Possible Representations

Sign Magnitude:	One's Complement	Two's Complement
000 = +0	000 = +0	000 = +0
001 = +1	001 = +1	001 = +1
010 = +2	010 = +2	010 = +2
011 = +3	011 = +3	011 = +3
100 = -0	100 = -3	100 = -4
101 = -1	101 = -2	101 = -3
110 = -2	110 = -1	110 = -2
111 = -3	111 = -0	111 = -1

- **Issues: balance, number of zeros, ease of operations**
- **Which one is best? Why?**

MIPS

- **32 bit signed numbers:**

0000 0000 0000 0000 0000 0000 0000 0000$_{two}$ = 0$_{ten}$
0000 0000 0000 0000 0000 0000 0000 0001$_{two}$ = + 1$_{ten}$
0000 0000 0000 0000 0000 0000 0000 0010$_{two}$ = + 2$_{ten}$
...
0111 1111 1111 1111 1111 1111 1111 1110$_{two}$ = + 2,147,483,646$_{ten}$
0111 1111 1111 1111 1111 1111 1111 1111$_{two}$ = + 2,147,483,647$_{ten}$ *maxint*
1000 0000 0000 0000 0000 0000 0000 0000$_{two}$ = – 2,147,483,648$_{ten}$ *minint*
1000 0000 0000 0000 0000 0000 0000 0001$_{two}$ = – 2,147,483,647$_{ten}$
1000 0000 0000 0000 0000 0000 0000 0010$_{two}$ = – 2,147,483,646$_{ten}$
...
1111 1111 1111 1111 1111 1111 1111 1101$_{two}$ = – 3$_{ten}$
1111 1111 1111 1111 1111 1111 1111 1110$_{two}$ = – 2$_{ten}$
1111 1111 1111 1111 1111 1111 1111 1111$_{two}$ = – 1$_{ten}$

Two's Complement Operations

- Negating a two's complement number: invert all bits and add 1
  - remember: "negate" and "invert" are quite different!
- Converting n bit numbers into numbers with more than n bits:
  - MIPS 16 bit immediate gets converted to 32 bits for arithmetic
  - copy the most significant bit (the sign bit) into the other bits

```
0010  -> 0000 0010
1010  -> 1111 1010
```

  - "sign extension" (lbu vs. lb)

Addition & Subtraction

- Just like in grade school (carry/borrow 1s)

```
  0111        0111        0110
+ 0110      - 0110      - 0101
------      ------      ------
```

- Two's complement operations easy
  - subtraction using addition of negative numbers

```
  0111
+ 1010
------
```

- Overflow (result too large for finite computer word):
  - e.g., adding two n-bit numbers does not yield an n-bit number

```
  0111
+ 0001
------
  1000
```

note that overflow term is somewhat misleading,
it does not mean a carry "overflowed"

Detecting Overflow

- No overflow when adding a positive and a negative number
- No overflow when signs are the same for subtraction
- Overflow occurs when the value affects the sign:
  - overflow when adding two positives yields a negative
  - or, adding two negatives gives a positive
  - or, subtract a negative from a positive and get a negative
  - or, subtract a positive from a negative and get a positive
- Consider the operations A + B, and A – B
  - Can overflow occur if B is 0 ?
  - Can overflow occur if A is 0 ?

Effects of Overflow

- An exception (interrupt) occurs
  - Control jumps to predefined address for exception
  - Interrupted address is saved for possible resumption
- Details based on software system / language
  - example: flight control vs. homework assignment
- Don't always want to detect overflow
  — new MIPS instructions: `addu, addiu, subu`

note: `addiu` *still sign-extends!*
note: `sltu`, `sltiu` *for unsigned comparisons*