Subtraction and Negative Numbers

Steven R. Bagley

Introduction

- Saw in the last lecture how to add two binary numbers
- Using just discrete logic gates
- But how do we do subtraction?
- And what about negative numbers?

Subtraction

$$\begin{array}{ccccc} 4 & 0 & 9 & 6 & \\ 1 & 0 & 2 & 4 & - \\ \hline \end{array}$$

Standard decimal subtraction

Subtraction

4	0	9	6	
1	0	2	4	-
			2	

Standard decimal subtraction

Subtraction

$$
\begin{array}{cccccc}
 & 4 & 0 & 9 & 6 & \\
 & 1 & 0 & 2 & 4 & - \\
\hline
 & 3 & 0 & 7 & 2 &
\end{array}
$$

Standard decimal subtraction

Subtraction

```
 8  1  9  2
 1  0  2  4  -
-------------
```

Standard decimal subtraction
Gets more interesting when some of the bottom numbers are bigger than the top (e.g. 2 and 4)

Subtraction

$$\begin{array}{ccccc} 8 & 1 & {}^{8}\cancel{9} & {}^{1}2 & \\ 1 & 0 & 2 & 4 & - \\ \hline \end{array}$$

Need to borrow one
Now 12 -4 which of course is 8

Subtraction

$$\begin{array}{ccccc} 8 & 1 & {}^{8}\cancel{9} & {}^{1}2 & \\ 1 & 0 & 2 & 4 & - \\ \hline & & 6 & 8 & \end{array}$$

Need to borrow one
Now 12 -4 which of course is 8
Then subtract 2 from the new 8 in the 10s column

Subtraction

$$\begin{array}{ccccc} 8 & 1 & {}^{8}\cancel{9} & {}^{1}2 & \\ 1 & 0 & 2 & 4 & - \\ \hline 7 & 1 & 6 & 8 & \end{array}$$

Need to borrow one
Now 12 -4 which of course is 8
Then subtract 2 from the new 8 in the 10s column
And repeat
Can sometimes have to borrow from the next column along

Subtraction

- Subtract each column from right
- If bottom row greater than top, we *borrow one* from the next column (or even further)
- Binary subtraction is the same…

Binary Subtraction

$$\begin{array}{r} 1\ 0\ 1\ 0 \\ 1\ 1\ - \\ \hline \end{array}$$

Standard decimal subtraction
Have to borrow ‘one’ to do first subtraction

Binary Subtraction

$$\begin{array}{ccccc} 1 & 0 & {}^{0}\cancel{1} & {}^{1}0 & \\ & & 1 & 1 & - \\ \hline \end{array}$$

Standard decimal subtraction
Have to borrow ‘one’ to do first subtraction

Binary Subtraction

$$
\begin{array}{ccccc}
1 & 0 & {}^{0}\cancel{1} & {}^{1}0 & \\
 & & 1 & 1 & - \\
\hline
 & & & 1 &
\end{array}
$$

Standard decimal subtraction
Have to borrow ‘one’ to do first subtraction 10 -1 is 1
Now need to borrow one from next-left, btu that is zero -- so that borrows from its left

Binary Subtraction

$$\begin{array}{ccccc} {}^{0}\cancel{1} & {}^{1}0 & {}^{0}\cancel{1} & {}^{1}0 & \\ & & 1 & 1 & - \\ \hline & & & 1 & \end{array}$$

Standard decimal subtraction
Have to borrow ‘one’ to do first subtraction 10 -1 is 1
Now need to borrow one from next-left, btu that is zero -- so that borrows from its left
Now we can borrow…

Binary Subtraction

$$
\begin{array}{ccccc}
{}^{0}\cancel{1} & {}^{0}{}^{1}\cancel{0} & {}^{10}\cancel{1} & {}^{1}0 & \\
 & & 1 & 1 & - \\
\hline
 & & & 1 &
\end{array}
$$

10 – 1 is 1 again

Binary Subtraction

$$
\begin{array}{ccccc}
{}^{0}\cancel{1} & {}^{0}{}^{1}\cancel{0} & {}^{10}\cancel{1} & {}^{1}0 & \\
 & & 1 & 1 & - \\
\hline
0 & 0 & 1 & 1 &
\end{array}
$$

10 - 1 is 1 again

Subtraction in Logic

- Can design a circuit
- Half-subtractor and full-subtractors
- But there's another way to subtract two numbers…

Subtraction by Addition

- Can also 'subtract' by adding a negative number
- So 4096-2048 is the same as 4096 + -2048
- Same holds true in binary…
- But how do we represent a negative number in binary?

