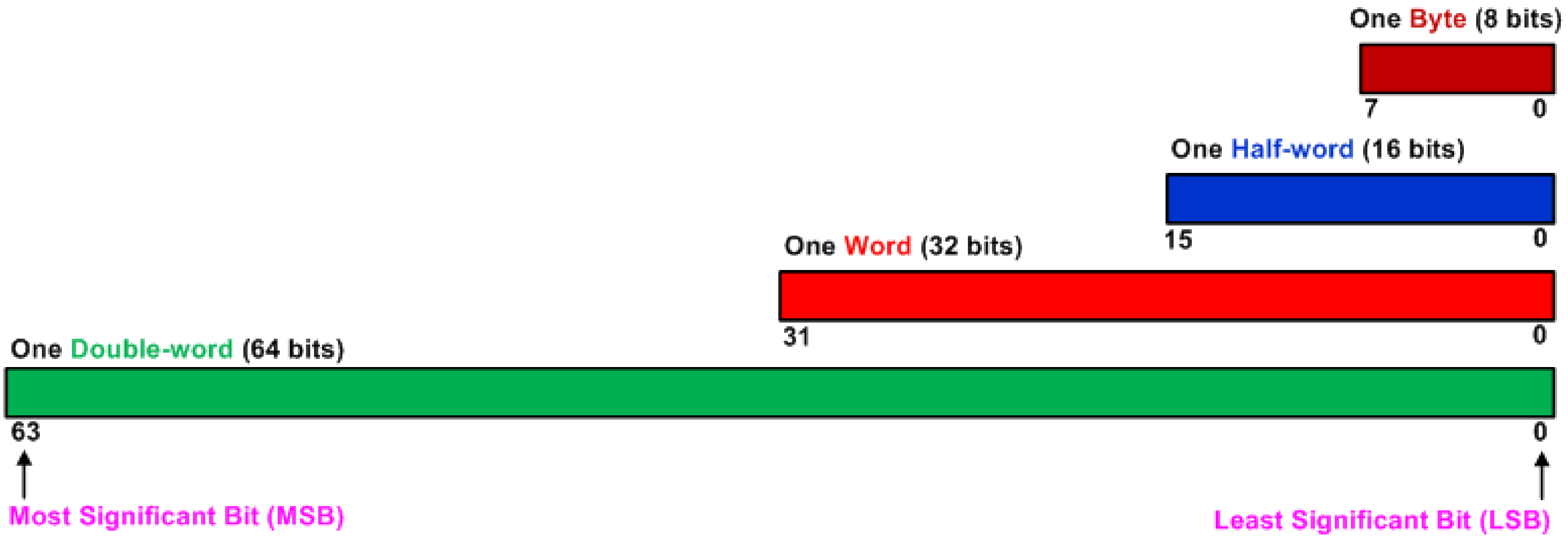
L1 (CHAPTER 2)

Data Representation

Acknowledgement: some slides taken from Yifeng Zhu's courseware

Bit, Byte, Half-word, Word, Double-Word

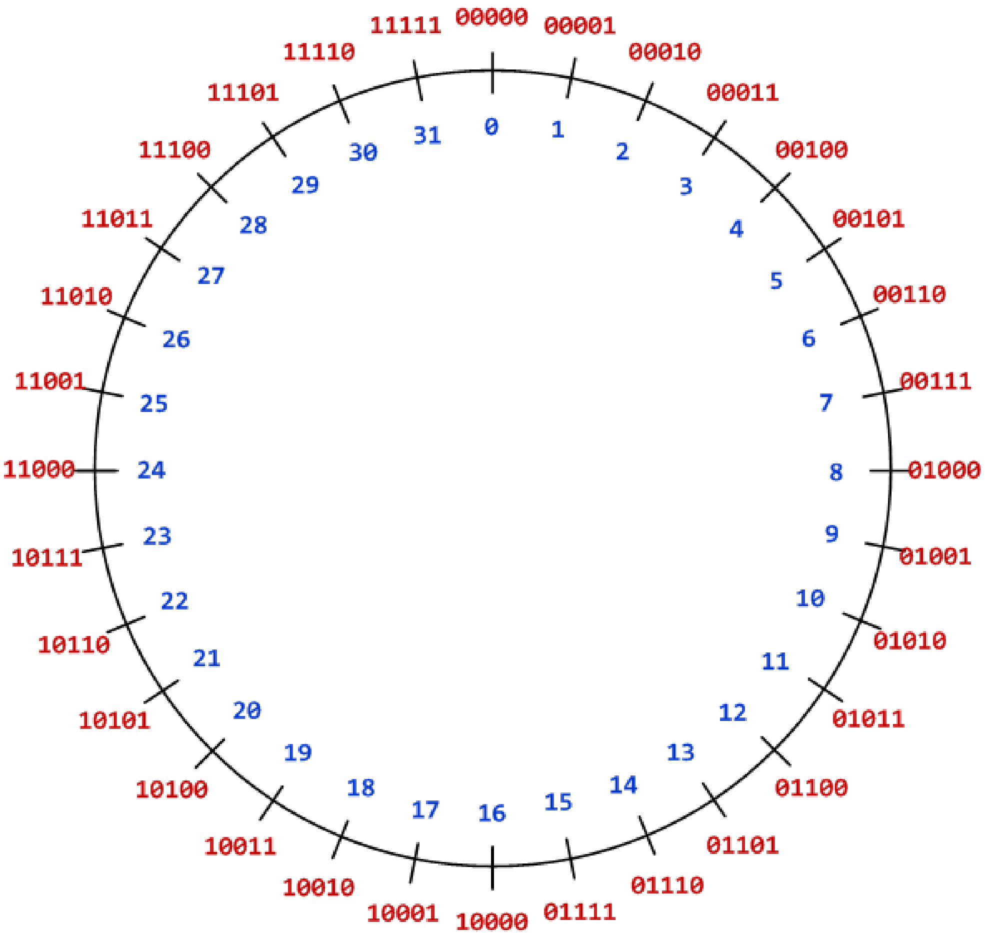


Decimal, Binary and Hex

Decimal	Binary	Hex
0	0000	0x0
1	0001	0x1
2	0010	0x2
3	0011	0x3
4	0100	0x4
5	0101	0x5
6	0110	0x6
7	0111	0x7
8	1000	0x8
9	1001	0x9
10	1010	0xA
11	1011	0xB
12	1100	0xC
13	1101	0xD
14	1110	0xE
15	1111	0xF

Prefix 0x denotes hex

Unsigned Integers



Five-bit binary code

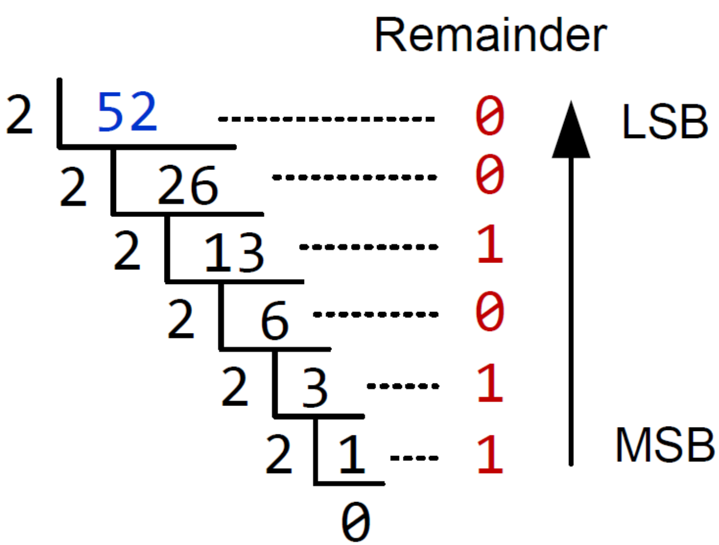
Convert from Binary to Decimal:

$$
\begin{aligned}
1011_2 &= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
&= 8 + 2 + 1 \\
&= 11
\end{aligned}
$$

