Lecture 2 Slides

Number Conversion

Binary Arithmetic

Codes (Decimal Codes)

Review

- Base r => Base 10 Conversion
  - $11010011_2 = ?_{10} =$

- How many bits are required represent the decimal value 356?

Number System Review

- Base r number system:
  - r coefficients [0 – (r-1)]
  - Implicit place values are powers of r

___	___	___	___	___ .	___	___	___
r^4	r^3	r^2	r^1	r^0	r^{-1}	r^{-2}	r^{-3}

- Base r => Base 10 Method
- $X_r = (?)_{10} = \Sigma_i\, a_i * r^i$
  - Sum each coefficient times its place value (powers of r)
  - Ex1: $11010_2 = 16+8+2 = 26_{10}$
  - Ex2: $1A5_{16} = 1*256 + 10*16 + 5*1 = 421_{10}$

MORE NUMBER SYSTEMS

Conversion: Base 10 to Base r

- $X_{10} = (?)_r$
- General Method (base 10 to arbitrary base r)
  - Division Method for integer portion or number
  - Multiplication Method for fractional portion of number
  - Split number into integer and fractional portion and convert them separately, then combine results

Fraction

$$(45.375)_{10}$$

45 — Division Method

.375 — Multiplication Method

Division Method Explanation

$$45_{10} = \frac{a_4}{2^4} \quad \frac{a_3}{2^3} \quad \frac{a_2}{2^2} \quad \frac{a_1}{2^1} \quad \frac{a_0}{2^0} \; . \; 0$$

$$45_{10} = a_4 2^4 + a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0$$

$$\frac{45_{10}}{2} = \frac{a_4 2^4 + a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0}{2}$$

$$22.5_{10} = a_4 2^3 + a_3 2^2 + a_2 2^1 + a_1 2^0 + a_0 2^{-1}$$

Division Method

- Converts integer portion of a decimal number to base r
- Informal Algorithm
  - Repeatedly divide number by *r* until equal to 0
  - Remainders form coefficients of the number base r
  - Remainder from last division = MSD (most significant digit)

$193_{10} = (??)_5$

5 | 193		
5 | 38	rem. = 3	LSD
5 | 7	rem. = 3	
5 | 1	rem. = 2	
0	rem. = 1	MSD

Keep dividing until you reach 0

Remainders form the number in base r (order from bottom up)

$193_{10} = (1233)_5$

Division Method Example

$45_{10} = (??)_2$

$2 \,\vert\, 45$	22	rem. = 1 (LSB)
$2 \,\vert\, 22$	11	rem. = 0
$2 \,\vert\, 11$	5	rem. = 1
$2 \,\vert\, 5$	2	rem. = 1
$2 \,\vert\, 2$	1	rem. = 0
$2 \,\vert\, 1$	0	rem. = 1 (MSB)

Keep dividing until you reach 0

Remainders form the number in base r (order from bottom up)

$45_{10} = (101101)_2$

How number conversion works

$$45_{10} = \underset{2^4}{\underline{a_4}}\ \underset{2^3}{\underline{a_3}}\ \underset{2^2}{\underline{a_2}}\ \underset{2^1}{\underline{a_1}}\ \underset{2^0}{\underline{a_0}}$$

More bits may be required for this actual example, but we'll use 5 to illustrate…

$$45_{10} = a_4 2^4 + a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0$$

$$\frac{45_{10}}{2} = \frac{a_4 2^4 + a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0}{2} = \overbrace{a_4 2^3 + a_3 2^2 + a_2 2^1 + a_1 2^0}^{\text{Quotient}} + \overset{\text{Rem.}}{\frac{a_0}{2}}$$

$$\frac{a_4 2^3 + a_3 2^2 + a_2 2^1 + a_1 2^0}{2} = \overbrace{a_4 2^2 + a_3 2^1 + a_2 2^0}^{\text{Quotient}} + \overset{\text{Rem.}}{\frac{a_1}{2}}$$

- Each time we divide by *r*, another coefficient "falls out" and all the other place values are reduced by a factor of *r*.

