Computer Organization & Architecture

Moazzam Ali Sahi

Computers and Arithmetic operations

- How do computers represent numbers?
  - How about negative numbers?
- How do computers add? subtract? multiply? etc…
  - Hardware or software?
  - What happens if the resulting number is bigger than the space, we have for it?
- How about fractions?

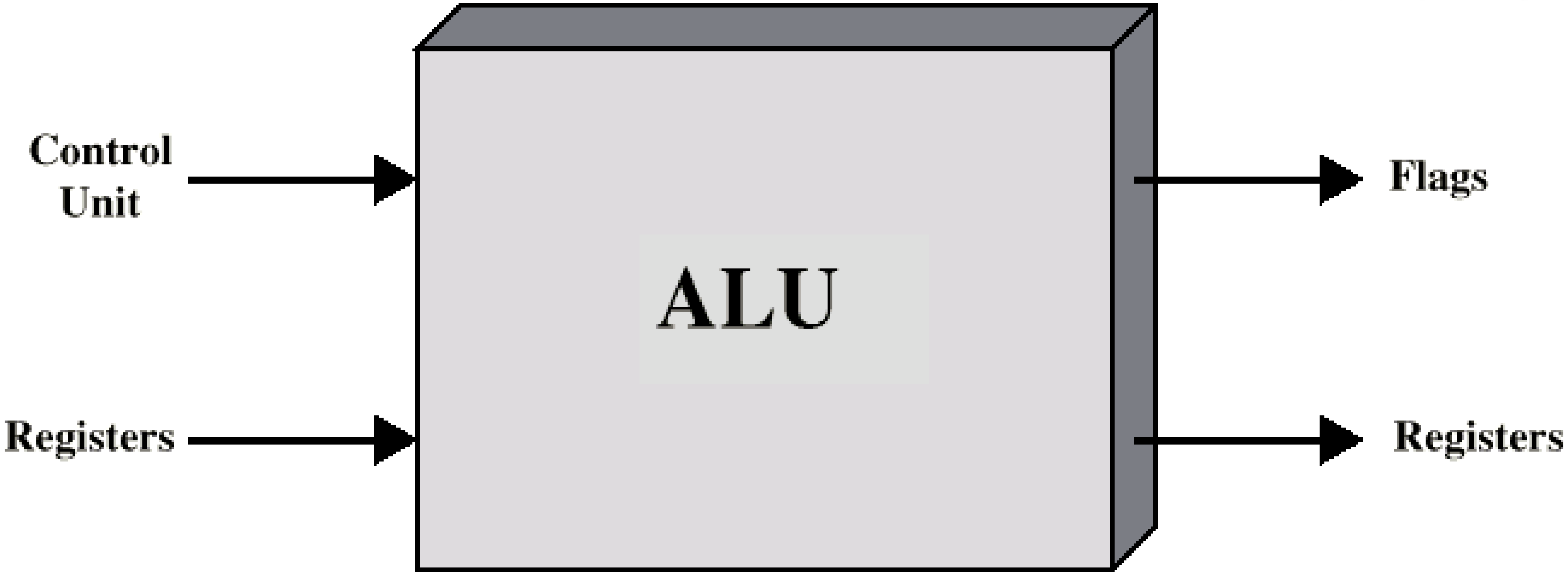
Chapter objectives

In this lecture we will focus on the representation of numbers and techniques for implementing arithmetic operations. Processors typically support two types of arithmetic: *integer* (or *fixed point)*, and *floating point*. For both cases, we first examine the representation of numbers and then discusses arithmetic operations.

