Trinity College Dublin

Coláiste na Tríonóide, Baile Átha Cliath

The University of Dublin

5.1 (Binary) Arithmetic

CSU11021 – Introduction to Computing I

Dr Jonathan Dukes | jdukes@tcd.ie

School of Computer Science and Statistics

Decimal numeral system

We are most familiar with the decimal numeral system

10 symbols: **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**

What happens if I want to represent this number of apples?

Counting the apples … 1, 2, 3, 4, 5, 6, 7, 8, 9 … I've run out of digits!

… but, if we write down a digit represent the count of 10s of apples

… followed by another digit representing the count of single (unit) apples

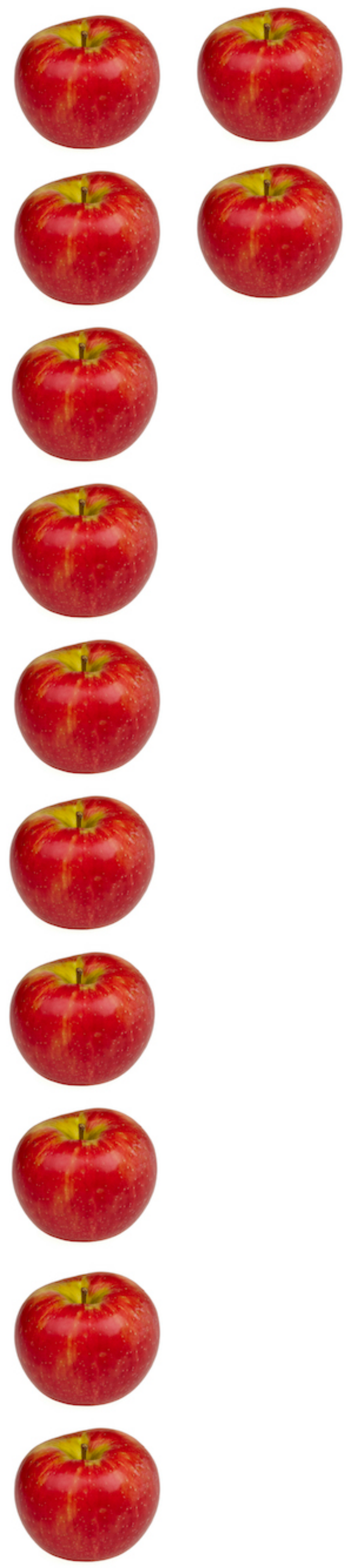
… then we can express the number of apples as **12** or $(1 \times 10^1) + (2 \times 10^0)$

This method of expressing a value is known as a "positional"

because the position of a digit corresponds to the magnitude of its contribution to the overall quantity (number of 1000s of apples, number of 100s of apples, number of 10s of apples and number of single apples)

with the rightmost digit (the least significant digit) corresponding to 10^0 (=1)

the next rightmost digit corresponding to 10^1, then 10^2, then 10^3 etc.



Binary numeral system

Binary is another positional numeral system

2 symbols: **0**, **1**

What happens if we want to r[illegible]he same number of apples in binary?

Counting the apples … 0, 1 … we've [illegible]gits!

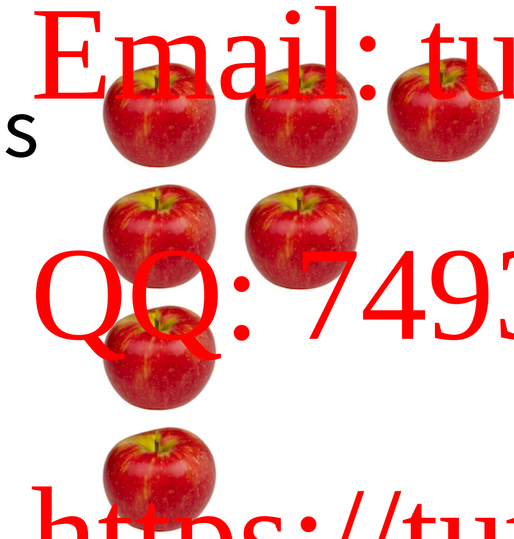
… but, if we write down a digit represent the count of 2s of apples

… followed by another digit representing the count of single (unit) apples

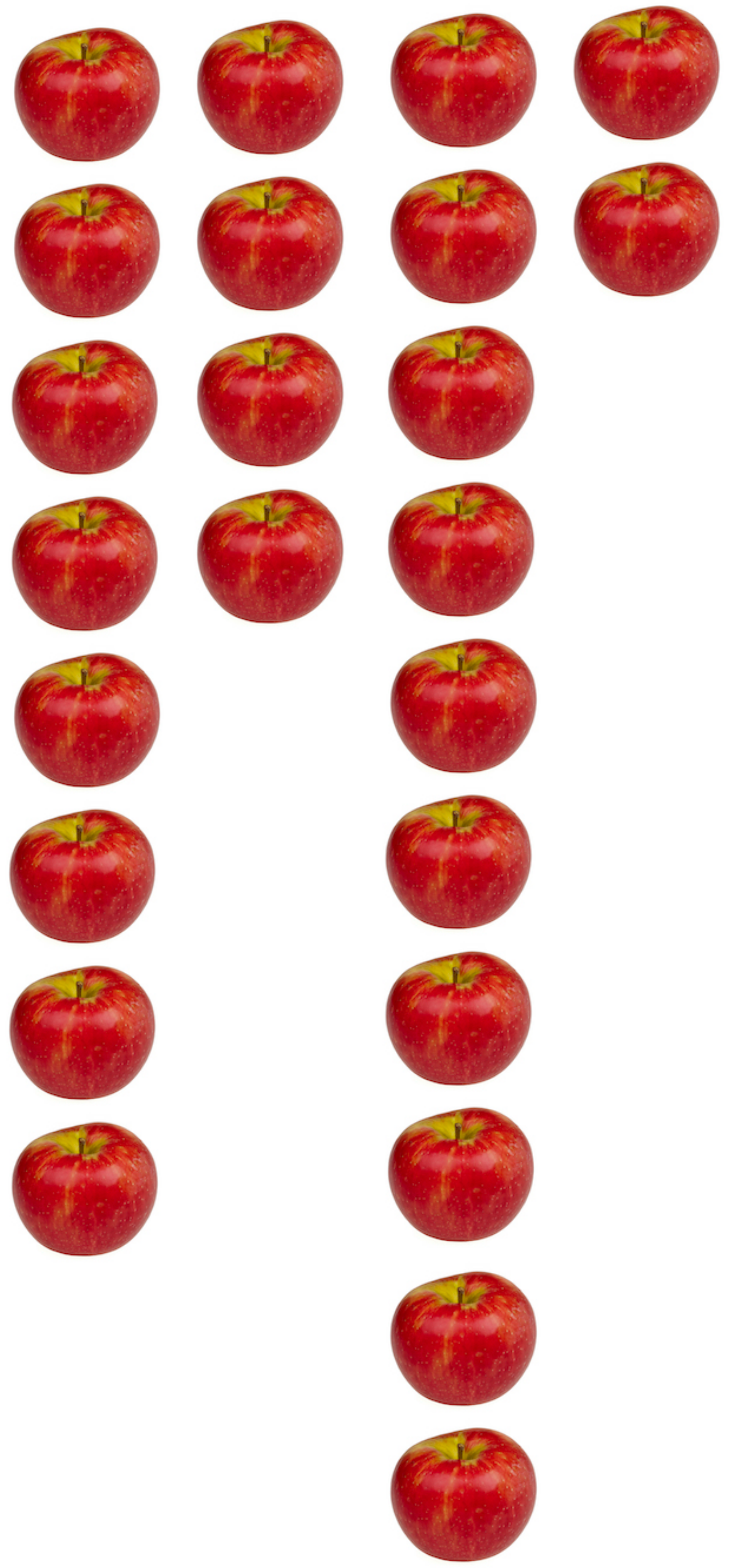
… we can count up to 11 apples

… so we need another digit, this time representing the count of 4s of apples (4=2^2)

… now we can represent 111 apples



Still not enough digits!



