Lecture 2 Slides

Number Conversion
Binary Arithmetic
Codes (Decimal Codes)
Review

• Base $r \Rightarrow$ 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 $r$

\[ r^4 \quad r^3 \quad r^2 \quad r^1 \quad r^0 \quad \ldots \quad r^{-1} \quad r^{-2} \quad r^{-3} \]

• Base $r$ => Base 10 Method

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

\[(45.375)_{10}\]

\[45 \quad .375\]

Division Method \hspace{1cm} Multiplication Method
Division Method Explanation

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

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

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

\[ 22.5_{10} = a_4 2^3 + a_3 2^2 + a_2 2^1 + a_1 2^0 + a_0 2^{-1} \]
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)

\[ 193_{10} = (??)_{5} \]

\[
\begin{array}{c|c}
5 & 193 \\
\hline
5 & 38 \quad \text{rem.} = 3 \\
\hline
5 & 7 \quad \text{rem.} = 3 \\
\hline
5 & 1 \quad \text{rem.} = 2 \\
\hline
0 & \quad \text{rem.} = 1 \\
\end{array}
\]

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

\[ 193_{10} = (1233)_{5} \]
Division Method Example

\[ 45_{10} = (??)_2 \]

2 \[ \begin{array}{c|c}
45 & \\
2 & 22 \\
2 & 11 \\
2 & 5 \\
2 & 2 \\
2 & 1 \\
0 & \\
\end{array} \]

rem. = 1
rem. = 0
rem. = 1
rem. = 1
rem. = 1
rem. = 0
rem. = 1

Keep dividing until you reach 0

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

\[ 45_{10} = (101101)_2 \]

© Mark Redekopp, All rights reserved
How number conversion works

\[ 45_{10} = a_4 a_3 a_2 a_1 a_0 \]
\[ \overline{2^4 2^3 2^2 2^1 2^0} \]

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

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

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

• 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

Each time we divide by \( r \), another coefficient “falls out” and all the other place values are reduced by a factor of \( r \).

\[
\frac{D_{10}}{r} = \left(\frac{a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + \cdots + a_1r^1 + a_0}{r}\right) = \frac{r}{r} \left(\frac{a_{n-1}r^{n-2} + \cdots + a_2r^1 + a_1 + a_0}{r}\right)
\]

\( a_0 \) is the remainder.
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)

$$0.375_{10} = (??)_{2}$$

\[
\begin{align*}
0.750 &= 2 \times 0.375 \\
1.50 &= 2 \times 0.750 \\
1.00 &= 2 \times 0.500
\end{align*}
\]

Multiply until the fractional part = 0

$$0.375_{10} = (0.011)_{2}$$
How number conversion works

\[
0.375_{10} = \frac{a_1}{2^{-1}} + \frac{a_2}{2^{-2}} + \frac{a_3}{2^{-3}} + \frac{a_4}{2^{-4}}
\]

\[
0.375_{10} = a_1 2^{-1} + a_2 2^{-2} + a_3 2^{-3} + a_4 2^{-4}
\]

\[
2 \times (0.375_{10}) = (a_1 2^{-1} + a_2 2^{-2} + a_3 2^{-3} + a_4 2^{-4}) \times 2 = (a_1 2^{0} + a_2 2^{-1} + a_3 2^{-2} + a_4 2^{-3})
\]

\[
2 \times (0.375_{10}) = (a_2 2^{-1} + a_3 2^{-2} + a_4 2^{-3}) \times 2 = (a_2 2^{0} + a_3 2^{-1} + a_4 2^{-2})
\]

- 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:

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

\( (1/10)_{10} = .1... \) but \( (1/10)_{2} = .000110... \)
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

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.
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

\[ 101001110.110 \]
\[ 516.6_8 \]

- 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

\[ 000101001110.1100 \]
\[ 14E.C_{16} \]
Octal or Hex to Binary

- Expand each octal digit to a group of 3 bits

317.2\textsubscript{8}

\[
011001111.010_2
\]

11001111.01_2

- Expand each hex digit to a group of 4 bits

D93.8\textsubscript{16}

\[
110110010011.1000_2
\]

110110010011.1_2
You’re Turn

• $6D.7E_{16} = ?_2 =$

• $111010.11_2 = ?_{16} =$
Conversion Methods

• Base $r \Rightarrow$ Base 10
  – Sum of coefficients $\times$ place values

• Base 10 $\Rightarrow$ Base $r$
  – Division method for integer portion
  – Multiplication method for fraction portion

