EE345

Number Representation

Binary Integers

- Also known as “base 2”
- Each place is a successive power of 2
  - With either ‘0’ or ‘1’ in that place
- E.g., $01101_{2\,=}\ 13_{10}$
  - $1*2^3 + 1*2^2 + 0*2^1 + 1*2^1$

Other Radices

- Can use any positive integer > 1 for base
  - "base" = "radix"
  - In fact, can use negative radix
- Very common to use radix = 16
  - Hexadecimal
  - Convert binary->hex: group each 4 bits
- E.g., 0001_1010 = $1A_{16} = 26_{10}$
  - A – F: digits with values 10-15
  - $1*16^4 + 10*16^0$

Two’s Complement

- One way to represent negative values in binary
- Some nice properties
  - Addition of 2’s complement values just works
    - Positive + negative yields correct result
  - Only *one* representation of 0: 00…00
    - Some other systems: positive and negative zero
  - MSBit == 1 means “value is negative”
- Not-so-nice property: not symmetric around 0
  - N bits: 2^{N-1} encodings < 0, but $2^{N-1}-1$ encodings > 0
  - E.g., $-4_{10} = 100_2$: can’t represent +4 in three-bit 2’s comp

Negation

- Negating a 2’s complement value
  - Invert each bit
  - Add 1
  - E.g., -(0100) yields 1100
    - Check: 0100 + 1100 == 0 (correct!)

Shifts

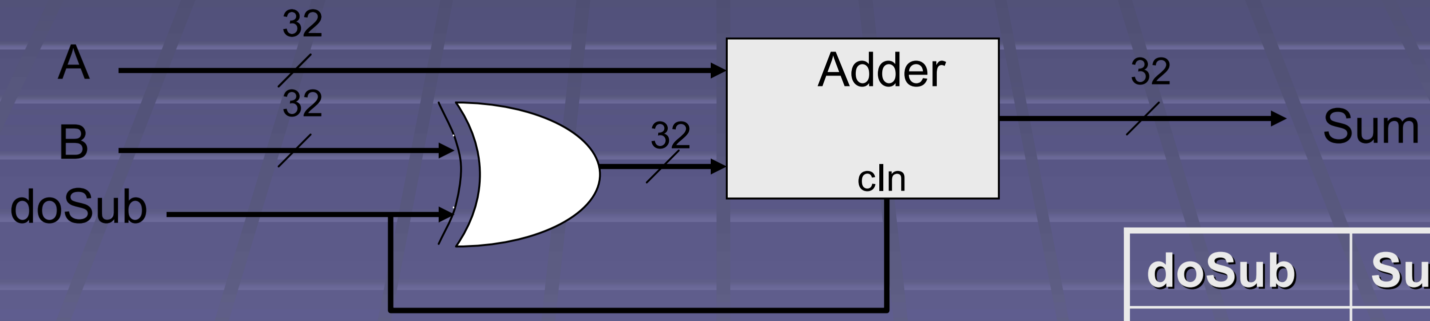
- Left shift
  - Same result as multiply by power of two
  - “<<“ is left shift operator in C and Verilog
  - E.g., 0011 << 1 (shift left by 1 digit)
  - 0011 << 1 yields 0110 ($3 * 2^1 == 6$)
- Right shift
  - Same result as division by power of two
  - “>>” is right shift operator
  - Need to replicate the MSB for “signed” shift!
  - E.g., 1100 >> 1 yields 1110 ($-4 / 2^1 == -2$)

Sign Extension

- When increasing number of bits
  - E.g., converting 8-bit 2’s complement to 16-bit
- Replicate MSB from narrow value into all upper bits of wider value
- E.g., convert 4-bit to 8-bit…
  - 0011 becomes 0000_0011
  - 1010 becomes 1111_1010

Subtract

- CPU has an adder
  - We would prefer *not* to also have a subtractor

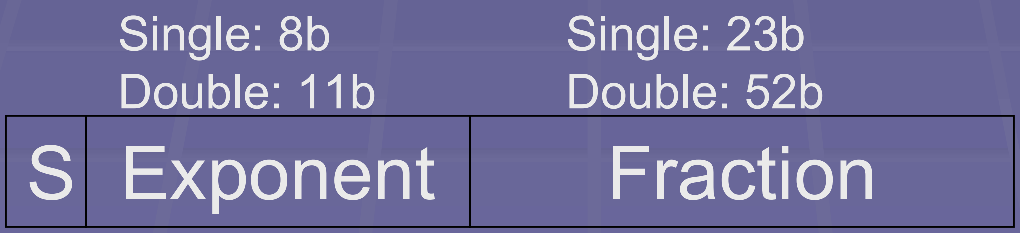


doSub	Sum
0	A + B
1	A + -B

- Standard Solution
  - To subtract: add two's complement of 2nd operand
  - If doSub: (A + ~B + 1)
    - ~B + 1 = two's complement of B