If we follow the same pattern with one more digit, we can represent the number of apples as **1100** or (1 x 2^3) + (1 x 2^2) + (0 x 2^1) + (0 x 2^0)

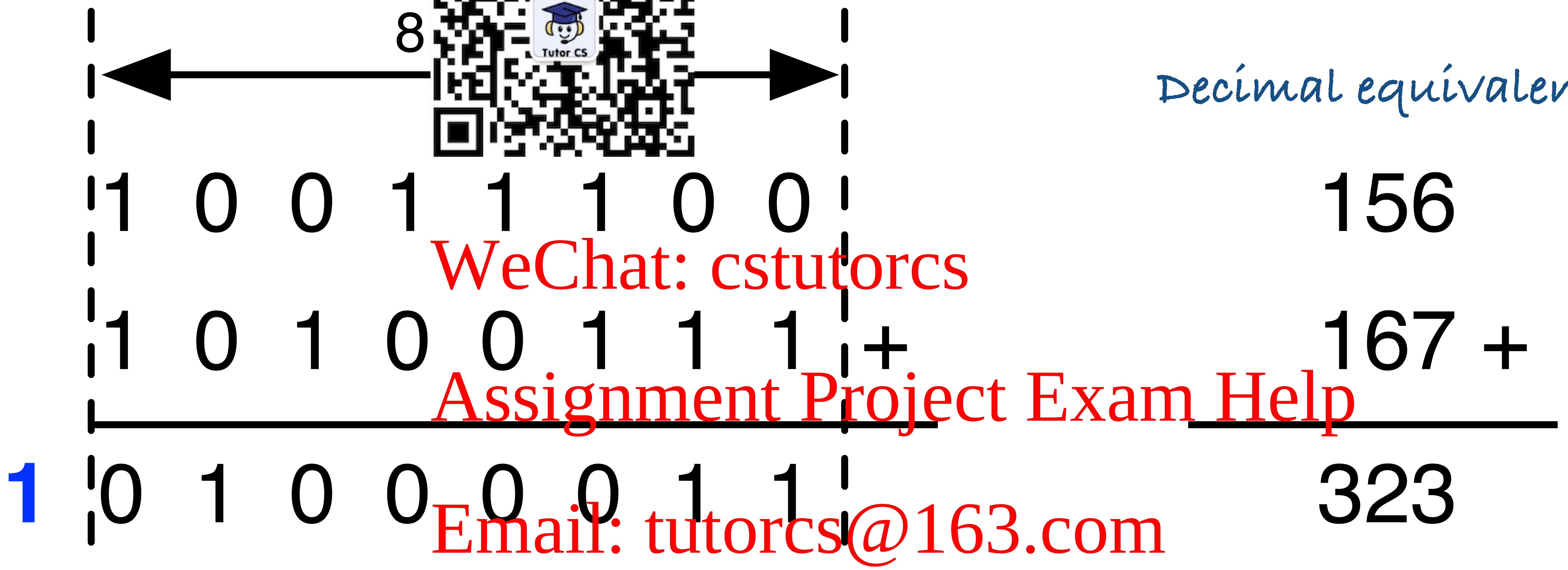
Binary Arithmetic – Addition

Binary	Decimal equivalent
0 0 0 0 0 1 1 0	6
0 0 0 0 1 0 1 1 +	11 +
0 0 0 1 0 0 0 1	17
0 0 0 1 0 1 1 0	22
0 0 0 0 1 0 1 1 +	11 +
0 0 1 0 0 0 0 1	33

Carry

What happens if we run out of digits?

Adding two numbers each stored in 1 byte (8 bits) may produce a 9-bit result



1 0 0 1 1 1 0 0 156

1 0 1 0 0 1 1 1 + 167 +

1 0 1 0 0 0 0 1 1 323

Added $156_{10} + 167_{10}$ and expected to get 323_{10}

8-bit result was 01000011_2 or 67_{10}

Largest number we can represent in 8-bits is 255

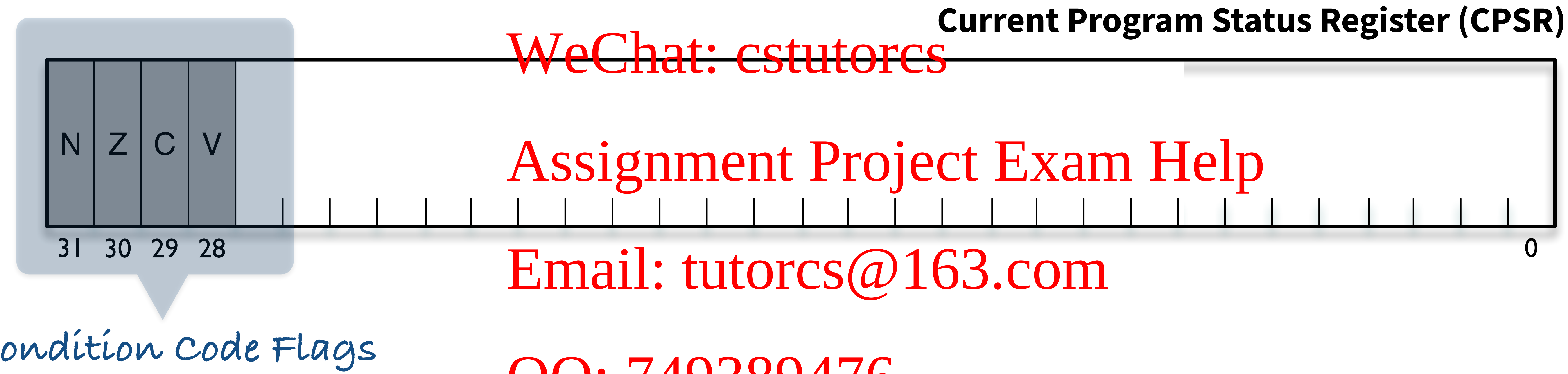
The “missing” or “left-over” 1 is called a ***carry*** (or ***carry-out***)

8-bits just for illustration here. Our ARM processor has 32-bit registers and performs 32-bit arithmetic so we get a carry-out if our result requires 33 bits.

Condition Code Flags

Some instructions can **optionally** update the Condition Code Flags to provide information ab[illegible]esult of the execution of the instruction

e.g. whether the result of an [illegible]s zero, or negative or whether a carry occurred



N – Negative	Z – Zero
V – oVerflow	C – Carry

Condition Code Flags

The Condition Code Flags (N, Z, C, V) can be **optionally** updated to reflect the result of an instruction

S-bit in a machine code instruction is used to tell the processor whether the Condition Code Flags should be updated, based on the result

e.g. want to update Condition Code Flags during an ADD instruction

Condition Code Flags only updated if (machine code) S-bit (bit 20) is 1

ADD instruction machine code

1	1	1	0	0	0	0	0	1	0	0	S	Rn	Rd	0	0	0	0	0	0	0	0	Rm
31										21	20	19 16	15 12	11							4	3 0

In assembly language, we cause the Condition Code Flags to be updated by appending “S” to the instruction mnemonic (e.g. ADDS, SUBS, MOVS)

Carry Flag

```
        LDR     R0, =0xC0000000
        LDR     R1, =0x70000000
        ADDS    R0, R0, R1

stop    B       stop
```

ADDS causes the Condition Code Flags to be updated

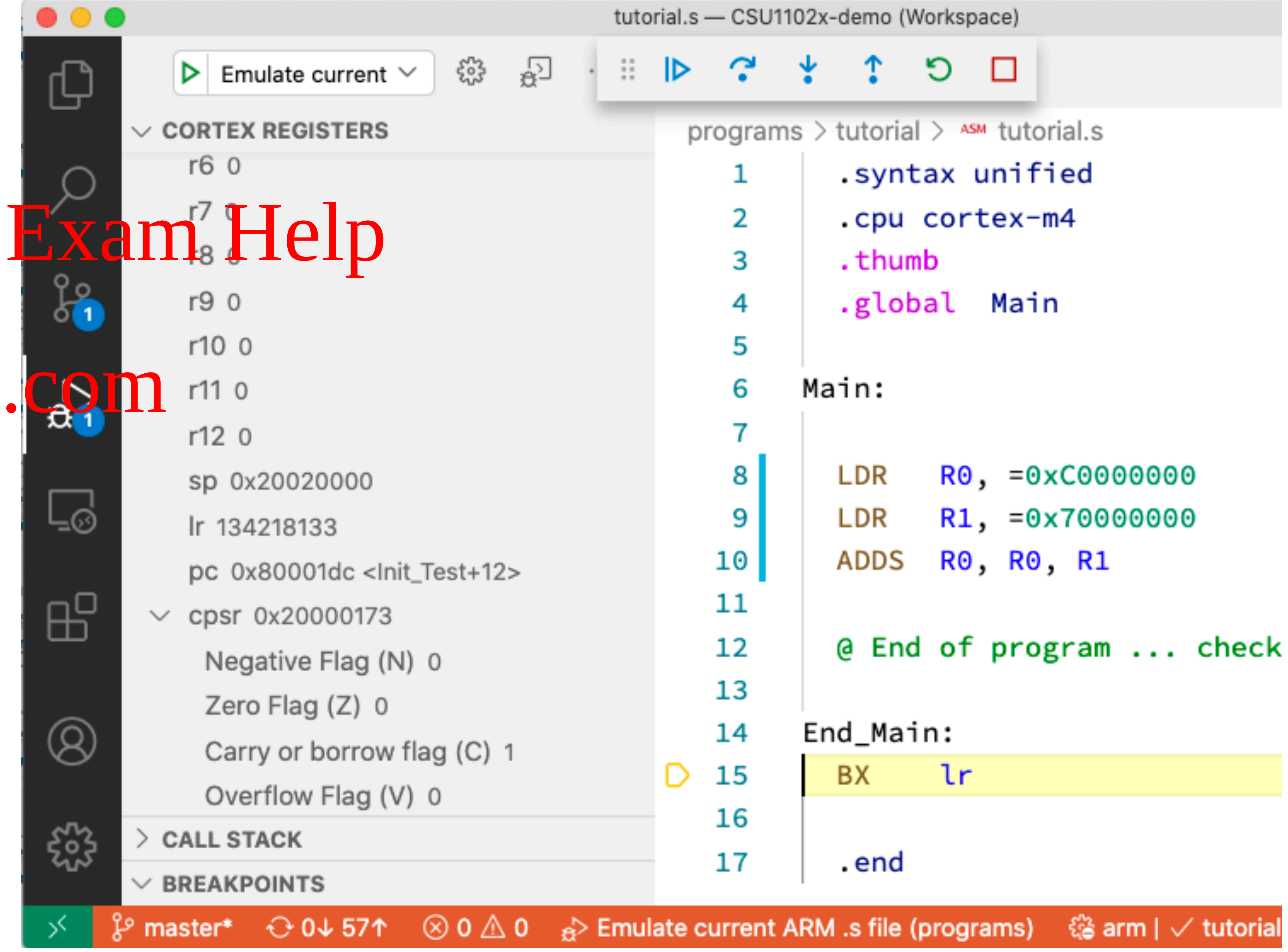
REMEMBER: 32-bit arithmetic!!

