Signed integers

Lecture B03

Lecture notes section B03

Last time

- Place value number systems
- Bases
  - binary (base 2)
  - hexadecimal (base 16)
  - octal (base 8)
- Thinking in binary
  - powers of two
  - binary 0 to 15
- Converting
  - from and to binary
- Unsigned integers

In this lecture

- Arithmetic
  - addition
- Signed integers
  - two's complement system
- More arithmetic
  - addition
  - subtraction
  - multiplication

Addition

- In decimal:

$$\begin{array}{cccc} & 0 & 1 & 0 \\ & 5 & 3 & 4 \\ + & 1 & 9 & 1 \\ \hline & 7 & 2 & 5 \end{array}$$

carry (zero or one): remaining digit of the sum (or zero)

← work right to left

sum (one digit): right-hand digit of the sum

Addition

- In unsigned binary

carry (0 or 1)

$$\begin{array}{ccccccccc} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & \\ & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ + & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

final carry used to detect overflow: 0 indicates result is correct; 1 indicates result is wrong

sum (0 or 1)

Addition

- To verify, do same addition in decimal

$$\begin{array}{c} \boxed{58} \\ + \\ \boxed{92} \\ = \\ \boxed{150} \end{array} \qquad \begin{array}{ccccccccc} & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ + & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

✓

Overflow

- An incorrect result occurs if the sum is too large to be represented in the number of bits available
- For addition of unsigned integers, overflow occurs if and only if carry out of MSB is 1

Overflow

- Try to add $200_{10} + 101_{10}$
  - 8-bit unsigned binary

	1	1	0	0	0	0	0	0
	1	1	0	0	1	0	0	0
+	0	1	1	0	0	1	0	1
	0	0	1	0	1	1	0	1

final carry is 1, result is incorrect

result is 45, not 301

Subtraction

- In decimal

	4	13	
	~~5~~	~~3~~	4
-	1	9	1
	3	4	3

if necessary, borrow ten from previous column

difference (one digit)

Subtraction

- Could do same with binary
  - borrow 2 from previous column if needed
- Subtraction means negative numbers are possible
  - need a system of representation of both positive and negative (signed) numbers
    - using just bits
  - borrow alone isn't enough

Signed integers

- Need to reserve one bit of integer for sign
- Simplest technique is signed magnitude
  - MSB = 0: number is positive
  - MSB = 1: number is negative

0	0	0	0	1	1	1	0	= +14
1	0	0	0	1	1	1	0	= −14

sign bit

Signed integers

- Signed magnitude has problems
  - more complex circuitry needed to perform addition/subtraction
  - two representations of zero!
    - +0 (all bits 0)
    - −0 (all bits 0 except MSB)
- Need a better system to represent signed numbers

Signed integers

- Almost all computers use two’s complement system
  - so called because negative numbers are represented as the difference (complement) from a power of two
- MSB’s place value is negative of normal

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
(-128)	(64)	(32)	(16)	(8)	(4)	(2)	(1)

MSB in 8-bit binary is bit 7

Signed integers

- Smallest value (8 bits)

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
(-128)	(64)	(32)	(16)	(8)	(4)	(2)	(1)
1	0	0	0	0	0	0	0
✓	✗	✗	✗	✗	✗	✗	✗

This bit pattern no longer represents +128 as it did in an unsigned representation

-128 = $\boxed{-128}$

Signed integers

- Largest value (8 bits)

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
(-128)	(64)	(32)	(16)	(8)	(4)	(2)	(1)
0	1	1	1	1	1	1	1
✗	✓	✓	✓	✓	✓	✓	✓

This bit pattern still represents +127 as it did in an unsigned representation

64 + 32 + 16 + 8 + 4 + 2 + 1 = $\boxed{+127}$

Signed integers

- In 8-bit binary
  - smallest representable value is -128
  - largest representable value is +127
  - All values between are representable
  - No other values are representable
- Positive values use same bit pattern as unsigned
- Negative values use bit patterns formerly representing positive numbers
  - positive range is reduced