Unsigned Integers

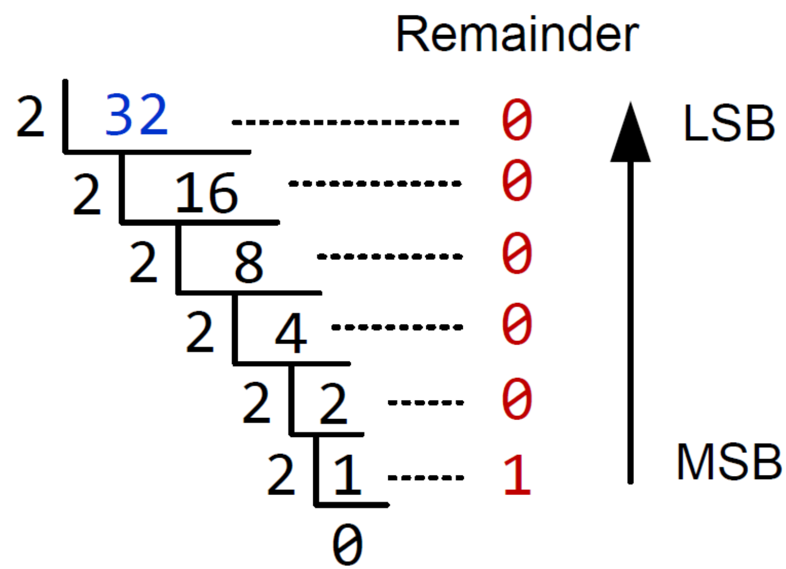
Convert Decimal to Binary

Example 1



$52_{10} = 110100_2$

Example 2



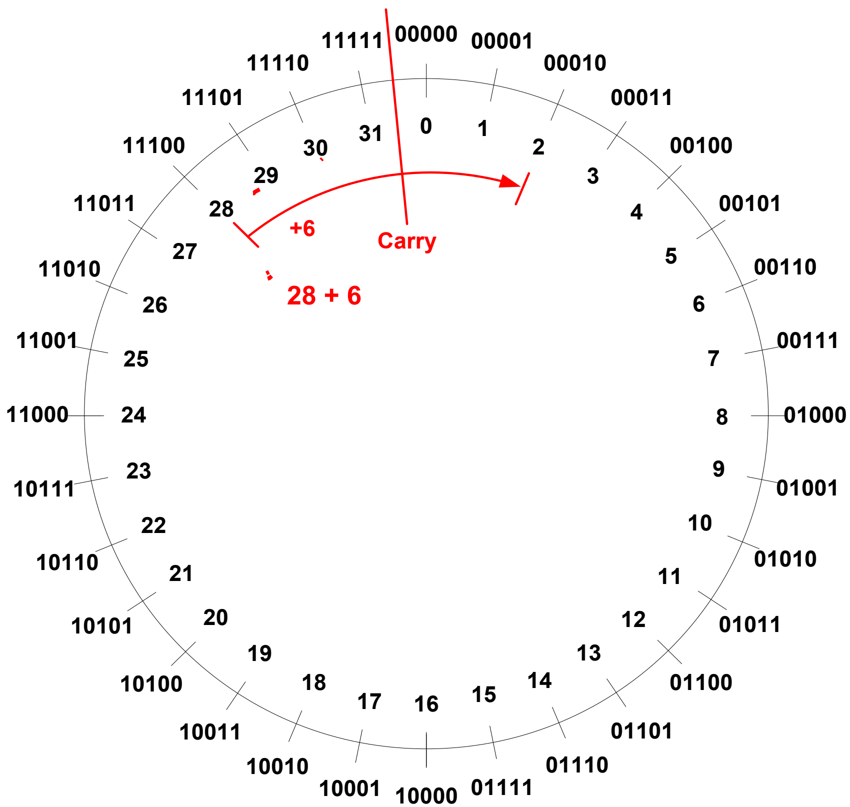
$32_{10} = 100000_2$

Carry/borrow flag bit for unsigned arithmetic

- When adding two unsigned numbers in an *n*-bit system, a carry occurs if the result is larger than the maximum unsigned integer that can be represented (*i.e.* $2^n - 1$).
- When subtracting two unsigned numbers, borrow occurs if the result is negative, smaller than the smallest unsigned integer that can be represented (*i.e.* 0).
- On ARM Cortex-M3 processors, the carry flag and the borrow flag are physically the same flag bit in the CPSR (Current Program Status Register).
  - Carry = NOT Borrow

