UM EECS 270 F22
Introduction to Logic Design

6. Binary Representation of Positive Numbers

Representation of Positive Numbers

- So far we've been looking at binary signals, yet most concepts in the real world are represented as base-10 (decimal) numbers. How can we represent these concepts on a computer? First let's fully understand our "normal" number system…
- We use a **positional number system**; one where a number is represented by a string of digits and each digit position has an associated weight
  - Example: 836.24 = 8*100 + 3*10 + 6*1 + 2*.1 + 4*.01
    $= 8*10^2 + 3*10^1 + 6*10^0 + 2*10^{-1} + 4*10^{-2}$

*10 is called the **radix** or **base***

*The "decimal point" is more generally referred to as the **radix point***

What is an example of a non-positional number system?

- Example: In radix 3 with digits {0, 1, 2}:

$$1021_3 = 1*3^3 + 0*3^2 + 2*3^1 + 1*3^0 = 34_{10}$$

- In general, for any radix *r*, the value of a number written

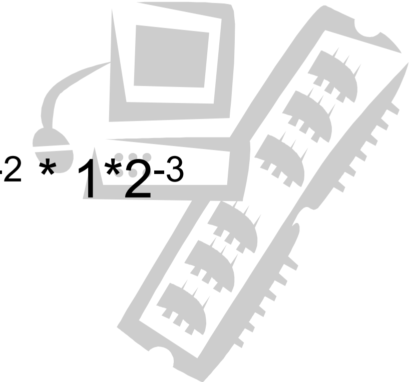
$$d_{p-1}d_{p-2}\ldots d_0.d_{-1}d_{-2}\ldots d_{-n}$$

can be expressed as the weighted sum:

$$\sum_{i=-n}^{p-1} d_i \cdot r^i$$

where d_i is a member of a set of digits

- In general, the lower the radix, the more digits are needed to represent a number
- Useful radices:
  - 10/decimal (10 fingers!)
  - 2/binary (0/1 – high/low voltage)
  - 8/octal and 16/hexadecimal (compact representation of binary)



- **Binary** (base-2): digits {0, 1}

$$1011.101_2 = 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 + 1*2^{-1} + 0*2^{-2} * 1*2^{-3}$$
$$= 8 + 2 + 1 + .5 + .125 = 11.625_{10}$$

- Large numbers require lots of digits!

$$27825_{10} = 110110010110001_2$$

- Often convenient to use octal or hexadecimal as a shorthand for binary

- **Octal** (base-8): digits {0, 1, 2, 3, 4, 5, 6, 7}

$$321_8 = 3*8^2 + 2*8^1 + 1*8^0 = 209_{10}$$

- Because 8 is a power of 2 (2^3), conversion from binary to octal and vice-versa is easy! Each octal digit represents exactly 3 binary digits.

Octal to binary: $6 \quad 2 \quad 1_8 =$

$110\ 010\ 001_2$

Make sure to pad with 0s when necessary!

Binary to octal: $001\ 011.110\ 1\mathbf{00}_2 =$

$1 \quad 3\ .\ 6 \quad 4_8$

- Hexadecimal (base-16):
  digits {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
  (can't borrow digits for 10-15 from base-10, so use A-F)

$$8B7_{16} = 8*16^2 + 11*16^1 + 7*16^0 = 2231_{10}$$

- Because 16 is a power of 2 (2^4), each hexadecimal digit represents 4 binary digits

  Hexadecimal to binary: A 8$_{16}$ =
  $1010\ 1000_2$

  Binary to hexadecimal: **0**110 1110$_2$ =
  6 E$_{16}$

- Often hex numbers are denoted with the prefix "0x" instead of a subscript 16 (0x3B8 ≡ $3B8_{16}$)
- How to convert from hex to octal and octal to hex??
  - Go through binary!
- Why hex? $27825_{10} = 110110010110001_2 = 6CB1_{16}$

- So far we know the following base conversions:
  - binary, octal, hex → decimal
  - binary ↔ octal ↔ hex
- What about decimal → binary, octal, or hex?

Use repeated division by radix:

$178_{10} = ?_2$

2 | 178
2 | 89 R 0 LSB
2 | 44 R 1
2 | 22 R 0
2 | 11 R 0
2 | 5 R 1
2 | 2 R 1
2 | 1 R 0
0 R 1 MSB

$178_{10} = 10110010_2$

$178_{10} = ?_8$

8 | 178
8 | 22 R 2 LSB
8 | 2 R 6
0 R 2 MSB

$178_{10} = 262_8$

$178_{10} = ?_{16}$

16 | 178
16 | 11 R 2 LSB
0 R 11 (B) MSB

$178_{10} = \text{B2}_{16}$

Why does repeated division work?

$$N_r = (d_3 * r^3 + d_2 * r^2 + d_1 * r^1 + d_0 r^0)_{10}$$

(remember that $d_i < r$)

