C335
Computer Structures

Number Representation
Part #2

Dr. Liqiang Zhang

Department of Computer and Information Sciences

Adapted from Morgan Kaufmann and others

How to Represent Negative Numbers?

- So far, unsigned numbers
- Obvious solution: define leftmost bit to be sign!
  - $0 \Rightarrow +$, $1 \Rightarrow -$
  - Rest of bits can be numerical value of number
- Representation called sign and magnitude
- MIPS uses 32-bit integers. $+1_{ten}$ would be:

0000 0000 0000 0000 0000 0000 0000 0001

- And -1_{ten} in sign and magnitude would be:

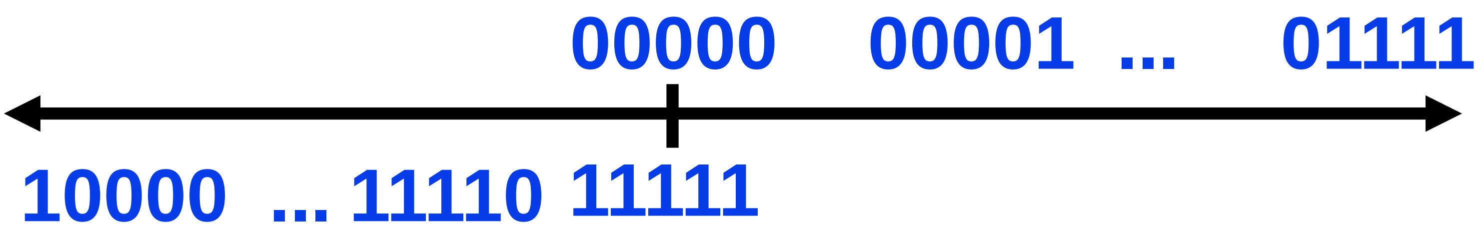
1000 0000 0000 0000 0000 0000 0000 0001

Shortcomings of sign and magnitude?

- Arithmetic circuit complicated
  - Special steps depending whether signs are the same or not
  - Extra step to set the sign
- Also, two zeros
  - 0x00000000 = $+0_{ten}$
  - 0x80000000 = -0_{ten}
  - What would two 0s mean for programming?
- Therefore sign and magnitude abandoned

Another try: complement the bits

- Example: $7_{10} = 00111_2$ $-7_{10} = 11000_2$
- Called One's Complement
- Note: positive numbers have leading 0s, negative numbers have leadings 1s.



- What is -00000 ?
  - Answer: 11111
- How many positive numbers in N bits?
- How many negative numbers?