Carry/borrow flag bit for unsigned arithmetic

If result of addition crosses the boundary between 0 and $2^n - 1$, the carry flag is set.




```
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.

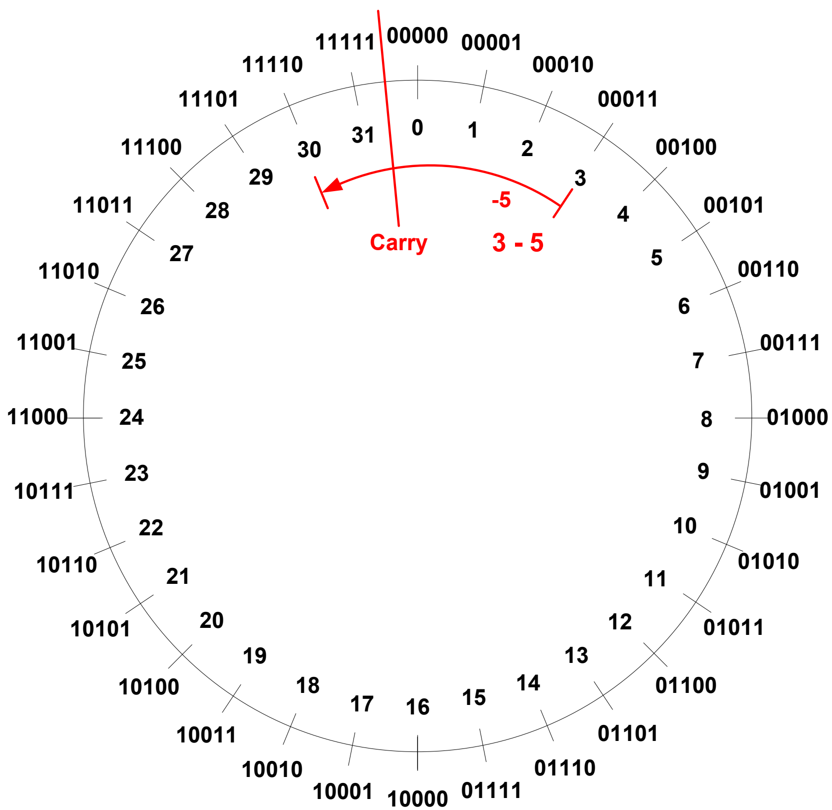
5-bit result

Carry flag = 1, since true result $34 > 2^5 - 1$.

For 5-bit system, a carry occurs when adding 28 and 6

Carry/borrow flag bit for unsigned arithmetic

If result of subtraction crosses the boundary between 0 and $2^n - 1$, the borrow flag is set.



Borrow	1 1 1 0 0	
	0 0 0 1 1	3
-	0 0 1 0 1	- 5
	1 1 1 1 0	30

$\underbrace{1\ 1\ 1\ 1\ 0}_{\text{5-bit result}}$

Carry flag = 0 (Borrow flag = 1), since true result $-2 < 0$.

For 5-bit system, a borrow occurs when subtracting 5 from 3.

Signed Integer Representation Overview

- Three ways to represent signed binary integers:
  - Signed magnitude
    - $\boldsymbol{value} = (-\mathbf{1})^{\boldsymbol{sign}} \times \boldsymbol{Magnitude}$
  - One's complement ($\widetilde{\boldsymbol{\alpha}}$)
    - $\boldsymbol{\alpha} + \widetilde{\boldsymbol{\alpha}} = \mathbf{2}^{\boldsymbol{n}} - \mathbf{1}$
  - Two's complement ($\overline{\boldsymbol{\alpha}}$)
    - $\boldsymbol{\alpha} + \overline{\boldsymbol{\alpha}} = \mathbf{2}^{\boldsymbol{n}}$

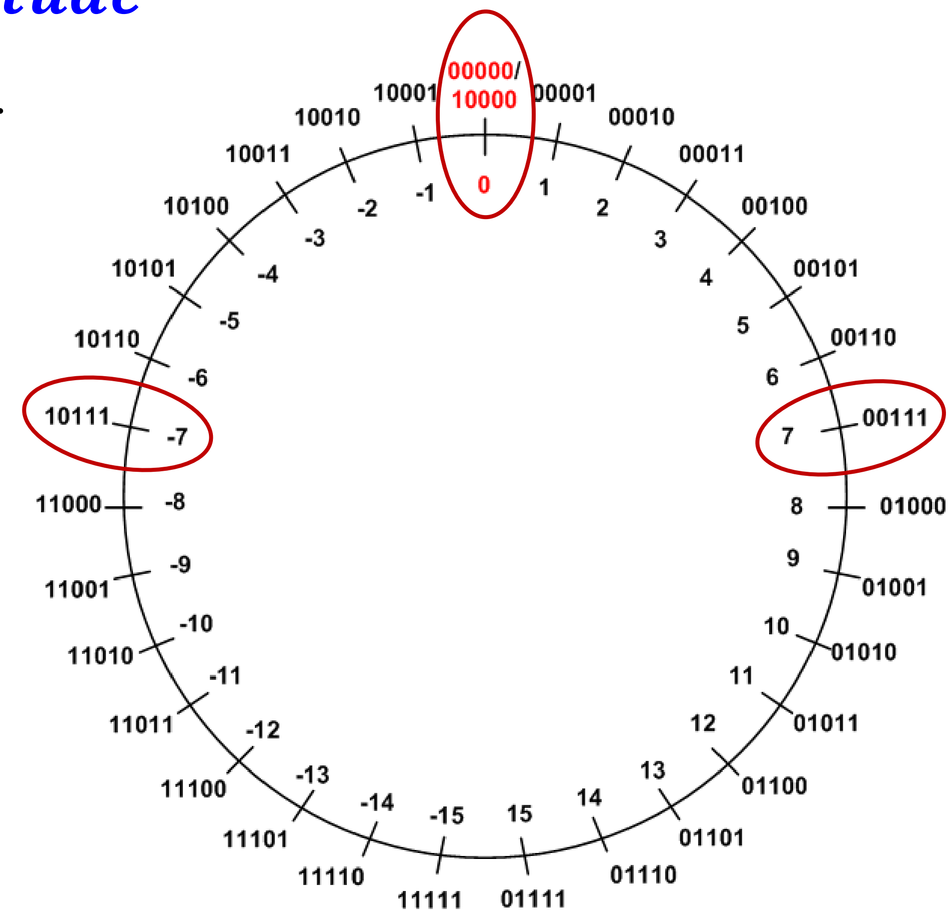
	Sign-and-Magnitude	One's Complement	Two's Complement
Range	$[-2^{n-1}+1, 2^{n-1}-1]$	$[-2^{n-1}+1, 2^{n-1}-1]$	$[-2^{n-1}, 2^{n-1}-1]$
Zero	Two zeroes (± 0)	Two zeroes (± 0)	One zero
Unique Numbers	$2^n - 1$	$2^n - 1$	2^n

Signed Integers
Method 1: Signed magnitude

Sign-and-Magnitude:

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

- The most significant bit is the sign.
- The rest bits are magnitude.

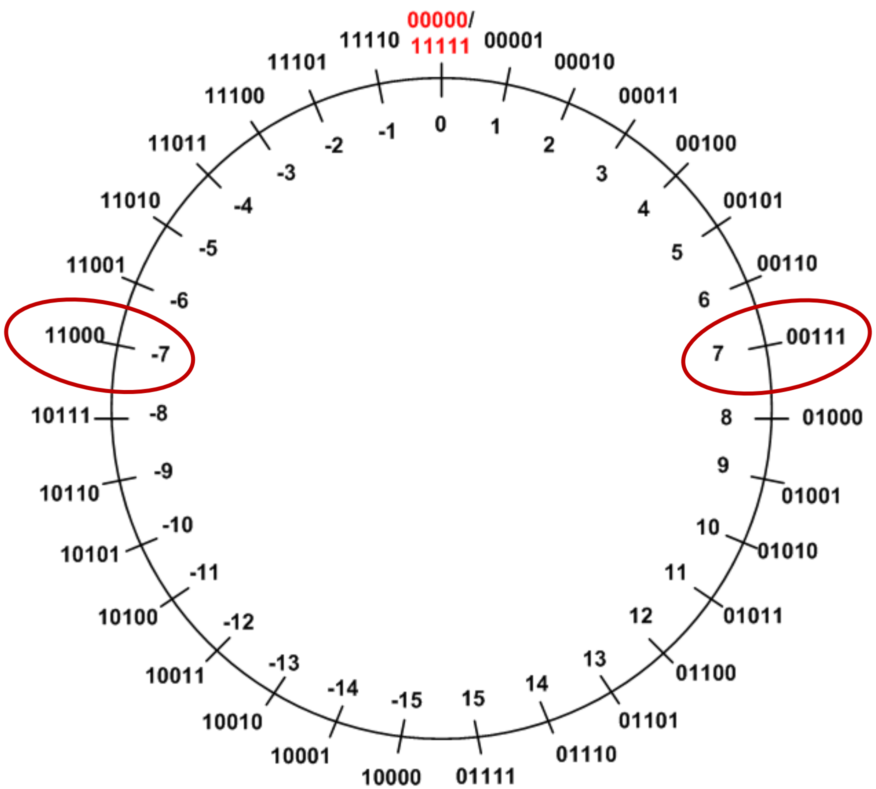


- Example: in a 5-bit system
  - $+7_{10} = 00111_2$
  - $-7_{10} = 10111_2$
- Two ways to represent zero
  - $+0_{10} = 00000_2$
