Lecture 18 Binary Math

- A major use for digital circuits is to do math operations on binary numbers
- Often us numerical representation to describe set of bits

⇒ Need to understand how numbers are represented

Binary = base 2

Write 01001101

MSB
= most significant bit
= highest power of two

LSB
= least sig. bit
= lowest power of two

Note that MSB is not always on left, need to check convention!

Convert to decimal

⇒ $1 \times 2^0 = 1$
$+ 0 \times 2^1 = 0$
$+ 1 \times 2^2 = 4$
$+ 1 \times 2^3 = 8$
$+ 0 \times 2^4 = 0$
$+ 0 \times 2^5 = 0$
$+ 1 \times 2^6 = 64$
$+ 0 \times 2^7 = 0$

$= 77$ in decimal

I'll write 77_{dec}

Converting to binary:

$$56_{dec} = 32 + 16 + 8$$

$$= 00111000_{bin}$$

Also common to use hexadecimal = base 16

digits: 0, 1, 2, ..., 9, A, B, C, D, E, F

Since $16 = 2^4$, one hex digit = 4 binary digits

So convert in groups of 4:

$$\underbrace{0100}_{=4_{dec}=4_{hex}} \quad \underbrace{1101}_{=13_{dec}=D_{hex}} = 4D_{hex}$$

and $A3_{hex} = (10_{dec})(3_{dec})$

$$= 1010\ 0011$$

Notations: not always obvious whether number is hex or dec

$39 = 3\times10+9$ or $= 3\times16+9$?

Use 0x prefix: 0x 39

Or h/H suffix: 39H 39h

I'll use hex/dec subscript

4 bits to one digit is convenient

But people don't work in hex

Alternative: binary-coded decimal (BCD)

Use set of 4 bits to encode one decimal digit

So 27_{dec} → (0010)(0111) = 00100111_{BCD}
2 7

Compare to standard binary 27_{dec} = 16+8+2+1

= 00011011_{bin}

BCD often convenient for base 10 computations

But wastes bits: some combinations aren't used

Impossible to distinguish BCD from binary,
& no standard notation

Need to check what type of encoding your application uses.

Negative numbers

No +/- signs in binary, need to encode somehow

Standard & best encoding is "wrap" values
from max positive to max negative

Called "2s complement" encoding

(vs "unsigned")

Example: 4 bit, 16 values

	unsigned	2s complement
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1

Benefit: math operations don't depend on encoding

$0010 + 1011 = 1101$

either $(2) + (11) = (13)$

or $(2) + (-5) = (-3)$ valid either way

Of course, $6+7$ has trouble, $6+7 = -3$?

but some problem for $12+13$ in unsigned...

Set errors when you exceed limits of representation

Converting signed values

Binary → decimal:

Binary value is negative iff first digit is 1
(need to know total number of digits in use)

To convert:

1. Invert all bits (0 ↔ 1)
2. Add one
3. Convert to decimal
4. Add minus sign

So $1011_{2s} \rightarrow 0100 \rightarrow 0101 \rightarrow 5 \rightarrow -5_{dec}$

Decimal → binary:

1. Add (negative) value to 2^N using N bits
2. Convert result to binary as if unsigned

∴ So with N=4, $-5_{dec} \rightarrow 16-5 = 11 \rightarrow 1011_{2s}$

Bigger example, N = 8

$-120_{dec} \rightarrow 256-120 = 136$
$= 128+8$
$= 10001000_{2s}$

Or $10001000_{2s} \rightarrow 01110111+1 = 01111000$
$= 64+32+16+8 = 120$
$\rightarrow -120_{dec}$