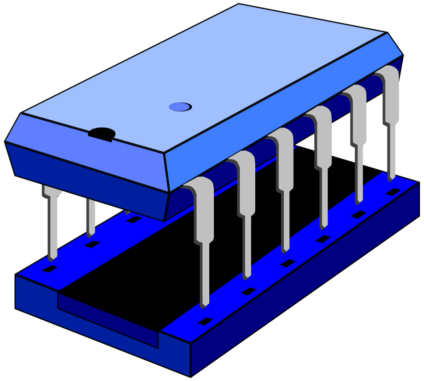

BINARY ARITHMETIC

Bernhard Kainz (with thanks to **A. Gopalan, N. Dulay** and **E. Edwards**)

b.kainz@imperial.ac.uk

Binary Arithmetic

- Unsigned
  - Addition, Subtraction, Multiplication and Division

- Signed
  - Two's Complement Addition, Subtraction, Multiplication and Division
    - Chosen because of its widespread use

Binary Arithmetic

- Couple of definitions
  - Subtrahend: what is being subtracted
  - Minuend: what it is being subtracted from
  - Example: 612 - 485 = 127
  - 485 is the subtrahend, 612 is the minuend, 127 is the result

Binary Addition – Unsigned

- Reasonably straight forward
- Example: Perform the binary addition 111011 + 101010

Carry		1	1	1		1		
A			**1**	**1**	**1**	**0**	**1**	**1**
B			**1**	**0**	**1**	**0**	**1**	**0**
Sum		**1**	**1**	**0**	**0**	**1**	**0**	**1**
Step		*7*	*6*	*5*	*4*	*3*	*2*	*1*

In Decimal: 59 + 42 = 101

Binary Subtraction – Unsigned

- Reasonably straight forward as well ☺
- Example: Perform the binary subtraction 1010101 - 11100

A''		0	1	10				
A'		1	0	0	10			
A		**1**	**0**	**1**	**0**	**1**	**0**	**1**
B			**–**	**1**	**1**	**1**	**0**	**0**
Diff		**0**	**1**	**1**	**1**	**0**	**0**	**1**
Step		*7*	*6*	*5*	*4*	*3*	*2*	*1*

Step k	**$A_k – B_k = Diff_k$**
1	1 – 0 = 1
2	0 –0 = 0
3	1 – 1 = 0
4	0 – 1 Borrow by subtracting 1 from $A_{7..5}$=101 to give $A'_{7..5}$=100 and A'_4=10. **Now use A' instead of A**, e.g. $A'_4 – B_4$ 10 – 1 =1
5	0 – 1 Subtract 1 from $A'_{7..6}$ =10 to give $A''_{7..6}$ =01, A''_5 = 10. **Now use A'' instead of A'**, e.g. $A''_5 – B_5$ 10 – 1 =1
6	1 – 0 = 1 i.e. $A''_6 – B_6$
7	0 – 0 = 0

Binary Multiplication – Unsigned

- Example: Perform the binary multiplication 11101 x 111

A				**1**	**1**	**1**	**0**	**1**
B					x	**1**	**1**	**1**
				1	1	1	0	1
			1	1	1	0	1	
		1	1	1	0	1		
Answer	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**
Carry	1	10	10	1	1			

Binary Division – Unsigned

- Recall:
  - Division is: $\frac{dividend}{divisor} = quotient + \frac{remainder}{divisor}$
  - Or: $dividend = quotient \times divisor + remainder$
  - Left as an exercise ☺
    - Can use long division

Binary Arithmetic – Signed

- Two's complement Arithmetic because of it's widespread use
- Recall

  - Addition and subtraction in two's complement works without having a separate sign bit

- Overflow
  - Result of an arithmetic operation is too large or too small to fit into the resultant bit-group (E.g.: 9 can't fit into 4-bits in Two's complement)
  - Normally left to programmer to deal with this situation

Two’s Complement – Addition

- Add the values and discard any carry-out bit

- Example: Add –8 to +3 and -2 and -5 using 8-bit two’s complement

(+3)	0000 0011		(–2)	1111 1110	
+(–8)	1111 1000		+(–5)	1111 1011	
(–5)	1111 1011		(–7)	1 1111 1001	
				↑ Discard Carry-Out	

Two's Complement – Addition

- Overflow
  - Occurs if and only if 2 Two's Complement numbers are added and they both have the same sign (both positive or both negative) and the result has the opposite sign
    - Adding two positive numbers must give a positive result
    - Adding two negative numbers must give a negative result
  - Never occurs when adding operands with different signs
  - E.g.
    - (+A) + (+B) = -C
    - (-A) + (-B) = +C

