Chapter 4

Interfacing to a microprocessor

ECE 3120

Dr. Mohamed Mahmoud

http://iweb.tntech.edu/mmahmoud/
mmahmoud@tntech.edu
Outline

4.1 Basic Concepts of Parallel I/O Ports

4.2 Interfacing with Simple I/O Devices

4.4 Advanced Parallel I/O Devices
- I/O ports are used to connect input and output devices to the microcontroller
- Examples for output devices: 7 segments, LEDs, LCDs,.. etc
- Examples for input devices: switches, pushbuttons, keypad, .. etc
- Output ports can output values from microcontroller to output device
- Input ports can input values (zero or one) from input devices to the microcontroller
- I/O ports are connected to the data, address, and control buses
- Output ports read from data bus
- Input ports write to the data bus
- The address on the address bus can enable only one port
The HCS12 I/O ports

Table 4.5 Number of pins available in each parallel port

<table>
<thead>
<tr>
<th>Port Name</th>
<th>No. of Pins</th>
<th>Pin Name</th>
<th>Port names in mc9s12dg256.inc</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>8</td>
<td>PA7~PA0</td>
<td>PORTA</td>
</tr>
<tr>
<td>B</td>
<td>8</td>
<td>PB7~PB0</td>
<td>PORTB</td>
</tr>
<tr>
<td>E</td>
<td>8</td>
<td>PE7~PE0</td>
<td>PORTE</td>
</tr>
<tr>
<td>H</td>
<td>8</td>
<td>PH7~PH0</td>
<td>PTH</td>
</tr>
<tr>
<td>J</td>
<td>4</td>
<td>PJ7~PJ0</td>
<td>PTJ</td>
</tr>
<tr>
<td>K</td>
<td>7</td>
<td>PK4~PK0</td>
<td>PORTK</td>
</tr>
<tr>
<td>M</td>
<td>8</td>
<td>PM7~PM0</td>
<td>PTM</td>
</tr>
<tr>
<td>P</td>
<td>8</td>
<td>PP7~PP0</td>
<td>PTP</td>
</tr>
<tr>
<td>S</td>
<td>8</td>
<td>PS3~PS0</td>
<td>PTS</td>
</tr>
<tr>
<td>T</td>
<td>8</td>
<td>PT7~PT0</td>
<td>PTT</td>
</tr>
<tr>
<td>PAD1, PAD0</td>
<td>16</td>
<td>PAD15~PAD0</td>
<td>PORTADO and PORTAD1</td>
</tr>
<tr>
<td>L</td>
<td>8</td>
<td>PL7~PL0</td>
<td>Not available</td>
</tr>
<tr>
<td>U</td>
<td>8</td>
<td>PU7~PU0</td>
<td>Not available</td>
</tr>
<tr>
<td>V</td>
<td>8</td>
<td>PV7~PV0</td>
<td>Not available</td>
</tr>
<tr>
<td>W</td>
<td>8</td>
<td>PW7~PW0</td>
<td>Not available</td>
</tr>
</tbody>
</table>

Available in H family devices only
Our board's has D family microcontroller
- Each I/O port has a set of pins and several registers to support its operation.

- Registers of I/O ports have unique addresses

- Memory instructions can be used with I/O registers

- Each port has two main registers
  (1) Data Direction Register configures the ports as input or output
  (2) Data Register by which data is exchanged with peripheral

**1) Data Direction Register:**

Can configure each bit individually as input or output.

- To configure a pin for output, write a ‘1’ to the associated bit in the data direction register.

- To configure a pin for input, write a ‘0’ to the associated bit in the data direction register.
- The addresses of the data direction registers are defined in mc9s12dg256.inc

- Name is formed by adding the letters “DDR” as the prefix to the port name. For example, DDRA, DDRB, and DDRT.

```assembly
movb #$FF,DDRA ; configure Port A for output
movb #0,DDRB  ; configure Port B for input
movb #$AA,DDRB ; $AA = %1010 1010 configured Port A odd pins for output, and even pins for input
movb #$0F,DDRB ; configure the first 4 bits in Port B as output and the last 4 bits as input
bset DDRA,$81 ; $81 = %1000 0001 configure Port A pin 7 and 1 for output
```
Simple 2-bit port

Microcontroller

Outside world

Outside world

Microcontroller

Tri state Buffer

Output = input
(2) I/O Data Register:

- When inputting data from input port, the user reads from the port data register.

- When outputting data to an output port, the user writes to the port data register.

- Each I/O port register is assigned to an address. For example, Port A data register is assigned to address 0

- The addresses of the data registers are defined in mc9s12dg256.inc

- Using names to access the I/O registers can improve program readability and reduce the chance of errors.

- The name of port data register is formed by adding letters “PT” or “PORT” as the prefix to the port name. For example, PORTA, PORTB, and PTT.
movb #$35,0 ; output $35 to Port A
PORTA equ 0 ; defined in mc9s12dg256.inc
movb #$35, PORTA ; output $35 to Port A
staa PORTB ; output the contents of A to port B
movb #$67,PORTB ; output the value $67 to Port B
movb PORTB,ibuf ; read the contents of Port B and save it
                  ; at the memory location represented by ibuf
ldaa PORTA ; accumulator A = port B input

Notice all the instructions used for memory locations can be used with ports
Generating a Digital Waveform

- A square wave can be generated by pulling a pin to high for a certain amount of time and then pull it to low for the same amount of time and repeat.

Example: Write a program to generate a 1-kHz periodic square wave from the PT5 pin.

Steps

1. Configure the PT5 pin for output
2. Pull the PT5 pin to high
3. Wait for 0.5 ms
4. Pull PT5 pin to low
5. Wait for 0.5 ms
6. Go to step 2
include 'mc9s12dg256.inc'

org $1500
lds #$1500

bset DDRT,#00010000 ; configure PT5 pin for output

Forever: bset PTT,#00010000 ; pull PT5 pin to high

ldy #10 ; wait for 0.5 ms
jsr delayby50us ;

bclr PTT, #00010000 ; pull PT5 pin to low

ldy #10 ; wait for 0.5 ms
jsr delayby50us ;

bra forever

Delayby50us is a subroutine that makes a delay equal to Y x 50us
Y is the content of register y
Square waveforms at all pins of port B with different frequencies

Pin 0: 10KHz
Pin 1: 5KHz
Pin 2: 2.5 KHz
Pin 3: 1.25 KHz

1- A = 0
2- PortB = A
3- Wait 50us
3- Increment A
4- Loop to 2

At pin 0: duration of 0 and 1 is 50us
At pin 1: duration of 0 and 1 is double of those of pin 0 = 100us
At pin 2: duration of 0 and 1 is double of those of pin 1 = 200us

