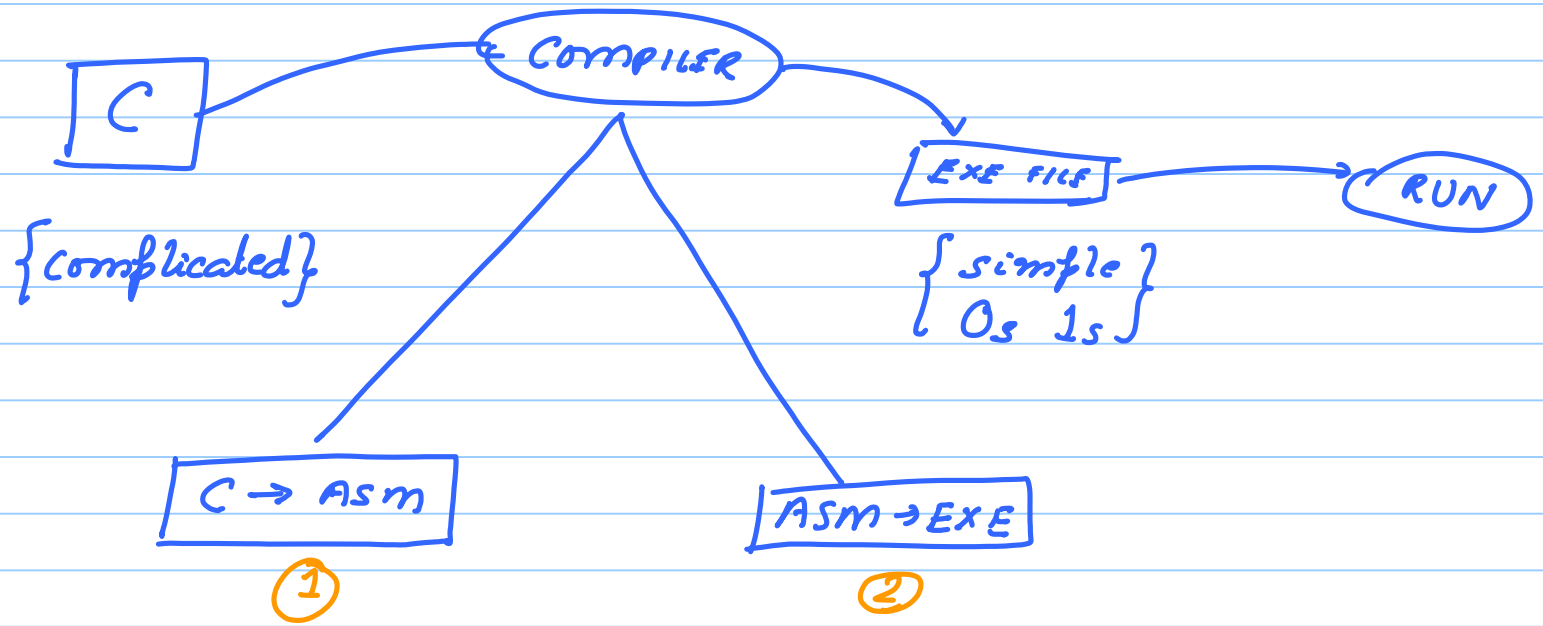
Aug 3rd

Note Title 03-08-2012

- Assembly Language



C

```
int a, b, c;
c = a + b;
```

Assembly (ARM)

13 variables

R_0 . . . . . R_{12}

Register → int (4 bytes)

1) Register assignment

$a \longleftrightarrow R_0$

$b \longleftrightarrow R_1$

$c \longleftrightarrow R_2$

```
ADD R2, R0, R1
```

ADD: opcode, R_2: dest, R_0: src1, R_1: src2

$C = a - b$

```
SUB R2, R0, R1
```

(c) (a) (b)

$c \leftarrow a - b$

Logical operations: { Bitwise and, Bitwise or, Bitwise XOR }

```
 1011
&0101
 0001
```

Opcode (and =):
- and: AND
- or: ORR
- xor: EOR

AND R_3, R_2, R_1

$R_3 = R_2 \& R_1$

Shift: C

`<<` Left shift `>>` Right Shift

1011 << 2 → 101100

1011 >>2 → 10 (= 2)

Binary rep: 1011

$2^3 \times 1 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 1 = 11$

$x << k$

$x \times 2^k$

$11/4$

Q	R
2	3
⟵	x

Left shift → Multiplication by a power of 2

right shift → division " " " " "

$11 \rightarrow 8 + 0 + 2 + 1$

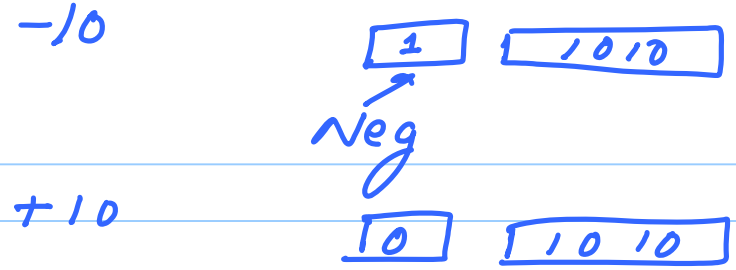
$(1\ 0\ 1\ 1)_{binary}$

Negative numbers in binary:

$-10 \rightarrow (1010)_b = 10$

$(0101)_3 = 5$

Simple soln: [sign bit] [number]



\+ simple
— difficult to work on them

Better soln (biased) : 8 bit number system

0 – 255

assume : 128 represents 0

[-128 to +127]

10 ⇝ 138

-10 ⇝ 118

rep → + simple
— hard to perform ops

Best soln:
(2's complement)

negative number: $(-x)$

$2^n - x$

4 bit number system.

