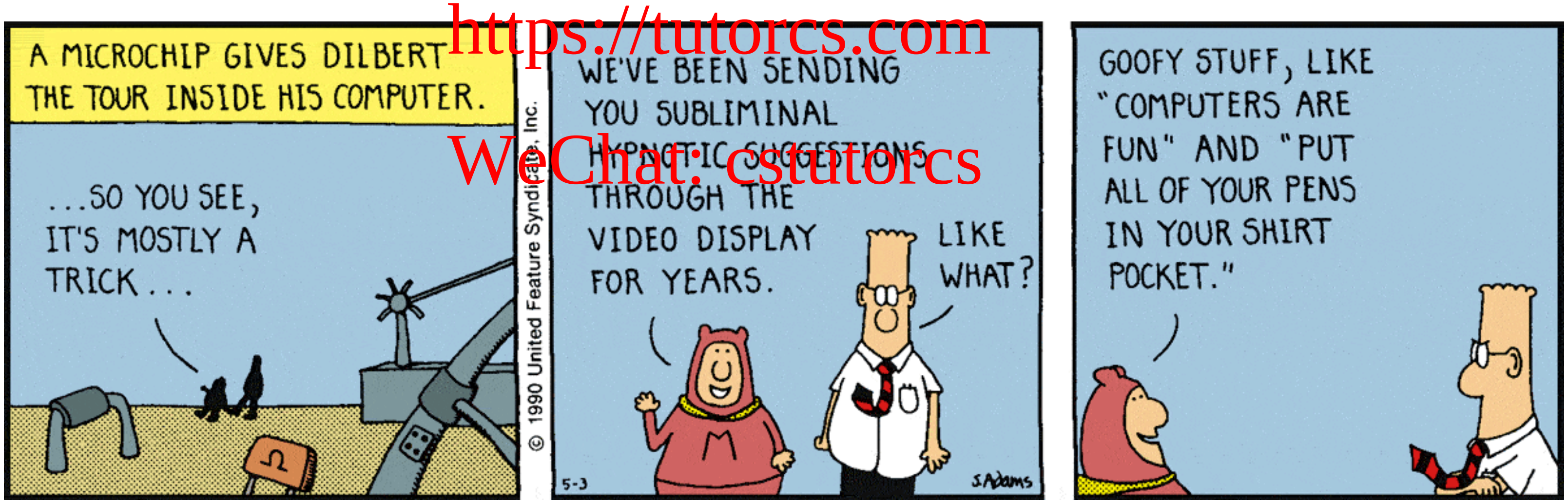
Lecture 3

Signed and Unsigned Numbers



Office Hours

Tentative time and space

- Tuesdays 1 pm – 2 pm   Dreese 660
- Tuesdays 2 pm – 3 pm   Dreese 331
- Thursdays 1 pm – 2 pm   Dreese 331

Will post Quiz #1 on Carmen today (or tomorrow)

n-bit Unsigned Numbers

Unsigned number = positive number

8-bits unsigned numbers range from 0 to 255

n-bit unsigned numbers range from 0 to $2^n - 1$

Binary	Decimal
0000 0000	0
0000 0001	1
0000 0010	2
...	
...	
1111 1101	253
1111 1110	254
1111 1111	255

Signed Numbers – First Attempt

How do we represent negative integers in binary?

In decimal we represent negative numbers by prefixing them with a "–" sign

$(-0110)_2 = (-6)_{10}$ does **not** work on computers!!

There is no (–) sign

But we could still use **sign bit** and **magnitude**

sign bit	magnitude							
0	0	0	0	0	1	1	0	$(6)_{10}$
1	0	0	0	0	1	1	0	$(-6)_{10}$

0 for positive
1 for negative

Difficult to add two numbers

- Check signs: if both signs are the same, add both numbers …
- … if not compare magnitudes: subtract smaller number from larger one …
- … decide on the sign of the result

Yikes! We need to do better!