Arithmetic & Logic Unit

- Does the calculations
- Everything else in the computer is there to service this unit
- Handles integers
- May handle floating point (real) numbers
- May be separate (math co-processor)

ALU Inputs and Outputs



Review: Decimal Numbers

- Integer Representation
  - number is sum of DIGIT * “place value”

Range
0 to $10^n - 1$

d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0
0	0	0	0	3	7	9	2
10^7	10^6	10^5	10^4	10^3	10^2	10^1	10^0

$$\begin{aligned} 3792_{10} &= 3 \times 10^3 + 7 \times 10^2 + 9 \times 10^1 + 2 \times 10^0 \\ &= 3000 + 700 + 90 + 2 \end{aligned}$$

Review: Decimal Numbers

Adding two decimal numbers

- add by "place value", one digit at a time

```
    3792
+    531
--------
     ???
```

```
     1
   3 7 9 2
 + 0 5 3 1
 ---------
 0 4 3 2 3
```

"carry 1" because 9+3 = 12

Binary Numbers

- Humans can naturally count up to 10 values, but computers can count only up to 2 values (0 and 1)
- (Unsigned) Binary Integer Representation "base" of place values is 2, not 10

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
0	1	1	0	0	1	0	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

$$
\begin{aligned}
01100100_2 &= 2^6 + 2^5 + 2^2 \\
&= 64 + 32 + 4 \\
&= 100_{10}
\end{aligned}
$$

Range

0 to $2^n - 1$

Binary Numbers

If a number is represented in n = 8-bits

Value in Binary:

a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0

Value in Decimal:

$$2^7.a_7 + 2^6.a_6 + 2^5.a_5 + 2^4.a_4 + 2^3.a_3 + 2^2.a_2 + 2^1.a_1 + 2^0.a_0$$

Value in Binary:

a_{n-1}	a_{n-2}	…	a_4	a_3	a_2	a_1	a_0

Value in Decimal:

$$2^{n-1}.a_{n-1} + 2^{n-2}.a_{n-2} + \ldots + 2^4.a_4 + 2^3.a_3 + 2^2.a_2 + 2^1.a_1 + 2^0.a_0$$

Binary Arithmetic

carry-in bits 1 1 1 0 0
A bits 1 1 1 0
B bits + 0 1 1 1
sum bits 0 1 0 1
carry-out bits 1 1 1 1 0

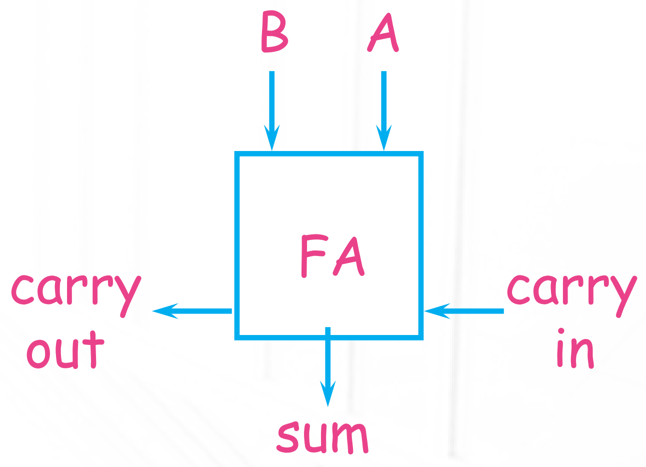
Add up to 3 bits at a time per place value

- A and B
- "carry in"

Output 2 bits at a time

- sum bit for that place value
- "carry out" bit (becomes carry-in of next bit)

Can be done using a function with 3 inputs, 2 outputs



Integer Representation

- Only have 0 & 1 to represent everything
- Positive numbers stored in binary
  - e.g. 41=00101001
- No minus sign
- No period
- Sign-Magnitude
- Two's complement

Sign-Magnitude

- Left most bit is sign bit
- 0 means positive
- 1 means negative
- +18 = 00010010
- -18 = 10010010

Problems:

- Need to consider both sign and magnitude in arithmetic
- Two representations of zero (+0 and -0)

Two’s Complement

- +3 = 00000011
- +2 = 00000010
- +1 = 00000001
- +0 = 00000000

-3 = 11111101
-2 = 11111110
-1 = 11111111
-0 = 00000000

Two's Complement

If a number is represented in n = 8-bits

Value in Binary:

a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0

Value in Decimal:

$$2^7.a_7 + 2^6.a_6 + 2^5.a_5 + 2^4.a_4 + 2^3.a_3 + 2^2.a_2 + 2^1.a_1 + 2^0.a_0$$