  - $-0_{10} = 10000_2$
- Not used in modern systems
  - Hardware complexity
  - Two zeros

Signed Integers
Method 2: One's Complement

One's Complement ($\widetilde{\alpha}$):

$$\boldsymbol{\alpha + \widetilde{\alpha} = 2^n - 1}$$



The one's complement representation of a negative binary number is the bitwise NOT of its positive counterpart.

Example: in a 5-bit system

$+7_{10} = 00111_2$

$-7_{10} = 11000_2$

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

$= 11111_2$

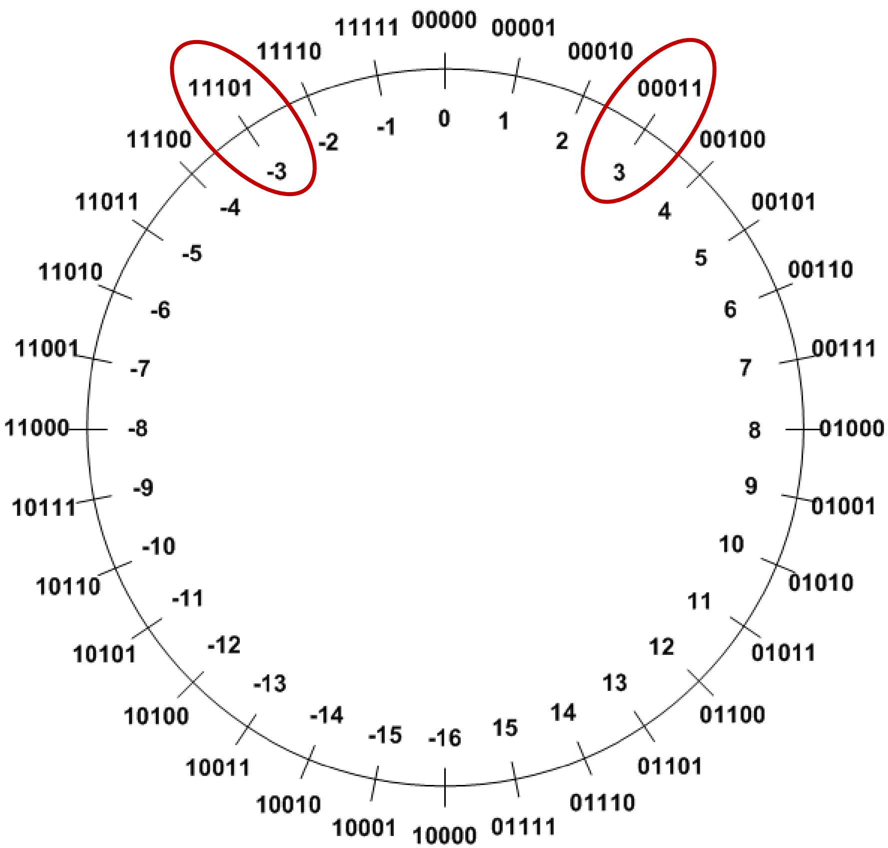
$= 2^5 - 1$

IMPORTANT

Signed Integers
Method 3: Two's Complement

Two's Complement $(\overline{\alpha})$**:**

$$\alpha + \overline{\alpha} = 2^n$$



TC of a number can be obtained by its bitwise NOT plus one.

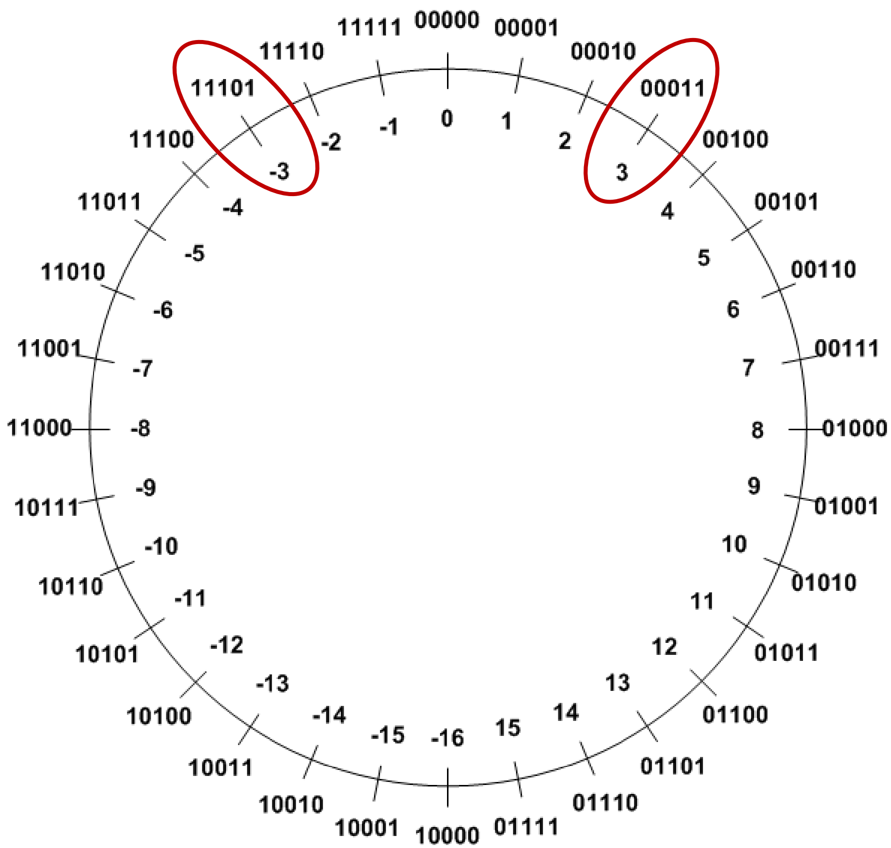
Example **1**: TC(3)

	Binary	Decimal
Original number	00011	3
Step 1: Invert every bit	11100	
Step 2: Add 1	+ 00001	
Two's complement	11101	-3

Signed Integers
Method 3: Two's Complement

Two's Complement (TC)

$$\alpha + \overline{\alpha} = 2^n$$



TC of a number can be obtained by its bitwise NOT plus one.

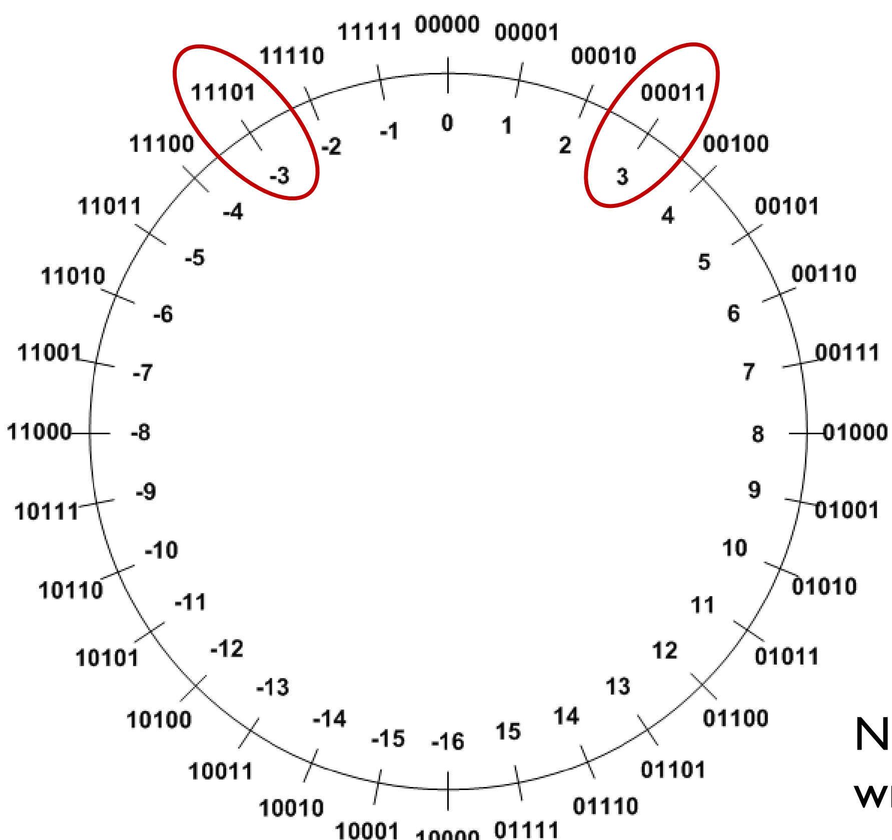
Example 2: TC(-3)

	Binary	Decimal
Original number	11101	-3
Step 1: Invert every bit	00010	
Step 2: Add 1	+ 00001	
Two's complement	00011	3

Signed Integers
Method 3: Two's Complement

Two's Complement (TC)

$$\alpha + \overline{\alpha} = 2^n$$



TC of a number can be obtained by its bitwise NOT plus one.

Example 2: TC(-16)

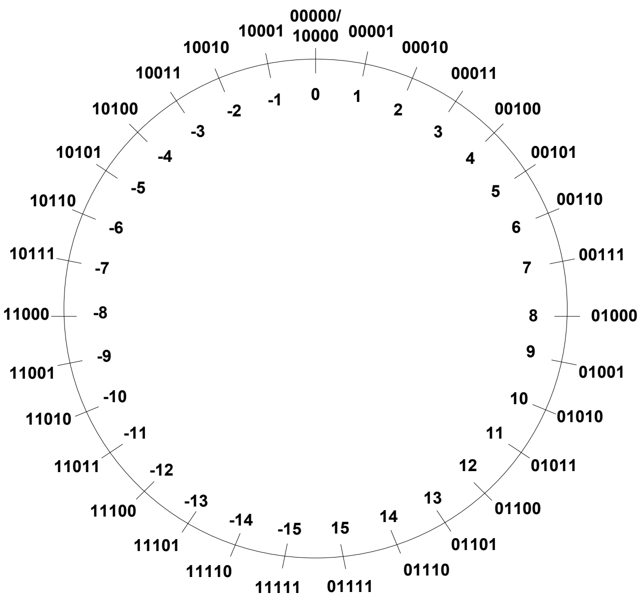
	Binary	Decimal
Original number	10000	-16
Step 1: Invert every bit	01111	
Step 2: Add 1	+ 10000	
Two's complement	10000	-16

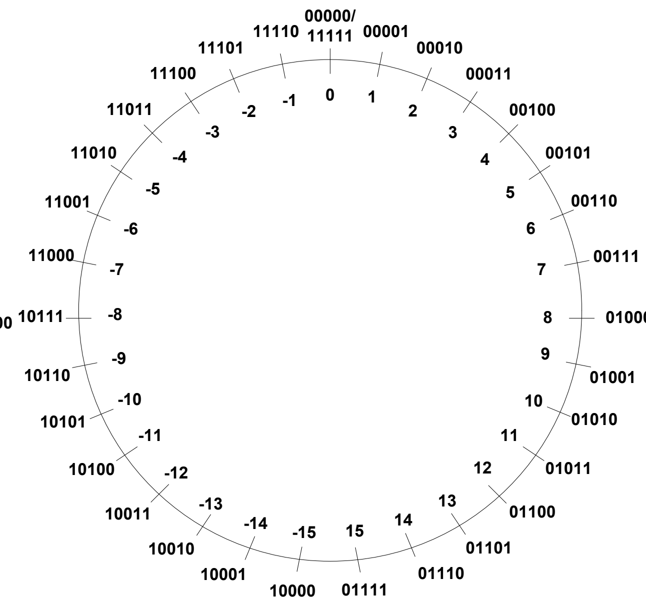
Negation of -16 in 5-bit two's complement wraps back to itself, meaning the most negative number's two's complement is itself. (Number range is [-16, 15], so 16 is out of range)

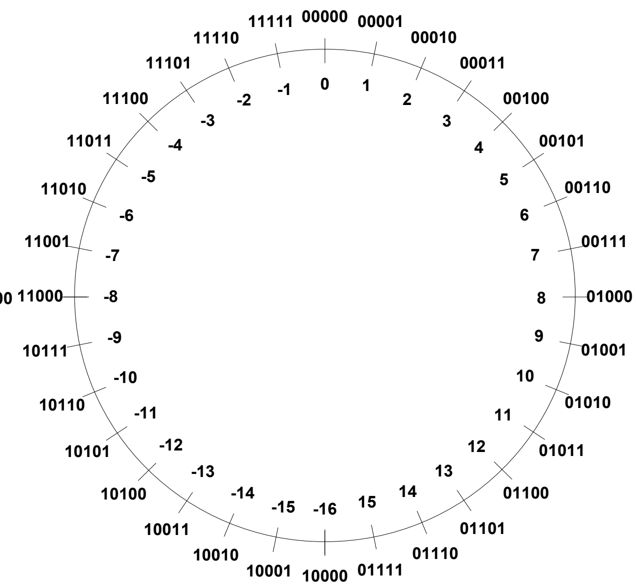
Quiz