• Binary $\Leftrightarrow$ Octal
  – 3-bits = 1 octal digit

• Binary $\Leftrightarrow$ 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 = 0
  0 + 1 = 1
  1 + 0 = 1
  1 + 1 = 10

no need to carry  no need to carry  no need to carry  carry 1 into next column of bits
  sum bit         sum bit         sum bit
```
Binary Addition

\[
\begin{array}{c}
110 \\
0110 \ (6) \\
+ \ 0111 \ (7) \\
\hline
1101 \ (13)
\end{array}
\]
Binary Addition

\[
\begin{array}{cccc}
\text{0} & \text{0110 (6)} & \text{0} \\
\text{+} & \text{0111 (7)} & \text{+} & \text{1} \\
\hline
\text{1101 (13)} & \text{01} & \\
\text{carry bit} & \text{sum bit}
\end{array}
\]
Binary Addition

\[
\begin{array}{c}
10 \\
0110 \ (6) \\
+ \ 0111 \ (7) \\
\hline
1101 \ (13)
\end{array}
\]

\[+ 1\]

10

\[\text{carry bit}\]

\[\text{sum bit}\]
Binary Addition

\[
\begin{array}{c}
110 \\
0110 \ (6) \\
+ \ 0111 \ (7) \\
\hline
1101 \ (13)
\end{array}
\]

\[
\begin{array}{c}
1 \\
+ 1 \\
\hline
11
\end{array}
\]

\[1+1+1 = 3_{10} = 11_2\]

carry bit  
sum bit
Binary Addition

\[ \begin{array}{c c c}
  & 1 & 1 & 0 \\
+ & 0 & 1 & 1 & 0 & (6) \\
\hline
1 & 1 & 0 & 1 & (13) \\
\end{array} \]

carry bit  

\[ \begin{array}{c c c c c}
  & 1 & 1 & 0 & 1 & 1 & (7) \\
+ & 0 & & & & & (0) \\
\hline
1 & 1 & 0 & 1 & 1 & (13) \\
\end{array} \]

sum bit

01
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

\[
\begin{array}{c}
\text{1 0 1 0 (10)} \\
- \text{0 1 1 1 (7)} \\
\hline
\text{0 0 1 1 (3)}
\end{array}
\]
Binary Subtraction

\[
\begin{array}{c}
1 & 0 & 1 & 0 \quad \text{(10)} \\
- & 0 & 1 & 1 & 1 \quad \text{(7)} \\
\hline
0 & 0 & 1 & 1 \quad \text{(3)}
\end{array}
\]

Can’t perform 0 – 1, so we must borrow.
Binary Subtraction

\[
\begin{array}{cccc}
0 \\
1 & 0 & \textbf{1} & 0 \\
\text{(10)} \\
- & 0 & 1 & 1 & 1 \\
\text{(7)} \\
\hline
0 & 0 & 1 & 1 \\
\text{(3)}
\end{array}
\]

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
Can’t perform 0 – 1, so we must borrow.
Binary Subtraction

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 $10 - 1 = 1$.

\[
\begin{array}{ccc}
\phantom{0} & \phantom{0} & 110 \\
\hline
\phantom{0} & \phantom{0} & 10 \\
\phantom{0} & \phantom{0} & \phantom{0} (10) \\
\hline
\phantom{0} & \phantom{0} & 111 \\
\phantom{0} & \phantom{0} & \phantom{0} (7) \\
\hline
\phantom{0} & \phantom{0} & 0111 \\
\phantom{0} & \phantom{0} & \phantom{0} (3) \\
\end{array}
\]
Binary Subtraction

We can perform $1 - 1 = 0$
Binary Subtraction

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

\[
\begin{array}{cccc}
0 & 1 & 1 & 0 \\
\times & 0 & 1 & 0 & 1 \\
\hline
0 & 1 & 1 & 1 & 1 & 0 \\
\end{array}
\]

(6) * (5) = (30)
Binary Multiplication

\[
\begin{array}{cccc}
0 & 1 & 1 & 0 \ (6) \\
\times & 0 & 1 & 0 \ 1 \ (5) \\
\hline
& 0 & 1 & 1 & 0 \\
\end{array}
\]

First partial product
Binary Multiplication

\[
\begin{array}{cccc}
0 & 1 & 1 & 0 \ (6) \\
\times & 0 & 1 & 0 & 1 \ (5) \\
\hline
0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 \\
\end{array}
\]

Second partial product
Binary Multiplication

\[
\begin{array}{c}
0 & 1 & 1 & 0 \ (6) \\
\times & 0 & 1 & 0 & 1 \ (5) \\
\hline
0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0
\end{array}
\]

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

\[
\begin{array}{cccc}
0 & 1 & 1 & 0 \quad \text{(6)} \\
\times & 0 & 1 & 0 & 1 \quad \text{(5)} \\
\hline
0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 \\
+ & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 1 & 1 & 1 & 1 & 0 \\
\end{array}
\]

Sum the partial products up
Binary Division

• Use the same long division techniques as in decimal

\[ \begin{array}{cccccc}
\text{10} & \text{1} & \text{0} & \text{1} & \text{1} \\
\text{1} & \text{0} & \text{1} & \text{1} \\
\text{0} & \text{1} \\
\text{0} & \text{1} \\
\end{array} \]

\[ \begin{array}{cccccc}
\text{2} & \text{1} \\
\text{0} \\
\text{0} \\
\end{array} \]

\[ \text{(2)}_2 \text{ \ 10 \ 0 \ 1 \ 1 \ r.1 \ (5 \ r.1)_10} \]

\[ \begin{array}{cccccc}
\text{10} & \text{0} & \text{1} & \text{1} & \text{1} \\
\text{1} & \text{0} & \text{1} & \text{1} \\
\text{0} & \text{1} \\
\text{0} & \text{0} \\
\end{array} \]

\[ \begin{array}{cccccc}
\text{1} & \text{1} \\
\text{1} \\
\end{array} \]

\[ \text{(11)}_{10} \]
Binary Division

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

\[
\begin{array}{c}
0 \\
10 \overline{1 0 1 1}
\end{array}
\]
Binary Division

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

\[
\begin{array}{cccc}
0 & 1 \\
\hline
10 & \underline{1} & 0 & 1 & 1 \\
-1 & 0 & \\
\hline
0 & 1
\end{array}
\]
Binary Division

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

\[
\begin{array}{c}
10 \\
\hline
010 \\
\hline
1011 \\
\hline
11 \\
\end{array}
\]
Binary Division

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