Expected result?

Does the result fit in 32-bits?

Will the carry flag be set?

Examine by running the program …



tutorial.s — CSU1102x-demo (Workspace)

Emulate current

CORTEX REGISTERS

```
r6 0
r7 0
r8 0
r9 0
r10 0
r11 0
r12 0
sp 0x20020000
lr 134218133
pc 0x80001dc <Init_Test+12>
cpsr 0x20000173
  Negative Flag (N) 0
  Zero Flag (Z) 0
  Carry or borrow flag (C) 1
  Overflow Flag (V) 0
```

CALL STACK

BREAKPOINTS

programs > tutorial > tutorial.s

```
    .syntax unified
    .cpu cortex-m4
    .thumb
    .global  Main

Main:

    LDR     R0, =0xC0000000
    LDR     R1, =0x70000000
    ADDS    R0, R0, R1

    @ End of program ... check

End_Main:
    BX      lr

    .end
```

master* 0↓ 57↑ ⊗ 0 ⚠ 0 Emulate current ARM .s file (programs) arm | ✓ tutorial

CMP Instruction and the Condition Code Flags

CMP (CoMPare) instruction performs a subtraction **without storing the result of the subtraction**

Processor remembers the properties of the CMP result by **updating the Condition Code Flags**

Allows us to determine equality (=) or inequality (< ≤ ≥ >)

Don’t care about absolute value of result
(i.e. don’t care ***by how much*** *x* is greater than *y*, only whether it is or not.)

CMP always sets the Condition Code Flags (so no need for **CMPS**)

```
        CMP     R2, #0   @ subtract 0 from R2, ignoring result but
                         @ updating the CC flags
        BEQ     EndWh    @ if the result was zero then branch to EndWh
        ...     ...      @ otherwise (if result was not zero) then keep
                         @ going (with sequential instruction path)
EndWh:
```

BEQ – Branch if EQual

(or more precisely branch if the Zero flag is set)

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

5.2 Negative numbers and 2s complement

CSU11021 – Introduction to Computing I

Dr Jonathan Dukes | jdukes@tcd.ie
School of Computer Science and Statistics

Negative Numbers

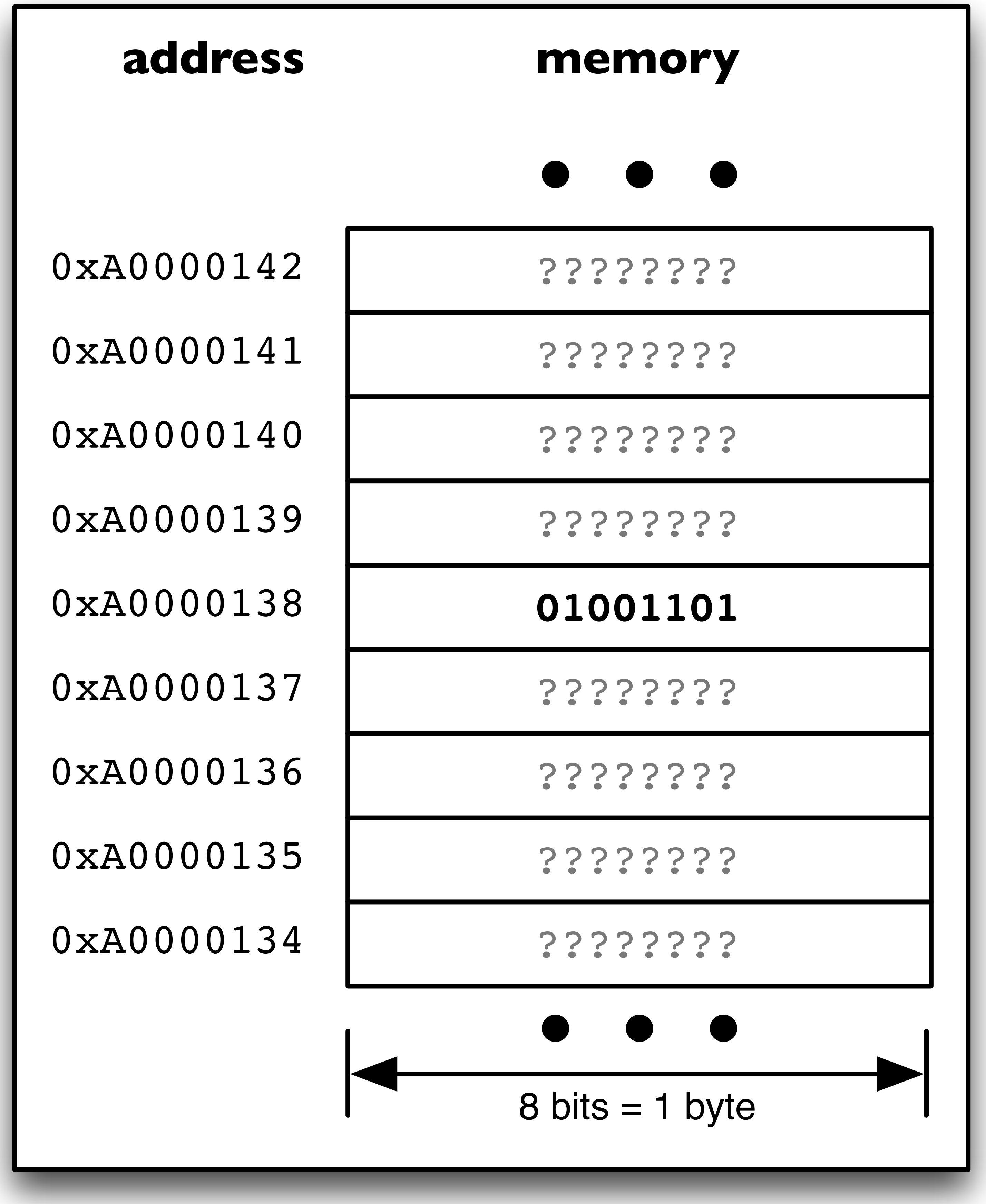
What does the binary value stored in memory at address 0xA0000138 represent?

Interpretation!

How can we represent signed values, and negative values such as -17_{10} in particular, in memory?

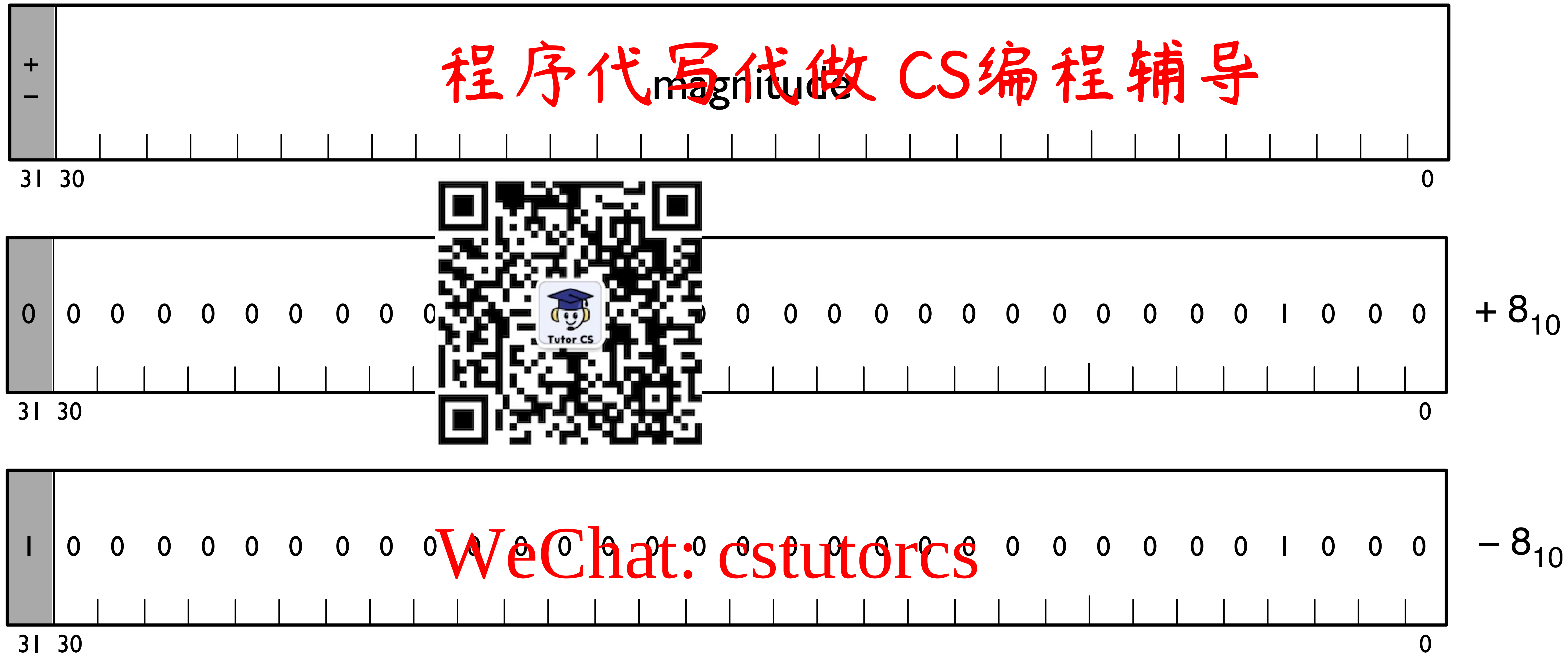
How can we tell whether any given value in memory represents an unsigned value, a signed value, an ASCII character or something else?