- Calculate TC(-6) for 6-bit system
- For a 6-bit two's complement number: the range of representable integers is from −32 to +31.
- −16 in 6-bit two's complement is 110000
  - Write 16 in binary: 010000
  - Take the two's complement (invert bits and add 1):
  - Invert bits: 101111, Add 1: 101111+1=110000
- To take the negation of this (i.e., find the two's complement of 110000):
  - Invert bits: 001111, Add 1: 001111+1=010000
  - This is 16 in binary, so the negation of −16 is +16 as expected.
- Unlike the 5-bit case where −16 is the minimum and its negation wraps onto itself, in 6 bits −16 behaves normally with correct negation.

Comparison: different signed reps





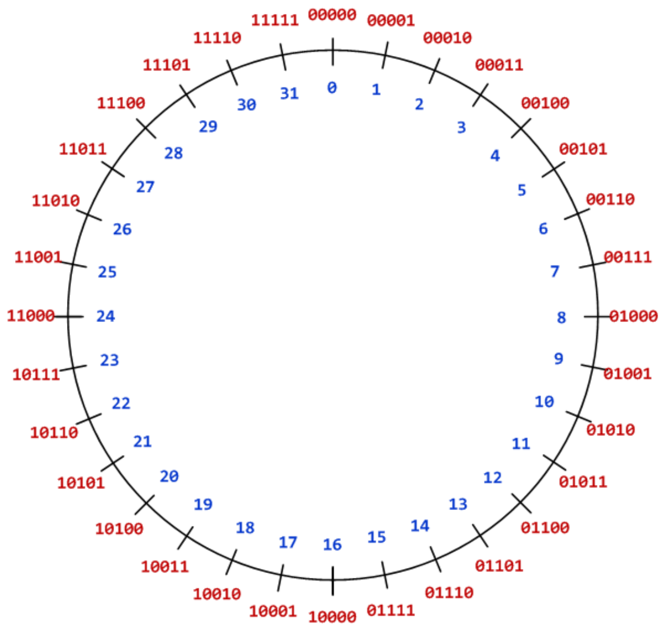


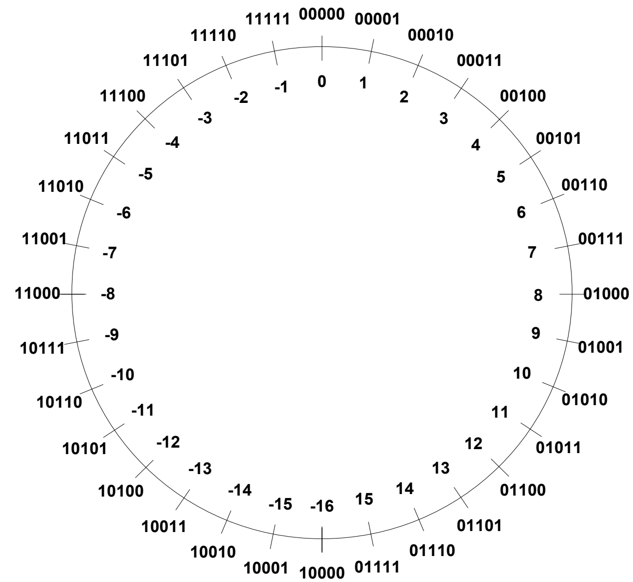
Signed magnitude representation
Range [-15,15]
0 = positive
1 = negative

One's complement representation
Range [-15,15]
Negative = invert all bits of a positive

Two's Complement representation
Range [-16,15]
TC = invert all bits, then plus 1

Comparison: unsigned vs. signed





Unsigned int representation
Range [0, 31]

Two's Complement representation
Range [-16,15]
TC = invert all bits, then plus 1

Unsigned vs. Signed (TC)

	Unsigned	Two's Complement Signed
Range	$[0, 2^{n-1}]$	$[-2^{n-1}, 2^{n-1} - 1]$
Zero	One zero	One zero
Unique Numbers	2^n	2^n

Two's Complement for 8-bit System

8-bit signed Int (Two's Complement)	8-bit unsigned Int	Binary
-128	128	1 0 0 0 0 0 0 0
-127	129	1 0 0 0 0 0 0 1
…	…	…
-2	254	1 1 1 1 1 1 1 0
-1	255	1 1 1 1 1 1 1 1
0	0	0 0 0 0 0 0 0 0
1	1	0 0 0 0 0 0 0 1
…	…	. . .
127	127	0 1 1 1 1 1 1 1

Note: Most significant bit (MSB) equals sign for signed int

Sign Extension

Decimal	Binary		
	4-bit	8-bit	32-bit
3_{ten}	0011_{two}	$0000\ 0011_{two}$	$0000\ 0000\ 0000\ 0011_{two}$
-3_{ten}	1101_{two}	$1111\ 1101_{two}$	$1111\ 1111\ 1111\ 1101_{two}$

- Assignment differs for signed (above table) and unsigned numbers