How number conversion works

$$\frac{D_{10}}{r} = \frac{\overbrace{\left(a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + \cdots + a_1 r^1 + a_0\right)}^{D_{10}\text{ written as coefficients * place values}}}{r} = \overbrace{\cancel{\frac{r}{r}}\left(a_{n-1}r^{n-2} + \cdots + a_2 r^1 + a_1\right)}^{\text{Factor r out of numerator terms with } a_{n-1} - a_1} + \frac{a_0}{r}$$

a_0 is the remainder

- Each time we divide by *r*, another coefficient "falls out" and all the other place values are reduced by a factor of *r*.

Fractional Conversion to Base r

- Converts fractional portion of a decimal number to base r
- Informal Algorithm
  - Repeatedly multiply (just the fractional) part by r until the fractional part equals 0
  - After each multiplication, remove integer result
  - Integer results form the coefficients of fraction base r (MSD = first integer result)

$0.375_{10} = (??)_2$

MSD

0 — $0.750 = 2 * 0.375$

integer part *fractional part*

1 — $1.50 = 2 * 0.750$

1 — $1.00 = 2 * 0.500$

LSD

Multiply until the fractional part = 0

$0.375_{10} = (0.011)_2$

How number conversion works

$$0.375_{10} = \underset{2^{-1}}{\underline{a_{-1}}}\ \underset{2^{-2}}{\underline{a_{-2}}}\ \underset{2^{-3}}{\underline{a_{-3}}}\ \underset{2^{-4}}{\underline{a_{-4}}}$$

$$0.375_{10} = a_{-1}2^{-1} + a_{-2}2^{-2} + a_{-3}2^{-3} + a_{-4}2^{-4}$$

$$2*(0.375_{10}) = (a_{-1}2^{-1} + a_{-2}2^{-2} + a_{-3}2^{-3} + a_{-4}2^{-4})*2 = (\underset{\text{Int.}}{a_{-1}2^{0}} + \underbrace{a_{-2}2^{-1} + a_{-3}2^{-2} + a_{-4}2^{-3}}_{\text{Fraction}})$$

$$2*(0.375_{10}) = (a_{-2}2^{-1} + a_{-3}2^{-2} + a_{-4}2^{-3})*2 = (\underset{\text{Int.}}{a_{-2}2^{0}} + \underbrace{a_{-3}2^{-1} + a_{-4}2^{-2}}_{\text{Fraction}})$$

- Each time we multiply by *r*, another coefficient "falls out" and all the other place values are increased by a factor of *r*.

Your Turn

$1629_{10} = ?_{16} =$

$0.1875_{10} = ?_8 =$

Note: Each base has some fractions that do not have finite representations:

$(1/3)_{10} = .333\ldots$ but $(1/3)_3 = .1$

$(1/10)_{10} = .1\ldots$ but $(1/10)_2 = .000110\ldots$

Making change…A less formal approach

MORE DECIMAL TO BASE R

Decimal to Binary

- To convert a decimal number, *x,* to binary:
  1. Find place values that add up to the desired values, starting with larger place values and proceeding to smaller values
  2. Place a 1 in those place values and 0 in all others

0	1	1	0	0	1	$= 25_{10}$
32	16	8	4	2	1	

For 25_{10} the place value 32 is too large to include so we include 16. Including 16 means we have to make 9 left over. Include 8 and 1.

Like making change…use larger denominations giving as many as possible without going over then move to smaller denominations

You're Turn

- $73_{10} = ?_2 =$
- $151_{10} = ?_2 =$
- $0.625_{10} = ?_2 =$
- $18_{10} = ?_{16} =$

Shortcuts for Conversion between base 2, 8, 16

OCTAL, BINARY, HEX

Binary, Octal, and Hexadecimal

- Octal (base 8 = 2^3)
- 1 Octal digit can represent: 0-7
- 3 bits of binary can represent: 000-111 = 0 – 7
- Conclusion…
  1 Octal digit = 3 bits

