the result is incorrect for an unsigned subtraction

$$C = \overline{Y_{31} \& R_{31} \parallel \overline{X_{31}} \& R_{31} \parallel \overline{X_{31}} \& Y_{31}}$$

$\& = \text{AND}$, $\parallel$=OR

- V is set if the result is incorrect for an signed subtraction.

$$V = X_{31} \& \overline{Y_{31}} \& \overline{R_{31}} \parallel \overline{X_{31}} \& Y_{31} \& R_{31}$$

Additional Examples

Example 11: What condition flags do you check to determine if A >= B

a) if A and B are signed numbers

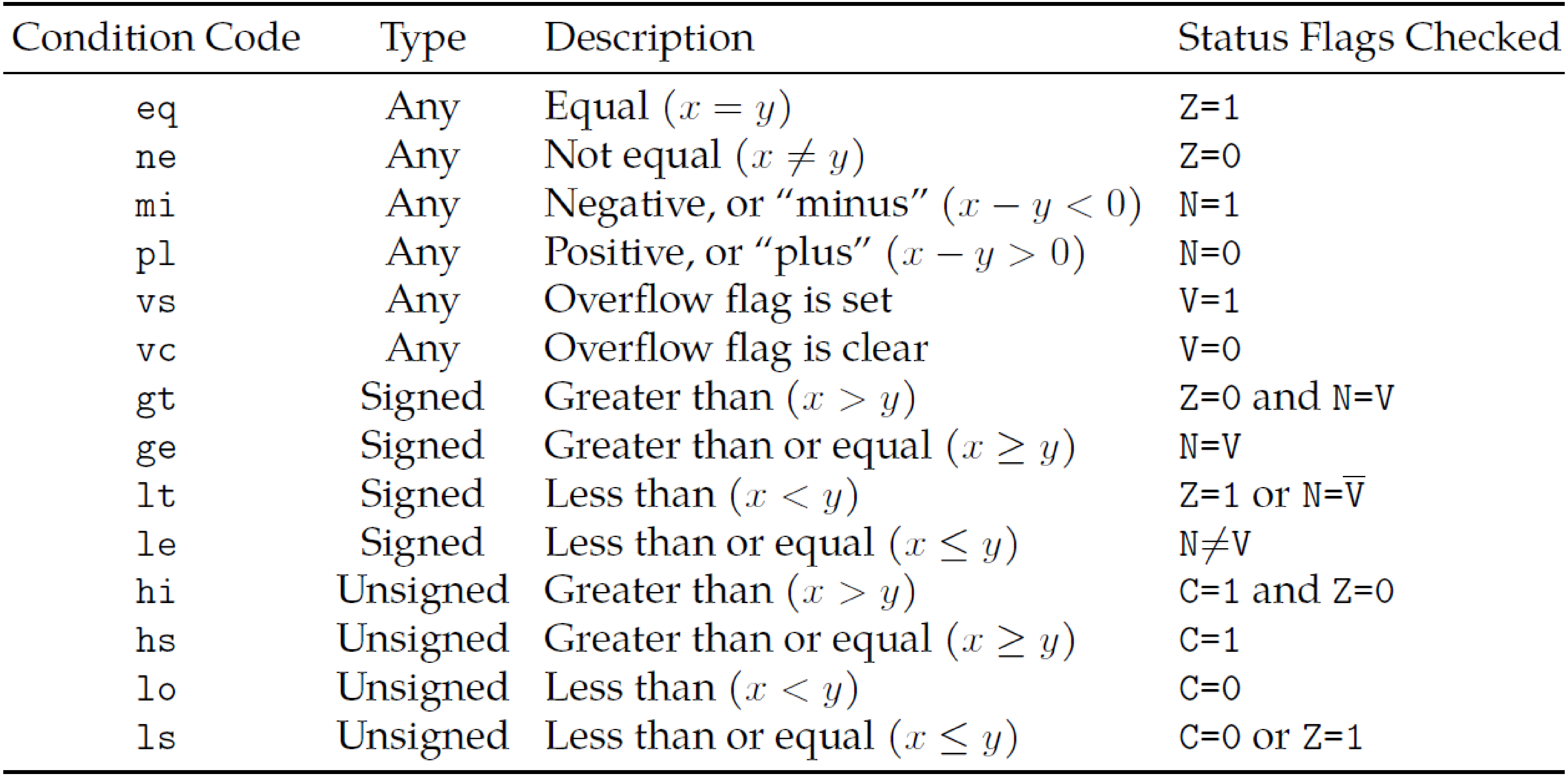
b) if A and B are unsigned numbers

Example 12: What condition flags do you check to determine if A < B

a) if A and B are signed numbers

b) if A and B are unsigned numbers

Additional Examples



Condition Code	Type	Description	Status Flags Checked
eq	Any	Equal $(x = y)$	Z=1
ne	Any	Not equal $(x \neq y)$	Z=0
mi	Any	Negative, or “minus” $(x - y < 0)$	N=1
pl	Any	Positive, or “plus” $(x - y > 0)$	N=0
vs	Any	Overflow flag is set	V=1
vc	Any	Overflow flag is clear	V=0
gt	Signed	Greater than $(x > y)$	Z=0 and N=V
ge	Signed	Greater than or equal $(x \geq y)$	N=V
lt	Signed	Less than $(x < y)$	Z=1 or N=$\overline{\text{V}}$
le	Signed	Less than or equal $(x \leq y)$	N$\neq$V
hi	Unsigned	Greater than $(x > y)$	C=1 and Z=0
hs	Unsigned	Greater than or equal $(x \geq y)$	C=1
lo	Unsigned	Less than $(x < y)$	C=0
ls	Unsigned	Less than or equal $(x \leq y)$	C=0 or Z=1