  - Compiler knows (from type declaration)
  - Different assembly instructions for copying signed/unsigned data

Overflow flag for signed arithmetic

- When adding or subtracting two signed numbers in an *n*-bit system, an overflow occurs if the true result is larger than the maximum signed integer (*i*.e. $2^{n-1} - 1$) or smaller than the minimum signed integer (*i*.e. -2^{n-1}) that can be represented

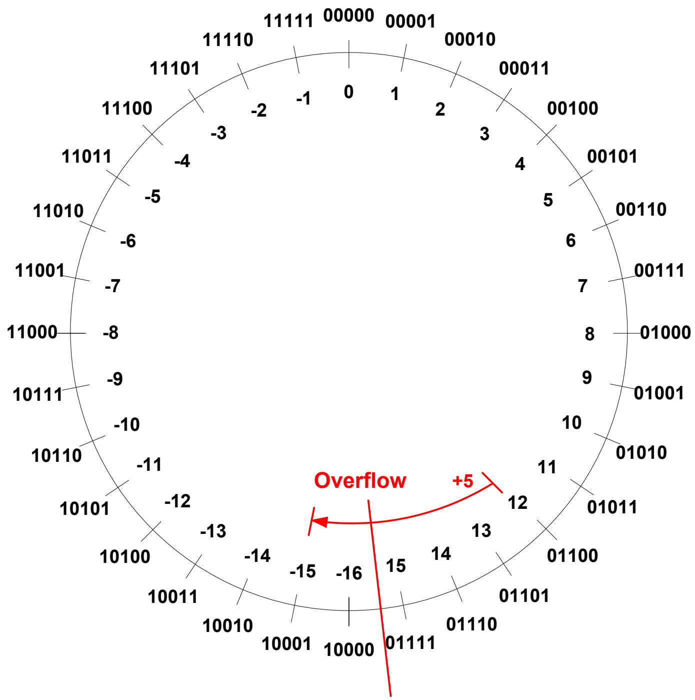
- Overflow may occur when adding 2 operands with the same sign, or subtracting 2 operands with different signs, including:
  1. adding two positive numbers
  2. adding two negative numbers
  3. subtracting a positive number from a negative number
  4. subtracting a negative number from a positive number

- Overflow cannot occur when adding 2 operands with different signs or when subtracting 2 operands with the same sign.
  - Why?

Overflow flag for signed arithmetic

- Overflow cannot occur when adding 2 operands with different signs or when subtracting 2 operands with the same sign. Proof:
  - A n-bit signed int has the range $[-2^{n-1}, 2^{n-1}-1]$
    - n = 4, number range [-16, 15]
  - 2 operands with different signs: positive one in the range of $[0, 2^{n-1}-1]$, negative one in the range of $[-2^{n-1}, -1]$. So the range of their sum must be $[0-2^{n-1}, 2^{n-1}-1+(-1)]=[-2^{n-1}, 2^{n-1}-2] \in [-2^{n-1}, 2^{n-1}-1]$
    - Positive number range [0, 15], negative number range [-16, -1]. Range of their sum [0-16, 15-1]=[-16, 14]
  - 2 operands with the same sign: if both are positive and in the range of $[0, 2^{n-1}-1]$, then the range of their difference must be $[0-(2^{n-1}-1), 2^{n-1}-1-0]=[-(2^{n-1}-1), 2^{n-1}-1]$; if both are negative and in the range of $[-2^{n-1}, -1]$, then the range of their difference must be $[-2^{n-1}-(-1), -1-(-2^{n-1})]=[-2^{n-1}+1, 2^{n-1}-1] \in [-2^{n-1}, 2^{n-1}-1]$