- Hex (base 16=2^4)
- 1 Hex digit can represent: 0-F (0-15)
- 4 bits of binary can represent: 0000-1111= 0-15
- Conclusion…
  1 Hex digit = 4 bits

Binary to Octal or Hex

- Make groups of 3 bits starting from radix point and working outward
- Add 0's where necessary
- Convert each group of 3 to an octal digit

$$\underbrace{101}_{5}\,\underbrace{001}_{1}\,\underbrace{110}_{6}\,.\,\underbrace{11\;0}_{6}$$

$$516.6_8$$

- Make groups of 4 bits starting from radix point and working outward
- Add 0's where necessary
- Convert each group of 4 to an octal digit

$$\underbrace{0\,0\,0\,1}_{1}\,\underbrace{0100}_{4}\,\underbrace{1110}_{E}\,.\,\underbrace{11\;0\,0}_{C}$$

$$14E.C_{16}$$

Octal or Hex to Binary

- Expand each octal digit to a group of 3 bits

317.2_8

$\overbrace{011}\overbrace{001}\overbrace{111}.\overbrace{010}_2$

11001111.01_2

- Expand each hex digit to a group of 4 bits

$D93.8_{16}$

$\overbrace{1101}\overbrace{1001}\overbrace{0011}.\overbrace{1000}_2$

110110010011.1_2

You're Turn

- $6D.7E_{16} = ?_2 =$

- $111010.11_2 = ?_{16} =$

Conversion Methods

- Base r => Base 10
  - Sum of coefficients * place values
- Base 10 => Base r
  - Division method for integer portion
  - Multiplication method for fraction portion
- Binary ⇔ Octal
  - 3-bits = 1 octal digit
- Binary ⇔ Hex
  - 4-bits = 1 hex digit

BINARY ARITHMETIC

Binary Arithmetic

- Can perform all arithmetic operations (+,-,*,÷) on binary numbers
- Use same methods as in decimal
  - Still use carries and borrows, etc.
  - Only now we carry when sum is 2 or more rather than 10 or more (decimal)
  - We borrow 2's not 10's from other columns
- Easiest method is to add bits in your head in decimal (1+1 = 2) then convert the answer to binary ($2_{10} = 10_2$)

Binary Addition

- In decimal addition we carry when the sum is 10 or more
- In binary addition we carry when the sum is 2 or more
- Add bits in binary to produce a sum bit and a carry bit

```
   0        0        1       1
                              1
 + 0      + 1      + 0     + 1
 ---      ---      ---     ---
  00       01       01      10
```

0 + 0 = 00: no need to carry; sum bit	0 + 1 = 01: no need to carry; sum bit	1 + 0 = 01: no need to carry; sum bit	1 + 1 = 10: carry 1 into next column of bits; sum bit

Binary Addition

```
      110
      0110 (6)
  +   0111 (7)
  ----------
      1101 (13)
      8 4 2 1
```

Binary Addition

```
      0
   0110 (6)
 + 0111 (7)
 ---------
   1101 (13)
```

```
   0
 + 1
 ---
  01
```

carry bit   sum bit

Binary Addition

```
       10                        0
     0110 (6)                    1
+    0111 (7)                 +  1
---------                     ----
     1101 (13)                  10
```

carry bit — sum bit

Binary Addition

```
    110
    0110 (6)
 +  0111 (7)
 ---------
    1101 (13)
```

```
      1
      1
  +   1
  -----
     11
```

$1+1+1 = 3_{10} = 11_2$

carry bit

sum bit

Binary Addition

```
   110             1
   0110 (6)        0
+  0111 (7)     +  0
---------       ----
   1101 (13)      01
```

carry bit   sum bit

Binary Subtraction

- If you can't perform subtraction in one column borrow from higher order columns
- When you borrow you are borrowing a 2, not a 10 as in decimal

```
    1 0 1 0 (10)
-   0 1 1 1 (7)
-----------
    0 0 1 1 (3)
```

Binary Subtraction

```
       1 0 1 0 (10)
  -    0 1 1 1 (7)
  ----------------
       0 0 1 1 (3)
```

Can't perform 0 – 1, so we must borrow.