Signed integers

- 8-bit two’s complement

binary	as unsigned value	as signed value
10000000_2	128_{10}	-128_{10}
10000001_2	129_{10}	-127_{10}
10000010_2	130_{10}	-126_{10}
…	…	…
11111110_2	254_{10}	-2_{10}
11111111_2	255_{10}	-1_{10}
00000000_2	0_{10}	0_{10}
00000001_2	1_{10}	$+1_{10}$
00000010_2	2_{10}	$+2_{10}$
…	…	…
01111101_2	125_{10}	$+125_{10}$
01111110_2	126_{10}	$+126_{10}$
01111111_2	127_{10}	$+127_{10}$

negative numbers have 1 in MSB

positive numbers have 0 in MSB

unsigned and signed values differ by 256 (2^8) for negative numbers

signed and unsigned values are same for positive numbers

zero is effectively positive

Signed integers

- In n-bit binary
  - represent $-x$ as $2^n - x$
    - e.g., represent -2 as $2^8 - 2 = 254$
  - number r represents value $r - 2^n$
    - e.g., 254 really represents 254 – 256 = -2
- Called two’s complement
  - number is represented as difference (complement) from power of two
- Range is -2^{n-1} to $+2^{n-1} - 1$

Signed integers

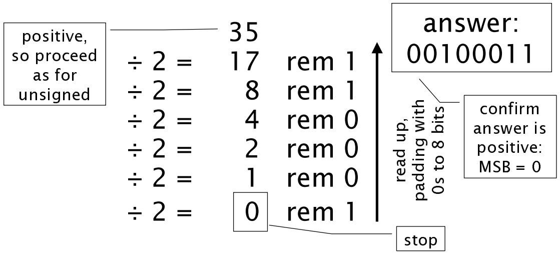
- C has signed integers of various sizes
- 8 bits (`signed char`)
  - −128 to +127
- 16 bits (`short`)
  - −32768 to +32767
- 32 bits (`long`)
  - −2 147 483 648 to +2 147 483 647
- 64 bits (`long long`)
  - −9 223 372 036 854 775 808 to +9 223 372 036 854 775 807
- `int` is either `short` or `long`

Signed integers

- Properties of two's complement
  - positive numbers (including zero)
    - MSB is 0
    - zero value is all 0 bits
    - same bit pattern as unsigned numbers
  - negative numbers start with 1
    - MSB is 1
    - −1 value is all 1 bits
  - negative range is bigger than positive range by 1

Signed integers: to binary

- What is $+35_{10}$ in 8-bit two's complement?



positive, so proceed as for unsigned

	35	
÷ 2 =	17	rem 1
÷ 2 =	8	rem 1
÷ 2 =	4	rem 0
÷ 2 =	2	rem 0
÷ 2 =	1	rem 0
÷ 2 =	0	rem 1

read up, padding with 0s to 8 bits

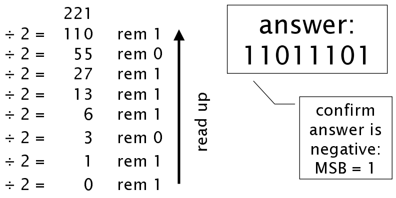
stop

answer: 00100011

confirm answer is positive: MSB = 0

Signed integers: to binary

- What is -35_{10} in 8-bit two's complement?
  - represent −35 as 2^8 − 35 = 256 − 35 = 221



	221	
÷ 2 =	110	rem 1
÷ 2 =	55	rem 0
÷ 2 =	27	rem 1
÷ 2 =	13	rem 1
÷ 2 =	6	rem 1
÷ 2 =	3	rem 0
÷ 2 =	1	rem 1
÷ 2 =	0	rem 1

read up

answer: 11011101

confirm answer is negative: MSB = 1

Signed integers: to binary

- Another way to convert to signed
  - −35 represented as 256 − 35
  - 256 − 35 = (255 + 1) − x = (255 − 35) + 1
  - 255 is highest representable 8-bit unsigned value
    - In binary, all bits are 1
  - Easy to subtract a bit from 1
    - 1 − 0 = 1, 1 − 1 = 0
    - no borrow, just flip the bit
  - Easy to subtract (positive) 35 from 255
    - just flip all the bits
  - Still need to add 1 afterwards