    - Both positive numbers [0, 15], range of difference [0-15, 15-0]=[-15, 15]
    - Both negative numbers [-16, -1], range of difference [-16-(-1), -1-(-16)]=[-15, 15]

Overflow bit flag for signed arithmetic



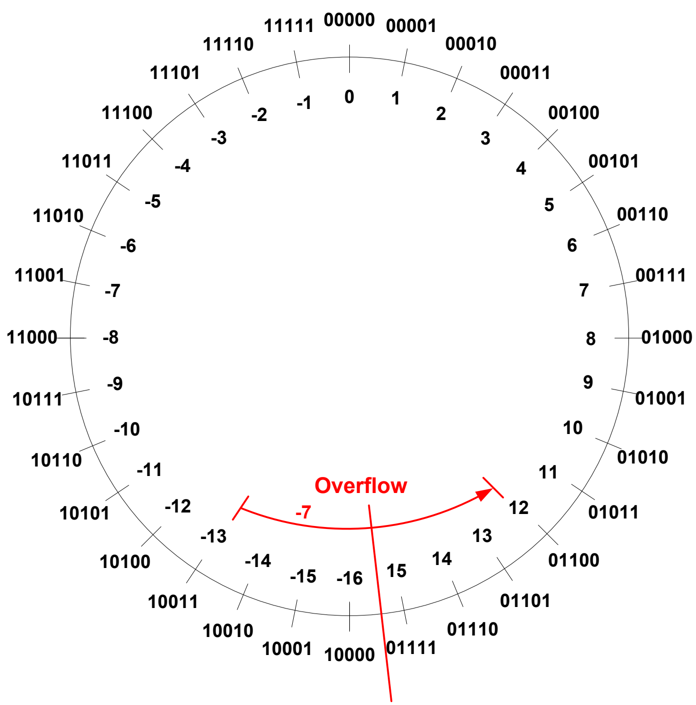
$$
\begin{array}{rccccc}
 & 0 & 1 & 1 & 0 & 0 \\
+ & 0 & 0 & 1 & 0 & 1 \\
\hline
 & 1 & 0 & 0 & 0 & 1
\end{array}
\qquad
\begin{array}{rr}
 & 12 \\
+ & 5 \\
\hline
 & -15
\end{array}
$$

$\underbrace{1\ 0\ 0\ 0\ 1}_{\text{5-bit result}}$

Overflow flag = 1, since true result $17 > 2^4 - 1$.

An overflow occurs when adding two positive numbers and getting a negative result.

Overflow bit flag for signed arithmetic



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

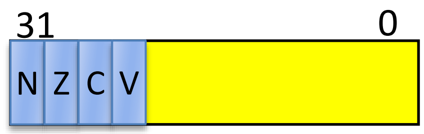
Extra bit is discarded.

5-bit result

Overflow flag = 1, since true result $-20 < -2^4$.

An overflow occurs when adding two negative numbers and getting a positive result.

Carry and Overflow Flags in CPSR



CPSR (Current Program Status Register)

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

25

IMPORTANT

Summary of Carry and Overflow Flags

Carry flag C = 1 (Borrow flag = 0) upon an **unsigned** addition if the answer is wrong (true result > 2^n-1)

Carry flag C = 0 (Borrow flag = 1) upon an **unsigned** subtraction if the answer is wrong (true result < 0)

Overflow flag V =1 upon a **signed** addition or subtraction if the answer is wrong (true result > 2^{n-1}-1 or true result < -2^{n-1})

	Unsigned Addition	Unsigned Subtraction	Signed Addition or Subtraction
Carry flag	true result > 2^n-1 ➔ Carry flag=1 (Result incorrect)	true result < 0 ➔ Carry flag=0 (Result incorrect)	N/A
Overflow flag	N/A	N/A	true result > 2^{n-1}-1 or true result < -2^{n-1} ➔ Overflow flag=1 (Result incorrect)

26

Signed or unsigned

- Whether the carry flag or the overflow flag should be used depends on the programmer's intention.

