2.4 BCD

- Binary Coded Decimal is just what it says it is. Here the decimal digits 0 - 9 are coded into binary. For each digit we need 4 bits. 0000, 0001, ..., 1001. The remaining 4-bit numbers are not used.

\[137_{10} = 0001 \ 0011 \ 0111 \text{ (BCD)}\]

- This is not the same as Binary! looks like binary but...
- BCD provides ease of interfacing for digital displays e.g. 7 segment displays.

- BCD is wasteful of bits.
- The conversion between BCD and binary is complicated. Thus most arithmetic operations are done with binary rather than other representations. (However it is possible to implement arithmetic in BCD but with a carry between the columns).

2.5 Signed numbers

- Sooner or later it becomes necessary to represent negative numbers. Can’t perform arithmetic easily without signed numbers.

2.5.1 Sign-magnitude representation

- Perhaps the simplest way to represent a negative number is to devote the MSB to indicating the sign.
- Sign-magnitude is often used where numbers are to be displayed (The MSB is simply tied to the sign display).
- Sign-magnitude is also used in some A/D conversion schemes.
- However sign-magnitude is not the best for carrying out computation.
2.5.2 Offset Binary representation (Excess-K)

- Offset Binary is where one subtracts K (usually half the largest possible number) from the representation to get the value.

- Has the advantage that the number sequence from the most negative to the most positive is a simple binary progression, which makes it a natural for binary counters.

- Note that the MSB still carries the sign information.

- Excess K is used in conjunction with floating point representations for the exponent. We will meet this again shortly.

- A note on arithmetic in Excess K:

  Assume $a$, $b$, $c$ are three values:

  $$\begin{align*}
  (a + b) &= c \quad \text{(values)} \\
  (a + k) + (b + k) &= (a + b) + 2k \\
  &= (c + k) + k
  \end{align*}$$

  Rewriting $A$, $B$, $C$ in Excess-K representations:

  $$\begin{align*}
  A + B &= C + k \\
  C &= (A + B) - k
  \end{align*}$$

- Try this for $(-1) + (+1) = 0$

2.5.3 2’s complement

- 2’s complement represents the method most widely used for integer computation.

- Positive numbers are represented in simple unsigned binary.

- The system is rigged so that a negative number is represented as the binary number that when added to a positive number of the same magnitude gives zero.

- To get the two’s complement, first take the ones complement, then add one.

2.5.4 1’s complement

- Exchange all the 1’s for 0’s and vice versa.
### 2.6 Performing Arithmetic

#### 2.6.1 In 1’s complement

Some examples of arithmetic with 1’s complement.

**Example 2.6.1 (addition)**

$$
\begin{align*}
0011 & \quad (+3) \\
+0010 & \quad (+2) \\
\hline
0101 & \quad (+5)
\end{align*}
$$

**Example 2.6.2 (subtraction)**

$$
\begin{align*}
0011 & \quad (+3) \\
+1101 & \quad (-2) \\
(1)0000 & \quad (0?)
\end{align*}
$$

<table>
<thead>
<tr>
<th>value</th>
<th>1’s complement</th>
<th>2’s complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>+7</td>
<td>0111</td>
<td>0111</td>
</tr>
<tr>
<td>+6</td>
<td>0110</td>
<td>0110</td>
</tr>
<tr>
<td>+5</td>
<td>0101</td>
<td>0101</td>
</tr>
<tr>
<td>+4</td>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>+3</td>
<td>0011</td>
<td>0011</td>
</tr>
<tr>
<td>+2</td>
<td>0010</td>
<td>0010</td>
</tr>
<tr>
<td>+1</td>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>-1</td>
<td>1110</td>
<td>1111</td>
</tr>
<tr>
<td>-2</td>
<td>1101</td>
<td>1110</td>
</tr>
<tr>
<td>-3</td>
<td>1100</td>
<td>1101</td>
</tr>
<tr>
<td>-4</td>
<td>1011</td>
<td>1100</td>
</tr>
<tr>
<td>-5</td>
<td>1010</td>
<td>1011</td>
</tr>
<tr>
<td>-6</td>
<td>1001</td>
<td>1010</td>
</tr>
<tr>
<td>-7</td>
<td>1000</td>
<td>1001</td>
</tr>
<tr>
<td>-8</td>
<td>-</td>
<td>1000</td>
</tr>
<tr>
<td>-0</td>
<td>1111</td>
<td>-</td>
</tr>
</tbody>
</table>
Example 2.6.3 (subtraction from a negative)

\[
\begin{array}{c|c}
1100 & (-3) \\
+1101 & (-2) \\
(1)1001 & (-6?) \\
\end{array}
\]

*The solution is to wrap the carry back in to the LSB.*

**Exercise 2.6.4** Can you explain why this works?

### 2.6.2 In 2’s complement

The Arithmetic operations are perhaps easiest in 2’s complement.