Negative Numbers

- Four approaches
  - Sign and Magnitude
  - One’s complement
  - Excess-*n*
  - Two’s complement

4-bit signed encodings

Sign and Magnitude		Excess-3		One's Complement		Two's Complement	
1111	−7	0000	−3	1000	−7	1000	−8
1110	−6	0001	−2	1001	−6	1001	−7
1101	−5	0010	−1	1010	−5	1010	−6
1100	−4	0011	0	1011	−4	1011	−5
1011	−3	0100	+1	1100	−3	1100	−4
1010	−2	0101	+2	1101	−2	1101	−3
1001	−1	0110	+3	1110	−1	1110	−2
1000	−0	0111	+4	1111	−0	1111	−1
0000	+0	1000	+5	0000	+0	0000	0
0001	+1	1001	+6	0001	+1	0001	+1
0010	+2	1010	+7	0010	+2	0010	+2
0011	+3	1011	+8	0011	+3	0011	+3
0100	+4	1100	+9	0100	+4	0100	+4
0101	+5	1101	+10	0101	+5	0101	+5
0110	+6	1110	+11	0110	+6	0110	+6
0111	+7	1111	+12	0111	+7	0111	+7

Sign and Magnitude

Sign and Magnitude		Excess-3		One's Complement		Two's Complement	
1111	-7	0000	-3	1000	-7	1000	-8
1110	-6	0001	-2	1001	-6	1001	-7
1101	-5	0010	-1	1010	-5	1010	-6
1100	-4	0011	0	1011	-4	1011	-5
1011	-3	0100	+1	1100	-3	1100	-4
1010	-2	0101	+2	1101	-2	1101	-3
1001	-1	0110	+3	1110	-1	1110	-2
1000	-0	0111	+4	1111	-0	1111	-1
0000	+0	1000	+5	0000	+0	0000	0
0001	+1	1001	+6	0001	+1	0001	+1
0010	+2	1010	+7	0010	+2	0010	+2
0011	+3	1011	+8	0011	+3	0011	+3
0100	+4	1100	+9	0100	+4	0100	+4
0101	+5	1101	+10	0101	+5	0101	+5
0110	+6	1110	+11	0110	+6	0110	+6
0111	+7	1111	+12	0111	+7	0111	+7

Use top bit to represent sign (just as we do in decimal)
Addition is 'tricky'
Have 'two zeroes'

Excess-*n*

Sign and Magnitude		Excess-3		One's Complement		Two's Complement	
1111	-7	0000	-3	1000	-7	1000	-8
1110	-6	0001	-2	1001	-6	1001	-7
1101	-5	0010	-1	1010	-5	1010	-6
1100	-4	0011	0	1011	-4	1011	-5
1011	-3	0100	+1	1100	-3	1100	-4
1010	-2	0101	+2	1101	-2	1101	-3
1001	-1	0110	+3	1110	-1	1110	-2
1000	-0	0111	+4	1111	-0	1111	-1
0000	+0	1000	+5	0000	+0	0000	0
0001	+1	1001	+6	0001	+1	0001	+1
0010	+2	1010	+7	0010	+2	0010	+2
0011	+3	1011	+8	0011	+3	0011	+3
0100	+4	1100	+9	0100	+4	0100	+4
0101	+5	1101	+10	0101	+5	0101	+5
0110	+6	1110	+11	0110	+6	0110	+6
0111	+7	1111	+12	0111	+7	0111	+7

Add a fixed offset to all values
Seen this used with sampled audio data…

One's Complement

Sign and Magnitude		Excess-3		One's Complement		Two's Complement	
1111	−7	0000	−3	1000	−7	1000	−8
1110	−6	0001	−2	1001	−6	1001	−7
1101	−5	0010	−1	1010	−5	1010	−6
1100	−4	0011	0	1011	−4	1011	−5
1011	−3	0100	+1	1100	−3	1100	−4
1010	−2	0101	+2	1101	−2	1101	−3
1001	−1	0110	+3	1110	−1	1110	−2
1000	−0	0111	+4	1111	−0	1111	−1
0000	+0	1000	+5	0000	+0	0000	0
0001	+1	1001	+6	0001	+1	0001	+1
0010	+2	1010	+7	0010	+2	0010	+2
0011	+3	1011	+8	0011	+3	0011	+3
0100	+4	1100	+9	0100	+4	0100	+4
0101	+5	1101	+10	0101	+5	0101	+5
0110	+6	1110	+11	0110	+6	0110	+6
0111	+7	1111	+12	0111	+7	0111	+7

In this case, negative values are inverted (notted)
Has two zeroes again, but addition now works as with unsigned numbers (more or less)

