

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) (HKN),
- 2) A wire stripper for 22 or 24 gauge wire
- 3) Soldering iron and solder (if you have a friend with one, borrowing is OK)

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

- Logical operations
- Shift operations
- Arithmetic operations (introduction)

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.

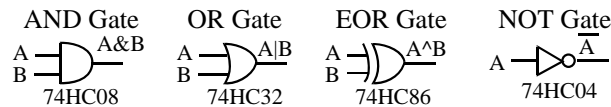


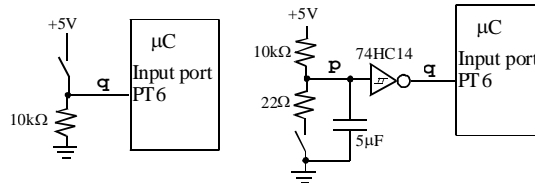
Figure 2.12. implemented with discrete digital gates.

A	$\sim A$
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
    
```

a7	a6	a5	a4	a3	a2	a1	a0	value of PTT
0	1	0	0	0	0	0	0	\$40 constant
0	a6	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 |= 0x03; /*PT1,PT0 outputs */
```

```
ldaa DDRT    ;read previous value
oraa #$03    ;set bits 4 and 5
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.

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

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

b7	b6	b5	b4	b3	b2	b1	b0	value of PTT
1	0	0	0	0	0	0	0	\$80 constant
~b7	b6	b5	b4	b3	b2	b1	b0	result of the eora instruction

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

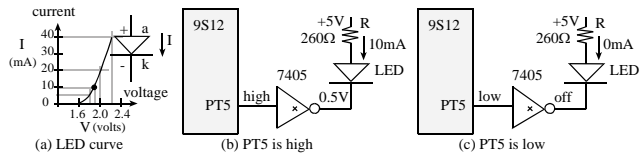


Figure 2.16. Positive logic LED interface (Lite-On LTL-10223W).

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

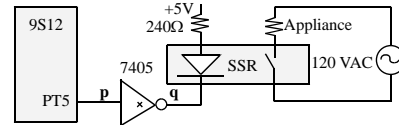


Figure 3.10. Solid state relay interface using a 7405 open collector driver.

```
; turn off relay
PTT &= 0xDF; /* PT5 becomes 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

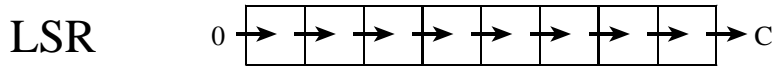


Figure 3.13. 8-bit logical shift right.

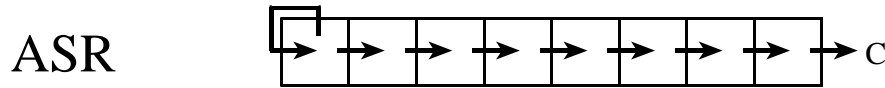


Figure 3.15. 8-bit arithmetic shift right.

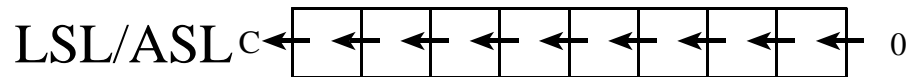


Figure 3.16. 8-bit shift left.

```

asla      ;RegA=RegA*2
lsla      ;RegA=RegA*2
asra      ;RegA=RegA/2
lsra      ;RegA=RegA/2

```

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

High and Low are unsigned 4-bit components, which will be combined into a single unsigned 8-bit Result.

```
Result = (High<<4) | Low;
```

The assembly code for this operation is

```

ldaa High    ;read value of High
lsla         ;shift into position
lsla
lsla
lsla
oraa Low     ;combine the two parts
staa Result  ;save answer

```

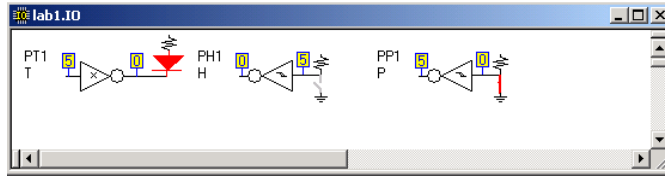
0	0	0	0	h ₃	h ₂	h ₁	h ₀	value of High
0	0	0	h ₃	h ₂	h ₁	h ₀	0	after first lsla
0	0	h ₃	h ₂	h ₁	h ₀	0	0	after second lsla
0	h ₃	h ₂	h ₁	h ₀	0	0	0	after third lsla
h ₃	h ₂	h ₁	h ₀	0	0	0	0	after last lsla
0	0	0	0	l ₃	l ₂	l ₁	l ₀	value of Low
h ₃	h ₂	h ₁	h ₀	l ₃	l ₂	l ₁	l ₀	result of the oraa instruction

Lab 1. Logic Function

The specific function you will implement is

$$T = P \& \overline{H}$$

This means the LED will be on if and only if the **H** switch is not pressed and the **P** switch is pressed, as shown in Figure 1.1.



TExaS IO window showing the door is unlocked.

```
void main(void){
    DDRH = 0x00;    // make Port H an input,  PH2 is H
    DDRP = 0x00;    // make Port P an input,  PP2 is P
    DDRT = 0xFF;    // make Port T an output, PT2 is T
    while(1){
        PTT = (~PTH)&PTP; // LED on iff PP1=1 and PH1=0
    }
}
```

The first C program to illustrate Lab 1.

Approach -> start with template

```
***** Lab1.RTF *****
; Program written by: Your Name
; Date Created: 7/29/2009 6:06:11 PM
; Last Modified: 7/29/2009 6:06:18 PM
; Section 1-2pm      TA: Nachiket Kharalkar
; Lab number: 1
; Brief description of the program
; The overall objective of this system is a digital lock
; Hardware connections
; PH1 is switch input H
; PP1 is switch input P
; PT1 is LED output T (on means unlocked)
; The specific operation of this system
; unlock if P is pressed and H is not pressed
; I/O port definitions on the 9S12DP512
PTH      equ $0260  ; Port H I/O Register
DDRH     equ $0262  ; Port H Data Direction Register
PTP      equ $0258  ; Port P I/O Register
DDRP     equ $025A  ; Port P Data Direction Register
PTT      equ $0240  ; Port T I/O Register
DDRT     equ $0242  ; Port T Data Direction Register
        org $0800   ; RAM
        ; Global variables (none required for this lab)
        org $4000   ; flash EEPROM

main
; Software performed once at the beginning
loop
; Software repeated over and over
    bra loop
    org $FFFE
    fdb main      ; Starting address
*****start TExaS show Lab1.rtf*****
```

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?

```

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

```

ldaa first      ;start at first
adda second     ;move CW by second

```

Explain subtraction on the number wheel

```

ldaa first      ;start at first
suba second     ;move CCW by second

```

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