Signed integers: to binary

- To write a negative number in two's complement binary:
  - start with the binary representation of the equivalent positive number
  - negate the number by doing these two steps
    - flip all the bits (0 ⇔ 1)
    - add 1 to the result
- If flip-and-add-1 is repeated, original positive number is returned
  - −(−x) = x

Signed integers: to binary

- What is -35_{10} in 8-bit two's complement?

start with equivalent positive number

$+35_{10} = 00100011_2$

flip bits

11011100

$+ 1$

add 1

$\boxed{1}1011101$

confirm result is negative: MSB = 1

answer: 11011101

Signed integers: from binary

- What value does 8-bit two's complement 01100011 have?
  - MSB = 0, so positive
  - just convert from binary like unsigned

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
(-128)	(64)	(32)	(16)	(8)	(4)	(2)	(1)
0	1	1	0	0	0	1	1
✗	✓	✓	✗	✗	✓	✓	✓

64 + 32 + 4 + 2 + 1 = $\boxed{+103}$

Signed integers: from binary

- What value does 8-bit two's complement 11000100 have?
  - can convert as normal with MSB worth -128

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
(-128)	(64)	(32)	(16)	(8)	(4)	(2)	(1)
1	1	0	0	0	1	0	0
✓	✓	✗	✗	✗	✓	✗	✗

-128+64 + 4 = $\boxed{-60}$

Signed integers: from binary

- What value does 8-bit two's complement 11000100 have?
  - or, find positive equivalent using flip-and-add-1

11000100

flip bits

00111011

add 1

$+ 1$

convert from binary to get 60_{10}

00111100

don't forget minus sign

Answer: -60_{10}

Signed addition

- What's so great about two's complement?
  - addition is same for positive and negative numbers
    - positive + positive
    - positive + negative
    - negative + positive
    - negative + negative

Signed addition

- -13 + (-10)
  - 8-bit signed

final carry no longer indicates overflow

	1	1	1	1	0	1	1	0	
		1	1	1	1	0	0	1	1
+		1	1	1	1	0	1	1	0
		1	1	1	0	1	0	0	1

-13

-10

-23 ✓

Signed addition

- -51 + 36
  - 8-bit signed

0 0 0 0 1 1 0 0

1 1 0 0 1 1 0 1 — -51

\+ 0 0 1 0 0 1 0 0 — 36

1 1 1 1 0 0 0 1

-15 ✓

Signed addition overflow

- 58 + 92
  - 8-bit signed

0 1 1 1 1 0 0 0

signed overflow occurs when these two carries differ

0 0 1 1 1 0 1 0 — 58

\+ 0 1 0 1 1 1 0 0 — 92

1 0 0 1 0 1 1 0

-106 ✗

or, more obviously, if the result has the wrong sign (here, pos + pos = neg?)

Signed subtraction

- Subtraction is easy in two's complement
  - x - y = x + (-y)
  - negate RHS of subtraction
    - with flip-and-add-1
  - then do normal addition
  - overflow rules are same as for addition

Signed subtraction

- 42 - 19
  - 8-bit signed

1 1 1 0 1 0 0 0

flip-and-add-1 to replace +19 with -19, and add

0 0 1 0 1 0 1 0 — 42

~~– 0 0 0 1 0 0 1 1~~ — 19

\+ 1 1 1 0 1 1 0 1 — -19

0 0 0 1 0 1 1 1

23 ✓

Binary multiplication

- Using same techniques as for decimal

10 — 00001010

11 — × 00001011

00001010

0001010

\+ 01010

for every 1 in second number, write first number, aligned at LSB

110 — 01101110

add to produce result

Covered in this lecture

- Arithmetic
  - unsigned addition
- Signed integers
  - two's complement system
- More arithmetic
  - signed addition
  - subtraction
  - multiplication

Going further

- Hardware implementation of addition and subtraction
  - adder circuit
- Binary division
  - same technique as long division in decimal

Next time

- Floating point
  - representing real numbers

Reading:
Lecture notes section B04

Copyright