---
include 'mc9s12dg256.inc'

org $1500
lds #$1500

movb DDRA,#%11111111 ;configure port A as output
clrb

Forever:
    stab PORTA
    jsr delayby50us
    incb
    bra forever
Outline

4.1 Basic Concepts of Parallel I/O Ports

4.2 Interfacing with Simple I/O Devices

4.4 Advanced Parallel I/O Devices
- Configurations A and B can be used when the LED needs 1mA to 2mA to produce enough brightness
- Configuration C is used when the LED needs more than 2mA
- Buffer is current amplifier. Used when the port current is not sufficient
- The resistors R1, R2, are R3 are used to limit the current flowing to the LED. R1 and R2 can be 1.5 K Ohm to 2K Ohm

Figure 4.15 An LED connected to a CMOS inverter through a current-limiting resistor.
LED Circuit in the Draong12-Plus Demo Board

Port J pin 1 is low, the LEDs are enabled to light
Port J pin 1 is high, the LEDs are disabled
Write a program to drive the LEDs on Dragon 12 board, so that one LED lights at a time from top to bottom and then from bottom to top with each LED lighted for half a second. E-Clock is 24MHz

- To turn on the LEDs one at a time, we output the values %1000 0000, %0100 0000, ……………… %0000 0001, which are equivalent to $80, $40, …,$01.

- Only one bit is 1 and the LED that corresponds to this pin lights

Steps

1- Configure port B and pin 1 or port J as output.
2- Store the values $80, $40, …. $01 in a table. Use register X to point to the starting address of the table. PJ1 pin is low to enable the LEDs
2- Output the value pointed by X to port B and increment X
3- Wait half a second
4- If X points to the end of the table, let X point to the beginning of the table
5- Go to step 2
include 'mc9s12dg256.inc'

org $1000

lpcnt ds.b 1

led_tab dc.b $80,$40,$20,$10,$08,$04,$02,$01
dc.b $01,$02,$04,$08,$10,$20,$40,$80

org $1500

Start: movb #$FF,DDRB ; configure port B for output
bset DDRJ,$02 ; configure PJ1 pin for output
bclr PTJ,$02 ; enable LEDs to light
movb #$FF,DDRP ; disable 7 segments that are connected
movb #$0F,PTP ;

forever: movb #16,lpcnt ; initialize LED pattern count
ldx #led_tab ; Use X as the pointer to LED pattern table

led_lp: movb 1,x+,PORTB ; turn on one LED

ldy #1000 ; wait for half a second
jsr delayby1ms ;

dec lpcnt ; reach the end of the table yet?
bne led_lp
bra forever ; start from beginning
; Make a delay for 1ms, given that E-clock = 24MHz. Each E-clock time interval is 41.67 ns
; To make this delay, we need 1ms/41.67 ns E-clocks = 24000 E-clocks. One way to do that is
; a loop for 1000 times for a sequence of code that needs 24 E-cycles

delayby1ms:
pshx

outerloop: ldx #1000  

innerloop: psha    ; 2 E cycles
           pula     ; 3 E cycles
           psha    ; 2 E cycles
           pula    ; 3 E cycles
           psha    ; 2 E cycles
           pula    ; 3 E cycles
           psha    ; 2 E cycles
           pula    ; 3 E cycles
           nop     ; 1 E cycle
           nop     ; 1 E cycle
           nop     ; 1 E cycle
           nop     ; 1 E cycle
           dbne x,innerloop
           dbne y,outerloop
           pulx
           rts

24 E-clocks

24,0000 E-clocks = 1 ms delay

Y x 1 ms delay
2- Interfacing with Seven-Segment Displays

- Seven-segment displays are mainly used to display decimal digits and a small set of letters.
- In dragon board, segments a-g are connected to the pins PB0 to PB6
- The microcontroller must send an appropriate value to Port B in order to display a certain value.

<table>
<thead>
<tr>
<th>Decimal digit</th>
<th>Segments</th>
<th>Value to send to port B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 1 1 1 1 1 1</td>
<td>$3F</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 0 1 1 0</td>
<td>$06</td>
</tr>
<tr>
<td>2</td>
<td>1 0 1 1 0 1 1</td>
<td>$5B</td>
</tr>
<tr>
<td>3</td>
<td>1 0 0 1 1 1 1</td>
<td>$4F</td>
</tr>
<tr>
<td>4</td>
<td>1 1 0 0 1 1 0</td>
<td>$66</td>
</tr>
<tr>
<td>5</td>
<td>1 1 0 1 1 0 1</td>
<td>$6D</td>
</tr>
<tr>
<td>6</td>
<td>1 1 1 1 1 0 1</td>
<td>$7D</td>
</tr>
<tr>
<td>7</td>
<td>0 0 0 0 1 1 1</td>
<td>$07</td>
</tr>
<tr>
<td>8</td>
<td>1 1 1 1 1 1 1</td>
<td>$7F</td>
</tr>
<tr>
<td>9</td>
<td>1 1 0 1 1 1 1</td>
<td>$6F</td>
</tr>
</tbody>
</table>
Write a program to make one 7-segment display count from 0 to 9 and repeats. The display should change every one second.

**Steps:**
1. Make a table having a to g segments’ values for the digits 0, 1, 2, ..9
2. Enable only one display.
3. X points at the beginning of the table
4. Output the value pointed by X to port B
5. Wait one second
6. If X points to the last element, then X = the starting address of the table
include 'mc9s12dg256.inc'

Org $1000
DispTab dc.b $3F, $06, $5B, $4F, $66, $6D, $7D, $07, $7F, $6F

org $1500
Start: lds #$2000
movb #$FF,DDRB ; Configure port B as output port
movb #$0F,DDRP ; configure PP0 to PP3 as output pins
movb #$FE,PTP ; enable the first display

forever: ldx #DispTab ; set X to point to the display table
loopi: movb 1,X+,PORTB ; output segment pattern

ldy #1000
jsr delayby1ms ; wait for 1 second

cpx #DispTab+10 ; reach the end of the table?
bne loopi
bra forever
RTI can make interrupt every one second. Repeat previous program but by using

Strategy

- Main program
  - Initialize SP
  - Set up ports B and P for output
  - Set up RTI rate
  - Initialize RTI vector
  - Enable RTI interrupts (RTIE)
  - Turn on interrupt system (CLI)
  - Loop forever