N_r	$d_3 * r^3 + d_2 * r^2 + d_1 * r^1 + d_0 r^0$	
Divide by *r*:	$d_3 * r^2 + d_2 * r^1 + d_1 r^0$	remainder d_0
Divide again by *r*:	$d_3 * r^1 + d_2 r^0$	remainder d_1
Divide again by *r*:	$d_3 r^0$	remainder d_2
Divide again by *r*:	0	remainder d_3

Fraction conversion: use repeated *multiplication* by radix

$.59375_{10} = ?_2$

$$\begin{array}{lclc}
 & & .59375 & * 2 \\
\text{MSB} & 1 & .\ 1875 & * 2 \\
\downarrow & 0 & .\ 375 & * 2 \\
\downarrow & 0 & .\ 75 & * 2 \\
\downarrow & 1 & .\ 5 & * 2 \\
\text{LSB} & 1 & .\ 0 &
\end{array}$$

$.59375_{10} = .10011_2$

Watch for repeating digits…

$.7_{10} = ?_2$

$$\begin{array}{clc}
 & .\ 7 & * 2 \\
1 & .\ 4 & * 2 \\
0 & .\ 8 & * 2 \\
1 & .\ 6 & * 2 \\
1 & .\ 2 & * 2 \\
0 & .\ 4 & * 2
\end{array}$$

$.7_{10} = .1\overline{0110}_2$

Sometimes conversion won't be exact…

$.73_{10} = ?_2$

$$\begin{array}{lclc}
 & & .\ 73 & * 2 \\
\text{MSB} & 1 & .\ 46 & * 2 \\
\downarrow & 0 & .\ 92 & * 2 \\
\downarrow & 1 & .\ 84 & * 2 \\
\downarrow & 1 & .\ 68 & * 2 \\
\downarrow & 1 & .\ 36 & * 2 \\
\downarrow & 0 & .\ 72 & * 2 \\
\text{LSB} & 1 & .\ 44 & * 2
\end{array}$$

$.73_{10} \approx .1011101_2$

In Class Exercise

$101.45_{10} = ?_2, ?_{16}$

In Class Exercise

$$101.45_{10} = ?_2, ?_{16}$$

$101_{10} = ?_2$

$$\begin{array}{r|l}
2 & 101 \\
2 & 50 \text{ R } 1 \quad \text{LSB} \\
2 & 25 \text{ R } 0 \\
2 & 12 \text{ R } 1 \\
2 & 6 \text{ R } 0 \\
2 & 3 \text{ R } 0 \\
2 & 1 \text{ R } 1 \\
 & 0 \text{ R } 1 \quad \text{MSB}
\end{array}$$

$101_{10} = 1100101_2$

$.45_{10} = ?_2$

$$\begin{array}{l}
\ .45 * 2 \\
0\ .\ 9 * 2 \\
1\ .\ 8 * 2 \\
1\ .\ 6 * 2 \\
1\ .\ 2 * 2 \\
0\ .\ 4 * 2 \\
0\ .\ 8 * 2
\end{array}$$

$.45_{10} = .01\overline{1100}_2$

$101.45_{10} = 1100101.01\overline{1100}_2$

$0110\ 0101.0111\ \overline{0011}_2$

$= 6 \quad 5\ .\ 7 \quad 3_{16}$

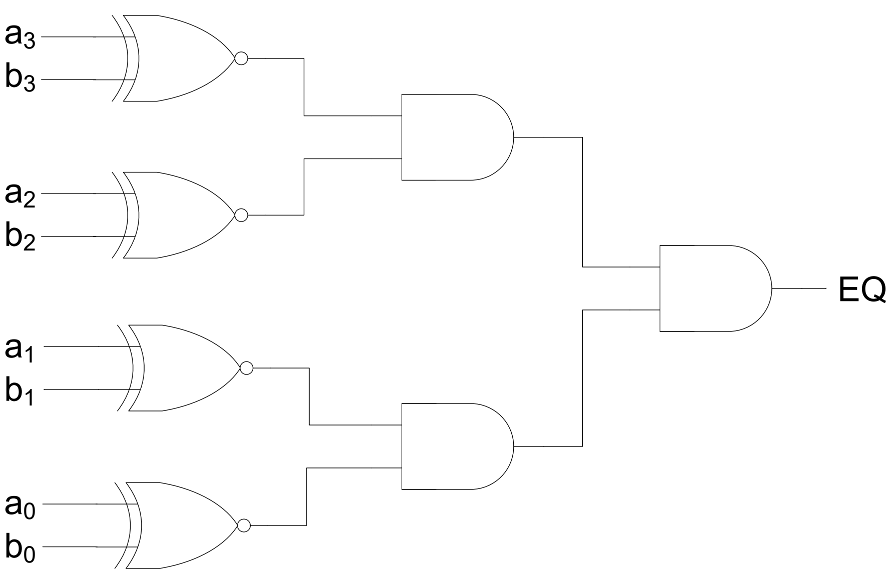
$= 65.7\overline{3}_{16}$

Equality Comparator

- Want a signal, EQ, that is 1 iff two 4-bit numbers, A and B, are equal. Under what logic conditions are A and B equal?
  – Equal if $a_3 = b_3$ AND $a_2 = b_2$ AND $a_1 = b_1$ AND $a_0 = b_0$
- Which gate performs an equality comparison?
  – XNOR!

