

Negative Numbers

CS31

Pascal Van Hentenryck



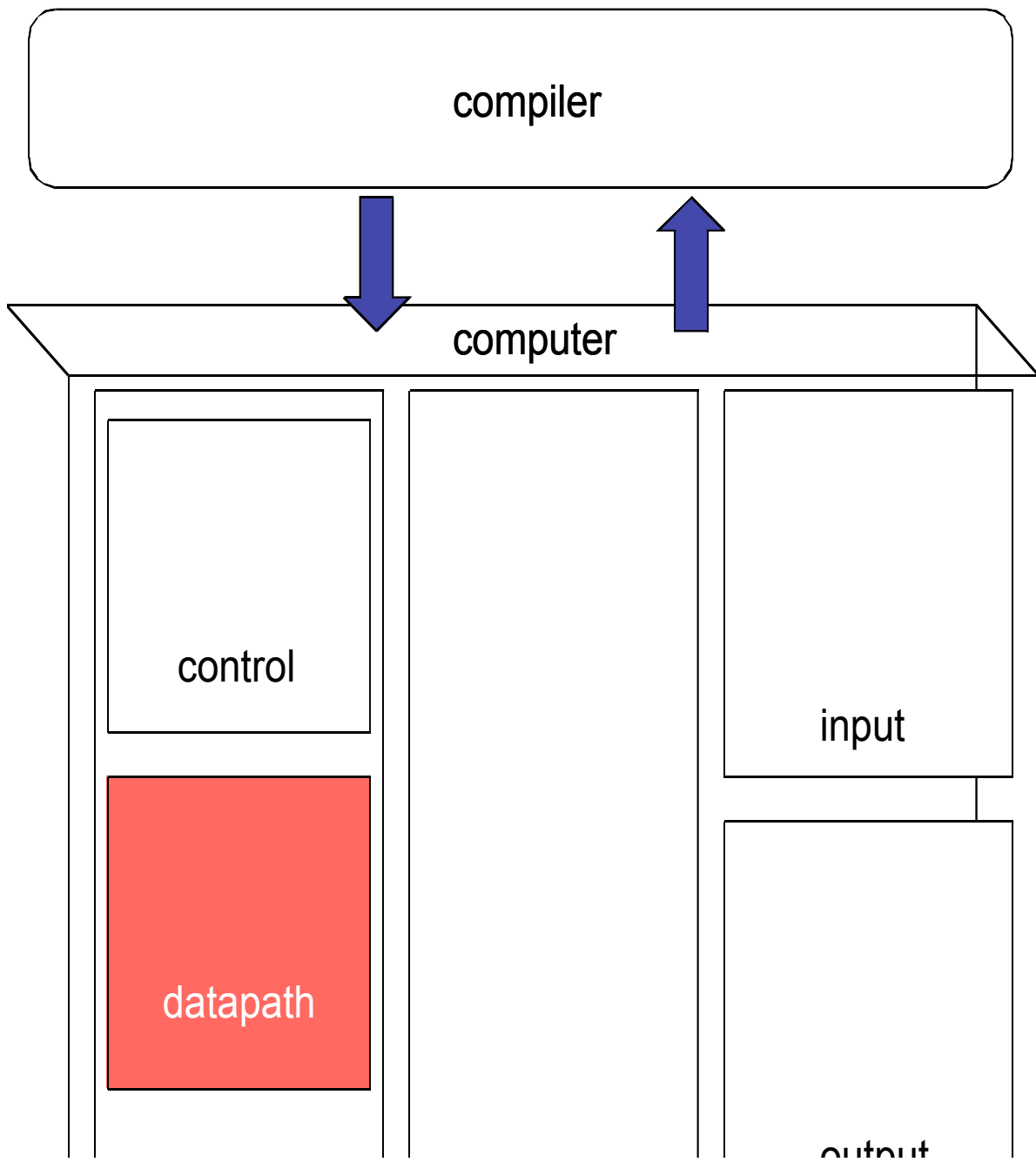
BROWN

Overview

Negative Numbers

- Signed Numbers
- Addition and Subtraction
- Properties
- Overflows

The Big Picture



Abstraction Hierarchy

Programming Language

Assembly Language

Machine Language

Sequential Circuit

Combinational Circuit

Binary Value

Voltage

Signed Numbers

Two issues in representing negative numbers

- How to represent the sign?
- How to represent the value?