Binary Subtraction

$$
\begin{array}{cccccc}
 & & & 0 & & \\
 & 1 & 0 & \cancel{1} & 10 & (10) \\
- & 0 & 1 & 1 & 1 & (7) \\
\hline
 & 0 & 0 & 1 & 1 & (3)
\end{array}
$$

We borrow the 1 (which is worth 2) from the next column and now we can perform 10-1 = 1.

And now we go to the next column

Binary Subtraction

```
            0
    1  0  1 1 0  (10)
-   0  1  1  1   (7)
----------------
    0  0  1  1   (3)
```

Can't perform 0 – 1, so we must borrow.

Binary Subtraction

We get the borrow from this column and work back to the current column

0 1 10

~~1~~ 1 ~~0~~ ~~1~~ 1 0 (10)

\- 0 1 1 1 (7)

0 0 1 1 (3)

We can't borrow from the column next to us (it is 0 as well) so we must try to borrow from the next column and then work our way back to the current column where we perform 10 – 1 = 1

Binary Subtraction

0 1 10

~~1~~ 1 ~~0~~ ~~1~~ 1 0 (10)

\- 0 1 1 1 (7)

0 0 1 1 (3)

We can perform 1 – 1 = 0

Binary Subtraction

```
   0   1  10
   1̸   0̸  1̸10  (10)
-  0   1   1   1  (7)
-------------------
   0   0   1   1  (3)
```

We can perform 0 – 0 = 0

Binary Multiplication

- Like decimal multiplication, find each partial product and shift them, then sum them up
- Multiplying two *n*-bit numbers yields at most a *2*n*-bit product

```
        0 1 1 0 (6)
*       0 1 0 1 (5)
-----------
  0 1 1 1 1 0 (30)
```

Binary Multiplication

```
      0 1 1 0 (6)
*     0 1 0 1 (5)
  -------------
      0 1 1 0
```

First partial product

Binary Multiplication

```
          0 1 1 0 (6)
    *     0 1 0 1 (5)
    -------------
          0 1 1 0
        0 0 0 0        <---- Second partial product
```

Binary Multiplication

```
          0 1 1 0 (6)
    *     0 1 0 1 (5)
   ---------------
          0 1 1 0
        0 0 0 0
      0 1 1 0
```

Third partial product

Binary Multiplication

```
         0 1 1 0 (6)
   *     0 1 0 1 (5)
   ---------------
         0 1 1 0
       0 0 0 0
     0 1 1 0
   0 0 0 0
```

Fourth partial product

Binary Multiplication

```
            0 1 1 0 (6)
     *      0 1 0 1 (5)
       ----------------
            0 1 1 0
          0 0 0 0
        0 1 1 0
    + 0 0 0 0
    -------------------
      0 0 1 1 1 1 0
```

Sum the partial products up

Binary Division

- Use the same long division techniques as in decimal

```
                 0 1 0 1 r.1   (5 r.1)10
              __________
(2)1      10 | 1 0 1 1         (11)10
0             -1 0
              -----
                 0 1
                -0 0
                -----
                   1 1
                  -1 0
                  -----
                   0 1
```

Binary Division

10 (2) goes into 1, 0 times. Since it doesn't, bring in the next bit.

```
      0
   ________
10 | 1 0 1 1
```

Binary Division

10 (2) goes into 10, 1 time. Multiply, subtract, and bring down the next bit.

```
      0 1
   ┌────────
10 │1 0 1 1
   -1 0 ↓
  ─────
      0 1
```

Binary Division

10 (2) goes into 01, 0 times. Multiply, subtract, and bring down the next bit.

```
       0 1 0
    _________
10 | 1 0 1 1
    -1 0
    ----
       0 1
      -0 0
      ----
         1 1
```