Two’s Complement – Addition

- Overflow

  - Example: Using 4-bit Two's Complement numbers ($-8 \le x \le +7$), calculate (-7) + (-6)

(–7)	1001	
+(–6)	1010	
(+3)	1 0011	**“Overflow”**

Two’s Complement – Subtraction

- Accomplished by negating the subtrahend and adding it to the minuend
  - Any carry-out bit is discarded

- Example: Calculate 8 – 5 using an 8-bit two’s complement representation
  - Recall: 8 – 5 → 8 + (- 5)

(+8)	0000 1000		0000 1000
–(+5)	0000 0101	-> Negate ->	+ 1111 1011
(+3)			1 0000 0011
			Discard

Two's Complement – Subtraction

- Overflow

  - Occurs if and only if 2 two's complement numbers are subtracted, and their signs are different, and the result has the same sign as the subtrahend

  - E.g.
    - (+A) – (–B) = –C
    - (–A) – (+B) = +C

Two's Complement – Subtraction

- Overflow

  - Example: Using 4-bit Two's Complement numbers (−8 ≤ x ≤ +7), calculate 7 – (-6)

(+7)	0111
(–6)	1010

(+7)	0111
–(–6)	0110 (Negated)
(–3)	1101 **"Overflow"**

Two’s Complement – Summary

- Addition
  - Add the values, discarding any carry-out bit

- Subtraction
  - Negate the subtrahend and add, discarding any carry-out bit

- Overflow
  - Adding two positive numbers produces a negative result
  - Adding two negative numbers produces a positive result
  - Adding operands of unlike signs never produces an overflow
  - **Note** - discarding the carry out of the most significant bit during Two's Complement addition is a normal occurrence, and does not by itself indicate overflow

Two’s Complement – Multiplication and Division

- Cannot be accomplished using the standard technique

- Example: consider $X * (-Y)$

  - Two’s complement of $-Y$ is $2^n - Y$ ➔ $X * (-Y) = X * (2^n - Y) = 2^n X - XY$

  - Expected result should be $2^{2n} - XY$

Signed multiplication

- Booth's multiplication algorithm
- Let **m** and **r** be the multiplicand and multiplier, respectively; and let x and y represent the number of bits in **m** and **r**.
- Determine the values of A and S, and the initial value of P. All of these numbers should have a length equal to $(x + y + 1)$.
  - A: Fill the most significant (leftmost) bits with the value of **m**. Fill the remaining $(y + 1)$ bits with zeros.
  - S: Fill the most significant bits with the value of (−**m**) in two's complement notation. Fill the remaining $(y + 1)$ bits with zeros.
  - P: Fill the most significant x bits with zeros. To the right of this, append the value of **r**. Fill the least significant (rightmost) bit with a zero.
- Determine the two least significant (rightmost) bits of P.
  - If they are 01, find the value of $P + A$. Ignore any overflow.
  - If they are 10, find the value of $P + S$. Ignore any overflow.
  - If they are 00, do nothing. Use P directly in the next step.
  - If they are 11, do nothing. Use P directly in the next step.
- Arithmetically shift the value obtained in the 2nd step by a single place to the right. Let P now equal this new value.
- Repeat steps 2 and 3 until they have been done y times.
- Drop the least significant (rightmost) bit from P. This is the product of **m** and **r**.

Booth's multiplication example

- Find 3 × (−4), with **m** = 3 and **r** = −4, and x = 4 and y = 4:
- m = 0011, -m = 1101, r = 1100
- A = 0011 0000 0
- S = 1101 0000 0
- P = 0000 1100 0
- Perform the loop four times:
  - P = 0000 110**0 0**. The last two bits are 00.
    - P = 0000 0110 0. Arithmetic right shift.
  - P = 0000 011**0 0**. The last two bits are 00.
    - P = 0000 0011 0. Arithmetic right shift.
  - P = 0000 001**1 0**. The last two bits are 10.
    - P = 1101 0011 0. P = P + S.
    - P = 1110 1001 1. Arithmetic right shift.
  - P = 1110 100**1 1**. The last two bits are 11.
    - P = 1111 0100 1. Arithmetic right shift.
- The product is 1111 0100, which is −12.

https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm

Two’s Complement – Multiplication and Division

- Can perform multiplication and division by converting the two's complement numbers to their absolute values and then negate the result if the signs of the operands are different

- Most architectures implement more sophisticated algorithms (Booth's multiplication algorithm, Wallace tree, Dadda multiplier)