- +3 = 00000011
- +2 = 00000010
- +1 = 00000001
- +0 = 00000000

-3 = 11111101

-2 = 11111110

-1 = 11111111

-0 = 00000000

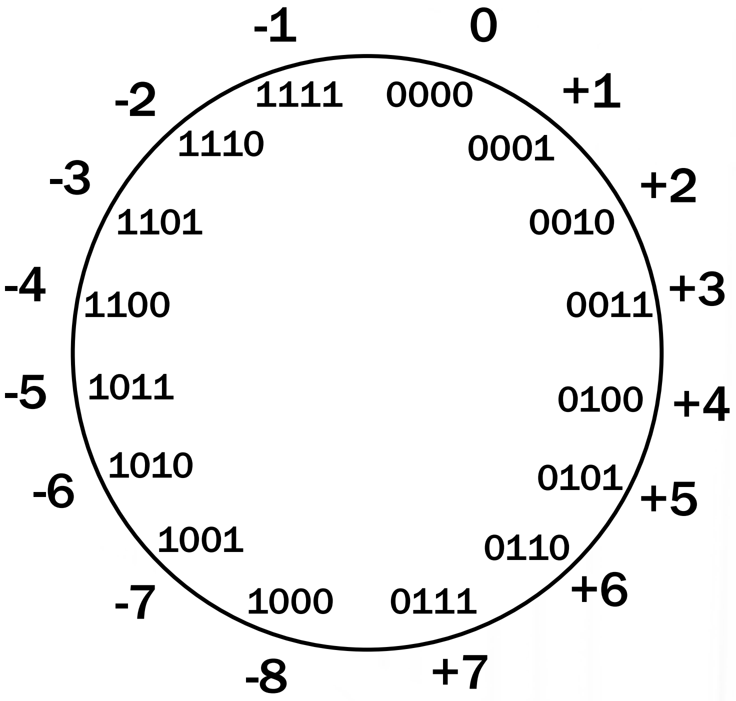
Benefits

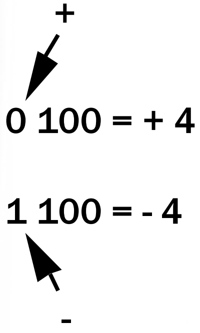
- One representation of zero
- Arithmetic works easily (see later)
- Negating is fairly easy
  - 3 = 00000011
  - Boolean complement gives 11111100
  - Add 1 to LSB 11111101