Binary Division

10 (2) goes into 11, 1 time. Multiply and subtract. The remainder is 1.

```
       0 1 0 1 r.1
     ________
  10 | 1 0 1 1
      -1 0
     -----
         0 1
        -0 0
        -----
           1 1
          -1 0
          -----
           0 1
```

Text and Codes

OTHER BINARY SYSTEMS

Binary Representation Systems

- Rational Numbers
  - Unsigned
    - Unsigned (Normal) binary
  - Signed
    - Signed Magnitude
    - 2's complement
    - *1's complement**
    - *Excess-N**
- Floating Point*
  - For very large and small (fractional) numbers

- Codes
  - Text
    - ASCII / Unicode
  - Decimal Codes
    - Weighted Codes
      - BCD (Binary Coded Decimal) / (8421 Code)
      - *2421 Code**
      - 84-2-1 Code
    - Non-weighted Codes
      - Excess-3

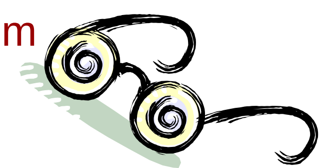
* = Not covered in this class

Interpreting Binary Strings

- Given a string of 1's and 0's, you need to know the *representation system* being used, before you can understand the value of those 1's and 0's.
- Information (value) = Bits + Context (System)

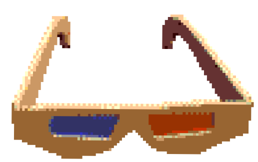
01000001 = ?

Unsigned Binary system



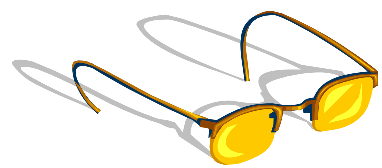
65_{10}

BCD System



41_{BCD}

ASCII system



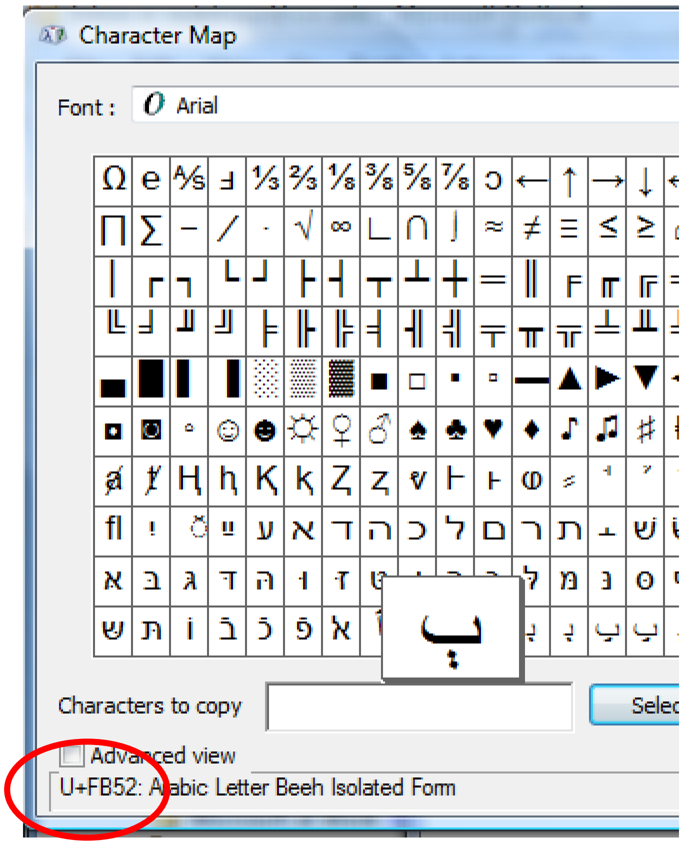
'A'_{ASCII}

ASCII Code

- Used for representing text characters
- Originally 7-bits but usually stored as 8-bits in a computer
- Example:
  - `printf("Hello\n");`
  - Each character is converted to ASCII equivalent
    - 'H' = 0x48, 'e' = 0x65, …
    - \n = newline character
      - CR = carriage return character (moves cursor to start of current line)
      - LF = line feed (moves cursor down a line)