- RTI routine
  - Clear RTIF
  - Copy the current digit to Port B
  - X points to the next digit
  - RTI
include 'mc9s12dg256.inc'
Org $1000
DispTab dc.b $3F, $06, $5B, $4F, $66, $6D, $7D, $07, $7F, $6F
rti_count ds.b 1
count ds.b 1
org $FFF0
dc.w RTI_ISR ; load RTI_ISR vector
   org 2000
clr count
clr rti_count ; Configure ports
   movb #$FF,DDRB ; Configure port B as output port
   movb #$0F,DDRP ; configure PP0 to PP3 as output pins
   movb #$FE,PTP ; enable the first display
   movb #$FF,DDRJ ; disable 7 segments
   movb #$2,PTJ
; configure RTI
   movb #$7F,RTICTL ; set up slowest RTI rate
   bset CRGINT,#$80 ; RTIE=1 (enables RTI interrupts)
   lds #$4000 ; initialize the stack pointer
   andcc #%10111111 ; enable nonmaskable interrupts
   cli ; enable interrupt systems
Loop: bra Loop ; wait for interrupt
We use memory location “count” instead of register X because Interrupt routines do not memorize X resulted from last routine call. Because it saves X at the beginning of the routine and return the original value back at the end.

RTI_ISR:
    movb #$80,CRGFLG ;clear RTIF by writing a 1 to it.
    inc rti_count
    ldaa rti_count
    cmpa #8 ;check if count =8
    bne RTI_done ;if no, we are done
    clr rti_count ;if yes clear count and
    ; a second is every 8 interrupts
    ldx #DispTab
    ldaa count
    movb a,x,PORTB ; output segment pattern
    inc count
    ldaa count
    cmpa #10
    bne RTI_done
    clr count
    RTI_done: rti

movw #RTI_ISR,$FFF0 did not work use org as shown in previous slide
Displaying Multiple Seven-Segment Displays

- A time-multiplexing technique is used to display multiple digits simultaneously.

- Light one digit for a short period of time and then switch to the next digit.

- Within a second, each digit is lighted in turn many times. Due to the persistence of vision, all digits appear to be lighted at the same time.

![Circuit Diagram]

Only one bit in PP0 – PP$ is zero to enable one display at a time.
- The common cathodes of all the segments are connected to port P through the buffer chip 74HC367

- For Port P, to enable one display at a time, only one bit 0 and the other bits are 1. The display its cathode is connected to 0 is on.

- To display 3 on the display #3

  movb #$FF,DDRB     ; configure Port B for output
  movb #$0F,DDRP     ; configure P0-P3 for output
  movb #$07,PTP      ; enable display #3 to be lighted
  movb #$4F,PORTB    ; send out the segment pattern of 7
Write a program to display 3456 on the four seven-segment displays in the dragon board

Steps:-
1- Define a table with the values that should be outputted at ports B and P

<table>
<thead>
<tr>
<th>Seven-Segment Display</th>
<th>Displayed BCD Digit</th>
<th>Port B</th>
<th>Port P</th>
</tr>
</thead>
<tbody>
<tr>
<td>#3</td>
<td>3</td>
<td>$7D</td>
<td>$07</td>
</tr>
<tr>
<td>#2</td>
<td>4</td>
<td>$6D</td>
<td>$0B</td>
</tr>
<tr>
<td>#1</td>
<td>5</td>
<td>$66</td>
<td>$0D</td>
</tr>
<tr>
<td>#0</td>
<td>6</td>
<td>$4F</td>
<td>$0E</td>
</tr>
</tbody>
</table>

2- Set X points at the address of the first element in the table
3- Output the byte pointed by X to Port B then increment X
4- Output the byte pointed by X to Port P then increment X
5- Wait 1ms
6- If X points at the last location go to 2 else go to 3
include 'mc9s12dg256.inc'

Org $1000
DispTab dc.b $7D,$07
dc.b $6D,$0B
dc.b $66,$0D
dc.b $4F,$0E

org $1500
Start: lds #$2000
        movb #$FF,DDRB
        movb #$0F,DDRP

forever: ldx #DispTab ; set X to point to the display table

loopi:   movb 1,x+,PORTB   ; output segment pattern
          movb 1,x+,PTP    ; output display select

        ldy #1
        jsr delayby1ms ; wait for 1 ms

cpx #DispTab+8 ; reach the end of the table?
bne loopi
bra forever
Repeat previous program but with using RTI

```
include 'mc9s12dg256.inc'

Org $1000
DispTab dc.b $7D,$07
    dc.b $6D,$0B
    dc.b $66,$0D
    dc.b $4F,$0E

count ds.b 1
    org $FFF0
    dc.w RTI_ISR ;load RTI ISR vector
    org $1500
    lds #$4000 ; initialize the stack pointer
c    clr count

; Configure ports
    movb #$FF,DDRB  ; Configure port B as output port
    movb #$0F,DDRP  ; configure PP0 to PP3 as output pins
    movb #$FE,PTP   ; enable the first display
    movb #$FF,DDRJ  ; disable 7 segments
    movb #$2,PTJ

; configure RTI
    movb #$10,RTICTL ;10 instead of 7F low delay is needed
    bset CRGINT,#$80 ; RTIE=1 (enables RTI interrupts)
```
; enable interrupts
    andcc #10111111 ; enable nonmaskable interrupts
    cli ; enable interrupt systems

Loop: brset PTH,#00000001,Loop ; if switch off go to loop waiting for interrupt
    ; switch is connected so change the status of the interrupt on to off or off to on
    ldaa CRGINT
    eora #$80 ; bit 7 is enables RTI interrupts
    staa CRGINT
    bra Loop ; wait for interrupt

RTI_ISR:
    movb #$80,CRGFLG ; clear RTIF by writing a 1 to it.
    ldx #DispTab
    ldaa count
    movb a,x,PORTB ; output segment pattern
    inc count
    ldaa count
    movb a,x,PTP ; output segment pattern
    inc count
    ldaa count
    cmpa #8
    bne RTI_done
    clr count

RTI_done: rti
A question in the assignment or lab:-

Write a program to count on the displays from 00 to 99 and repeats. The 7 segment is updated every one second.