Goal of representation

- Simplifying hardware as much as possible
- Addition, subtraction should be simple

Desirable Properties

- Unique representation of zero
- As many positive and negative numbers

Signed Numbers

Three main schemes

- Signed magnitude
- 1's compliment
- 2's compliment

Commonalities

- How they represent the sign
- How they represent positive values

Main difference

- How to represent negative values

Addition and Subtraction

To add signed numbers

- If they have the same sign, perform unsigned addition on the magnitudes and preserve the sign
- If they have different signs, subtract the smaller magnitude from the larger and take the sign of the larger

To subtract

- Change the sign of the subtrahend and add

Issues

- Many special cases
- Hairy to design the circuit
- Multiple representations of zero

1's Complement

Main Idea

A negative number has the unsigned representation of its magnitude in the lower order bits, but with all 1s changed to 0s and vice-versa

Example

$$+36_{10} = 00100100_2$$

$$-36_{10} = 11011011_2$$

Useful questions

- What's the largest possible value representable in n bits?
 $(2^{n-1}) - 1$
- What's the smallest (most negative) value?
 $-(2^{n-1}) + 1$
- How is 0 represented?
00000000 OR 11111111

1's Complement

Alternative Definition

- The 1's complement is obtained by subtracting its magnitude from $2^n - 1$

Example

$$+36_{10} = 00100100_2$$

$$-36_{10} = 11011011_2$$

$$\begin{array}{r} 11111111_2 \\ - 00100100_2 \\ \hline 11011011_2 \end{array}$$

1's Complement

$$\begin{array}{r} 00010_2 \\ + 11110_2 \\ \hline 1\ 00000_2 \end{array}$$

$$\begin{array}{r} 00100_2 \\ + 11101_2 \\ \hline 1\ 00001_2 \end{array}$$

$$\begin{array}{r} 00101_2 \\ + 11101_2 \\ \hline 1\ 00010_2 \end{array}$$

1's Complement

Why such a bizarre representation?

- Addition of two positive numbers is exactly like the addition of two unsigned numbers
- Addition of two negative numbers is almost exactly like the addition of two unsigned numbers
- Addition of a positive and a negative number is almost exactly like the addition of two unsigned numbers

However

- Having two representations for zero is really annoying
- Still need some special cases

2's Complement

Main Idea

- The 2's complement of a negative number is obtained by adding 1 to the 1's complement of its magnitude

Example

$$+36_{10} = 00100100_2$$

$$-36_{10} = 11011011_2 \quad (1's \text{ comp.})$$

$$-36_{10} = 11011100_2 \quad (2's \text{ comp.})$$

2's Complement

Alternative Definition

- The 2's complement is obtained by subtracting its magnitude from 2^n

Example

$$+36_{10} = 00100100_2$$

$$- 36_{10} =$$

$$\begin{array}{r} 10000000_2 \\ - \quad 00100100_2 \\ \hline 11011100_2 \end{array}$$

2's Complement

The value of an n-bit signed number

$$-b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$$

Useful Questions

- What's the largest possible value representable in n bits?
 $(2^{n-1}) - 1$
- What's the smallest (most negative) value?
 $-(2^{n-1})$
- How is 0 represented?
00000000

2's Complement

$$-83_{10} = 10101101$$

$$1_{10} = 00000001$$

$$01010010_2$$

$$+ \underline{\hspace{1.5cm}} 1_2$$

$$+ 83_{10} = 01010011_2$$

$$11111110_2$$

$$+ \underline{\hspace{1.5cm}} 1_2$$

$$- 1_{10} = 11111111_2$$

2's Complement Counting

Decimal	2's Complement
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

You can get from any number to the next by adding 1.

- This is a very important property

2's Complement

The only problem

- What is the 2's complement of the largest negative value?

the same number

Example (4 bits representation)

$$-8_{10} = 1000_2 \text{ (in 2's complement)}$$

2's complement

$$1000_2$$

$$- \underline{1000_2}$$

$$1000_2$$

This nuisance is unavoidable

- Having same number of positive and negative numbers and a single representation of zero requires an odd number of values
- Words always have an even number of values

2's Complement Addition

We can add 2's complement numbers using ordinary unsigned addition

- Don't have to worry about signs
- Do have to worry about overflow

$$+ 6 \quad 0110_2$$

$$\underline{- 2 \quad 1110_2}$$

$$+ 4 \quad 10100_2$$

$$- 5 \quad 1011_2$$

$$\underline{- 1 \quad 1111_2}$$

$$- 6 \quad 11010_2$$

$$- 7 \quad 1001_2$$

$$\underline{+ 5 \quad 0101_2}$$

$$- 2 \quad 1110_2$$

$$+ 5 \quad 0101_2$$

$$\underline{- 1 \quad 1111_2}$$

$$+ 4 \quad 10100_2$$

Why does it work?