Signed Numbers & Complements

The sign and magnitude method does not work well on computers

– at least not for integers or fixed point arithmetic

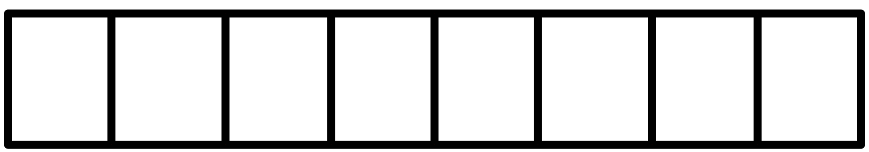
Modern computers use **2's complement** for signed numbers

Both 1's and 2's complement work only in context of a **fixed word length**

Two ingredients for complements:

1. **n** = word length in bits

= size of the register



n = 8 bits

2. **N** = Binary number we want to complement

n-bit Ones' Complement

n-bit 1's complement of a binary number is obtained by flipping its bits

Given binary number **N** and register size **n**

- fill the register – i.e., zero pad the number as needed to have n bits
- flip all bits – i.e., swap a 0 with a 1 and vice versa

e.g. N = 101101 n = 8 bits $\Rightarrow$ **8-bit ones' complement**

N = 101101	0	0	1	0	1	1	0	1
8-bit ones' complement	1	1	0	1	0	0	1	0

Same idea for n = 16 bits – only more bits to fill and toggle

Ones' Complement

Why is it called **ones' complement**?

N = 101101		0	0	1	0	1	1	0	1
8-bit ones' complement of N	+	1	1	0	1	0	0	1	0
all ones $2^8 - 1$		1	1	1	1	1	1	1	1

n-bit ones' complement of N $= 2^n - 1 - N$

Ones' Complement

What purpose does the one's complement serve?

⇒ Not much – at least in today's computers

However, some earlier computers used 1's complement for signed numbers

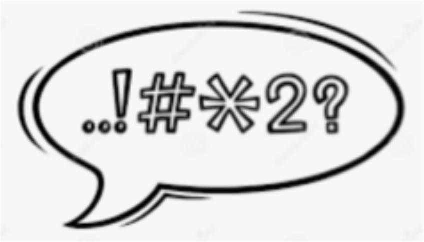
i.e., to express – 41 use 1's complement of 41, N = 101001

Does it work? Yes, it does. But there are some issues

e.g., normally 41 + (– 41) = 0

with ones' complement method

00101001 + 11010110 = 11111111



there are two representations of zero with ones' complement

00000000 and 11111111

Two's Complement

The **2's complement** of a binary number is obtained by adding 1 to its ones' complement

Given binary number **N** and register size **n**

- fill the register – i.e., zero pad the number as needed to have n bits
- flip all bits – i.e., 0 ↔ 1
- **add 1**

N = 101101	0	0	1	0	1	1	0	1	
8-bit ones' complement	1	1	0	1	0	0	1	0	+ 1
=	1	1	0	1	0	0	1	1	⇒ **8-bit two's complement**

2’s Complement – The Shortcut

There is a shortcut to write the **2’s complement** of a binary number

Given binary number **N** and register size **n**

- fill the register – i.e., zero pad the number as needed to have n bits
- **leave the least significant zeros and first 1 unchanged**
- **flip all remaining bits**

N = 110100

0	0	1	1	0	1	0	0

1	1	0	0	1	1	0	0
flip					leave unchanged		

$\Rightarrow$ **8-bit two’s complement**

Two’s Complement

Why is it called **two’s complement**? **Power of two’s complement**

N = 101101

0	0	1	0	1	1	0	1

8-bit two’s complement **+**

1	1	0	1	0	0	1	1

Power of 2 1

0	0	0	0	0	0	0	0

$= 2^8$

n-bit two’s complement of N $= 2^n - N$ if $N \neq 0$

Two’s Complement

A better definition of **two’s complement**

$$\text{n-bit two's complement of N} = \begin{cases} 2^n - N & \text{if } N \neq 0 \\ 0 & \text{if } N = 0 \end{cases}$$

Compare to ones’ complement

$$\text{n-bit ones' complement of N} = 2^n - 1 - N$$

We see:

2’s complement = 1’s complement + 1

Works for zero when restricted to n-bits

Signed Numbers w/ 2's Complement

Use **two's complement representation** for signed numbers

modern computers – including our MCU – use this method

- If a number N is positive, use binary representation of N
- If N is negative, use two's complement of absolute value of N

e.g. +/- 43

43		0	0	1	0	1	0	1	1
- 43	+	1	1	0	1	0	1	0	1
	1	0	0	0	0	0	0	0	0

2's complement of |-43|

Does this work?