**Steps**

**Main program**

1- Value = 0, table should have 7 segment values for 0, 1, ...9
2- Configure RTI interrupt for 1 second
3- Enable display 1, output the value in the table pointed by X + BCD1
4- Enable display 2, output the value in the table pointed by X + BCD2
5- Loop to 3

**RTI routine**

every 1 second increment Value. BCD addition to 1 using dda

<table>
<thead>
<tr>
<th>Value</th>
<th>display</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$3F</td>
</tr>
<tr>
<td>1</td>
<td>$06</td>
</tr>
<tr>
<td>2</td>
<td>$5B</td>
</tr>
<tr>
<td>3</td>
<td>$4F</td>
</tr>
<tr>
<td>4</td>
<td>$66</td>
</tr>
<tr>
<td>5</td>
<td>$6D</td>
</tr>
<tr>
<td>6</td>
<td>$7D</td>
</tr>
<tr>
<td>7</td>
<td>$07</td>
</tr>
<tr>
<td>8</td>
<td>$7F</td>
</tr>
<tr>
<td>9</td>
<td>$6F</td>
</tr>
</tbody>
</table>
A question in the assignment or lab:

Write a program to count on the displays the numbers of pulses from a sensor (e.g., numbers of customers, cars, boxes, ....)

Steps

Main program

1- Value = 0, table should have 7 segment values for 0, 1, ...9
2- Configure IRQ interrupt
3- Enable display 1, output the value in the table pointed by X + BCD
4- Enable display 2, output the value in the table pointed by X + BCD
5- loop to 3

IRQ routine

1- Each time a pulse comes, the IRQ routine is called.
2- Increment Value (BCD addition) by 1
   ldaa Value
   adda #1
   daa

table:

<table>
<thead>
<tr>
<th>Value</th>
<th>BCD2</th>
<th>BCD1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A question in the assignment or lab:-

Write a program to program a clock on the 7 segments. The first two displays are for seconds and the second two displays are for minutes.

Steps

Main program

1- Seconds = 0, Minutes = 0, table should have 7 segment values for 0, 1, ... 9

2- Configure RTI interrupt for 1 second

3- Enable display 1, output the value in the table pointed by $X + \text{BCD}_1$

4- Enable display 2, output the value in the table pointed by $X + \text{BCD}_2$

5- Enable display 3, output the value in the table pointed by $X + \text{BCD}_3$

6- Enable display 4, output the value in the table pointed by $X + \text{BCD}_4$

7- loop to 3
RTI routine

1- Every 1 second increment Seconds (BCD addition with 1)

2- if Seconds = 60 BCD, then Seconds = 00 and Minutes = Minutes +1 (BCD increment)
- In Dragon board there is a buzzer connected to pin 5 in port T

- A sound can be generated by creating a digital waveform in the audible range and using it to drive a speaker or a buzzer.

- By alternating the frequency of the generated waveform between two different values, a two tone siren can be generated

- The duration of the siren tone is variable

- The siren would sound more urgent if the tone duration were shorter

Example Write a program to generate a two-tone siren that alternates between 250Hz and 500 Hz with each tone lasting for half of a second.
include 'mc9s12dg256.inc'

org $1500
lds #$1500
bset DDRT,#%00100000; configure PT5 pin for output

forever:

  ldx #250 ; repeat 500 Hz waveform 250 times
tone1: bset PTT,#%00100000 ; pull PT5 pin to high
  ldy #1
  jsr delayby1ms
  bclr PTT,#%00100000
  ldy #1
  jsr delayby1ms
  dbne x,tone1

  ldx #125 ; repeat 250 Hz waveform for 125 times
tone2: bset PTT,#%00100000
  ldy #2
  jsr delayby1ms
  bclr PTT,#%00100000
  ldy #2
  jsr delayby1ms
  dbne x,tone2
bra forever
4- Interfacing with DIP switches and push buttons

- In the dragon board, the four push buttons are connected to PH0 – PH3 while the eight dip switches are connected to PH0 PH7

1. Push button: 1 released, 0 pressed
2. DIP switch: 1 open, 0 closed

The pin is zero if DIP switch or push button is connected. No way to know which one caused the zero.
- How to read the status of the switches

```assembly
movb  #0,DDRH  ; configure port A for input
ldaa PTH     ; read into accumulator A
```

- Check the individual bits of A to know the status of the switches.

- A switch is closed if the corresponding bit is 0

Example: Develop a program which continuously monitors the dip switches on the Dragon12 and echo the switches on the LEDs. A LED is on if the corresponding switch is connected.

**Steps:**
1. Configure port B (LEDs’ port) and pin1 in port J (to enable LEDs) as output
2. Configure port H (switches’ port) as input.
3. Read switches at port H, and complement the number to make 1 corresponds to connected switch and thus corresponding LED turns on
4. Send the number to LEDs at port B
5. Go to 3
include 'mc9s12dg256.inc'

org $1500

movb #$FF,DDRB ; configure port B for output
bset DDRJ,$02  ; configure PJ1 pin for output
bclr PTJ,$02   ; enable LEDs to light
movb #0,DDRH   ; configure port A for input
movb #$FF,DDRP ; disable 7 segments that are connected
movb #$0F,PTP  ; 

forever:
    ldaa PTH       ; read switches into accumulator A
    coma           ; bit = 1 when a switch is pressed so
                    ; that a LED is on when switch is connected
    staa PORTB
    bra forever    ; start from beginning

In next slide, we update the program to use switch 0 as on/off switch. If the switch is connected, the LEDs stop displaying the status of the switches.
include 'mc9s12dg256.inc'

org $1500

movb #$FF, DD RB           ; configure port B for output
b set  DDRJ, $02           ; configure PJ1 pin for output
b cl r  PTJ, $02           ; enable LEDs to light
movb #0, DDRH              ; configure port A for input
movb #$FF, DDRP            ; disable 7 segments that are connected
movb #$0F, PTP             ; '

forever:

br clr  PTH, %#00000001, forever ; test bit0 if 0 do not update LEDs

ldaa PTH                   ; read switches into accumulator A
coma                       ; bit =1 when a switch is pressed so
                          ; that a LED is on when switch is connected
staa PORTB
bra forever                ; start from beginning

In next slide, we update the program to use push button on/off switch. OFF when the button is pressed – ON again when button is pressed
include 'mc9s12dg256.inc'