```
uint a;
uint b;
…
c = a+/-b
…
```

C Program

Check the carry/borrow Flag for unsigned addition/ subtraction

```
int a;
int b;
…
c = a+/-b
…
```

C Program

Check the overflow flag for signed addition/subtraction

Signed or Unsigned

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

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

- CPU does not know; it sets up both the carry flag and the overflow flag.
- It is software's (programmer/compiler) responsibility to interpret the flags.
  - The C compiler uses either the carry or the overflow flag based on how this integer is declared in source code ("uint" or "int").

Signed or Unsigned

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

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

If unsigned:

```
uint a, b;
a = 16
b = 16
c = a + b
  = 32 > 2^5-1
Carry flag set
```

If signed:

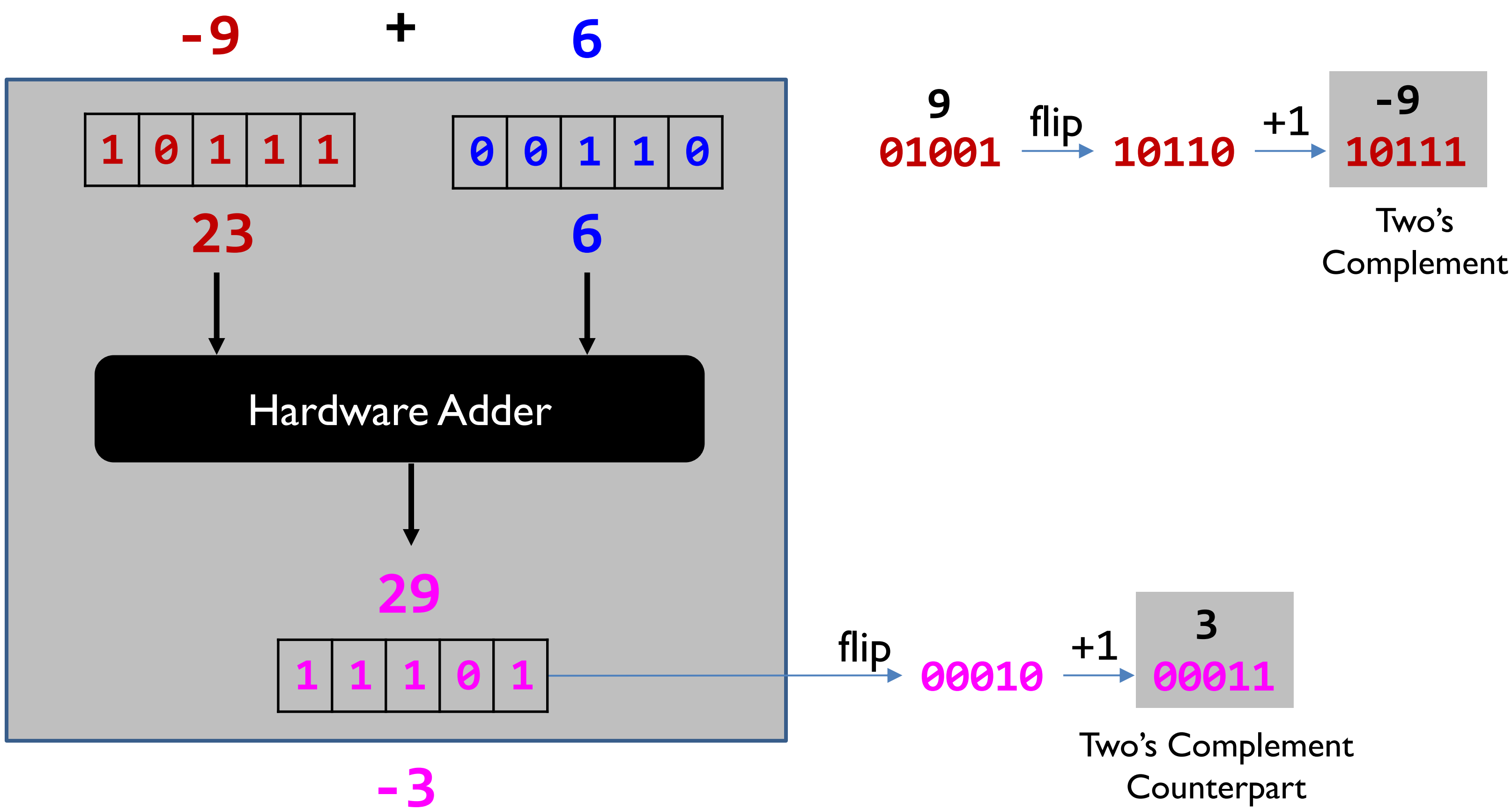
```
int a, b;
a = -16
b = -16
c = a + b
  = -32 < -2^4
Overflow flag set
```

Why use Two's Complement

Two's complement representation simplifies hardware

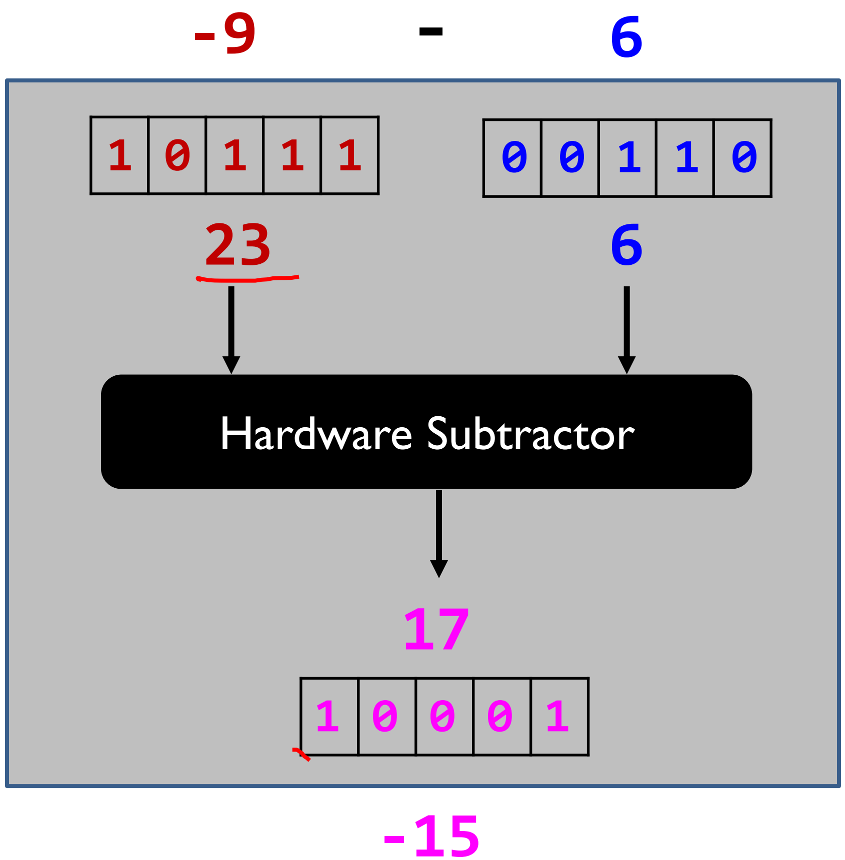
Operation	Are signed and unsigned operations the same?
Addition	Yes
Subtraction	Yes
Multiplication	Yes if the product has the same number of bits as operands (not discussed in class)
Division	No (not discussed in class)

Adding two integers



- Same bit patterns, different interpretation.
  - Unsigned addition: 23+6=29
  - Signed addition: -9+6=-3
- This example shows that the hardware adder for adding unsigned numbers, also works correctly for adding signed numbers.

Subtracting two signed integers: (-9) - 6



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

Two's Complement

15 → 01111 $\xrightarrow{\text{flip}}$ 10000 $\xrightarrow{+1}$ 10001 (-15)

- Same bit patterns, different interpretation.
  - Unsigned subtraction: 23-6=17
  - Signed subtraction: -9-6=-15
- This example shows that the hardware subtractor for subtracting unsigned numbers, also works correctly for subtracting signed numbers.

Two's Complement Simplifies Hardware Implementation

- In two's complement, the same hardware works correctly for both signed and unsigned addition/subtraction.
- If the product has the same number of bits as operands, the same hardware works correctly for both signed and unsigned multiplication.
- However, this is not true for division.

American
Standard
Code for
Information
Interchange

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	NUL	32	20	SP	64	40	@	96	60	‘
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	’	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Encoding 128 characters

ASCII

char str[13] = “ARM Assembly”;
// The length has to be at least 13
// even though it has 12 letters. The
// NULL terminator should be included.

Memory Address	Memory Content	Letter
str + 12 →	0x00	\0
str + 11 →	0x79	y
str + 10 →	0x6C	l
str + 9 →	0x62	b
str + 8 →	0x6D	m
str + 7 →	0x65	e
str + 6 →	0x73	s
str + 5 →	0x73	s
str + 4 →	0x41	A
str + 3 →	0x20	space
str + 2 →	0x4D	M
str + 1 →	0x52	R
str →	0x41	A

String Comparison

Strings are compared based on their ASCII values

- “j” < “jar” < “jargon” < “jargonize”
- “CAT” < “Cat” < “DOG” < “Dog” < “cat” < “dog”
- “12” < “123” < “2”< “AB” < “Ab” < “ab” < “abc”

Find out String Length

- Stings are terminated with a null character (NUL, ASCII value `0x00`)

Pointer dereference operator *	Array subscript operator [ ]

<pre>int strlen (char *pStr){
    int i = 0;
 	
    // loop until pStr[i] is NULL
    while(pStr[i])
        i++;
 	
    return i;
}</pre>
	<pre>int strlen (char *pStr){
	    int i = 0;
	
	    // loop until *pStr is NULL
	    while(*pStr) {
	        i++;
	        pStr++;
	    }
	    return i;
	}</pre>

Convert to Upper Case

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59	5A

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	71	72	73	74	75	76	77	78	79	7A

‘a’ – ‘A’ = 0x61 – 0x41 = 0x20 = 32

Pointer dereference operator *

```
void toUpper(char *pStr){
  for(char *p = pStr; *p; ++p){
    if(*p >= ’a’ && *p <= ’z’)
      *p -= ‘a’ – ‘A’;
      //or: *p -= 32;
  }
}
```

Array subscript operator [ ]

```
void toUpper(char *pStr){
  char c = pStr[0];
  for(int i = 0; c; i++, c = pStr[i];) {
    if(c >= ’a’ && c <= ’z’)
      pStr[i] -= ‘a’ – ‘A’;
      // or: pStr[i] -= 32;
  }
}
```

Summary

- Unsigned integer arithmetic
- Signed integer arithmetic
  - 2's complement
- ASCII strings

References

- Lecture 1. Why use two's complement?
  - https://www.youtube.com/watch?v=lJCefqV80ck&list=PLRJhV4hUhIymmp5CCeIFPyxbknsdcXCc8&index=1
- Lecture 2: Carry flag for unsigned addition and subtraction
  - https://www.youtube.com/watch?v=MxGW2WurKuM&list=PLRJhV4hUhIymmp5CCeIFPyxbknsdcXCc8&index=2
- Lecture 3: Overflow flag for signed addition and subtraction
  - https://www.youtube.com/watch?v=BIn6iyYIGio&list=PLRJhV4hUhIymmp5CCeIFPyxbknsdcXCc8&index=3