```
  10 | 0 1 0 1 . 1
     | 1 0 1 1
     --------
        1 0
        -1 0
           -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 = ?

65_{10}  
41_{BCD}  
‘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 $\Rightarrow 2^7 = 128$ 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 $\Rightarrow 65,536$ characters
  – Represents many languages alphabets and characters
  – Used by Java as standard character code
  
Unicode hex value (i.e. FB52 $\Rightarrow 111101101010010$)
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.

**Important: All decimal codes represent each decimal digit with a separate group of bits.**

**BCD Representation:**

This is not the binary representation of 439, it is the Binary Coded Decimal (BCD) representation.
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
  – \((972)_{10} = (1111\ 1001\ 0110)_{84-2-1}\)
  – \(9 = 8+4+(-2)+(-1)\), \(7 = 8+(-1)\), \(2 = 4+(-2)\)

• 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

\[ 518_{10} = ( \quad )_{BCD} \]

\[ 518_{10} = ( \quad )_{XS3} \]

\[ 518_{10} = ( \quad )_{84-2-1} \]

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

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

\[ (0111 0101 1000)_{84-2-1} = ?_{10} \]
Sample Conversions

\[ 518_{10} = (0101 0001 1000)_{\text{BCD}} \]
\[ 518_{10} = (1000 0100 1011)_{\text{XS3}} \]
\[ 518_{10} = (1011 0111 1000)_{84\text{-}2\text{-}1} \]

\[ (0111 0101 1000)_{\text{BCD}} = 758_{10} \]
\[ (0111 0101 1000)_{\text{XS3}} = 415_{10} \]
\[ (0111 0101 1000)_{84\text{-}2\text{-}1} = 138_{10} \]

• 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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Illegal</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Illegal</td>
<td>Illegal</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>Illegal</td>
<td>Illegal</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>Illegal</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Illegal</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Illegal</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Illegal</td>
<td>Illegal</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Illegal</td>
<td>Illegal</td>
<td>Illegal</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Illegal</td>
<td>Illegal</td>
<td>Illegal</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Illegal</td>
<td>9</td>
<td>Illegal</td>
</tr>
</tbody>
</table>
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

Important: All decimal codes represent each decimal digit with a separate group of bits
BCD & 7-Segment Displays

4-bit BCD digit

BCD-to-7 segment decoder

7-Segment Displays for outputting decimal numbers

0 0 0 1
0 1 1 1
0 1 0 1

0 0 0 1
0 1 0 1
0 1 1 1
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