Org $1000
Onoff dc.b 0 ; Onoff = FF on, Onoff = 00 off
org $1500

movb #$FF,DDRB ; configure port B for output
bset DDRJ,$02 ; configure PJ1 pin for output
bclr PTJ,$02 ; enable LEDs to light
movb #0,DDRH ; configure port A for input
movb #$FF,DDRP ; disable 7 segments that are connected
movb #0,DDRH ; configure port A for input

forever: brset PTH,#%00000001,start
    ; switch is pressed
ldaa Onoff ; complement onoff
coma
staa Onoff

start: ldab Onoff
cmpb #0
beq forever ; do not update LEDSs
ldaa PTH ; read switches into accumulator A
coma    ; bit =1 when a switch is pressed so
        ; that a LED is on when switch is connected
staa PORTB
bra forever ; start from beginning
Write a program to turn on LED number 0 and turn off the other LEDs initially. Each time a push button number 3 is pressed, the LED is off and the next one on left is on. Always only one LED is on and each switch is connected to the on LED shifts to left.

```assembly
org $1000

movb #$FF,DDRB       ; port b is output
bset DDRJ,#$02       ; configure PJ1 pin for output
bclr PTJ,#$02        ; enable LEDs to light

movb #%00000000,DDRH ; port H is input - push buttons

movb #%00000001,PORTB ; first LED is on

LOOP: brset PTH,#%00001000,LOOP ; loop as long as the switch is not pressed
    lsl PORTB
    bne SKIP
    movb #%00000001,PORTB ; reinitialize to 01h
SKIP: bra LOOP
```
Update the counting segment program in slide 4-24 as follows. When a switch is connected counting works, otherwise it stops

A very simple idea: turn RTI interrupt off when the switch is disconnected and interrupt on when it is connected

In the program in slide 4-24, add:
moveb #0,DDRH ; port H is input - push buttons

replace this code (Loop: bra Loop) with:-

Loop:
    brset PTH,#%00000001,Loop
; change the status of the interrupt on to off or off to on

    ldaa CRGINT
    eora #$80  ; change the status of bit 7 to enables/disable RTI interrupts
    staa CRGINT

    bra Loop   ; wait for interrupt
Write a program to increment the number on two 7-segment displays when a push button is pressed and decrement the number when another push button is pressed

1- Read the switches  
2- If increment button is pressed, increment the display  
3- If decrement button is pressed, decrement the display

Write a program to continuously read the value on switches and display it on 3 seven segment displays max. should be 255

1- Read the switches, and complement the number.  
2- Convert the binary number to BCD  
   2.1 divide the number by 10 then BCD1 = remainder  
   2.2 divide the quotient by 10 then BCD2 = remainder  
   2.3 divide the quotient by 10 then BCD3 = remainder  
3- display the three BCDs
Write a program to display a number 7 on display 3 and each time a button is pushed the number moves to the next display

1- Port B = $07 to display 7
2- Port P = 0000 0001 to enable display 3
3- If the switch is not pressed, does nothing
4- If the switch is pressed, shift left for port P to enable next display, If P = 00010000, then P = 0000 0001
Outline

4.1 Basic Concepts of Parallel I/O Ports

4.2 Interfacing with Simple I/O Devices

4.4 Advanced Parallel I/O Devices
A common problem in mechanical switches

When a switch is asserted, we expect a signal like this:

However, the actual signal is:

Switch timing showing bounce on touch and release.

For a short period of time, the switch signal is bouncing.
Why? because switch contacts do not come to rest immediately

**Problem:** although it is one press, the microcontroller may consider each low voltage as one press.

Humans cannot press a switch several time in 20 ms, but because the microcontroller is so fast, it can read several low voltages in 20 ms

**Solution: De-bouncing**

**1- Software solution:**

- The switch is closed if the voltage is low for about 10ms
- The switch is open after the voltage is high for about 10ms.

- **Wait and see:** the program waits for about 10 ms and re-examines the same key again to see if it is still pressed.
K2:
  brclr PTH,#%00000001,Test2 ; The branch is taken when bit 0 in port H is zero

  bra K2  ; If bit 0 is 1, go to K2

Test2:
  ; de-bouncing: wait and re-examine

  LDY 15

  JSR delayby1ms

  brclr PTH,#%00000001,realpress ; The branch is taken when bit 0 in port H is zero

  bra K2  ; If bit 0 is 1, go to K2

Realpress:
  ; It comes here when the first pushbutton switch is pressed
; De-bouncing when IRQ interrupt comes from a switch

IRQ_ISR:

; when interrupt happens, wait for some time and re-examine
to make sure it is real interrupt not bouncing

ldy #30

JSR delayby1ms

; if IRQ pin (pin2 in port E is 0, then it is real interrupt go
to execute the IRQ routine)

brclr PORTE,#%00000010,realbounce

rti ;if IRQ pin = 1, this is bouncing leave the routine

realbounce:

Your IRQ code here

rti
2- H/W solutions

Several methods.

A simple solution: using a capacitor

- When the switch is closed, $V_{out} = 0$
- Because of bouncing the capacitor charges for a short time making a small voltage on the capacitor.
- This small voltage will be interrupted as logic 0

(c) Integrating RC circuit debouncer
Interfacing the HCS12 to a Keypad

- Array of switches.
- **When a switch is pressed, it connects two conductors one row and one column**

![Keypad diagram](image)

The keypad’s connection in Dragon board

Input pins:
- PA3
- PA2
- PA1
- PA0

Output pins:
- PA4
- PA5
- PA6
- PA7
Is any key pressed?

1- PA4 ~ PA7 = 1111
2- Read PA0 ~ PA3
3- If PA0 ~ PA3 = 0000, no key is pressed, else a key is pressed

0 PA0 ~ PA3 = 1000, a key in column one is pressed {1, 4, 7, or *}

0 PA0 ~ PA3 = 0100, a key in column two is pressed {2, 5, 8, or 0}

0 PA0 ~ PA3 = 0010, a key in column three is pressed {3, 6, 9, or #}

0 PA0 ~ PA3 = 0001, a key in column four is pressed {A, B, C, or D}
Which key is pressed?

Scan rows and columns

1- Test if there is a key connected on row 1

(1) PA7 ~ P4 = 0001

Put 1 at the row you wanna test

(2) Check PA0 to PA3

If all of them are zeros, then there is no key pressed in row 1 because if a key is pressed, it would connect the one at PA4 to one pin in PA0~PA3.

Assume key 9 is connected

Move to the next row to check if a key is pressed
1- Test if there is a key connected on row 2

(1) \[ PA7 \sim P4 = 0010 \]

Put 1 at the row you wanna test

(2) Check PA0 to PA3

If all of them are zeros, then there is no key pressed in row 2

because if a key is pressed, it would connect the one at PA5 to one pin in PA0~PA3.

Assume key 9 is connected

Move to the next row to check if a key is pressed
Which key is pressed?

