2.9. Conversions

**ASCII to binary**

\[
n = 100 \times (\text{Data}[0] - 0x30) \\
  + 10 \times (\text{Data}[1] - 0x30) \\
  + \quad (\text{Data}[2] - 0x30);
\]

This 3-digit ASCII string could also be calculated as

\[
n = (\text{Data}[2] - 0x30) \\
+ 10 \times ((\text{Data}[1] - 0x30) + 10 \times (\text{Data}[0] - 0x30));
\]

**binary to ASCII**

Assume \( n \) is an unsigned integer less than or equal to 999:

\[
\text{Data}[0] = n/100 + 0x30; \\
n = n \% 100; \quad /* \text{0 and 99} */ \\
\text{Data}[1] = n/10 + 0x30; \\
n = n \% 10; \quad /* \text{0 and 9} */ \\
\text{Data}[2] = n + 0x30;
\]

2.10. Fixed-point numbers

*Why:*

- express values with noninteger values
- no floating point hardware support

*When:*

- range of values is known
- range of values is small

*How:*

1) **variable integer**, called \( I \).
   - may be signed or unsigned
may be 8, 16 or 32 bits (precision)

2) **fixed constant**, called $\Delta$ (resolution)
value is fixed, and can not be changed
not stored in memory
specify this fixed content using comments

The value of the fixed-point number

fixed-point number $\equiv I \cdot \Delta$

**decimal fixed-point**, $\Delta = 10^m$

decimal fixed-point number $= I \cdot 10^m$
nice for human input/output

**binary fixed-point**, $\Delta = 2^m$

binary fixed-point number $= I \cdot 2^m$
easier for computers to perform calculations

**Checkpoint 2.46:** Give an approximation of $\pi$ using the decimal fixed-point ($\Delta = 0.001$) format.

**Analog to digital converter (ADC)**
analog input range is 0 to +5 V,
digital output varies 0 to 255

$$V_{in} = 5*N/255 = 0.019607843*N$$
Decimal fixed-point is chosen because the voltage data for this voltmeter will be displayed.

A fixed-point resolution of $\Delta=0.01$ V is chosen because it is slightly smaller (better) than the ADC resolution.

<table>
<thead>
<tr>
<th>$V_{\text{in}}$ (V)</th>
<th>N</th>
<th>I (10 mV)</th>
<th>LCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog in</td>
<td>digital out</td>
<td>variable part</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0.02</td>
<td>1</td>
<td>2</td>
<td>0.02</td>
</tr>
<tr>
<td>1</td>
<td>51</td>
<td>100</td>
<td>1.00</td>
</tr>
<tr>
<td>2.5</td>
<td>128</td>
<td>250</td>
<td>2.50</td>
</tr>
<tr>
<td>5</td>
<td>255</td>
<td>500</td>
<td>5.00</td>
</tr>
</tbody>
</table>

Table 2.22. Performance data of voltmeter.

The software performs the following fixed-point calculation to convert N into I.

$$V_{\text{in}} = \frac{5\times N}{255} \quad \text{how ADC works}$$

$$V_{\text{in}} = I \times 0.01 \quad \text{definition of fixed point}$$

$$I = \frac{5\times 100\times N}{255} \quad \text{substitution}$$

$$I = \frac{100\times N}{51} \quad \text{simplify}$$

$$I = \frac{100\times N+25}{51} \quad \text{round to closest integer}$$

There are two mistakes that can happen.