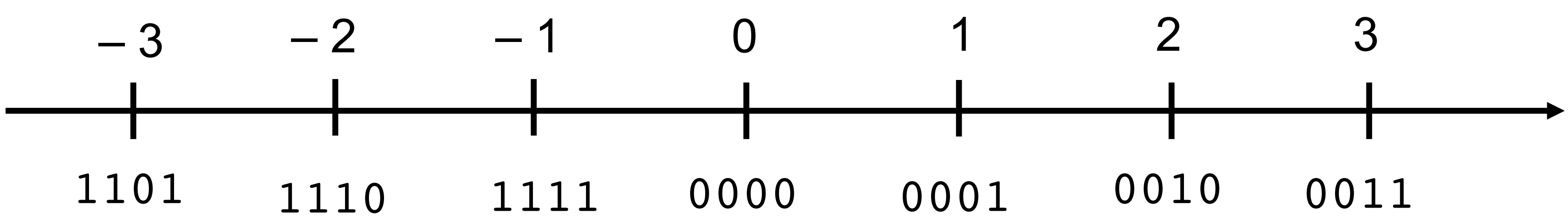
Is this consistent with the rules of arithmetic?

- $N + (-N) = 0$ Previous slide
- $-(-N) = N$ We get the original N when we complement twice
- Successors and predecessor relationships are consistent with incrementing and decrementing

$(1)_{10}$ = 0001
$(-1)_{10}$ = 1111

$(2)_{10}$ = 0010
$(-2)_{10}$ = 1110

$(3)_{10}$ = 0011
$(-3)_{10}$ = 1101



Signed Numbers

8-bits can represent 256 distinct values

8-bit unsigned numbers range from -128 to 127

n-bit signed numbers

-2^{n-1} to $2^{n-1}-1$

not sign-magnitude

Binary	Decimal
1000 0000	-128
1000 0001	-127
...	
1111 1110	-2
1111 1111	-1
0000 0000	0
0000 0001	1
...	
0111 1110	126
0111 1111	127

negative numbers start with a "1"

2's complement of the absolute value

positive numbers start with a "0"

binary representation of the number

Signed Numbers

Given **2's complement signed numbers** find the decimal values

- `0110 1001` Positive Number 105
- `1101 0001` Negative Number $(11010001)_2$ = 209

  - 47 2's complement is 256 – 209 = 47

  2's complement of `11010001` is `00101111`

  $(00101111)_2$ = 47
- `0010 1010` Positive Number 42
- `1110 1110` Negative Number - 18

Addition of Signed/Unsigned Numbers

Computers add all numbers using the same hardware – they do not distinguish between signed or unsigned numbers

Unsigned Number Interpretation		Signed Number Interpretation
43	00101011	43
+ 18	+ 00010010	+ 18
61	00111101	61
overflow possible did not happen		overflow possible did not happen

Addition of Signed/Unsigned Numbers

Computers add all numbers using the same hardware – they do not distinguish between signed or unsigned numbers

Unsigned Number Interpretation		Signed Number Interpretation
213	11010101	-43
+ 18	+ 00010010	+ 18
231	11100111	-25

overflow possible did not happen

overflow **not** possible!

Addition of Signed/Unsigned Numbers

Computers add all numbers using the same hardware – they do not distinguish between signed or unsigned numbers

Unsigned Number Interpretation		Signed Number Interpretation
43	00101011	43
+ 238	+ 11101110	+ -18
281	1 $\boxed{00011001}$	25

overflow!

overflow **not** possible!

carry out of “sign bit”

Addition of Signed/Unsigned Numbers

Computers add all numbers using the same hardware – they do not distinguish between signed or unsigned numbers

Unsigned Number Interpretation

213
+ 238
451

overflow!

11010101
+ 11101110
1 $\boxed{11000011}$

Signed Number Interpretation

-43
+ -18
-61

overflow possible did not happen