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

A	B	A&B	A|B	A^B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Table 2.14. Logical operations.

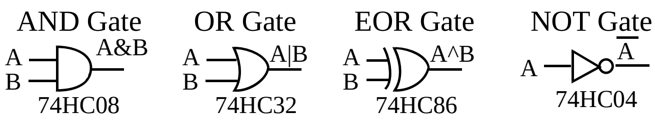


Figure 2.12. implemented with discrete digital gates.

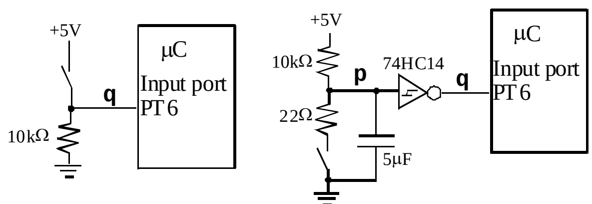
A	~A
0	1
1	0

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

```
Pressed = PTT&0x40;  // true if the switch is pressed
```



Interface of a switch to a microcomputer input.

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

a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	value of PTT
0	1	0	0	0	0	0	0	$40 constant
0	a_6	0	0	0	0	0	0	result of the **anda** instruction

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.

```
DDRT = DDRT|0x03; // PT1,PT0 outputs
DDRT |= 0x03;     // PT1,PT0 outputs

ldaa DDRT   ;read previous value
oraa #$03   ;set bits 1 and 0
staa DDRT   ;update
```

c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0	value of `DDRT`
0	0	0	0	0	0	1	1	`$03` constant
c_7	c_6	c_5	c_4	c_3	c_2	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.

```
  PTT = PTT^0x80;  // toggle PT7
  PTT ^= 0x80;     // toggle PT7

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

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0	value of `PTT`
1	0	0	0	0	0	0	0	`$80` constant
$\sim b_7$	b_6	b_5	b_4	b_3	b_2	b_1	b_0	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)

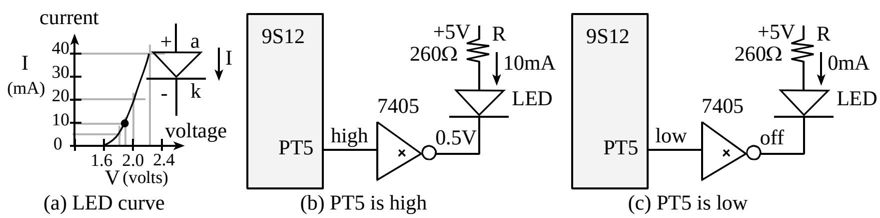


Figure 2.16. Positive logic LED interface (Lite-On LTL-10223W).

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

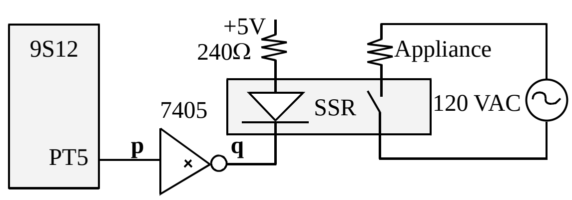


Figure 3.10. Solid state relay interface using a 7405 open collector driver.

```
; turn off relay
  PTT = PTT&0xDF;      //  PT5 becomes 0
  PTT &= 0xDF;         //  PT5 becomes 0
  PTT = PTT&(~0x20);   //  PT5 becomes 0
  PTT &= ~0x20;        //  PT5 becomes 0
  PTT_PTT5 = 0;

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

b7 b6 **b5** b4 b3 b2 b1 b0	value of PTT
1 1 **0** 1 1 1 1 1	$DF constant
b7 b6 **0** b4 b3 b2 b1 b0	result of the `anda` instruction

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

2.7. Shift operations

LSR

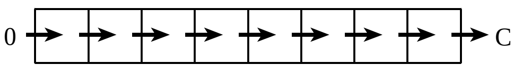


Figure 3.13. 8-bit logical shift right.

ASR

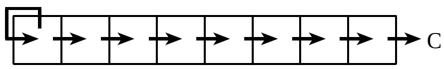


Figure 3.15. 8-bit arithmetic shift right.

LSL/ASL

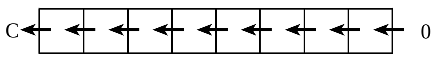


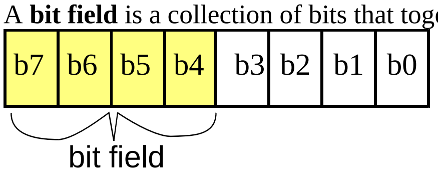
Figure 3.16. 8-bit shift left.

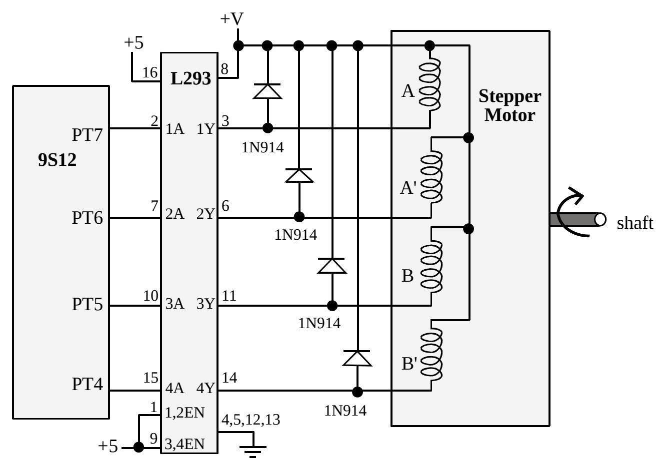
unsigned char	char

`unsigned char in,out;`
`  out = in<<1;`
`     ldaa in`
`     lsla   ;A=A*2`
`     staa out`
 	
`  out = in>>1;`
`     ldaa in`
`     lsra   ;A=A/2`
`     staa out`
	`char in,out;`
	`  out = in<<1;`
	`     ldaa in`
	`     asla   ;A=A*2`
	`     staa out`
	
	`  out = in>>1;`
	`     ldaa in`
	`     asra   ;A=A/2`
	`     staa out`

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.





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

The assembly code for this operation is

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

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

0	0	0	0	h_3	h_2	h_1	h_0	value of **data**
0	0	0	h_3	h_2	h_1	h_0	0	after first **lsla**
0	0	h_3	h_2	h_1	h_0	0	0	after second **lsla**
0	h_3	h_2	h_1	h_0	0	0	0	after third **lsla**
h_3	h_2	h_1	h_0	0	0	0	0	after last **lsla**
0	0	0	0	l_3	l_2	l_1	l_0	value of **PTT&0x0F**
h_3	h_2	h_1	h_0	l_3	l_2	l_1	l_0	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
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

```
adda  #w    ;RegA=RegA+w
adda  u     ;RegA=RegA+[u]
adda  U     ;RegA=RegA+[U]
suba  #w    ;RegA=RegA-w
suba  u     ;RegA=RegA-[u]
suba  U     ;RegA=RegA-[U]
```

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

bit	name	meaning after add or sub
N	negative	result is negative
Z	zero	result is zero
V	overflow	signed overflow
C	carry	unsigned overflow

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