**Overflow** result exceeds the range of number system promotion and ceiling/floor.

**Drop-out** occurs after an integer right shift or a divide intermediate result looses the information divide last when performing multiple calculations.
\[ 100 \cdot (N/51) \quad ? \quad (100 \cdot N)/51 \]

Let \( x, y, z \) be three fixed-point numbers with same \( \Delta \)
\( x = I \cdot \Delta, \ y = J \cdot \Delta, \) and \( z = K \cdot \Delta. \)
To perform **addition** \( z = x + y, \)
\( K = I + J \)
To perform **subtraction** \( z = x - y, \)
\( K = I - J \)

Let \( x, y, z \) be three fixed-point numbers with different \( \Delta \)
\( x = I \cdot 2^{-5}, \ y = J \cdot 2^{-2}, \) and \( z = K \cdot 2^{-3}. \)
To perform addition \( z = x + y, \) first convert to common
\( K = I/4 + 2 \cdot J \)

For multiplication, we have \( z = x \cdot y. \)
Let \( x, y, z \) be three fixed-point numbers with different \( \Delta \)
\( x = I \cdot 2^n, \ y = J \cdot 2^m, \) and \( z = K \cdot 2^p. \)
\[
K \cdot 2^p = I \cdot 2^n \cdot J \cdot 2^m \\
K = I \cdot J \cdot 2^{n+m-p}
\]

For division, we have \( z = x/y. \)
\[
K \cdot 2^p = (I \cdot 2^n)/(J \cdot 2^m) \\
K = I/J \cdot 2^{n-m-p} \text{ depends on if } (n-m-p)>0
\]

We must worry about overflow and drop out.

Jonathan W. Valvano
Consider the following digital filter calculation.
\[ y = x - 0.0532672 \cdot x_1 + x_2 + 0.0506038 \cdot y_1 - 0.9025 \cdot y_2 \]

The variables \( y, y_1, y_2, x, x_1, \) and \( x_2 \) are all integers
\[-0.0532672 \sim -14 \cdot 2^{-8} \]
\[0.0506038 \sim 13 \cdot 2^{-8} \]
\[-0.9025 \sim -231 \cdot 2^{-8} \]
\[y = x + x_2 + (-14 \cdot x_1 + 13 \cdot y_1 - 231 \cdot y_2) \gg 8\]

**Common Error:** Lazy or incompetent programmers use floating-point in many situations where fixed-point would be preferable.

**Observation:** As the fixed constant is made smaller, the accuracy of the fixed-point representation is improved, but the variable integer part also increases. Unfortunately, larger integers will require more bits for storage and calculations.

2.11. *Floating-point numbers*

**Observation:** If the range of numbers is unknown or large, then the numbers must be represented in a floating-point format.

**Observation:** Floating-point implementations on computers like the 6811/6812 that do not have hardware support are extremely long and very slow. So, if you really need
**floating point, a computer with hardware support is highly desirable.**

The floating-point format, $f$, for short real data type

- Bit 31: Mantissa sign, $s=0$ for positive, $s=1$ for negative
- Bits 30:23: 8-bit biased binary exponent $0 = e = 255$
- Bits 22:0: 24-bit mantissa, $m$, expressed as a binary fraction, a binary 1 as the most significant bit is implied.

$$m = 1.m_1m_2m_3...m_{23}$$

![Figure 2.36. 32-bit short real floating-point format.](image)

The value of a short real floating-point number is

$$f = (-1)^s \cdot 2^{e-127} \cdot m$$

The range of values is about $\pm 10^{-38}$ to $\pm 10^{+38}$. A precision of about 7 decimal digits.

**Short real floating point representation of 0.1**

- **Step 1:** 0.1 = $(-1)^0 \cdot 0.1$
- **Step 2:** 0.1 = $(-1)^0 \cdot 2^{-4} \cdot 1.6$
- **Step 3:** 0.1 = $(-1)^0 \cdot 2^{123-127} \cdot 1.6$
- **Step 4:** 0.1 = $(-1)^0 \cdot 2^{123-127} \cdot (1+0.6)$
- **Step 5:** 0.1 = $(-1)^0 \cdot 2^{123-127} \cdot (1+5033165 \cdot 2^{-23})$
- **Step 6:** 0.1 = $(-1)^0 \cdot 2^{87B-127} \cdot (1+$4CCCCD$ \cdot 2^{-23})$
- **Step 7:** 0.1 = $(0,7B,4CCCCD) = (0,01111011,10011001100110011001101)$

Jonathan W. Valvano
The following example shows the steps in finding the floating point approximation for $\pi$.