(we can't **tell** … as programmers we have to **know**)



address	memory
	• • •
0xA0000142	????????
0xA0000141	????????
0xA0000140	????????
0xA0000139	????????
0xA0000138	01001101
0xA0000137	????????
0xA0000136	????????
0xA0000135	????????
0xA0000134	????????
	• • •

Sign-Magnitude *(this is **not** how we usually represent negative numbers!!)*


+
−
magnitude
31 30 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 I 0 0 0
+ 8$_{10}$
31 30 0
I 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 I 0 0 0
− 8$_{10}$
31 30 0

Represent signed values in the range [$(-2^{31}-1) \dots (+2^{31}-1)$]

Two representations of zero (+0 and -0)

Would need special way to handle signed arithmetic (i.e. a separate circuit)

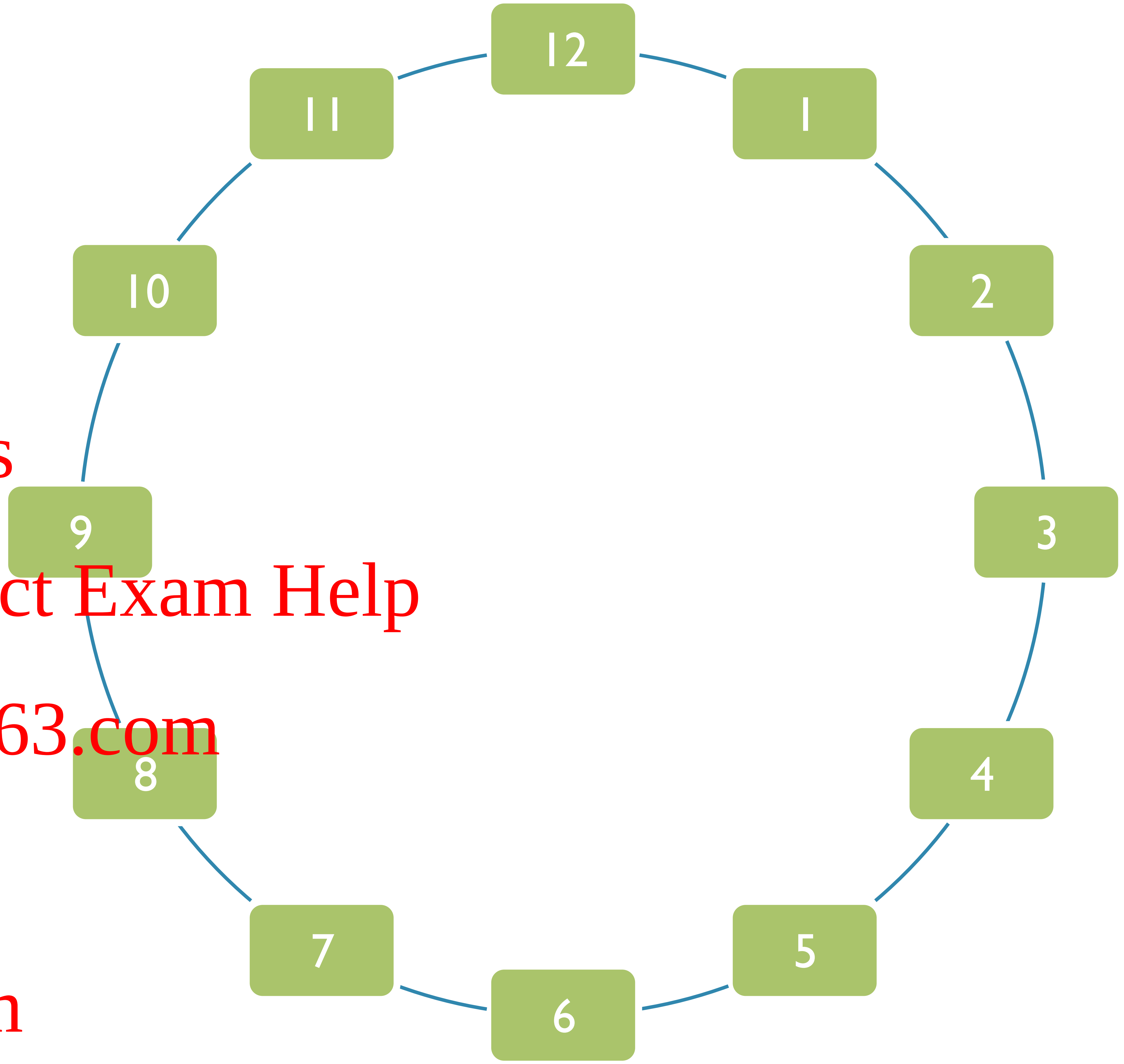
Remember: **interpretation!** (is it -8 or 2,147,483,656?)

Modulo Arithmetic

A 12-hour clock is an exa modulo-12 arithmetic

If we add 4 hours to 10 o’clock we get 2 o’clock

If we subtract 4 from 2 o’clock we get 10 o’clock (not -2 o’clock!)



2's Complement

Can represent 16 values with a 4-bit number system ($2^4 = 16$)

Ignoring carries from 4-bit addition gives us modulo-16 arithmetic (handy)