2's complement

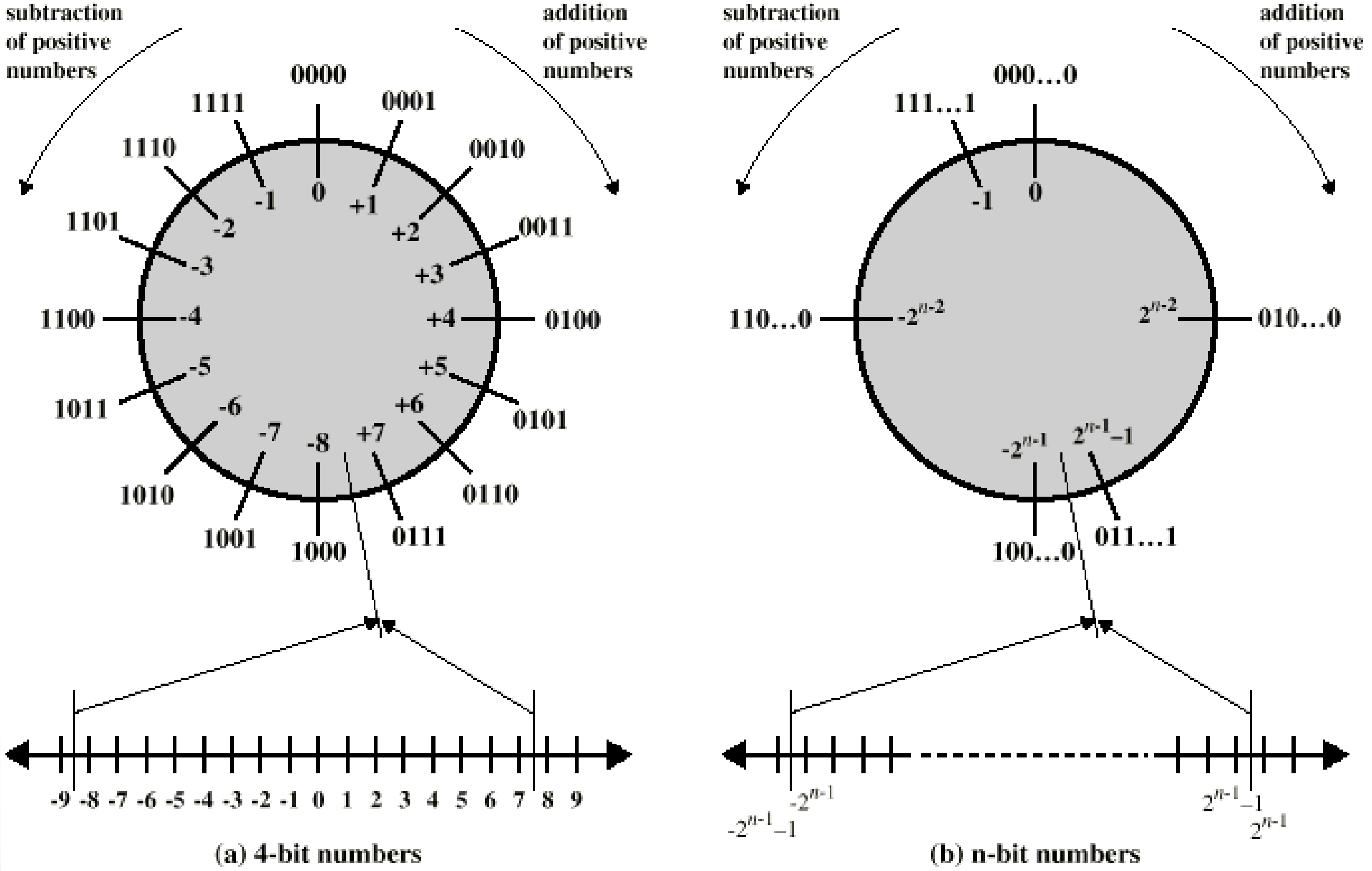
Only one representation for 0

One more negative number than positive numbers





Geometric Depiction of Two's Complement Integers



Negation Special Case 1

- 0 = 00000000
- Bitwise NOT 11111111
- Add 1 to LSB +1
- Result 1 00000000
- Overflow is ignored, so:
- $-0 = 0$ √

Negation Special Case 2

- -128 = 10000000
- bitwise NOT 01111111
- Add 1 to LSB +1
- Result 10000000
- So:
- -(-128) = -128 X
- Monitor MSB (sign bit)
- It should change during negation

Range of Numbers

- 8 bit 2's complement
  - +127 = 01111111 = 2^7 -1
  - -128 = 10000000 = -2^7
- 16 bit 2's complement
  - +32767 = 011111111 11111111 = 2^{15} - 1
  - -32768 = 100000000 00000000 = -2^{15}

Conversion Between Lengths

- Positive number pack with leading zeros
- +18 = 00010010
- +18 = 00000000 00010010
- Negative numbers pack with leading ones
- -18 = 10010010
- -18 = 11111111 10010010
- i.e. pack with MSB (sign bit)

Addition and Subtraction

- Normal binary addition
- Monitor sign bit for overflow
- Take two's complement of subtrahend and add to minuend
  - i.e. $a - b = a + (-b)$
- So we only need addition and complement circuits

Binary Subtraction

- 2's complement subtraction: add negative

5 - 3 = 2

0101

0011 flip
1100 +1
1101

-3 in 2's complement form

```
  0 1 0 1
 +1 1 0 1
 ---------
 1 0 0 1 0
```

ignore overflow

→ 2

3 - 5 = -2

0011

0101 flip
1010 +1
1011

-5 in 2's complement form

```
  0 0 1 1
 +1 0 1 1
 ---------
  1 1 1 0
```

→ -2

0001 ← flip
0010 ← +1

(flip+1 also gives positive of negative number)