Floating Point

- We won't use FP in EE345
  - But you should know a few basics
- Everyone now uses IEEE 754 for FP
  - Formerly, there were many standards
- Represent as sign, fraction, exponent
  - Multiple bit-widths available: 16, 32, 64, 128
  - Most common: 32b (single) and 64b (double)



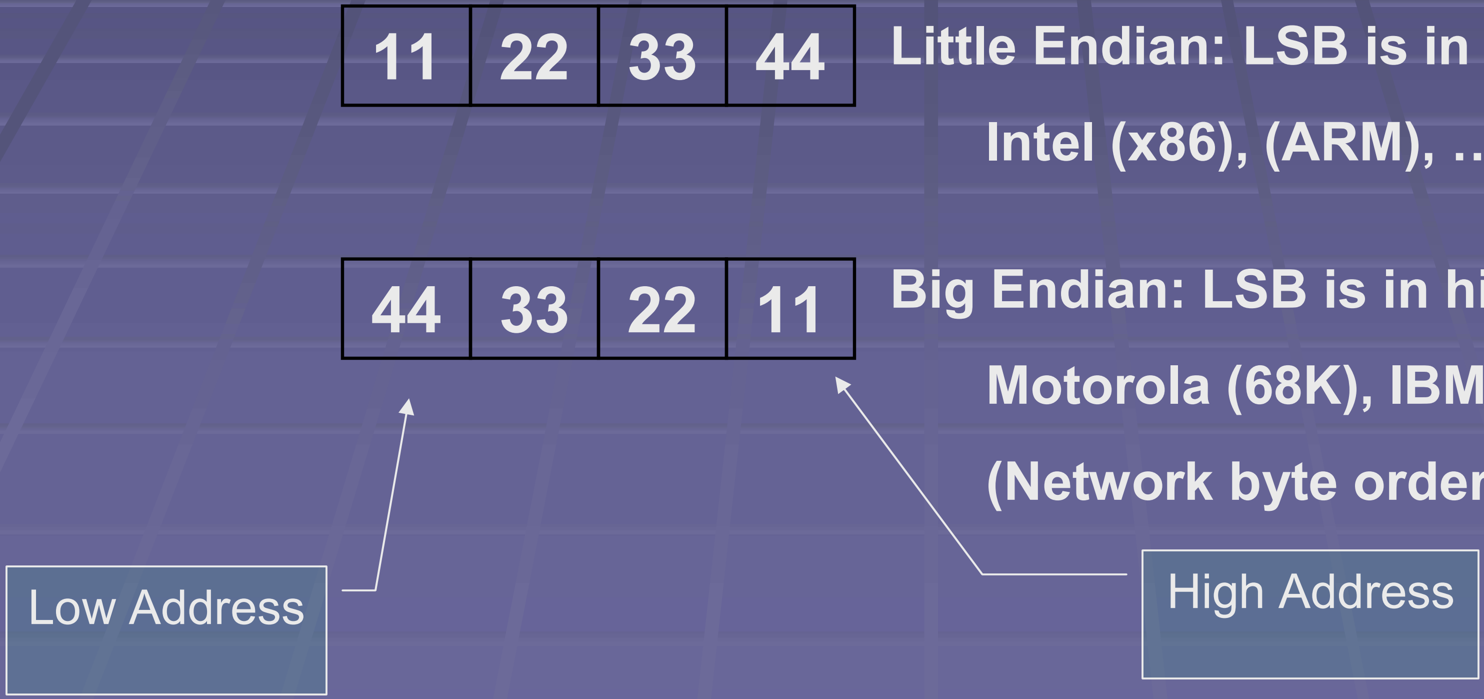
Byte Order

- Byte-addressible memory
  - Each memory address is for a single byte
  - E.g., 32b quantity in memory requires 4 bytes
    - So spans four addresses: address of byte 0, 1, 2, 3
  - Supported by most (all?) modern CPU's
- So, how to interpret the bytes of a value?
  - Need to assemble bytes into a single value
  - What order? I.e., which part of value comes first in memory?

Byte Order

- Example, value 32'h11223344 in memory
  - Two common ways to layout

We'll use little endian in EE345



Little Endian: LSB is in lower address

Intel (x86), (ARM), …

Big Endian: LSB is in higher address

Motorola (68K), IBM (Z series), …

(Network byte order)