(15 + 1) mod 16 = 0

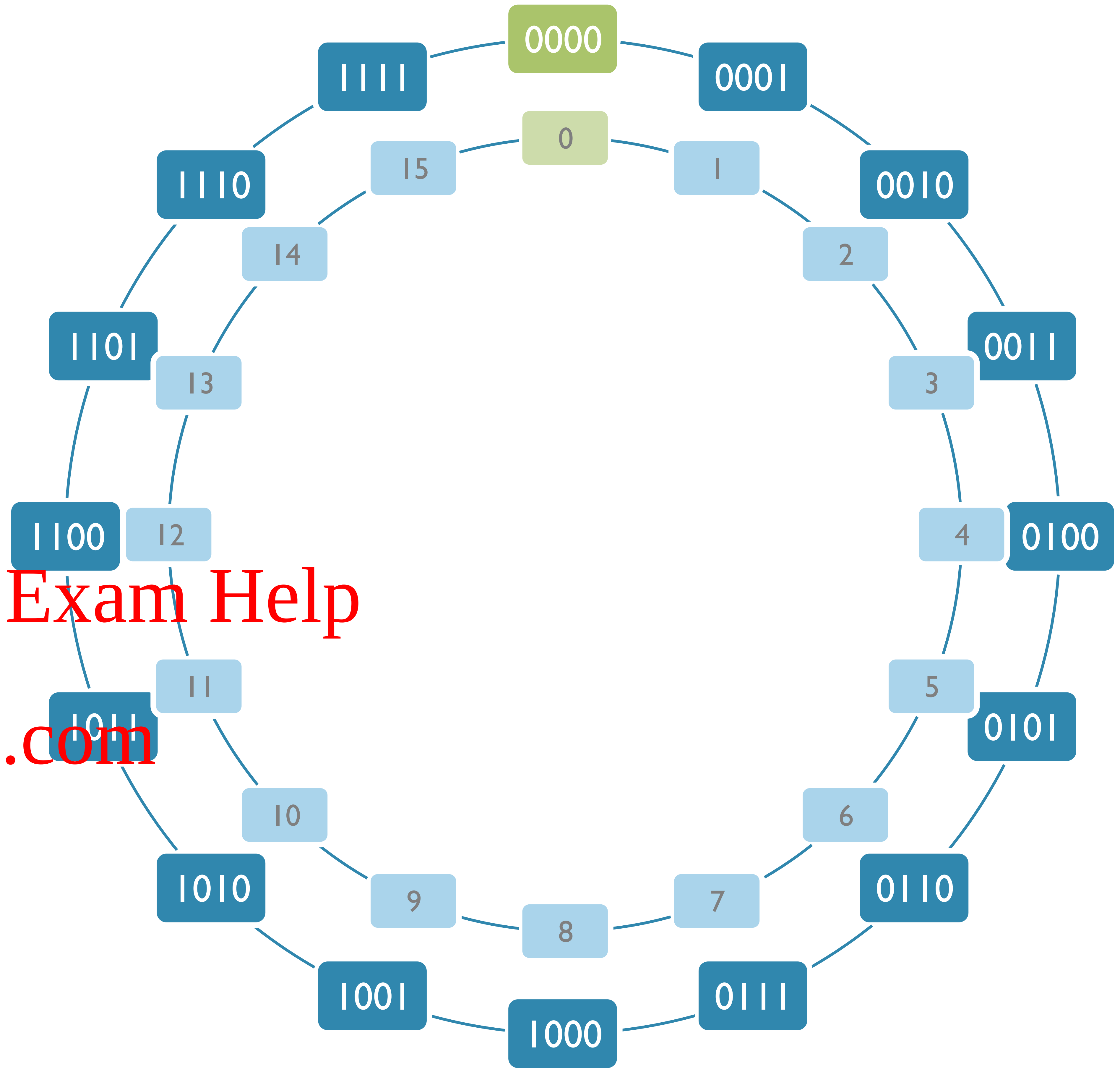
and -1 + 1 = 0

(14 + 2) mod 16 = 0

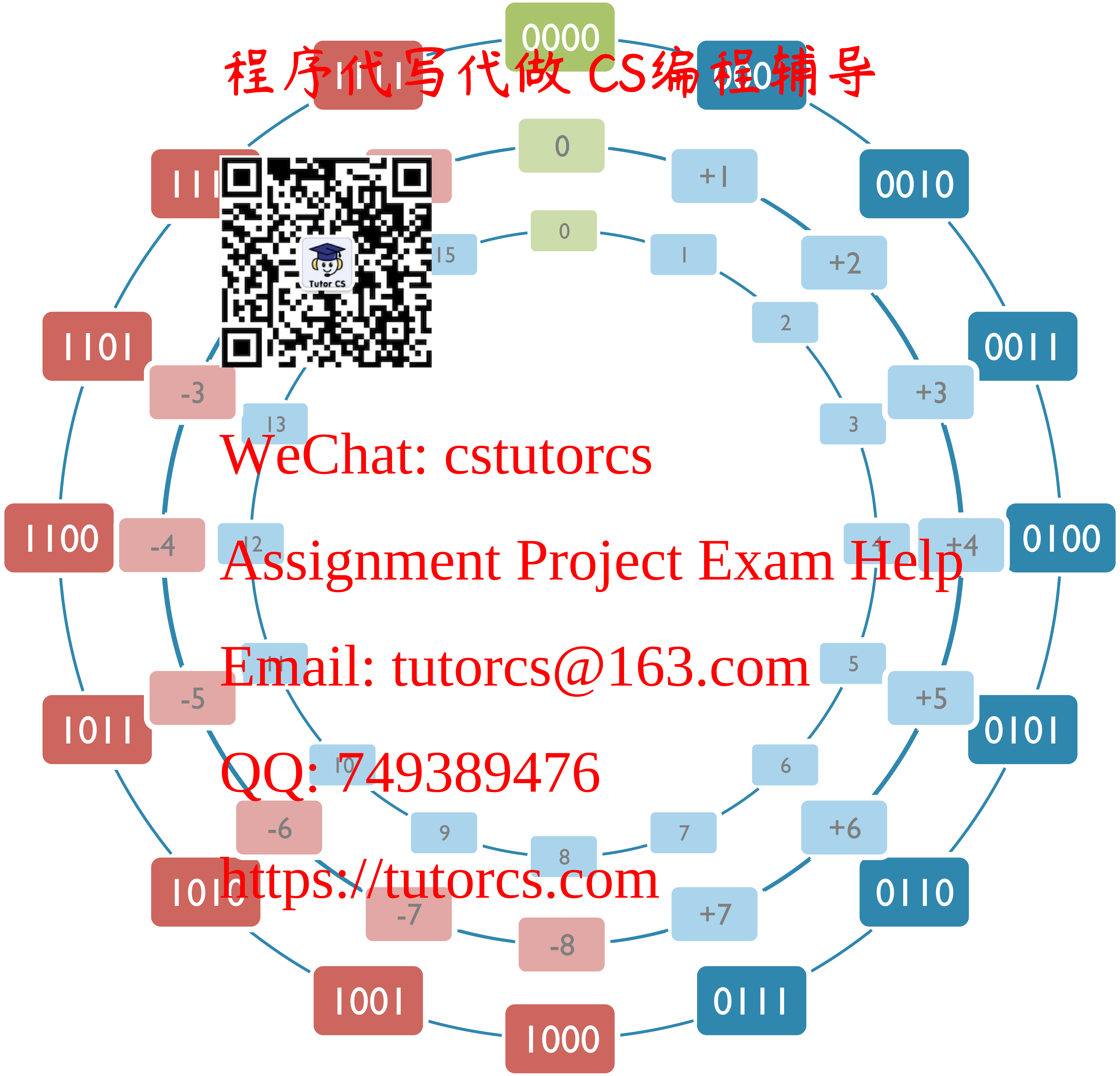
and -2 + 2 = 0

(14 + 4) mod 16 = 2

and -2 +4 = 2



2’s Complement



2's Complement Examples

Again, 8-bit values for illustration only here! In practice, we'll be working with 32-bit values

Represent -97_{10} using 2's complement

$97_{10} = 01100001_2$

Inverting gives 10011110_2

Adding 1 gives 10011111_2

Interpreted as a 2's complement signed integer

$10011111_2 = -97_{10}$

Interpreted as an unsigned integer

$1001\ 1111_2 = 159_{10}$

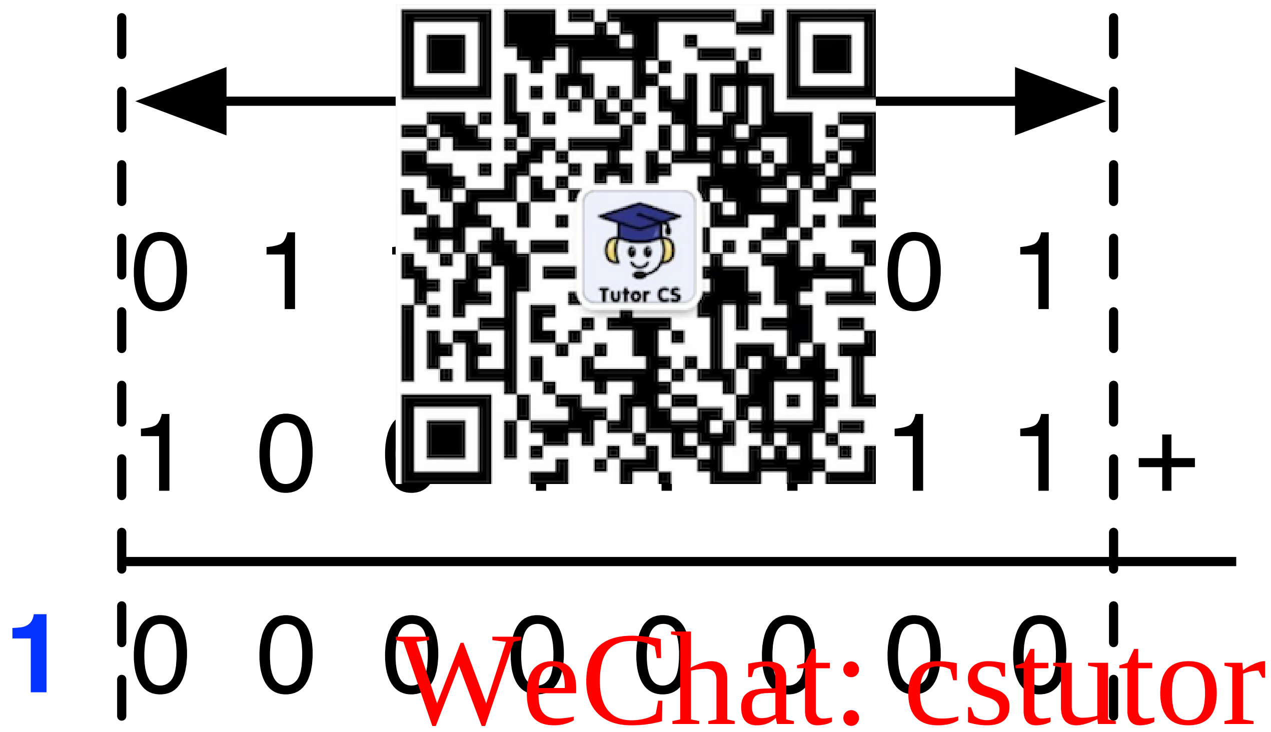
(159 + 97) mod 256 = 0

Correct interpretation is the responsibility of the programmer, not the CPU

CPU does not "know" whether a value 10011111_2 in R0 is -97_{10} or 159_{10}

2’s Complement Examples

Adding 01100001_2 ($+97_{10}$) and 10011111_2 (-97_{10})



Decimal equivalent

+97

–97 +

0

Ignoring the carry bit gives us the correct result of 0

Changing sign of $1001\ 1111_2$ (-97_{10})