Scan rows and columns

1- Test if there is a key connected on row 3

(1) PA7 ~ P4 = 0010

Put 1 at the row you wanna test

(2) Check PA0 to PA3

Since PA0 to PA3 does not equal to 0000, then there is a key connected in row 3

Done 🧄

Assume key 9 is connected

Giving that PA6 is connected to PA1, we can know that switch 9 is pressed
Computing the ASCII key of the pressed key

1- Define a table with the keys’ ASCIIIs

2- Using the pressed key’s row number and column number, get the pressed key’s ASCII

\[
\begin{align*}
\text{KCODE0 DC.B '123A'} & \quad \text{If row = 0, } X = \text{KCODE0} \\
\text{KCODE1 DC.B '456B'} & \quad \text{If row = 1, } X = \text{KCODE1} \\
\text{KCODE2 DC.B '789C'} & \quad \text{If row = 2, } X = \text{KCODE2} \\
\text{KCODE3 DC.B '*0#D'} & \quad \text{If row = 3, } X = \text{KCODE3}
\end{align*}
\]

The pressed Key’s ASCII = X + column number

Shift A, increment X , repeat until 1 is found (the column is reached)

SHIFT:

\[
\begin{align*}
\text{LSRA} & \quad ; \text{LOGICAL SHIFT RIGHT PORTA} \\
\text{BCS MATCH} & \quad ; \text{IF CARY SET COLUMN IS FOUND} \\
\text{INX} & \quad ; \text{IF CARY NOT CLEAR INCREMENT X} \\
\text{BRA SHIFT} & \quad ; \text{SHIFT RIGHT UNTIL CARY IS CLEAR.}
\end{align*}
\]

Match:
Write a subroutine called Keypad to read a character from the keypad and return the ASCII of the character in register A

keypad:
; Read one character from keypad and returns it in A
    MOVB #$F0, DDRA  ; Configure PA0-PA4 input as and PA5-PA7 output
    PSHY
    PSHB

    ;; 1- If the user presses and holds a key down, this must be one press
    K1:    MOVB #%11110000, PORTA  ; Set rows high
            LDAA PORTA    ; Capture port a
            ANDA #%00001111 ; Mask out rows
            CMPA #$00     ; If columns is zero no button pressed
            BNE K1        ; Do not move on until no button is pressed

    ;; 2- Check if a switch is pressed
    K2:     
            LDY 15
            JSR delayby1ms ; wait 15 ms
            LDAA PORTA    
            ANDA #%00001111 ;
            CMPA #$00     ; If cols !=0 then a button is pressed
            BEQ K2        ; If no button pressed keep checking
; FOR DEBOUNCING, WAIT 15ms AND CHECK AGAIN

LDY 15

JSR delayby1ms

LDAA PORTA ; READ PORT A

ANDA #%00001111 ; MASK OUT ROWS

CMPA #$00 ; CHECK FOR PRESS AFTER DEBOUNCE

BEQ K2 ; IF NO PRESS AFTER DEBOUNCE GO BACK

; 3- A SWITCH IS PRESSED – FIND THE ROW

OVER1:

;;;; Test Row 0

LDAA #%00010000 ; MAKE HIGH ROW0 THE REST GROUNDED

STAA PORTA ;

LDAB #$08 ; SET COUNT TO PROVIDE SHORT DELAY FOR STABILITY

; AFTER CHANGING THE PORT A OUTPUT

P1: DECB ; DECREMENT COUNT

BNE P1 ; IF COUNT NOT ZERO KEEP DECREMENTING

LDAA PORTA ; READ PORTA

ANDA #%00001111 ; MASK OUT ROWS

CMPA #$00 ; IS INPUT ZERO?

BNE R0 ; IF COLUMNS NOT ZERO THEN BUTTON IS IN ROW 0
Architect:  Test Row 1
LDAA  %00100000  ;IF ZERO THEN BUTTON NOT IN ROW0
STAA  PORTA     ;TURN ON ROW 1 TURN OFF ALL OTHERS
LDAB  $08       ;SHORT DELAY TO STABALIZE AFTER CHANGING THE PORT A OUTPUT
P2:  DECB        ;DECREMENT COUNT
     BNE  P2      ;IF COUNT NOT ZERO KEEP DECREMENTING
     LDAA  PORTA ;READ PORT A
     ANDA  %00001111 ;MASK OUT ROWS
     CMPA  $00    ;CHECK FOR KEY PRESS
     BNE  R1      ;IF Pressed KEY IS IN ROW1

Architect:  Test Row 2
LDAA  %01000000  ;IF ZERO BUTTON NOT IN ROW1
STAA  PORTA     ;TURN ON ROW2 ALL OTHERS OFF
LDAB  $08       ;SHORT DELAY TO STABALIZE PORTA
P3:  DECB        ;DECREMENT COUNT
     BNE  P3      ;DELAY LOOP
     LDAA  PORTA ;READ PORTA
     ANDA  %00001111 ;MASK OUT ROWS
     CMPA  $00    ;CHECK FOR PRESS
     BNE  R2      ;IF FOUND KEY IS IN ROW2
LDAA #%10000000 ; IF ZERO MOVE TO ROW3
STAA PORTA ; TURN ON ROW3 ALL OTHERS OFF
LDAB #$08 ; SHORT DELAY TO STABALIZE OUTPUT

P4:
DECB ; DECREMENT DELAY
BNE P4 ; DELAY LOOP
LDAA PORTA ; READ PORT A
ANDA #%00001111 ; MASK OUT ROWS
CMPA #$00 ; CHECK FOR PRESS
BNE R3 ; IF FOUND KEY IN ROW3
BRA K2 ; IF ROW NOT FOUND GO BACK TO START

R0: LDX #KCODE0 ; LOAD POINTER TO ROW0 ARRAY
BRA FIND ; GO FIND COLUMN

R1: LDX #KCODE1 ; LOAD POINTER TO ROW1 ARRAY
BRA FIND ; GO FIND COLUMN

R2: LDX #KCODE2 ; LOAD POINTER TO ROW2
BRA FIND ; GO FIND COLUMN

R3: LDX #KCODE3 ; LOAD POINTER TO ROW3
BRA FIND ; GO FIND COLUMN
FIND:    ;We knew the row number. Now we need to know the column number to get the key ASCII

ANDA #%00001111   ;MASK OUT ROWS   A = 0000 0001 or A = 0000 0010 or A = 0000 0100 or A = 0000 1000

SHIFT:                ;LOGICAL SHIFT RIGHT PORTA
    LSRA                ;IF CARY SET COLUMN IS FOUND
    BCS MATCH
    INX                 ;IF CARY NOT CLEAR INCREMENT POINTER TO ROW ARRAY
    BRA SHIFT           ;SHIFT RIGHT UNTIL CARY IS CLEAR.

