Number Systems

- Readings: 3.3.3, 3.3.5
- Problem: Implement simple pocket calculator
- Need: Display, adders & subtractors, inputs
  - Display: Seven segment displays
  - Inputs: Switches
- Missing: Way to implement numbers in binary

- Approach: From decimal to binary numbers (and back)

Decimal (Base 10) Numbers

- Positional system - each digit position has a value
  - \(2534 = 2 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 4 \times 10^0\)

- Alternate view: Digit position I from the right = Digit \(\times 10^I\) (rightmost is position 0)
  - \(2534 = 2 \times 10^3 + 5 \times 10^2 + 3 \times 10^1 + 4 \times 10^0\)
Base R Numbers
- Each digit in range 0..(R-1)
  - 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F ...
  - A = 10
  - B = 11
  - C = 12
  - D = 13
  - E = 14
  - F = 15

- Digit position I = Digit * R^I
  - D_3 D_2 D_1 D_0 (base R) = D_3*R^3+D_2*R^2+D_1*R^1+D_0*R^0

Number System (Conversion to Decimal)
- Binary: \((101110)_2 = \)

- Hexadecimal: \((E32)_{16} = \)
Conversion from Base R to Decimal

- Binary: \((110101)_2\)

- Hexadecimal: \((A6)_{16}\)

Conversion of Decimal to Binary (Method 1)

- For positive, unsigned numbers
- Successively subtract the greatest power of two less than the number from the value. Put a 1 in the corresponding digit position

- \(2^0=1\)
- \(2^1=2\)
- \(2^2=4\)
- \(2^3=8\)
- \(2^4=16\)
- \(2^5=32\)
- \(2^6=64\)
- \(2^7=128\)
- \(2^8=256\)
- \(2^9=512\)
- \(2^{10}=1024\) (1K)
- \(2^{11}=2048\) (2K)
- \(2^{12}=4096\) (4K)
- \(2^{13}=8192\) (8K)
Decimal to Binary Method 1

- Convert \((2578)_{10}\) to binary

- Convert \((289)_{10}\) to binary

Conversion of Decimal to Binary (Method 2)

- For positive, unsigned numbers
- Repeatedly divide number by 2. Remainder becomes the binary digits (right to left)
- Explanation:
Decimal to Binary Method 2

- Convert \((289)_{10}\) to binary

Decimal to Binary Method 2

- Convert \((85)_{10}\) to binary
Converting Binary to Hexadecimal

- 1 hex digit = 4 binary digits
- Convert \((11100011010111010011)_2\) to hex

- Convert \((A3FF2A)_{16}\) to binary

Converting Decimal to Hex

- Convert to binary, then to Hex
- Convert \((198)_{10}\) to Hexadecimal
# Arithmetic Operations

<table>
<thead>
<tr>
<th>Decimal:</th>
<th>Binary:</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 7 8 9 2</td>
<td>1 0 1 0 1 1 1</td>
</tr>
<tr>
<td>+ 7 8 9 5 6</td>
<td>+ 0 1 0 0 1 0 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decimal:</th>
<th>Binary:</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 7 8 9 2</td>
<td>1 0 1 0 0 1 1 0</td>
</tr>
<tr>
<td>- 3 2 9 4 6</td>
<td>- 0 0 1 0 1 0 1 0</td>
</tr>
</tbody>
</table>

---

# Arithmetic Operations (cont.)

<table>
<thead>
<tr>
<th>Decimal:</th>
<th>Binary:</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 0 1</td>
<td>1 0 0 1</td>
</tr>
<tr>
<td>* 2 1 4</td>
<td>* 1 0 1 1</td>
</tr>
</tbody>
</table>
**Half Adder**

<table>
<thead>
<tr>
<th>Ai</th>
<th>Bi</th>
<th>Carry</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>0 1</td>
<td>0 1</td>
</tr>
<tr>
<td>1 0</td>
<td>0 1</td>
<td>0 1</td>
<td>1 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 0</td>
<td>1 1</td>
<td>1 0</td>
</tr>
</tbody>
</table>

Carry = Ai Bi

Sum = \( A_i \land B_i \lor A_i \land \overline{B_i} \)

\( = A_i \oplus B_i \)

**Full Adder**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Ci</th>
<th>Co</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0 1</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0 0</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0 1</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1 0</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0 0</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1 1</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0 0</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1 1</td>
<td>1 1</td>
<td></td>
</tr>
</tbody>
</table>
Full Adder Implementation

Multi-Bit Addition

\[ A_3 A_2 A_1 A_0 \\
+ B_3 B_2 B_1 B_0 \]
Multi-Bit Addition in Verilog, Parameters

```verilog
module uadd #(parameter WIDTH=8)(out, a, b);
    output logic [WIDTH:0] out;
    input  logic [WIDTH-1:0] a, b;
    always_comb begin
        out = a + b;
    end
endmodule

module add4 #(parameter W=22)(out, a, b, c, d);
    output logic [W+1:0] out;
    input  logic [W-1:0] a, b, c, d;
endmodule
```