- *To add* ... just like in any other base.

**Example 2.6.5** (addition 5 + (-2):

\[
\begin{array}{c|c}
0101 & (+5) \\
+1110 & (-2) \\
0011 & (+3) \\
\end{array}
\]

- *To subtract* B from A take the 2’s complement of B and add to A.

**Example 2.6.6** (subtraction 2 - 5:

\[
\begin{array}{c|c}
0010 & (+2) \\
+1011 & (-5) \\
1101 & (-3) \\
\end{array}
\]

*since +5 = 0101*
2.6. **PERFORMING ARITHMETIC**

<table>
<thead>
<tr>
<th>value</th>
<th>Sign Magnitude</th>
<th>Offset Binary</th>
<th>2’s complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>+7</td>
<td>0111</td>
<td>1111</td>
<td>0111</td>
</tr>
<tr>
<td>+6</td>
<td>0110</td>
<td>1110</td>
<td>0110</td>
</tr>
<tr>
<td>+5</td>
<td>0101</td>
<td>1101</td>
<td>0101</td>
</tr>
<tr>
<td>+4</td>
<td>0100</td>
<td>1100</td>
<td>0100</td>
</tr>
<tr>
<td>+3</td>
<td>0011</td>
<td>1011</td>
<td>0011</td>
</tr>
<tr>
<td>+2</td>
<td>0010</td>
<td>1010</td>
<td>0010</td>
</tr>
<tr>
<td>+1</td>
<td>0001</td>
<td>1001</td>
<td>0001</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>1000</td>
<td>0000</td>
</tr>
<tr>
<td>-1</td>
<td>1001</td>
<td>0111</td>
<td>1111</td>
</tr>
<tr>
<td>-2</td>
<td>1010</td>
<td>0110</td>
<td>1110</td>
</tr>
<tr>
<td>-3</td>
<td>1011</td>
<td>0101</td>
<td>1101</td>
</tr>
<tr>
<td>-4</td>
<td>1100</td>
<td>0100</td>
<td>1100</td>
</tr>
<tr>
<td>-5</td>
<td>1101</td>
<td>0011</td>
<td>1011</td>
</tr>
<tr>
<td>-6</td>
<td>1110</td>
<td>0010</td>
<td>1010</td>
</tr>
<tr>
<td>-7</td>
<td>1111</td>
<td>0001</td>
<td>1001</td>
</tr>
<tr>
<td>-8</td>
<td>-</td>
<td>0000</td>
<td>1000</td>
</tr>
<tr>
<td>-0</td>
<td>1000</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

- Multiplication also works right in 2’s complement. Long multiplication reduces to shifts and adds

- We have implicitly used the concept of carry. In particular we dropped/ignored the carry bit in the case of the two’s complement number representation.

---

**Example 2.6.7** \(3 - 3 =\)

\[
\begin{align*}
0011 & \quad (3) \\
\pm1101 & \quad (-3)
\end{align*}
\]

\(\text{(c.f. above 1’s complement example)}\)
• It should be clear that for the unsigned binary the carry has relevance.

Example 2.6.8 (3 + 13 =)

\[
\begin{array}{c c}
0011 & (3) \\
+1101 & (+13) \\
(1) 0000 & (16)
\end{array}
\]

• We can also perform subtraction directly and still ignore the carry/borrow bit.

Example 2.6.9 (2’s complement borrow)

\[
\begin{array}{c c}
0011 & (3) \\
-0100 & (-4) \\
(1) 1111 & (-1)
\end{array}
\]

• However, for subtraction with the unsigned binary the borrow is important, particularly in multiple word operations.

Example 2.6.10 (multiple word addition/carry)

\[
\begin{array}{c c}
0011 \ 0011 & (51) \\
+0000 \ 1101 & (13) \\
0100 \ 0000 & (64)
\end{array}
\]

Example 2.6.11 (multiple subtraction/borrow)

\[
\begin{array}{c c}
0100 \ 0000 & (64) \\
-0000 \ 1101 & (-13) \\
0011 \ 0011 & (51)
\end{array}
\]

• These still could represent two’s complement, but low order words must treated as if unsigned.