Required equipment (you will need to buy these)
1) You will need a voltmeter (any cheap one will do, spending $10 to $20 is sufficient) (myDAQ is ok),
2) A wire stripper for 22 or 24 gauge wire
3) We will set up a soldering iron in lab (you will need to make 5 solder connections this semester)

Recap
9S12 decomposes the execution into bus cycles
Stack stores temp data and subroutine return address
Subroutines provide a mechanism for modularity
Parallel port, direction registers

Overview
Intro to C
Logical operations
Shift operations
Arithmetic operations (introduction)

What is C?
- C is a high-level language
  - Abstracts hardware
  - Expressive
  - Readable
  - Analyzable
- C is a procedural language
  - The programmer explicitly specifies steps
  - Program composed of procedures
    - Functions/subroutines
- C is compiled (not interpreted)
  - Code is analyzed as a whole (not line by line)

Why C?
- C is popular
- C influenced many languages
- C is considered close-to-machine
  - Language of choice when careful coordination and control is required
  - Straightforward behavior (typically)
- Typically used to program low-level software (with some assembly)
  - Drivers, runtime systems, operating systems, schedulers, …

How to program in C
- Preprocessor directives
- Variables and types
- Functions
  - Subroutines and functions
- Statements
- Expressions
- Names
- Operators
- Comments
- Syntax

2.6. Logical operations

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A&amp;B</th>
<th>A</th>
<th>B</th>
<th>A^B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.14. Logical operations.
Figure 2.12. implemented with discrete digital gates.

<table>
<thead>
<tr>
<th>A</th>
<th>~A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.11. Logical complement.

| anda #w | RegA=RegA&w |
| anda u  | RegA=RegA&[u] |
| anda U  | RegA=RegA&[U] |
| oraa #w | RegA=RegA|w |
| oraa u  | RegA=RegA|[u] |
| oraa U  | RegA=RegA|[U] |
| eora #w | RegA=RegA^w |
| eora u  | RegA=RegA^[u] |
| eora U  | RegA=RegA^[U] |
| coma    | RegA=~RegA |

The **and** operation to extract, or **mask**, individual bits

\[
\text{Pressed} = \text{PTT} \& 0x40; \quad // \text{true if the switch is pressed}
\]

Interface of a switch to a microcomputer input.

1daa PTT ;read input Port T
anda #$40 ;clear all bits except bit 6
staa Pressed ;true iff the switch is pressed

<table>
<thead>
<tr>
<th>a7 a6 a5 a4 a3 a2 a1 a0</th>
<th>value of PTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td>$40 constant</td>
</tr>
<tr>
<td>0 a6 0 0 0 0 0 0</td>
<td>result of the <strong>anda</strong> instruction</td>
</tr>
</tbody>
</table>

**Question 1.** Assume an input switch is interfaced to Port T bit 5. Write assembly code that reads the switch and branches to location **ItsSet** if the switch is set (PT5 is 1).

The **or** operation to set bits 1 and 0 of the register DDRT.
The other six bits of DDRT remain constant.

**Friendly** software modifies just the bits that need to be.

\[
\text{DDRT} = \text{DDRT}\|0x03; \quad // \text{PT1,PT0 outputs}
\]
\[
\text{DDRT} |= 0x03; \quad // \text{PT1,PT0 outputs}
\]

1daa DDRT ;read previous value
ora #03 ;set bits 1 and 0
staa DDRT ;update
c7 c6 c5 c4 c3 c2 c1 c0  value of DDRT
0 0 0 0 0 0 1 1 03  constant
/  c7 c6 c5 c4 c3 c2  1 1  result of the oraa instruction

**Maintenance Tip:** When interacting with just some of the bits of an I/O register it is better to modify just the bits of interest, leaving the other bits unchanged. In this way, the action of one piece of software does not undo the action of another piece.

**Question 2.** Write friendly code that makes Port T bit 7 (PT7) an output.

The **exclusive or** operation can also be used to toggle bits.

\[
\text{PTT} = \text{PTT} \oplus \text{0x80}; \quad \text{// toggle PT7} \\
\text{PTT} ^= \text{0x80}; \quad \text{// toggle PT7}
\]

ldaa PTT ; read output Port T
eora #$80 ; toggle bit 7
staa PTT ; update

\[
\begin{array}{cccccccc}
\text{b7} & b6 & b5 & b4 & b3 & b2 & b1 & b0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

value of PTT

\[
\begin{array}{cccccccc}
\text{~b7} & b6 & b5 & b4 & b3 & b2 & b1 & b0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]

result of the eora instruction

The output of an **open collector gate**, drawn with the 'x', has two states low (0V) and HiZ (floating.)