MATCH:                ;X point at the address of key’s ASCII
    LDAA 0,X
    ;LOAD ASCII FROM look up table

PULB
PULY
rts     ;end of keypad routine

; look up table
KCODE0 DC.B '123A'
KCODE1 DC.B '456B'
KCODE2 DC.B '789C'
KCODE3 DC.B '*0#D'
Do not forget this subroutine, it is needed in Keypad subroutine

delayby1ms:
   pshx
outerloop: ldx #1000
innerloop: psha ; 2 E cycles
   pula ; 3 E cycles
   psha ; 2 E cycles
   pula ; 3 E cycles
   psha ; 2 E cycles
   pula ; 3 E cycles
   psha ; 2 E cycles
   pula ; 3 E cycles
   pula ; 3 E cycles
   nop ; 1 E cycle
   nop ; 1 E cycle
   nop ; 1 E cycle
   nop ; 1 E cycle
   dbne x,innerloop
   dbne y,outerloop
   pulx
   rts
Using Keypad subroutine, write a program that reads a key from the keypad and display its ASCII on the LEDs

ABSENTRY Entry
INCLUDE 'mc9s12dp256.inc'

ORG $4000
Entry:

LDS #$4000 ;Initialize Stack

; ======== For LEDs =============
MOVB #$FF,DDRB ;MAKE PORTB OUTPUT
MOVB #$02,DDRJ ;ENABLE LED ARRAY ON PORTB OUTPUT
MOVB #$00,PTJ ;ENABLE LED ARRAY ON PORTB OUTPUT
MOVB #$00,PORTB ;INITIALIZE PORT B

; ======== For Seven segments =============
MOVB #$0F,DDRP ;
MOVB #$0F,PTP ;TURN OFF 7SEG LED

yy: BSR keypad ; returns the key’s ASCII in register A
STAA PORTB ;PUT ASCII TO PORTB
BRA yy ;BACK TO START
Interfacing the HCS12 to a LCD

- In dragon board, the LCD is connected to port K as shown in the figure.
- RS is connected to PK0.
  - RS = 0, write a command (clear, turn off, etc.) to instruction register (IR)
  - RS = 1, write data to data register (DR)
- E is the enable is connected to PK1
- 4 bit data (or command) are connected to PK2 ~ PK5

Figure 7.28 LCD interface example (4-bit bus, used in Dragon12)
- However, data and commands are 8 bits. We write to the LCD twice: 4 bits each time.
- The dragon LCD has two lines and 16 character per line.
- The LCD has a RAM and the data you wanna display should be written in the RAM.
- The address of the data RAM should be set up with a previous instruction.
- A counter is used to keep track of the address of the next RAM location to be accessed.
- To write to the LCD successfully, one must consider that the microcontroller is much faster than the LCD so delays will be used to consider this.
The procedure for writing a byte to the LCD data register

1- RS = high.
2- E = high.
3- Output the most significant 4 bits to PK2~PK5.
4- E = low and wait for some time to make sure that the internal operation is complete.
5- Repeat these procedures to send the least significant 4 bytes.

Write a subroutine called PUTCLCD, to display the character in register A in the LCD

```
PUTCLCD:     PSHA          ; save a copy of the data in the stack
            ANDA   #$F0   ; Clear the lower 4 bits – output the upper 4 bits first
            LSRA
            LSRA       ; match the upper 4 bits with the LCD
            STAA PORTK
            BSET PORTK,#$01 ; RS select LCD data register
            BSET PORTK,#$02 ; EN = high
            NOP         ; 3 NOP to extend the duration of EN
            NOP
            NOP
```
BCLR PORTK,#$02 ;EN = low

PULA ; repeat previous instructions to output the other 4 bits
ANDA #$0F
LSLA
LSLA
STAA PORTK
BSET PORTK,#$01 ;RS select LCD data register
BSET PORTK,#$02 ;EN = high
NOP
NOP
NOP
BCLR PORTK,#$02 ;E = low to complete the write cycle
LDY #$01 ;delay is needed until the LCD does all the internal
JSR delayby1ms ; operations
RTS

- To display a character, put its ASCII in register A and call this subroutine.

  LDAA #'H'
  JSR PUTCLCD

- Where the character will be written? As long as we did not specify it will be next to the last instruction.

- Later we will discuss that we can specify the location of the character to be written
Write a subroutine called CMD2LCD, to send a command in register A to the LCD

CMD2LCD :  PSHA ; save a copy of the command in the stack
ANDA  #$F0 ; Clear the lower 4 bits – output the upper 4 bits first
LSRA ; match the upper 4 bits with the LCD
LSRA ; two shifts because the 4 bits should start from bit 2
STAA PORTK
BCLR PORTK,#$01 ; RS select LCD instruction register
BSET PORTK,#$02 ; EN = high, 4 bits of the command are sent with RS and EN
NOP ; 3 NOP to extend the duration of EN
NOP
NOP
NOP