C → unsigned (0 → 15)
→ signed (−8 ... +7)

0	→	0 0 0 0
1	→	0 0 0 1
:		
7	→	0 1 1 1

MSB = 0

16

−8 .... +7

−1 :	1 1 1 1	$(2^4 - 1 = 15)$
−2 :	1 1 1 0	
.		
−7 :	1 0 0 1	(+9)
−8 :	1 0 0 0	(+8)

MSB = 1

Quick method of getting a 2s complement

n-bit

$x > 0$

compute $-x$

slow : $2^n - x$

fast : $2^n - x = [(2^n - 1) - x] + 1$

$2^n - 1 \rightarrow \overbrace{1\ 1\ \cdots\ 1}^{n}$

$n = 4$

$x \rightarrow 5$

$(2^n - 1) - x \rightarrow$

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

every bit flipped

$(+11 \rightarrow -5)$

- flip every bit
- add 1

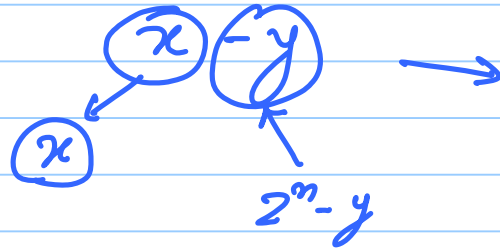
$3 \rightarrow 0011$

$-3 \rightarrow 1100 + 1 = 1101$

$-3 \rightarrow 1101$ → flip → $0010 + 1 = 0011 \rightarrow 3$

$x, -y \quad (x, y > 0)$

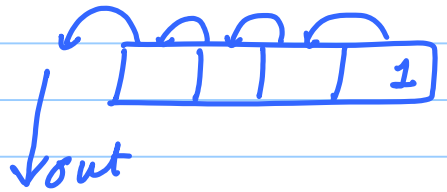
$16 \rightarrow 10000$



$\rightarrow 2^n + (x - y)$

$\downarrow$

$x - y$



$(-x) \times (-y) \quad (x, y > 0)$

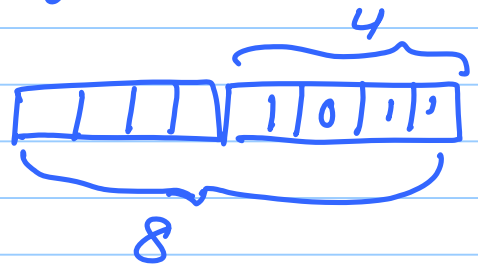
$(2^n - x) \times (2^n - y)$

$= [2^{2n} - 2^n \cdot y - x \times 2^n] + xy$

$= xy$

🚩 Try out different 2s complement operations by yourself

→ get a feel for it



(-5) → 1011

(+)ve : 7 → 0 111 } fill up the new positions with zeros

0 000 0 111

(-)ve : -5 → 1 0 11

$-x \ (x>0) \rightarrow 2^n - x$

$n \rightarrow m \quad (m > n)$

$2^m - x$?

$$2^m - x = 2^m - 2^n + (2^n - x)$$

$$= (2^m - 1) - (2^n - 1) + (2^n - x)$$

m

1 1 1 1 1

1 1 1 1

n

1 1 1 1 0 . . . . .

$(m-n)$

-5 : 1011

4

rew 1 | orig

-5 : 1111 | 1011

8

0 | orig

7 : 0000 | 0111

8

Sign extension shortcuts

Problem: convert a n-bit binary num
to a m-bit binary num
(m > n)

Soln:

(a) write the orig num

(b) for the new (m-n) positions
— fill in the msb of the
original number

Shift: << (multiplication by 2^n)

-5 << 2

1011 << 2
= 1100 (-4)

1011 << 1
= 0110 (+6)

Conc: Left-shift does not make a lot of sense in a 2s complement number system

(Logical)
LSR :

-5 : 1011 >> 2
= 0010 (2)

Right shift does not make sense

5 : 0101 >> 2
= 0001
5/4 = 1

(Arithmetic)
ASR

-5 : 1011 >>> 2
= 1110
(-2)
-5/4 = (4x -2) + 3

5 : 0101 >>> 2
= 0001
(1)
5/4 = 1x4 + 1

Logical right shift

1) fill the new bits with 0

2) (division) works for (+)ve not for (-)ve

Arithmetic right shift

1) fill the new bits with the MSB

2) performs division correctly for both (+)ve and (-)ve nums