- Invert bits and add 1 again
- Inverting gives 01100000_2
- Adding 1 gives 01100001_2 ($+97_{10}$)

Changing Sign in Assembly Language

Write an Assembly Language program to change the sign of the value stored in R0

Sign of a 2’s Complement value can be changed by inverting the value (bits) and adding 1

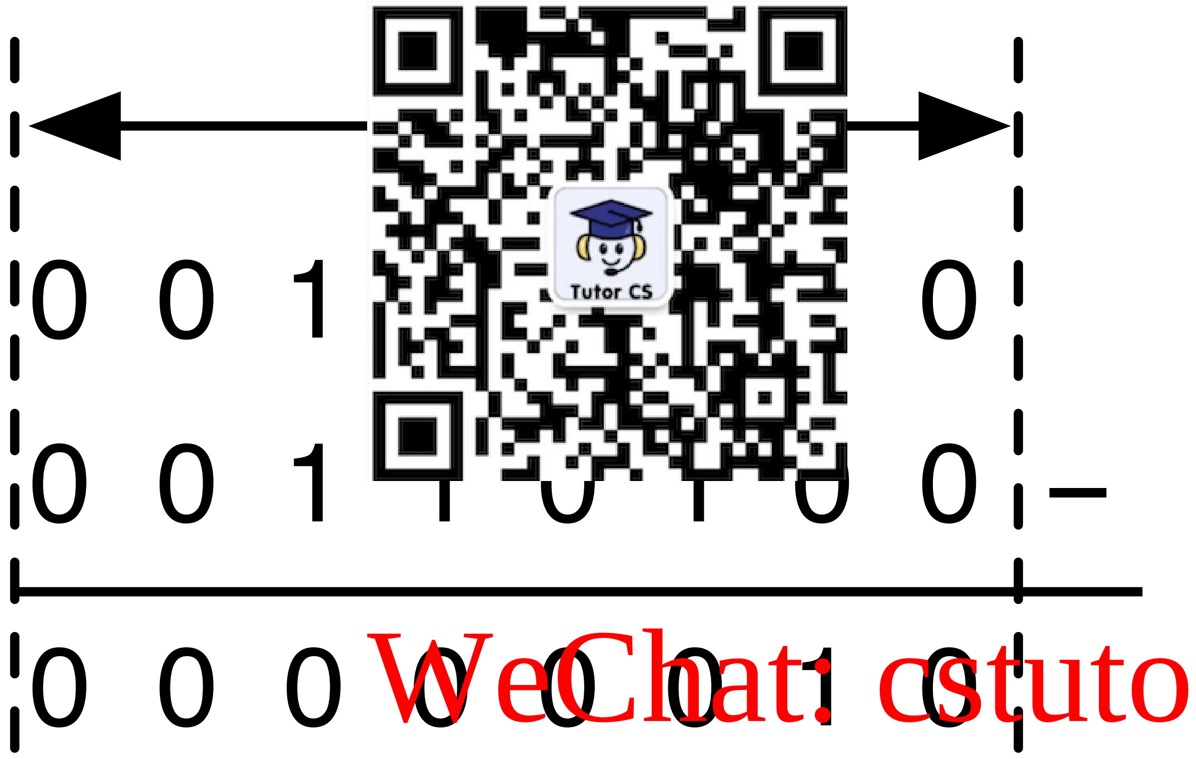
```
LDR     r0, =7          ; value = 7 (simple test value)
MVN     r0, r0          ; value = NOT value (invert bits)
ADD     r0, r0, #1      ; value = value + 1 (add 1)
```

ARM Instruction Set provides a single instruction for this purpose

```
LDR     r0, =7          ; value = 7 (simple test value)
NEG     r0, r0          ; value = -value
```

Subtraction

A – B



0 0 1 [illegible] 0

0 0 1 1 0 1 0 0 –

0 0 0 0 0 0 1 0

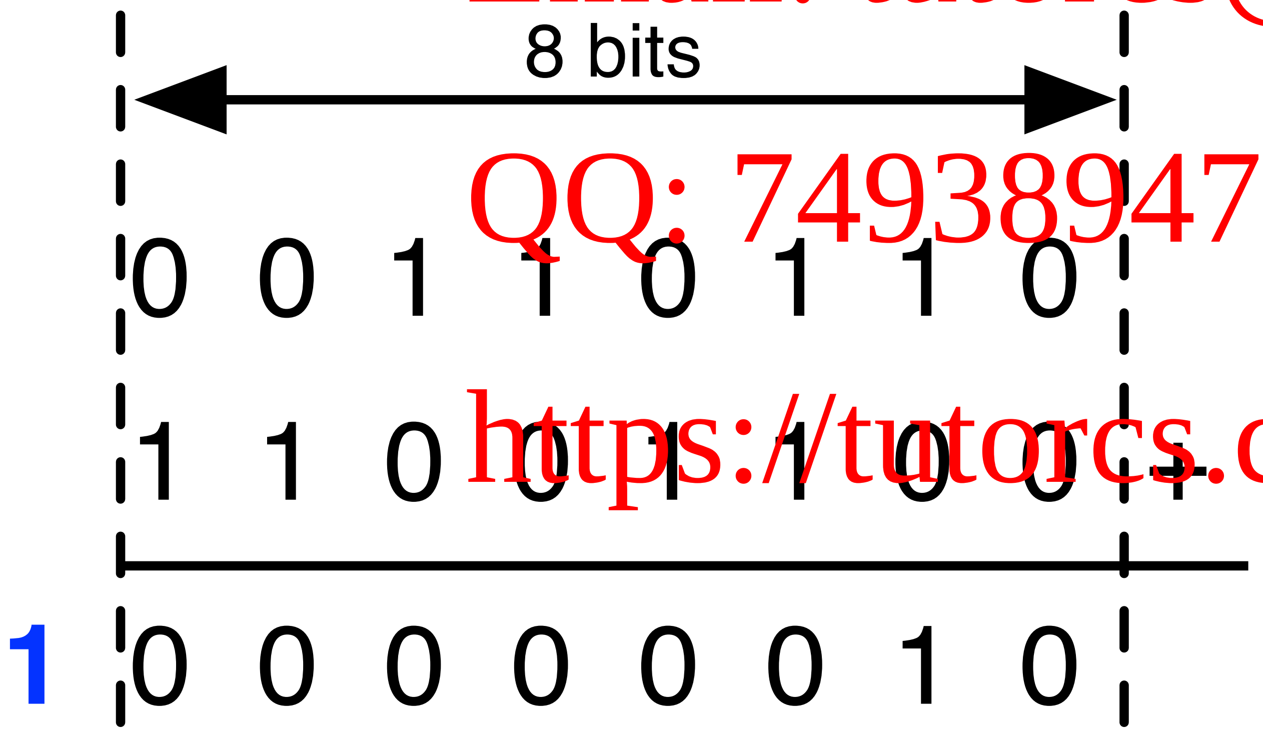
Decimal equivalent

+54

+52 –

+2

A + TwosComplement(B)