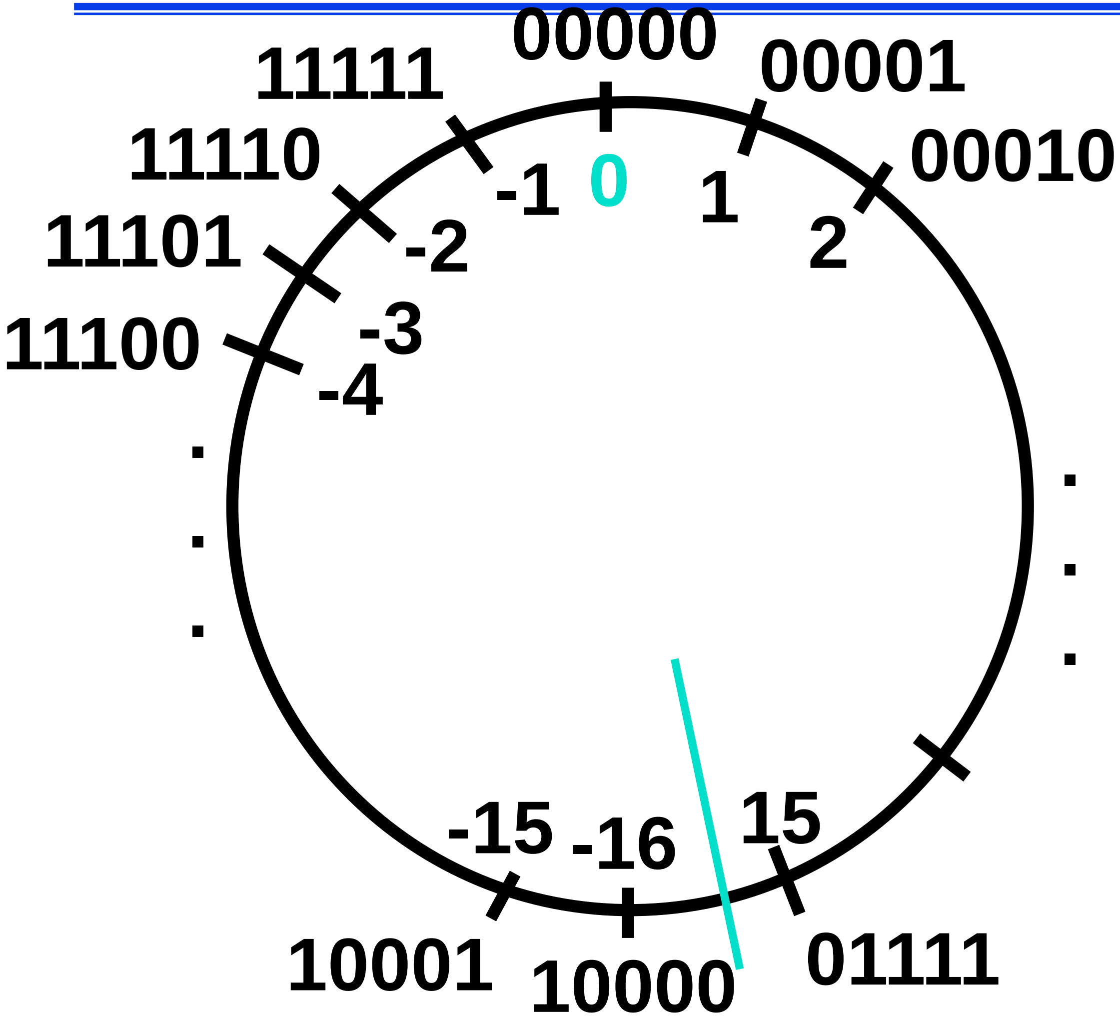
Shortcomings of One's complement?

- Arithmetic still somewhat complicated.
- Still two zeros
  - `0x00000000` = $+0_{ten}$
  - `0xFFFFFFFF` = -0_{ten}
- Although used for awhile on some computer products, one's complement was eventually abandoned because another solution was better.

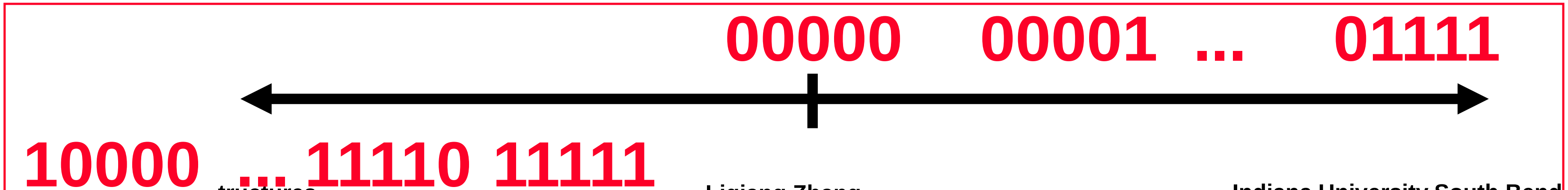
Standard Negative Number Representation

- What is result for unsigned numbers if trie to subtract large number from a small one?
  - Would try to borrow from string of leading 0s, so result would have a string of leading 1s
    - 3 - 5 ⇒ 00…0011 – 00…0101 = 11…1110
  - With no obvious better alternative, pick representation that made the hardware simple
  - As with 1's complement, leading 0s ⇒ positive, leading 1s ⇒ negative
    - 000000...xxx is ≥ 0, 111111...xxx is < 0
    - except 1…1111 is -1, not -0 (as in 1's complement.)
- This representation is <u>Two's Complement</u>

2's Complement Number "line": N = 5



- 2^{N-1} non-negatives
- 2^{N-1} negatives
- one zero
- how many positives?



