Lecture 25:
Other Number Systems

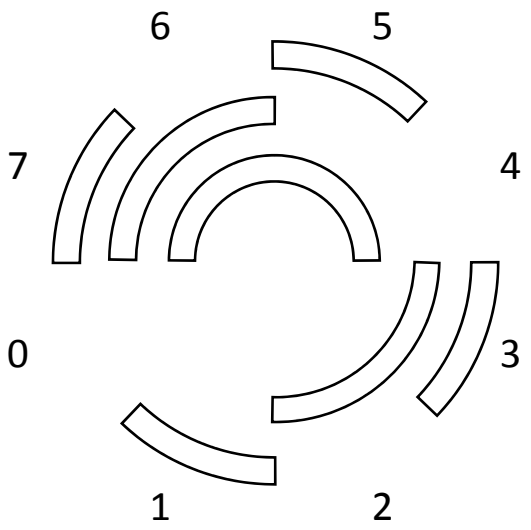
Today's Goals

- Learn about Gray Codes
- Use scaled binary numbers
- Learn the IEEE floating point number format

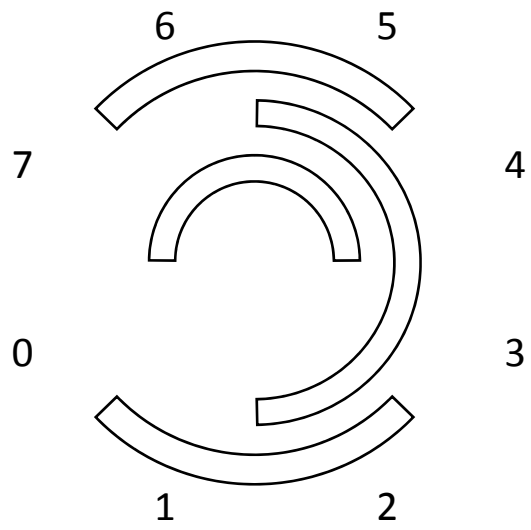
Gray Codes

Rotational Position Encoded with

Standard Binary



Gray Code



- Gray Code, also known as the reflected binary code, is a binary number system in which two adjacent values are different by only one bit.
- From Digital System 1, Karnaugh map (K-map) uses Gray Code to simplify Boolean algebra expressions.

Generating a Gray Code

- Start with a 1 bit code.
- Double the number of values by mirroring the current code, appending 0's to the beginning of the original copy and 1's to the beginning of the new copy.
- Repeat this process until enough values are available.
- Note:
  - You can create a slightly smaller Gray Code by deleting the same number of values from the top of the list as you do from the bottom.
  - A Gray Code must have an even number of values.

Scaled Binary

- When a program requires very large numbers, very small numbers, or number with fractions, programmers often turn to floating point numbers.

```
USAGDP  DC.W    1   ; USA's gross domestic product
                    ; in billions of dollars
CANGDP  DC.W    1   ; Canada's gross domestic product
                    ; in billions of dollars

WAVEON  DC.W    1   ; Waveform "on" time in 0.001's of a sec.
WAVEONF DC.W    1   ; Waveform "off" time in 0.001's of a sec.
```

- Note:
  - The resolution is determined by the scaling factor, which is the drawback to this approach.

Using Binary Fractions

- Floating point arithmetic takes much more time to perform than integer arithmetic, and it should be minimally used.
- A binary number can contain a fraction component if the radix point is moved to the left.
- In decimal, a radix point is commonly called the decimal point.
- Moving the radix point has the same effect in binary that it does in decimal, as illustrated with the examples below.
- Examples:
  - What decimal value does 000011.11_2 represent? 3.75_{10}
  - What decimal value does 11111.1111_{2c} represent? -0.0625_{10}
- Notes:
  - When performing arithmetic, all numbers must have the radix point in the same location.
  - The radix point is tracked by the programmer, not the processor.

Converting to a Binary Fraction

- The integer portion and fraction portion are converted separately.
  - The integer portion is converted as before.

- For the fraction,
  - 1. Multiply the **decimal fraction** by two.
  - 2. Record the digit to the left of the decimal point in the result as the next least significant digit.
  - Repeat (note: ignore the integer portion) the above two steps until the fraction is zero (terminating fraction), or the fraction is the same as from a previous step (repeating fraction).

- Examples:
  - Represent 6.375_{10} in 8-bit unsigned binary. Place the radix point where desired.
    - 00110.011_2
    - 0110.0110_2
    - 110.01100_2
  - Represent 0.4_{10} in 8-bit unsigned binary. Place the radix point where desired.
    - $.01100110_2$

Floating Point

- Floating point numbers were designed to be used when both very large and very small numbers needed to be used simultaneously.
- To do this, a floating point number includes the scaling factor that was assumed by the programmer when using scaled binary.
- A floating point number contains three values:
  - Sign bit – most floating point systems used today are sign and magnitude)
  - Exponent – scaling factors are limited to powers of 2
  - Fraction

IEEE Floating Point

- IEEE established a standard format, described briefly below.

- Except for 0, values are normalized. This means a number is scaled so that $1 \leq \text{fraction} < 2$.
- Since the normalized value is always 1.fraction, the leading 1 is assumed and not stored in the fraction.
- The value represented is $(-1)^S(2^{(E-127)})(1.\text{fraction})$
- An exponent of 255 is reserved for infinity and not-a-number.
- An exponent of 0 either means 0 (if fraction is 0) or the minimum exponent (if fraction is not 0).

Examples

$(-1)^{S}(2^{(E-127)})(1.fraction)$

```
0 00000000 00000000000000000000000 = 0
1 00000000 00000000000000000000000 = -0

0 11111111 00000000000000000000000 = Infinity
1 11111111 00000000000000000000000 = -Infinity

0 11111111 00000100000000000000000 = NaN
1 11111111 00100010001001010101010 = NaN

0 10000000 00000000000000000000000 = +1 * 2**(128-127) * 1.0 = 2
0 10000001 10100000000000000000000 = +1 * 2**(129-127) * 1.101 = 6.5
1 10000001 10100000000000000000000 = -1 * 2**(129-127) * 1.101 = -6.5

0 00000001 00000000000000000000000 = +1 * 2**(1-127) * 1.0 = 2**(-126)
0 00000000 10000000000000000000000 = +1 * 2**(-126) * 0.1 = 2**(-127)
0 00000000 00000000000000000000001 = +1 * 2**(-126) *
                                     0.00000000000000000000001 =
                                     2**(-149)  (Smallest positive value)
```

- More examples
  - $C11900000_{16}$ → -9.5625
  - $3CDC0000_{16}$ → 0.02685546875
- Calcuator: http://babbage.cs.qc.edu/IEEE-754/32bit.html

Questions?

Wrap-up

What we've learned

What to Come