8 bits

0 0 1 1 0 1 1 0

1 1 0 0 1 1 0 0 +

1 0 0 0 0 0 0 1 0

Decimal equivalent

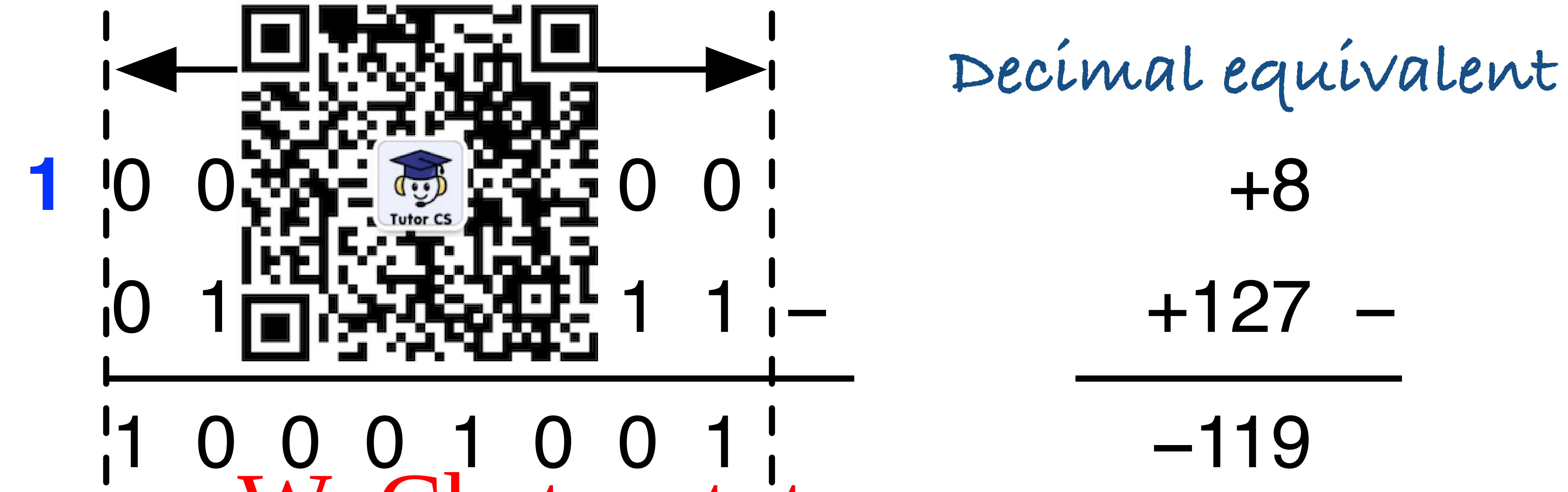
+54

–52 +

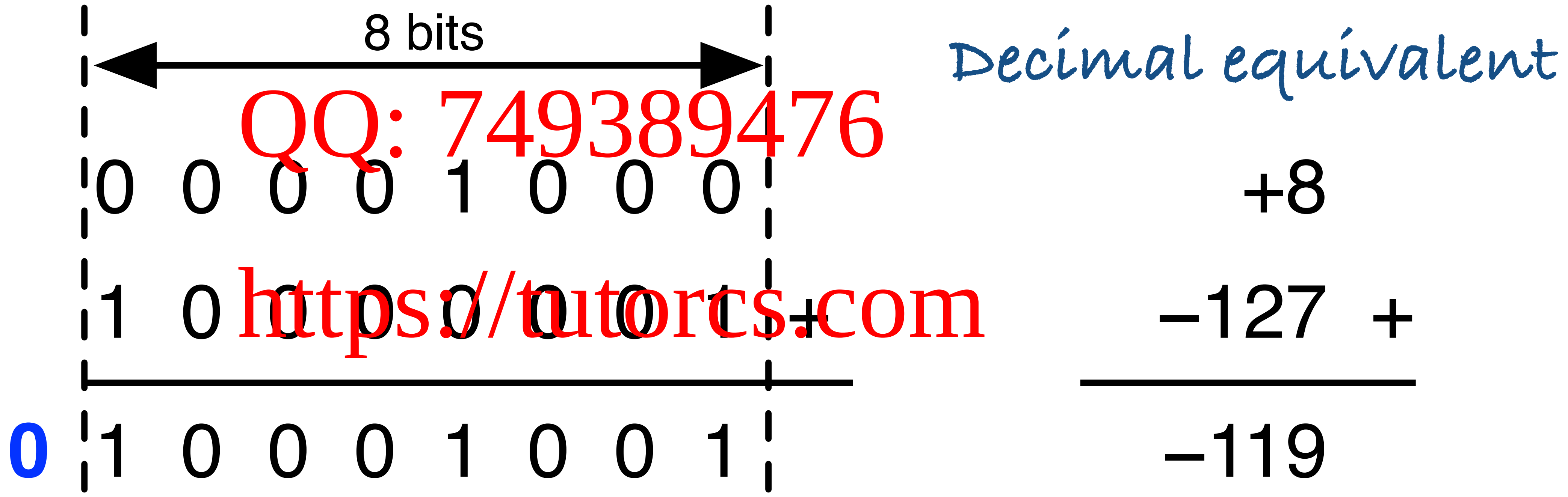
+2

Subtraction

A – B



A + TwosComplement(B)



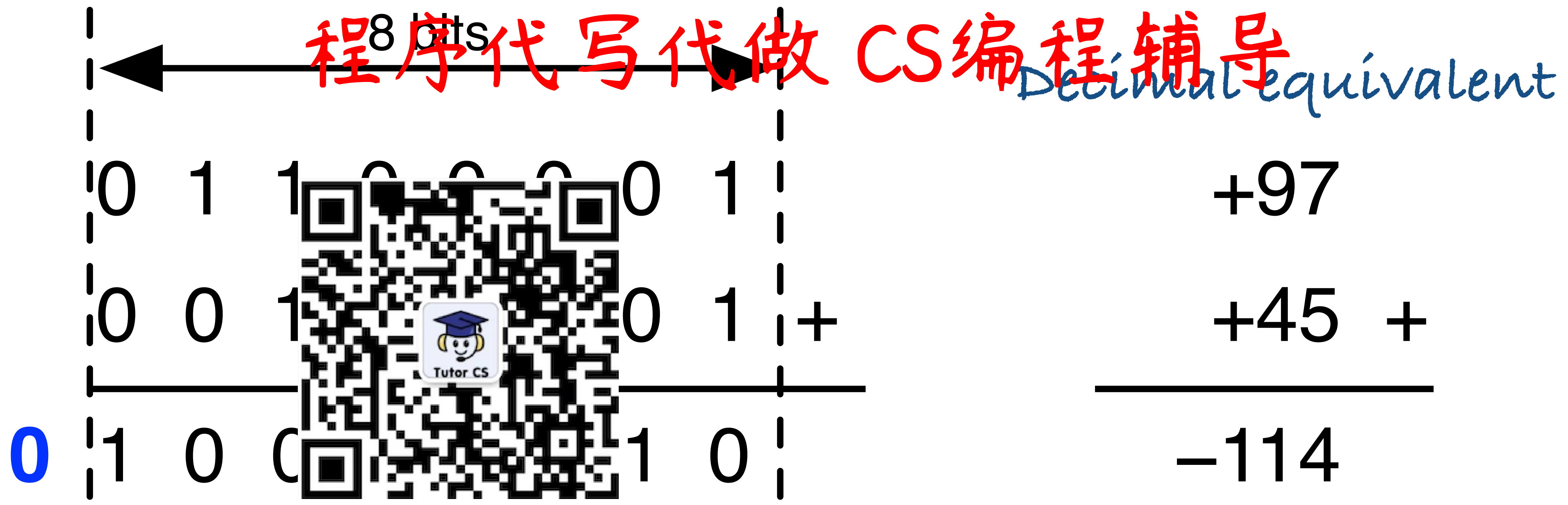
Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

5.3 oVerflow

CSU11021 – Introduction to Computing I

Dr Jonathan Dukes | jdukes@tcd.ie
School of Computer Science and Statistics

2’s Complement Example



Result is 10001110_2 (142_{10}, or -114_{10})

If we were interpreting the two added values and the result as **signed integers**, we got an incorrect result:

We added two +ve numbers and obtained a -ve result

With 8-bits, the highest +ve integer we can represent is +127

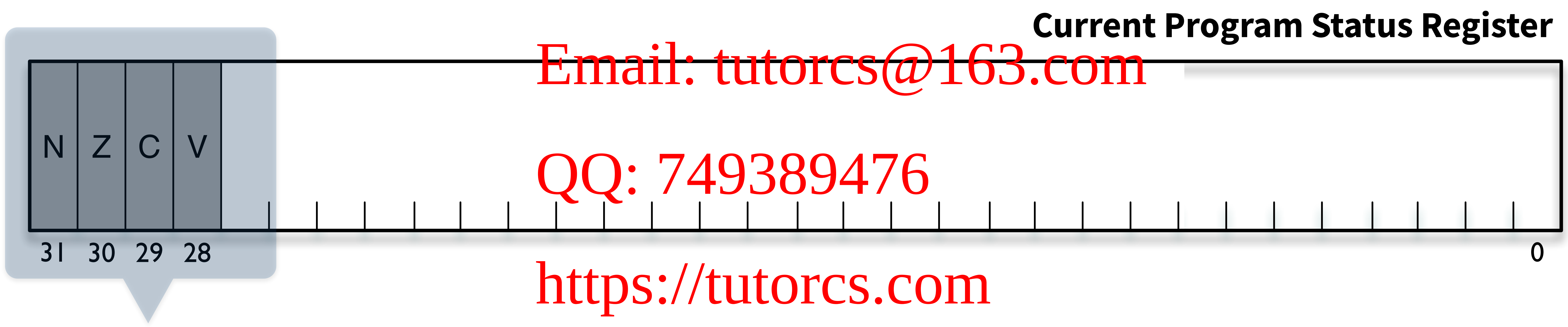
10001110_2 (-114_{10})

The result is outside the range of the signed number system

oVerflow

If the result of an addition or subtraction gives a result that is outside the range of the signed number system, then an oVerflow has occurred

The processor sets the oVerflow Condition Code Flag after performing an arithmetic operation to indicate whether an overflow has occurred



Condition Code Flags

Carry and oVerflow – Interpretation!