UniCode

- ASCII can represent only the English alphabet, decimal digits, and punctuation
  - 7-bit code => 2^7 = 128 characters
  - It would be nice to have one code that represented more alphabets/characters for common languages used around the world
- Unicode
  - 16-bit Code => 65,536 characters
  - Represents many languages alphabets and characters
  - Used by Java as standard character code



Unicode hex value
(i.e. FB52 => 1111101101010010)

Binary Codes

- Using binary we can represent any kind of information by coming up with a code
- Using *n* bits we can represent 2^n distinct items

Colors of the rainbow:
•Red = 000
•Orange = 001
•Yellow = 010
•Green = 100
•Blue = 101
•Purple = 111

Letters:
•'A' = 00000
•'B' = 00001
•'C' = 00010
.
.
.
•'Z' = 11001

Decimal Codes

- Rather than convert a decimal number to binary, decimal codes represent each decimal digit as a separate group of bits
- BCD (Binary-Coded Decimal) is a popular decimal code that represents each decimal digit as a separate 4-bit number

$$(439)_{10}$$

BCD Representation: 0100 0011 1001

This is not the binary representation of 439, it is the Binary Coded Decimal (BCD) representation

Important: All decimal codes represent each decimal digit with a <u>separate</u> group of bits

Example Decimal Codes

- BCD = Binary-Coded Decimal (a.k.a. 8421 Code)
  - Each decimal digit represented as 4-bit value with the weights (place values) of 8,4,2,1
  - $(972)_{10} = (1001\ 0111\ 0010)_{BCD}$
- 84-2-1
  - Each decimal digit represented by 4-bits with the weights (place values) of 8, 4, -2, -1
  - $(972)_{10} = (1111\ 1001\ 0110)_{84\text{-}2\text{-}1}$
  - 9 = 8+4+(-2)+(-1), 7 = 8+(-1), 2 = 4+(-2)
- Excess-3
  - Each decimal digit represented by 4-bits equal to the binary value of the digit plus 3
  - $(972)_{10} = (1100\ 1010\ 0101)_{XS3}$
  - 9 = 1001 + 0011, 7 = 0111 + 0011, 2 = 0010 + 0011

Decimal Code Review

$518_{10} = (\qquad\qquad\qquad\qquad\qquad\qquad)_{BCD}$

$518_{10} = (\qquad\qquad\qquad\qquad\qquad\qquad)_{XS3}$

$518_{10} = (\qquad\qquad\qquad\qquad\qquad\qquad)_{84\text{-}2\text{-}1}$

$(0111\ 0101\ 1000)_{BCD} = ?_{10}$

$(0111\ 0101\ 1000)_{XS3} = ?_{10}$

$(0111\ 0101\ 1000)_{84\text{-}2\text{-}1} = ?_{10}$

Sample Conversions

$518_{10} = (0101\ 0001\ 1000)_{BCD}$

$518_{10} = (1000\ 0100\ 1011)_{XS3}$

$518_{10} = (1011\ 0111\ 1000)_{84\text{-}2\text{-}1}$

$(0111\ 0101\ 1000)_{BCD} = 758_{10}$

$(0111\ 0101\ 1000)_{XS3} = 415_{10}$

$(0111\ 0101\ 1000)_{84\text{-}2\text{-}1} = 138_{10}$

- Question: With 4-bits we can make 16 combinations. Which combinations are illegal (not possible) when using 84-2-1 Code? Excess-3?

