CSE 210: Computer Architecture
Lecture 7: Negative Numbers, Overflow

Stephen Checkoway

Oberlin College

Oct. 18, 2021

Slides from Cynthia Taylor

Announcements

- Problem Set 2 due Friday 23:59

- Lab 1 due Sunday 23:59

- Office Hours Tuesday 13:30 – 14:30

How do we indicate a negative number?

- Sign and magnitude
- Ones’ Compliment
- Two’s Compliment

Ones’ Complement

- To make a number negative, just flip all its bits!

- Need to know how many bits: -5 in
  - 4 bits: 0101 => 1010
  - 8 bits: 00000101 => 11111010

A byte representing -6_{10} in Ones' Complement is

A. 00000110

B. 10000110

C. 11111001

D. 11110110

E. None of the above

Ones’ complement

- Two zeros: 00000000 and 11111111 (in 8 bits)

- Addition:
  - Perform normal n-bit addition
  - Add the carryout bit back to the result

Two's Complement

- Flip all the bits and add 1
- For n bits, the unsigned version of $-x = 2^n - x$
- Can represent –128 to 127 in 8 bits
  - In n bits, can represent -2^{n-1} to $2^{n-1} - 1$
- Only one zero (00000000 in 8 bits)
- Used in modern computers

-6 in Two's Complement

A. 11110110

B. 11111001

C. 11111010

D. 11111110

E. None of the above

Two's Complement: $11111101_2 = ?_{10}$

A. -2

B. -3

C. -4

D. -5

E. None of the above

The negation of 11110001_2 is $\_\_\_\_\_\__2$

A. 00001110

B. 00001111

C. 00011110

D. 01110001

E. None of the above

Addition and Subtraction

- Positive and negative numbers are handled in the same way.
- The carry out from the most significant bit is ignored.
- To perform the subtraction A − B, compute A + (two's complement of B)

For n bits, the sum of a number and its negation will be

A. $0_{n-1} \ldots 0_0$

B. $1_{n-1} 0_{n-2} \ldots 0_0$

C. $1_{n-1} \ldots 1_0$

D. It will vary

E. None of the above

$$11110110_2 + 00001100_2 = ?_2$$

A. 00000010

B. 00001100

C. 11110010

D. 11111110

E. None of the above

$1111 + 1000 = \_\__2$

A. 0111

B. 1000

C. 1111

D. 0000

E. None of the above

Overflow

- Overflow occurs when an addition or subtraction results in a value which cannot be represented using the number of bits available.

- In that case, the algorithms we have been using produce incorrect results.

Is overflow a problem in modern programs?

A. Nope, we have totally solved this business!

B. Yep, still a problem.

Handling Overflow

- Hardware can detect when overflow occurs

- Software may or may not check for overflow
  - Java guarantees two’s complement behavior!
  - In C, overflow is “undefined behavior” meaning, it can do anything
  - In Rust, overflow is checked in debug builds but not optimized builds!

How To Detect Overflow

- On an addition, an overflow occurs if and only if the carry into the sign bit differs from the carry out from the sign bit.

- Overflow occurs if adding two negative numbers produces a positive result or if adding two positive numbers produces a negative result.

Will $01111111_2 + 00000101_2$ result in overflow?

A. Yes

B. No

C. It depends

Unsigned Numbers

- Some types of numbers, such as memory addresses, will never be negative
- Some programming languages reflect this with types such as “unsigned int”, which only hold positive numbers
  - uint32_t in C99
  - u32 in Rust
  - Java only has signed types
- In an unsigned byte, values will range from 0 to 255

In MIPS

- add, sub, addi instructions cause exceptions on (signed) overflow
- addu, subu, addiu instructions do not

- Rationale: In C, unsigned types never cause overflow, they're defined to wrap (produce a value modulo 2^n)
- In practice: Since overflow is undefined behavior, it is assumed to never happen so compilers always use addu/subu/addiu

Reading

- Next lecture: How Instructions Are Represented
  – Section 2.5

- Problem Set 2 due Friday

- Lab 1 due Sunday