**Step 1**  
$\pi = (-1)^0 \cdot \pi$

**Step 2**  
$\pi \approx (-1)^0 \cdot 2^1 \cdot 1.570796327$

**Step 3**  
$\pi \approx (-1)^0 \cdot 2^{128-127} \cdot 1.570796327$

**Step 4**  
$\pi \approx (-1)^0 \cdot 2^{128-127} \cdot (1+0.570796327)$

**Step 5**  
$\pi \approx (-1)^0 \cdot 2^{128-127} \cdot (1+4788187\cdot2^{-23})$

**Step 6**  
$\pi \approx (-1)^0 \cdot 2^{80-127} \cdot (1+490FDB\cdot2^{-23})$

**Step 7**  
$\pi \approx (0,80,490FDB)$

When $e$ is 255, plus or minus infinity

When $e$ is 0, **denormalized**

$$f = (-1)^s \cdot 2^{-126} \cdot m$$

where

$$m = 0.m_1m_2m_3...m_{23}$$

**Roundoff** is the error that occurs as a result of an arithmetic operation, discarding the least significant bits of the product

*multiplication* of two 64-bit mantissas yields a 128-bit product. The final result is normalized into a normalized floating point number with a 64-bit mantissa.

Roundoff during *addition* and *subtraction* error results when smaller number is shifted right two n-bit numbers are added the result is n+1 bits
Truncation is the error that when a number is converted from one format to another. For example, 0.1 could not be exactly represented.

2.12. Tutorial 2. Arithmetic and logical operations

<table>
<thead>
<tr>
<th>format</th>
<th>descriptions</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>8-bit unsigned hexadecimal</td>
<td>$00 $12 $FF</td>
</tr>
<tr>
<td>d</td>
<td>8-bit unsigned decimal</td>
<td>0 18 255</td>
</tr>
<tr>
<td>b</td>
<td>8-bit unsigned binary</td>
<td>%00000000 %00010010</td>
</tr>
<tr>
<td>H</td>
<td>16-bit unsigned hexadecimal</td>
<td>$0000 $1234 $FFFF</td>
</tr>
<tr>
<td>D</td>
<td>16-bit unsigned decimal</td>
<td>0 4660 65535</td>
</tr>
<tr>
<td>B</td>
<td>16-bit unsigned binary</td>
<td>%0001000100110100</td>
</tr>
<tr>
<td>-h or +h</td>
<td>8-bit signed hexadecimal</td>
<td>-$80 +$12 +$7F</td>
</tr>
<tr>
<td>-d or +d</td>
<td>8-bit signed decimal</td>
<td>-128 +18 +127</td>
</tr>
<tr>
<td>-b or +b</td>
<td>8-bit signed binary</td>
<td>-%10000000 +%00010010</td>
</tr>
<tr>
<td>-H or +H</td>
<td>16-bit signed hexadecimal</td>
<td>-$8000 +$1234 +$7FF</td>
</tr>
<tr>
<td>-D or +D</td>
<td>16-bit signed decimal</td>
<td>-32768 +4660 +32767</td>
</tr>
<tr>
<td>-B or +B</td>
<td>16-bit signed binary</td>
<td>-+%0001000100110100</td>
</tr>
<tr>
<td>b3</td>
<td>3-bit binary (lsb)</td>
<td>%000 %111</td>
</tr>
<tr>
<td>b4</td>
<td>4-bit binary (lsb)</td>
<td>%0000 %1111</td>
</tr>
<tr>
<td>cc</td>
<td>8-bit binary CCR</td>
<td>sXhInzvc</td>
</tr>
<tr>
<td>c or C</td>
<td>ASCII character</td>
<td>'A' 'x' '0'</td>
</tr>
<tr>
<td>s or S</td>
<td>ASCII string</td>
<td>&quot;Hello World&quot;</td>
</tr>
<tr>
<td>v</td>
<td>address itself, unsigned dec</td>
<td>2048</td>
</tr>
<tr>
<td>V</td>
<td>address itself, unsigned hex</td>
<td>$0800</td>
</tr>
<tr>
<td>+v or −v</td>
<td>address itself, signed dec</td>
<td>-32768 +0 +32767</td>
</tr>
<tr>
<td>+V or −V</td>
<td>address itself, signed hex</td>
<td>-$8000 +0 +$7FF</td>
</tr>
</tbody>
</table>

Table 2.23. Available formats for displaying ViewBox.