Unused/Illegal Codes

- Decimal codes use 4-bits for each digit
  - $2^4 = 16$ combos
  - Only 10 dec. digits
- 6 unused/illegal codes per system

D3`	D2	D1	D0	Dec. Digit BCD	Dec. Digit 84-2-1 Code	Dec. Digit Excess-3
0	0	0	0	0	0	Illegal
0	0	0	1	1	Illegal	Illegal
0	0	1	0	2	Illegal	Illegal
0	0	1	1	3	Illegal	0
0	1	0	0	4	4	1
0	1	0	1	5	3	2
0	1	1	0	6	2	3
0	1	1	1	7	1	4
1	0	0	0	8	8	5
1	0	0	1	9	7	6
1	0	1	0	Illegal	6	7
1	0	1	1	Illegal	5	8
1	1	0	0	Illegal	Illegal	9
1	1	0	1	Illegal	Illegal	Illegal
1	1	1	0	Illegal	Illegal	Illegal
1	1	1	1	Illegal	9	Illegal

Decimal Codes

- Rather than convert a decimal number to binary, decimal codes represent each decimal digit as a separate group of bits
- BCD (Binary-Coded Decimal) is a popular decimal code that represents each decimal digit as a separate 4-bit number

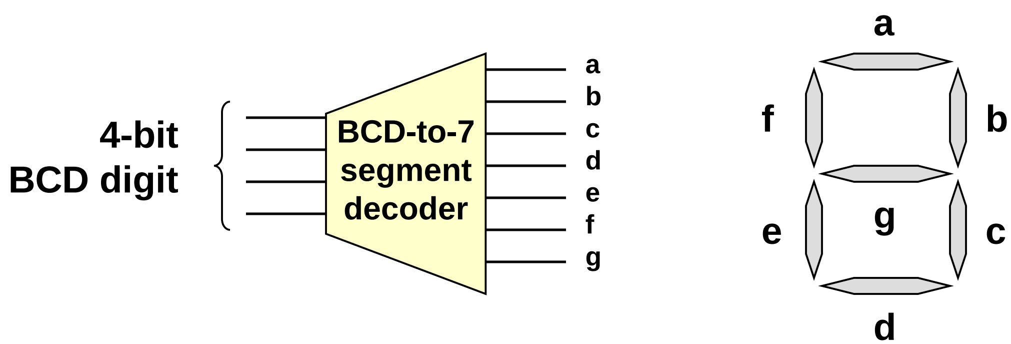
$$(439)_{10}$$

BCD Representation: $\overbrace{0100}\ \overbrace{0011}\ \overbrace{1001}$

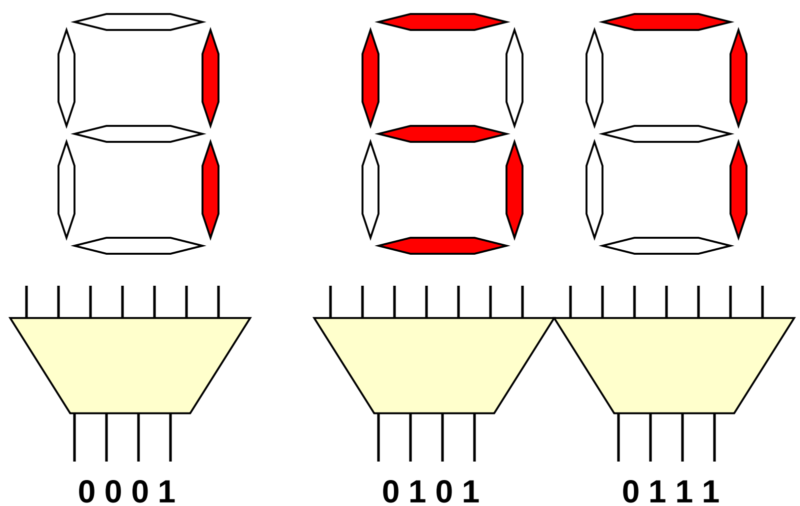
This is not the binary representation of 439, it is the Binary Coded Decimal (BCD) representation

Important: All decimal codes represent each decimal digit with a <u>separate</u> group of bits

BCD & 7-Segment Displays



7-Segment Displays for outputting decimal numbers



Concepts & Skills

- Concepts
  - Number conversion from any base
  - Arithmetic in other bases
  - Binary representation systems & codes
- Skills
  - Convert from decimal using division/multiplication method
  - Convert between Bin/Oct/Hex
  - Perform addition and subtraction in any base
  - Representing decimal numbers in binary decimal codes