Carry and oVerflow flags always set by the processor regardless of our signed or unsigned inter of stored values

Processor does not "kno our interpretation is

- e.g. we could interpret the binary value 10001110_2 as either 142_{10} (unsigned) or -114_{10} (signed)
- (we could also interpret it as the code for "A" or as the colour blue)

The C and V flags are set by the processor and it is our responsibility to choose:

- whether to interpret C or V (are we interpreting the values as unsigned or signed?)
- how to interpret C or V

oVerflow Rules

Addition rule (r = a + b)

V = 1 B(a) = MSB(b) and

B(r) ≠ MSB(a)

i.e. oVerflow accurs for addition if the operands have the same sign and the result has a different sign

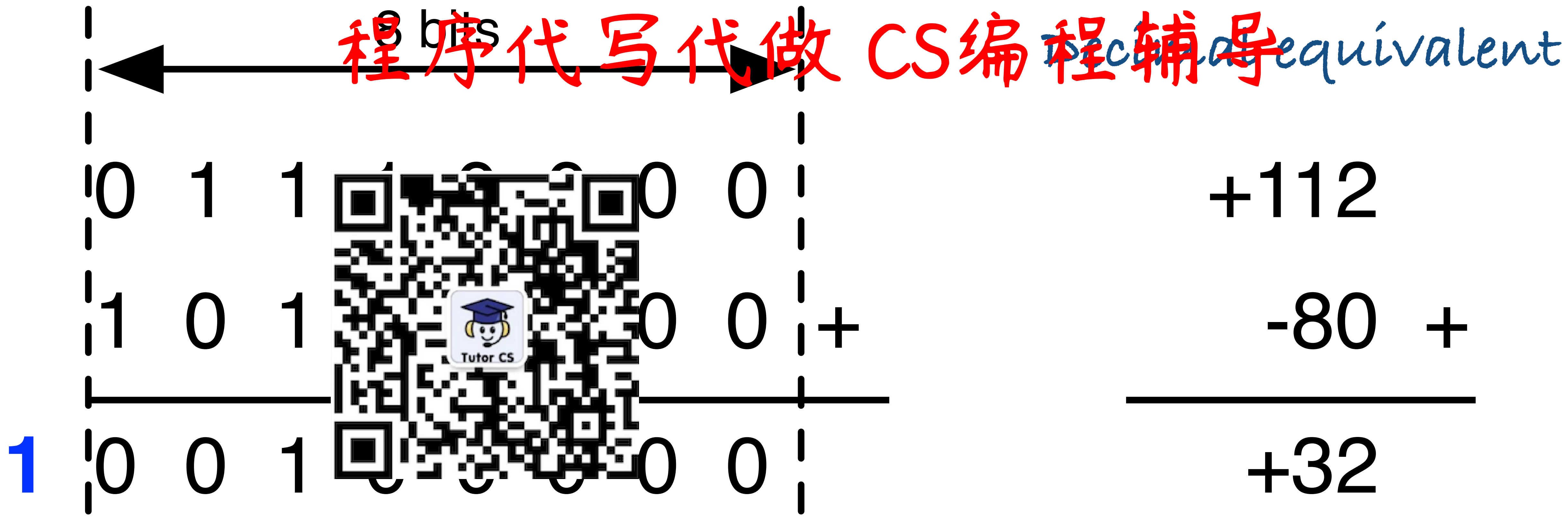
Subtraction rule (r = a – b)

V = 1 if MSB(a) ≠ MSB(b) and

MSB(r) ≠ MSB(a)

i.e. oVerflow occurs for subtraction if the operands have different signs and the sign of the result is different from the sign of the first operand

Carry and oVerflow Example



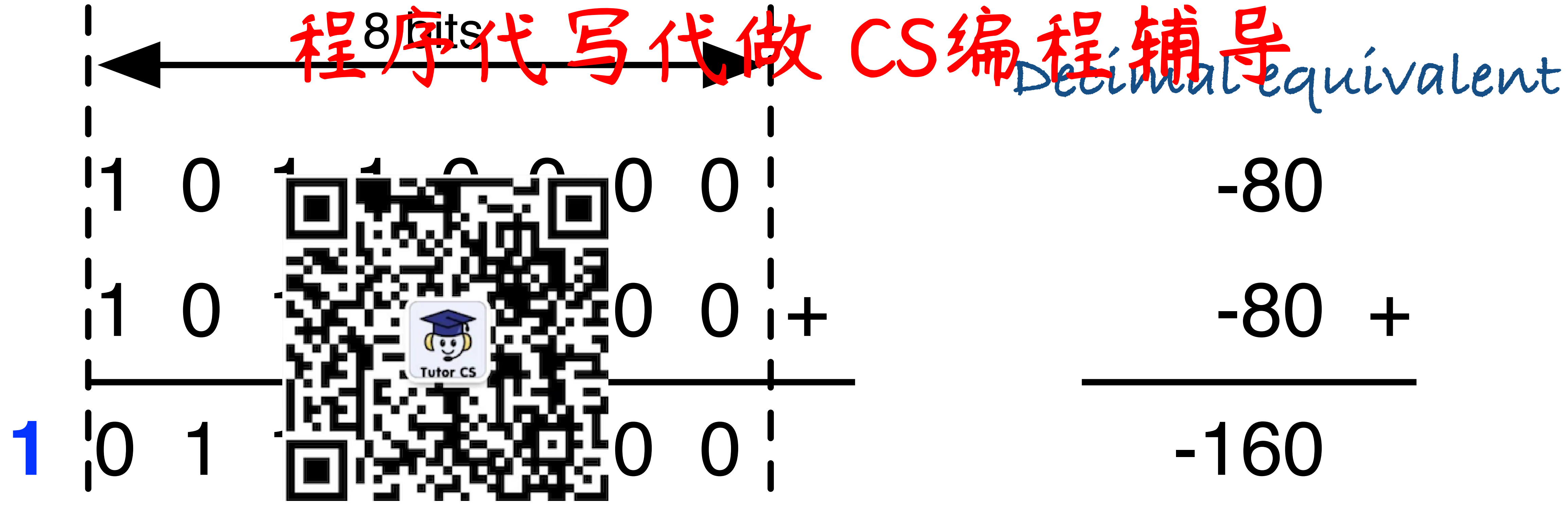
Signed interpretation: (+112) + (-80) = +32

Unsigned interpretation: 112 + 176 = 288

By examining the V flag (V = 0), we know that if were interpreting the values as signed integers, the result is correct

If we were interpreting the values as 8-bit unsigned values, C = 1 tells us that the result was too large to fit in 8-bits

Carry and oVerflow Example



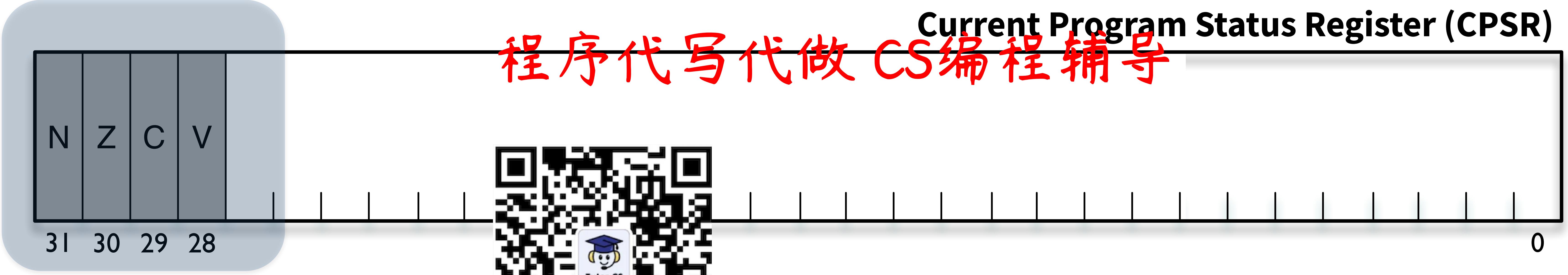
Signed: (-80) + (-80) = -160

Unsigned: 176 + 176 = 352

By examining the V flag (V = 1), we know that if were interpreting the values as signed integers, the result is outside the range of the signed number system

If we were interpreting the values as 8-bit unsigned values, C = 1 tells us that the result was too large to fit in 8-bits

Condition Code Flags – Recap


Current Program Status Register (CPSR)

Many instructions can optionally cause the processor to update the Condition Code Flags (N, Z, V, and C) to reflect certain properties of the result of an operation

Append "S" to instruction in assembly language (e.g. ADDS)

Set S-bit in machine code instruction

<u>Remember</u>: Processor updates NZVC regardless of our interpretation of values as signed or unsigned

N flag set to 1 if result is **<u>N</u>**egative (i.e. if MSB is 1)

Z flag is set to 1 if result is **<u>Z</u>**ero (i.e. all bits are 0)

C flag set if **<u>C</u>**arry occurs (addition) or borrow does not occur (subtraction)

V flag set if o**<u>V</u>**erflow occurs for addition or subtraction

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

5.4 Condition Code Flag examples

CSU11021 – Introduction to Computing I

Dr Jonathan Dukes | jdukes@tcd.ie
School of Computer Science and Statistics

Example 1

```
LDR     R0, =0xC0000000
LDR     R1, =0x700000
ADDS    R0, R0, R1
```

Is the Carry flag set?

Is the oVerflow flag set?

Is the Zero flag set?

Is the Negative flag set?

Example 2

```
LDR     R0, =0xC0000000
LDR     R1, =0x400000
ADDS    R0, R0, R1
```

Is the Carry flag set?

Is the oVerflow flag set?

Is the Zero flag set?

Is the Negative flag set?

Example 3

```
LDR     R0, =0xC0000000
LDR     R1, =0x900000
ADDS    R0, R0, R1
```

Is the Carry flag set?

Is the oVerflow flag set?

Is the Negative flag set?

Example 4

```
LDR      R0, =0xC0000000
LDR      R1, =0x3000000
ADD      R0, R0, R1
```

Is the Carry flag set?