Action. Execute TExaS and open the Chap2.rtf and Chap2.uc files. Bring the ViewBox to the front. We will begin talking about unsigned numbers, so we will use the “d” format.
to observe values in Register A, and use the “D” format for Register X.

**Question 2.1.** Register A contains an 8-bit integer. Its precision is 8 bits. As an unsigned number, its range of values is 0 to 255. What happens when you try to set Register A to 256?

**Question 2.2.** What happens when you try to set Register A to -1?

**Question 2.3.** Register X contains a 16-bit integer. Its precision is 16 bits. As an unsigned number, its range of values is 0 to 65535. What happens when you try to set Register X to 65536?

**Question 2.4.** What happens when you try to set Register X to -1?

**Action.** Next, we will study signed numbers. Change the format of Register A to the “+d”, and change the format of Register X to the “+D”. To change the format of a parameter in the ViewBox: click on the ViewBox entry, type the new format in the Format field, then hit enter.

**Question 2.5.** As a signed number, the range of values in Register A is -128 to +127. What happens when you try to set Register A to +128?

**Question 2.6.** What happens when you try to set Register A to -129?
Question 2.7. As a signed number, the range of values in Register X is -32768 to +32767. What happens when you try to set Register X to 32768?

Question 2.8. What happens when you try to set Register X to -32769?

Question 2.9: Use the help system of TExaS to look up the instruction ldaa and answer the question, “Even though the ldaa instruction does not perform any arithmetic or logical operations, does it modify the condition code bits, N, Z, V, and C?” Within the TExaS application execute Help->HelpTopics, double-click 6812 assembly language, double-click 6812 memory access instructions, click ldaa.

Action. Assemble the Chap2.rtf program, and bring the TheList.rtf TheLog.rtf and Chap2.uc windows to the front. Notice that the instructions lsla and asla have the same object code.

Question 2.10: First perform the following logical operations by hand, and record what you think the result will be in 8-bit unsigned hexadecimal. In addition, record your expectation for the N and Z bits. Within TExaS, change the format of Register A to unsigned hexadecimal “h”. Reset and single-step the program through part 1. Correct your answers by recording the proper values of Register A and the CCR bits N and Z. The logical operations clear the V bit. The complement instruction is the only one that sets the C bit, while the other logical operations only affect the N Z and V bits.
Question 2.11: First perform the following unsigned arithmetic operations by hand, and record what you think the result will be in 8-bit unsigned decimal format. In addition, record your expectation for the N Z and C bits. Although the processor will set the V bit during the calculation, we will ignore it when operating on unsigned integers. Within TExaS, change the format of Register A to unsigned decimal “d”. Single-step the program through part 2. Correct your answers by recording the proper values of Register A and the CCR bits N Z and C.

$0F&$85
$0F|$85
$0F^$85
~$0F

155>>1
50<<1
96+64
224+64
160−64
32−64

Question 2.12: First perform the following signed arithmetic operations by hand, and record what you think the result will be in 8-bit signed decimal. In addition, record your expectation for the N Z and V bits. Although the processor will set the C bit during the calculation, we will ignore it when operating on signed integers. Within TExaS, change the format of Register A to signed decimal “+d”. Single-step the program through part 3. Correct your answers by recording the proper values of Register A and the CCR bits N Z and V.

$-101>>1$
-50<<1
-32 + 64
96 + 64
32 - 64
-96 - 64