Two's Complement for N=32

$0000 \ldots 0000\ 0000\ 0000\ 0000_{two} =$	0_{ten}
$0000 \ldots 0000\ 0000\ 0000\ 0001_{two} =$	1_{ten}
$0000 \ldots 0000\ 0000\ 0000\ 0010_{two} =$	2_{ten}
. . .	
$0111 \ldots 1111\ 1111\ 1111\ 1101_{two} =$	$2,147,483,645_{ten}$
$0111 \ldots 1111\ 1111\ 1111\ 1110_{two} =$	$2,147,483,646_{ten}$
$0111 \ldots 1111\ 1111\ 1111\ 1111_{two} =$	$2,147,483,647_{ten}$
$1000 \ldots 0000\ 0000\ 0000\ 0000_{two} =$	$-2,147,483,648_{ten}$
$1000 \ldots 0000\ 0000\ 0000\ 0001_{two} =$	$-2,147,483,647_{ten}$
$1000 \ldots 0000\ 0000\ 0000\ 0010_{two} =$	$-2,147,483,646_{ten}$
. . .	
$1111 \ldots 1111\ 1111\ 1111\ 1101_{two} =$	-3_{ten}
$1111 \ldots 1111\ 1111\ 1111\ 1110_{two} =$	-2_{ten}
$1111 \ldots 1111\ 1111\ 1111\ 1111_{two} =$	-1_{ten}

- One zero; 1st bit called sign bit
- 1 “extra” negative:no positive $2,147,483,648_{ten}$

Two's Complement Formula

- Can represent positive and negative numbers in terms of the bit value times a power of 2:

  $d_{31} \times -(2^{31}) + d_{30} \times 2^{30} + \ldots + d_2 \times 2^2 + d_1 \times 2^1 + d_0 \times 2^0$

- Example: 1101_{two}

  $= 1 \text{x} -(2^3) + 1 \text{x} 2^2 + 0 \text{x} 2^1 + 1 \text{x} 2^0$

  $= -2^3 + 2^2 + 0 + 2^0$

  $= -8 + 4 + 0 + 1$

  $= -8 + 5$

  $= -3_{ten}$

Two's Complement shortcut: Negation

- Change every 0 to 1 and 1 to 0 (invert or complement), then add 1 to the result

- Proof*: Sum of number and its (one's) complement must be $111...111_{two}$

  However, $111...111_{two} = -1_{ten}$

  Let x' $\Rightarrow$ one's complement representation of x

  Then $x + x' = -1 \Rightarrow x + x' + 1 = 0 \Rightarrow \boxed{-x = x' + 1}$

- Example: -3 to +3 to -3

- x : $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two}$

  x': $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two}$

  +1: $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011_{two}$

  ()': $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$

  +1: $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two}$

You should be able to do this in your head…

Exercise

- Convert 0b11100111 to decimal (assume 2’s complement is used)

Two's comp. shortcut: Sign extension

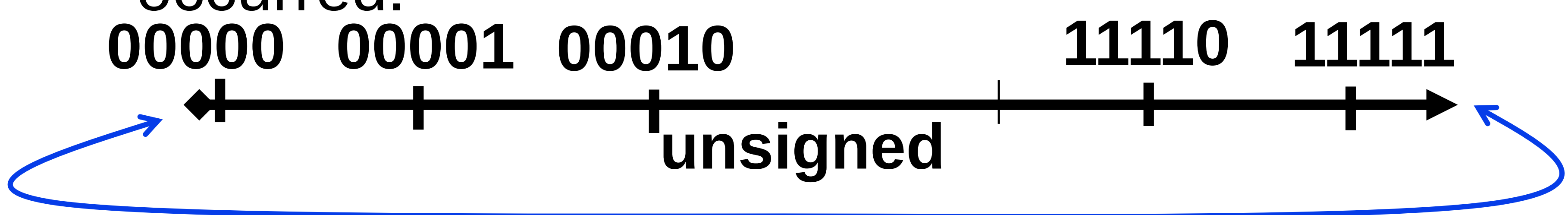
- Convert 2's complement number rep. using n bits to more than n bits
- Simply replicate the most significant bit (sign bit) of smaller to fill new bits
  - 2's comp. positive number has infinite 0s
  - 2's comp. negative number has infinite 1s
  - Binary representation hides leading bits; sign extension restores some of them
  - 16-bit -4_{ten} to 32-bit:

$$1111\ 1111\ 1111\ 1100_{two}$$

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{two}$$

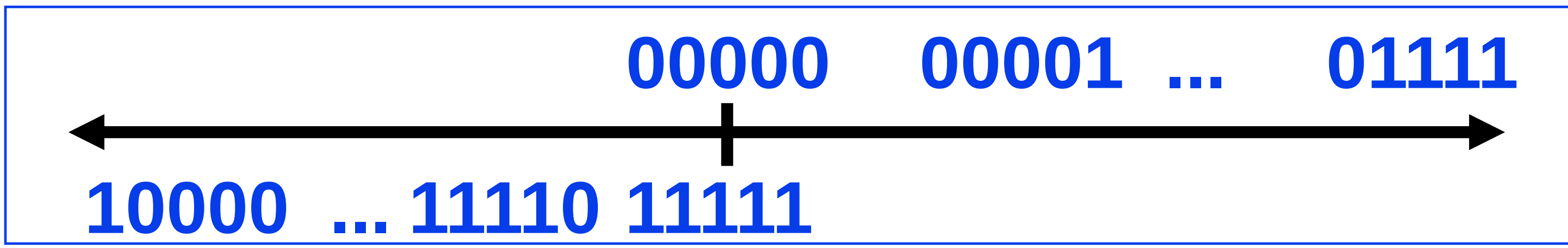
What if too big?

- Binary bit patterns above are simply <u>representatives</u> of numbers. Strictly speaking they are called “numerals”.
- Numbers really have an ∞ number of digits
  - with almost all being same (00…0 or 11…1) except for a few of the rightmost digits
  - Just don’t normally show leading digits
- If result of add (or -, *, /) cannot be represented by these rightmost bits, <u>overflow</u> is said to have occurred.

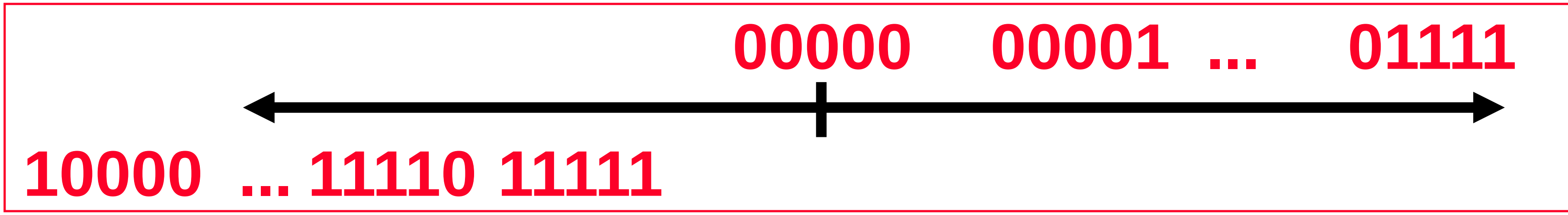


Number summary...

- We represent "things" in computers as particular bit patterns: N bits $\Rightarrow 2^N$
- Decimal for human calculations, binary for computers, hex to write binary more easily
- 1's complement - mostly abandoned



- 2's complement universal in computing: cannot avoid, so learn



- Overflow: numbers ∞; computers finite,errors!

Exercise

X = 1111 1111 1111 1111 1111 1111 1111 1100$_{two}$

Y = 0011 1011 1001 1010 1000 1010 0000 0000$_{two}$

A. X > Y (if signed)

B. X > Y (if unsigned)

C. To identify 30 registers, you need at least 5 bits

	ABC
1:	FFF
2:	FFT
3:	FTF
4:	FTT
5:	TFF
6:	TFT
7:	TTF
8:	TTT

Exercise

- Convert -98_{10} to binary (8 bits), then to hexadecimal (assume 1's complement is used)

- Convert -98_{10} to binary (8 bits), then to hexadecimal (assume 2's complement is used)

Exercise

- Convert 0xFFB5 to binary then to Decimal (assume 1's complement is used)

- Convert 0xFFB5 to binary then to Decimal (assume 2's complement is used)