**7406 in lab (same as 7405, but can handle more current)**

![Figure 2.16. Positive logic LED interface (Lite-On LTL-10223W).](image)

The **and** operation can be used to clear bits.

\[
\text{PTT} = \text{PTT} \& \text{0xDF}; \quad \text{// PT5 becomes 0} \\
\text{PTT} \&= \text{0xDF}; \quad \text{// PT5 becomes 0} \\
\text{PTT} = \text{PTT} \& (\text{~0x20}); \quad \text{// PT5 becomes 0} \\
\text{PTT} \&= \text{~0x20}; \quad \text{// PT5 becomes 0} \\
\text{PTT\_PT5} = 0;
\]

ldaa PTT ; read output Port T
anda #$DF ; clear just bit 5
staa PTT ; update

![Figure 3.10. Solid state relay interface using a 7405 open collector driver.](image)

; turn off relay
\[
\text{PTT} = \text{PTT} \& 0x0DF; \quad \text{// PT5 becomes 0} \\
\text{PTT} \&= 0x0DF; \quad \text{// PT5 becomes 0} \\
\text{PTT} = \text{PTT} \& (\text{~0x20}); \quad \text{// PT5 becomes 0} \\
\text{PTT} \&= \text{~0x20}; \quad \text{// PT5 becomes 0} \\
\text{PTT\_PT5} = 0;
\]

Jonathan W. Valvano
**Question 3.** Assume PH2 is an output. Write friendly code that clears bit 2 of Port H.

### 2.7. Shift operations

**LSR**

```
0 ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐
  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │
  └───┘ └───┘ └───┘ └───┘ └───┘ └───┘ └───┘ └───┘  
    C
```

*Figure 3.13. 8-bit logical shift right.*

**ASR**

```
    C
```

*Figure 3.15. 8-bit arithmetic shift right.*

**LSL/ASL**

```
0 ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐ ┌───┐
  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │  │
  └───┘ └───┘ └───┘ └───┘ └───┘ └───┘ └───┘  
```

*Figure 3.16. 8-bit shift left.*

#### Maintenance Tip:
Use the `asla` instruction when manipulating signed numbers, and use the `lsla` instruction when shifting unsigned numbers.

A **bit field** is a collection of bits that together have meaning.

```
b7 b6 b5 b4 b3 b2 b1 b0
```

**PTT** bits 7-4 be connected to a stepper motor.

Can take on the values 5, 6, 10 or 9

Let `data` be an 8-bit variable

Write code to set the stepper motor value equal to the variable

```
PTT = (PTT&0x0F)|((data<<4));
PITT = (PTT&0x0F)+(data<<4);
```

The assembly code for this operation is

```
ldaa data ;read value of data
lsia       ;shift into position
lsia
lsia
lsia
```

Jonathan W. Valvano
ldab PTT
andb #$0F   ;keep previous values of PTT
aba         ;combine the two parts
staa PTT    ;output to stepper

0 0 0 0  h3 h2 h1 h0  value of data
0 0 0 0  h3 h2 h1 h0 0  after first lsla
0  h3 h2 h1 h0 0 0  0  after second lsla
h3 h2 h1 h0 0 0  0  after third lsla
0 0 0 0  l3 l2 l1 l0  value of PTT&0x0F
h3 h2 h1 h0 l3 l2 l1 l0  result of the aba instruction

Write code to set the variable equal to the stepper motor
    data = (PTT&0xF0)>>4;
    data = PTT>>4;
The assembly code for this operation is
    ldaa PTT    ;read value of PTT
    lsra        ;shift into position
    lsra
    lsra
    staa data   ;result into variable

2.8. Arithmetic operations

**Question 4.** How many bits does it take to store the result of two unsigned 8-bit numbers added together?
**Question 5.** How many bits does it take to store the result of two unsigned 8-bit numbers subtracted?
**Question 6.** How many bits does it take to store the result of two unsigned 8-bit numbers multiplied together?

Q4, determine range of values after 8-bit add
0 + 0 = 0,   255+255 = 510  0 to 510 is 9 bits
Q5, determine range of values after 8-bit subtract
0 - 255 = -255,  255-0 = 255  -255 to +255 is 9 bits
Q6, determine range of values after 8-bit multiple
0 * 0 = 0,   255*255 = 65025 0 to 65025 is 16 bits

**Condition code register (CC or CCR)**

C set after an unsigned add if the answer is wrong
V set a signed add if the answer is wrong

<table>
<thead>
<tr>
<th>bit</th>
<th>name</th>
<th>meaning after add or sub</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>negative</td>
<td>result is negative</td>
</tr>
<tr>
<td>Z</td>
<td>zero</td>
<td>result is zero</td>
</tr>
<tr>
<td>V</td>
<td>overflow</td>
<td>signed overflow</td>
</tr>
<tr>
<td>C</td>
<td>carry</td>
<td>unsigned overflow</td>
</tr>
</tbody>
</table>

Table 3.16. Condition code bits.
Draw the number wheels for
  unsigned 8-bit numbers
  unsigned 16-bit numbers
  signed 8-bit numbers
  signed 16-bit numbers

Explain addition on the number wheel (overflow??)
  third = first+second;
  ldaa first    ;start at first
  adda second   ;move CW by second
  staa third

Explain subtraction on the number wheel (overflow??)
  third = first-second;
  ldaa first    ;start at first
  suba second   ;move CCW by second
  staa third

Observation: The carry bit, N, is set after an addition or subtraction when the result is on the left half (msbit = 1).

Observation: The overflow bit, Z, is set after an addition or subtraction when the result is zero.

Observation: The carry bit, C, is set after an unsigned addition or subtraction when the result is incorrect.

Observation: The overflow bit, V, is set after a signed addition or subtraction when the result is incorrect.

The bottom line
  Use OR to turn on individual bits
  Use AND to select individual bits
  Use AND to turn off individual bits
  Use EORA to toggle individual bits
  Remember the sign when shifting left
  Number wheel specifies the finite set
  CCR bits describe the state after an add or subtract