4-bit signed encodings

Sign and Magnitude		Excess-3		One's Complement		Two's Complement	
1111	−7	0000	−3	1000	−7	1000	−8
1110	−6	0001	−2	1001	−6	1001	−7
1101	−5	0010	−1	1010	−5	1010	−6
1100	−4	0011	0	1011	−4	1011	−5
1011	−3	0100	+1	1100	−3	1100	−4
1010	−2	0101	+2	1101	−2	1101	−3
1001	−1	0110	+3	1110	−1	1110	−2
1000	−0	0111	+4	1111	−0	1111	−1
0000	+0	1000	+5	0000	+0	0000	0
0001	+1	1001	+6	0001	+1	0001	+1
0010	+2	1010	+7	0010	+2	0010	+2
0011	+3	1011	+8	0011	+3	0011	+3
0100	+4	1100	+9	0100	+4	0100	+4
0101	+5	1101	+10	0101	+5	0101	+5
0110	+6	1110	+11	0110	+6	0110	+6
0111	+7	1111	+12	0111	+7	0111	+7

Very similar to one's complement but we invert negative numbers and add one
Only one zero, but we have one more negative number than positive
Addition same as with unsigned numbers

Adding negative

- Easiest to understand addition with sign and magnitude
- Lets add -18 and 25 (in 16-bit sign/mag format)

```
-18 = 1000000000010010
 25 = 0000000000011001
```

- Can't just add as normal binary…

Would give us -(18+25)

Adding Negative

- ```
  -18 = 1000000000010010
   25 = 0000000000011001
  ```

- Need to find the larger number and subtract the smaller from it (based on magnitudes)
- Then adjust the signs…
- Messy!

Still needs a dedicated subtractor circuit
  ```

One’s Complement

- Negative numbers formed by inverting the bits of the equivalent positive number
- But leads to two zeros which is annoying
- However, addition can be done using a standard full adder

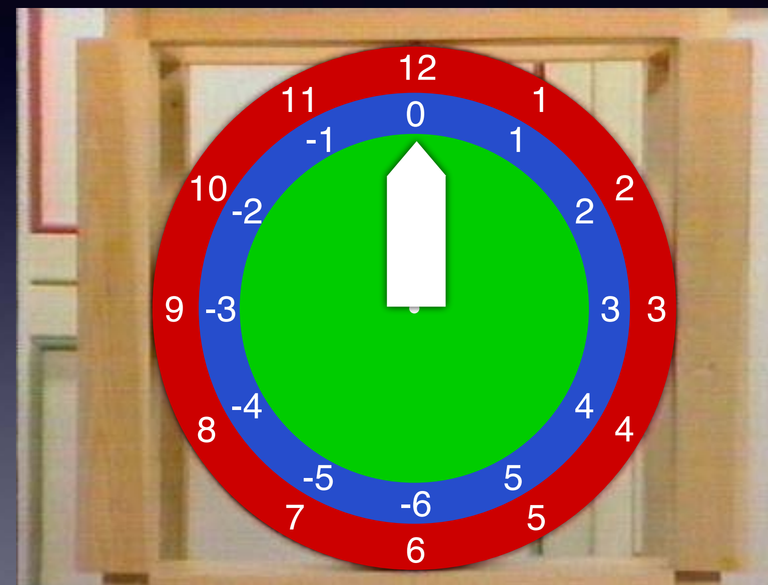
With some tweaks
Example add together -124 and 236 using one’s complement notation
124 == 0b0111 1100
236 == 0b1110 1100
Use 12 bits

Two's complement

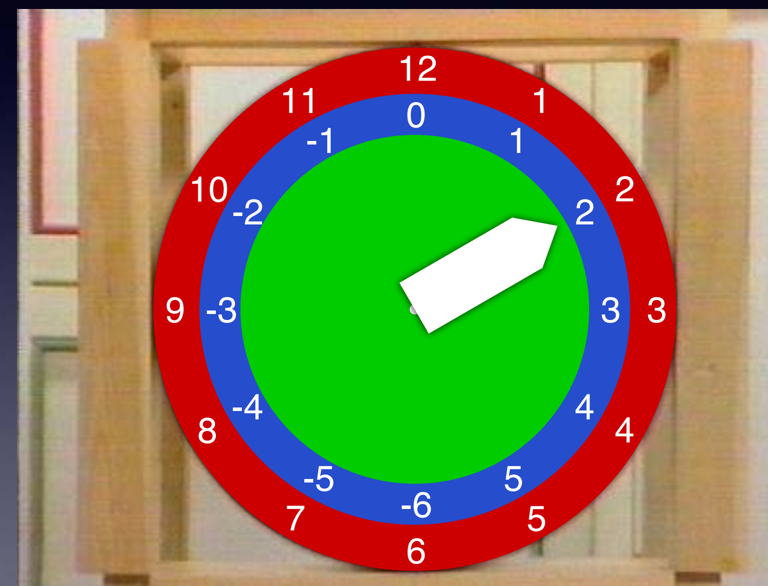
- Represent -`x` in n bits as $2^n - x$
- Quick way to calculate
  - Invert `x` and add `1` to result
- Addition identical to unsigned numbers (although carry bit is ignored)
- The standard on all CPUs for signed numbers

Show this in C

Two's Complement Clock



Two's Complement Clock



Two's Complement Clock