A	B	$A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

$$EQ = (a_3 \odot b_3) \cdot (a_2 \odot b_2) \cdot (a_1 \odot b_1) \cdot (a_0 \odot b_0)$$



How many gates are required to implement an *n*-bit "parallel" comparator?

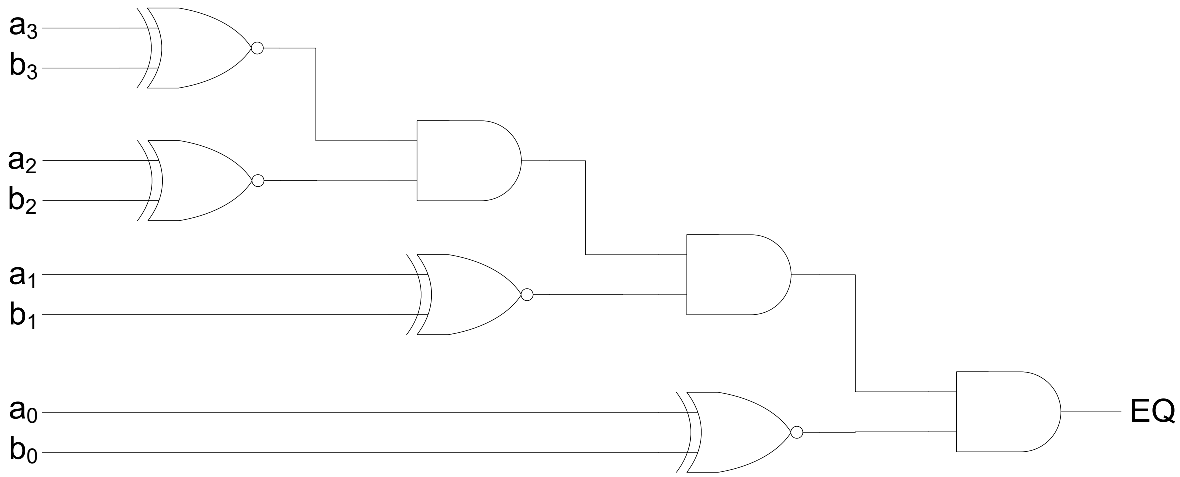
Assume $n = 2^k$

$\#\text{gates} = 2^k + 2^{k-1} + 2^{k-2} + \dots 2^0$

$= 2^{k+1} - 1$

$= 2n - 1$

A “Series” Comparator Implementation



How many gates are required to implement
an *n*-bit “series” comparator?

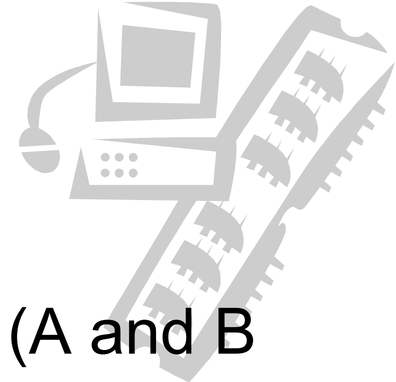
- Comparing first two bits: 3 gates

- Comparing each additional bit: 2 gates

- After comparing the first 2 bits, there are n-2 bits left to compare for a total of 2*(n-2) gates

-Total number of gates required: 2(n-2) + 3 = 2n – 1

- “Series” and “parallel” implementations require the same number of gates! Is there a reason to prefer one implementation over the other?



Magnitude Comparator

- Want to design a signal, AgrB, that is 1 iff A > B (A and B are 4-bit numbers). Under what logic conditions is A > B?
  - A > B if a_3 = 1 and b_3 = 0
  - A > B if a_2 = 1 and b_2 = 0 *and* $a_3 = b_3$
  - A > B if a_1 = 1 and b_1 = 0 and $a_2 = b_2$ and $a_3 = b_3$
  - A > B if a_0 = 1 and b_0 = 0 and $a_1 = b_1$ and $a_2 = b_2$ and $a_3 = b_3$

$$
\begin{aligned}
AgrB = a_3 \cdot \overline{b_3} &+ \\
(a_3 \odot b_3) \cdot a_2 \cdot \overline{b_2} &+ \\
(a_3 \odot b_3) \cdot (a_2 \odot b_2) \cdot a_1 \cdot \overline{b_1} &+ \\
(a_3 \odot b_3) \cdot (a_2 \odot b_2) \cdot (a_1 \odot b_1) \cdot a_0 \cdot \overline{b_0} &
\end{aligned}
$$