Two positive numbers: obvious

One positive number p and one negative number $-m$

- if $p > m$ then

$$(p + 2^n - m) \bmod 2^n$$

$$(2^n + (p - m)) \bmod 2^n$$

$$p - m$$

- if $p < m$ then

$$(2^n - (m - p)) \bmod 2^n$$

$$2^n - (m - p)$$

which is the 2's complement representation of $m - p$

Overflow

Only finitely many numbers can be represented

- An overflow happens when the result is too large to be represented

Adding two numbers of different signs cannot cause overflow

- Why? **Because the result will always have a smaller magnitude**

Adding two numbers of the same sign overflows when the result has a different sign from the addends

$$+ 6 \quad 0110_2$$

$$\underline{+ 5 \quad 0101_2}$$

$$+11 \quad 1011_2$$

$$- 4 \quad 1100_2$$

$$\underline{- 7 \quad 1001_2}$$

$$- 11 \quad 10101_2$$

Overflow

What to do in the presence of overflow?

- Machine dependent
- Raise an exception: see interruption later on

2's Complement Subtraction

Can subtract 2's complement numbers with unsigned subtraction.

Better to (we already have the hardware)

1. Complement the subtrahend
2. Add the minuend

Rewrite as

1. Digit-complement the subtrahend
2. Add the minuend + 1

We will see later that we can do this in hardware with only one addition by making the *carry in* be 1.

Subtraction Examples

$$\begin{array}{r} + 7 \quad 0111_2 \\ - 2 \quad 0010_2 \\ \hline + 5 \quad 0101_2 \end{array}$$

$$\begin{array}{r} 0111_2 \\ + 1110_2 \\ \hline 10101_2 \end{array}$$

$$\begin{array}{r} - 6 \quad 1010_2 \\ - - 3 \quad -1101_2 \\ \hline - 3 \quad 1101_2 \end{array}$$

$$\begin{array}{r} 1010_2 \\ + 0011_2 \\ \hline 1101_2 \end{array}$$

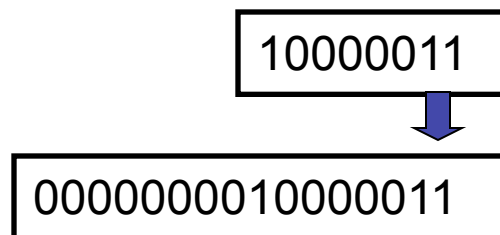
Check overflow in addition step

Sign Extension

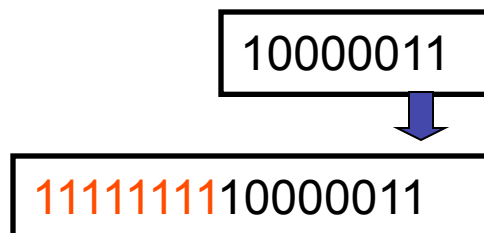
It is sometimes necessary to move from a n (e.g. 8) bit representation to a m (e.g. 16) bit representation.

Need to maintain the same value

Unsigned Numbers



Signed Numbers



Radix-Complement Representation

Generalization from binary to any radix

Useful when reading your programs

How to complement a number

- Subtract it from b^n
- Or complement the digits and add 1

10's complement (4 digits)

2491	7509
0006	9994
0097	9903
9999	0001

Radix Complement Examples

Given: 2491_{10}

Complement digits: 9999

2491

7508

Add 1: 7509

Given: 0001_{10}

Complement digits: 9999

0001

9998

Add 1: 9999

Representing Characters

We use ASCII (American Standard Code for Information Interchange)

There are other codes, but same basic idea

- 7 bits for each character
- usually stored one character per byte
- some characters are non-printing
BEL, CR, LF
- characters are in order, so we can use numerical sorts for alphabetical sorting
1000011, 1010011, 0110001,
0110010

Note: the byte representing 1 is different from the byte representing '1'

Other character representation

- Unicode (Java)
- Larger character set (more characters)

01010011 01100101 01100101
00100000 01111001 01101111
01110101 00100000 01101110
01100101 01111000 01110100
00100000 01110100 01101001
01101101 01100101 00100001

(See you next time!)