Negative Numbers

- Need an efficient way to represent negative numbers in binary
  - Both positive & negative numbers will be strings of bits
  - Use fixed-width formats (4-bit, 16-bit, etc.)
- Must provide efficient mathematical operations
  - Addition & subtraction with potentially mixed signs
  - Negation (multiply by -1)
Sign/Magnitude Representation

High order bit is sign: 0 = positive (or zero), 1 = negative
Three low order bits is the magnitude: 0 (000) thru 7 (111)
Number range for n bits = +/-2^{n-1} - 1
Representations for 0:

Sign/Magnitude Addition

0 0 1 0 (+2) + 0 1 0 0 (+4) = 1 0 1 0 (+6)

1 0 1 0 (-2) + 1 1 0 0 (-4) = 0 1 1 0 (-2)

Bottom line: Basic mathematics are too complex in Sign/Magnitude
Idea: Pick negatives so that addition works

- Let \(-1 = 0 - (+1):\)
  \[
  \begin{array}{c}
  0 0 0 0 \ (0) \\
  -0 0 0 1 \ (+1)
  \end{array}
  \]

- Does addition work?
  \[
  \begin{array}{c}
  0 0 1 0 \ (+2) \\
  +1 1 1 1 \ (-1)
  \end{array}
  \]

- Result: Two’s Complement Numbers

Two’s Complement

- Only one representation for 0
- One more negative number than positive number
- Fixed width format for both pos. & neg. numbers
Negating in Two’s Complement

- Flip bits & Add 1
- Negate (0010)\(_2\) (+2)
- Negate (1110)\(_2\) (-2)

Addition in Two’s Complement

\[
\begin{array}{ccc}
0 & 0 & 1 & 0 & (+2) & 1 & 1 & 1 & 0 & (-2) \\
+ & 0 & 1 & 0 & 0 & (+4) & + & 1 & 1 & 0 & 0 & (-4)
\end{array}
\]

\[
\begin{array}{ccc}
0 & 0 & 1 & 0 & (+2) & 1 & 1 & 1 & 0 & (-2) \\
+ & 1 & 1 & 0 & 0 & (-4) & + & 0 & 1 & 0 & 0 & (+4)
\end{array}
\]
Subtraction in Two’s Complement

- \( A - B = A + (-B) = A + \overline{B} + 1 \)

- 0010 - 0110

- 1011 - 1001

- 1011 - 0001

Overflows in Two’s Complement

*Add two positive numbers but get a negative number*

*or two negative numbers but get a positive number*

- \( 5 + 3 = -8 \)

- \( 7 - 2 = +7 \)
Overflow Detection in Two’s Complement

<table>
<thead>
<tr>
<th></th>
<th>0101</th>
<th>-7</th>
<th>1001</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0101</td>
<td>-7</td>
<td>1001</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>-2</td>
<td>1110</td>
</tr>
<tr>
<td>-8</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Overflow</td>
<td></td>
<td>Overflow</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0101</th>
<th>-3</th>
<th>1101</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0101</td>
<td>-3</td>
<td>1101</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>-5</td>
<td>1011</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>-8</td>
<td></td>
</tr>
<tr>
<td>No overflow</td>
<td></td>
<td>No overflow</td>
<td></td>
</tr>
</tbody>
</table>

Adder/Subtractor

\[ A - B = A + (-B) = A + B + 1 \]
Converting Decimal to Two’s Complement

- Convert absolute value to unsigned binary, then fixed width, then negate if necessary
- Convert \((-9)_{10}\) to 6-bit Two’s Complement
- Convert \((9)_{10}\) to 6-bit Two’s Complement

Converting Two’s Complement to Decimal

- If Positive, convert as normal;
  If Negative, negate then convert.
- Convert \((11010)_2\) to Decimal
- Convert \((01101)_2\) to Decimal
Sign Extension

- To convert from N-bit to M-bit Two’s Complement (N<M), simply duplicate sign bit:

- Convert \((0010)_2\) to 8-bit Two’s Complement

- Convert \((1011)_2\) to 8-bit Two’s Complement

Solving Complex Problems

- Many problems too complex to build as one system
  - Replace with communicating sub-circuits

- Design process:
  - Understand the problem
  - Break problem into subsystems, identifying connections
  - Design individual subsystems.
Complex Problem Example

- Design a digital clock, which can
  - Display the seconds, minutes and hours
- Have three inputs
  - Increment hour
  - Increment minute
  - Reset seconds

Complex Problem Example (cont.)