The procedure for writing a command to the IR register

1- RS = low.
2- E = low wait some time then E = high.
3- Output the most significant 4 bits to PK2~PK5.
4- E = low and wait for some time to make sure that the internal operation is complete.
5- Repeat these procedures to send the least significant 4 bytes
When you wanna send a command to the LCD, put the command in register A and call this subroutine.

```
BCLR PORTK,#$02 ;EN = low

PULA ; repeat previous instructions to output the other 4 bits
ANDA #$0F
LSLA
LSLA
STAA PORTK
BCLR PORTK,#$01 ;RS select LCD data register
BSET PORTK,#$02 ;EN = high
NOP
NOP
NOP
BCLR PORTK,#$02 ;E = low to complete the write cycle
LDY #$01 ;delay is needed until the LCD does all the internal
JSR delayby1ms ; operations
RTS
```
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Description</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear display</td>
<td>RS R/W B7 B6 B5 B4 B3 B2 B1 B0</td>
<td>Clears display and returns cursor to the home position (address 0).</td>
<td>1.64 ms</td>
</tr>
<tr>
<td>Cursor home</td>
<td>0 0 0 0 0 0 0 0 1 *</td>
<td>Returns cursor to home position (address 0). Also returns display being shifted to the original position. DDRAM contents remain unchanged.</td>
<td>1.64 ms</td>
</tr>
<tr>
<td>Entry mode set</td>
<td>0 0 0 0 0 0 0 1 I/D S</td>
<td>Set cursor move direction (I/D), specifies to shift the display (S). These operations are performed during data read/write.</td>
<td>40 µs</td>
</tr>
<tr>
<td>Display on/off control</td>
<td>0 0 0 0 0 0 1 D C B</td>
<td>Sets on/off of all display (D), cursor on/off (C) and blink of cursor position character (B).</td>
<td>40 µs</td>
</tr>
<tr>
<td>Cursor/display shift</td>
<td>0 0 0 0 0 0 1 S/C/R/L *</td>
<td>Sets cursor-move or display-(S/C), shift direction (R/L). DDRAM contents remains unchanged.</td>
<td>40 µs</td>
</tr>
<tr>
<td>Function set</td>
<td>0 0 0 0 1 DL N F *</td>
<td>Sets interface data length (DL), number of display line (N) and character font (F).</td>
<td>40 µs</td>
</tr>
<tr>
<td>Set CGRAM address</td>
<td>0 0 0 1</td>
<td>Sets the CGRAM address. CGRAM data is sent and received after this setting.</td>
<td>40 µs</td>
</tr>
<tr>
<td>Set DDRAM address</td>
<td>0 0 1</td>
<td>Sets the DDRAM address. DDRAM data is sent and received after this setting.</td>
<td>40 µs</td>
</tr>
<tr>
<td>Read busy flag and address counter</td>
<td>0 1 BF</td>
<td>Reads busy flag (BF) indicating internal operation is being performed and reads CGRAM or DDRAM address counter contents (depending on previous instruction).</td>
<td>0 µs</td>
</tr>
<tr>
<td>Write CGRAM or DDRAM</td>
<td>1 0</td>
<td>Writes data to CGRAM or DDRAM.</td>
<td>40 µs</td>
</tr>
<tr>
<td>Read from CGRAM or DDRAM</td>
<td>1 1</td>
<td>Reads data from CGRAM or DDRAM.</td>
<td>40 µs</td>
</tr>
</tbody>
</table>
Table 7.6 LCD instruction bit names

<table>
<thead>
<tr>
<th>Bit name</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/D</td>
<td>0 = decrement cursor position, 1 = increment cursor position</td>
</tr>
<tr>
<td>S</td>
<td>0 = no display shift, 1 = display shift</td>
</tr>
<tr>
<td>D</td>
<td>0 = display off, 1 = display on</td>
</tr>
<tr>
<td>C</td>
<td>0 = cursor off, 1 = cursor on</td>
</tr>
<tr>
<td>B</td>
<td>0 = cursor blink off, 1 = cursor blink on</td>
</tr>
<tr>
<td>S/C</td>
<td>0 = move cursor, 1 = shift display</td>
</tr>
<tr>
<td>R/L</td>
<td>0 = shift left, 1 = shift right</td>
</tr>
<tr>
<td>DL</td>
<td>0 = 4-bit interface, 1 = 8-bit interface</td>
</tr>
<tr>
<td>N</td>
<td>0 = 1/8 or 1/11 duty (1 line), 1 = 1/16 duty (2 lines)</td>
</tr>
<tr>
<td>F</td>
<td>0 = 5x8 dots, 1 = 5 x 10 dots</td>
</tr>
<tr>
<td>BF</td>
<td>0 = can accept instruction, 1 = internal operation in progress</td>
</tr>
</tbody>
</table>
; CLEAR DISPLAY, CURSOR AT THE BEGINNING, WRITES FROM THE
; BEGINNING
LDAA #$1
JSR CMD2LCD

; Return cursor to home position (left hand side) without 
; changing the content
; when you write, you overwrite the LCD contents
LDAA #$2
JSR CMD2LCD

; I/D (bit 1) bit is 0 - after each character you output, the 
cursor is decremented (move to left)
LDAA #$00000100
JSR CMD2LCD

; I/D (bit 1) bit is 1 - after each character output, the 
cursor is incremented (move to right)
LDAA #$00000110
JSR CMD2LCD
; S = 1 (bit 0), cursor is frozen, write from left to right
; with shifting the number to left
LDAA #%00000111
JSR CMD2LCD

; S = 0 (bit 0), cursor is shifted to right each time a
character is written
LDAA #%00000110
JSR CMD2LCD

; C bit = 0, cursor is off
LDAA #%00001100
JSR CMD2LCD

; C bit = 1, cursor is on
LDAA #%00001110
JSR CMD2LCD

; B bit = 0 cursor blinking is off
LDAA #%00001110
JSR CMD2LCD

; B bit = 1 cursor blinking is on
LDAA #%00001111
JSR CMD2LCD
; D bit = 0, turn off display
LDAA #%00001000
JSR CMD2LCD

; D bit = 1 turn on display it will display the last number
LDAA #%00001100
JSR CMD2LCD

; S/C =0 (bit3) to move cursor - R/L (bit2)= 0 for shift left
; - when write it writes to the new location
LDAA #%00010000
JSR CMD2LCD

; S/C =0 (bit3) to move cursor - R/L (bit2)= 1 for shift right - when write it writes to the new location
LDAA #%00010100
JSR CMD2LCD

; S/C =1 (bit3) to shift display shift the values on the display - R/L (bit2)= 0 for shift left - when write it writes to the new location - can be used in moving the display
LDAA #%00011000
JSR CMD2LCD
; S/C =1 (bit3) to shift display - R/L (bit2)= 1 for shift right - when write it writes to the new location
LDAA #%00011100
JSR CMD2LCD

; THE BEGINING OF NEXT LINE NEXT LINE
LDAA #$C0
JSR CMD2LCD

How to write in a specific locations

;-- write zero at digit 16 (1111) in line 0 because bit 6 =0
LDAA #%10001111
JSR CMD2LCD ;Set the starting address to display information
           ; to digit 16 line 0
LDAA #$30
JSR PUTCLCD ; write 0

;-- write zero at digit 16 (1111) in line 1 because bit 6 =1
LDAA #%11001111
JSR CMD2LCD ;set the address to digit 16 line 1
LDAA #$30
JSR PUTCLCD ; write 0
Addresses:

LDAA #%10001111
JSR CMD2LCD

Line 0, bit 6 = 0 and the first 4 bits takes numbers from 0000 to 1111 for the 16 locations in line 0

Line 1, bit 6 = 1 and the first 4 bits takes numbers from 0000 to 1111 for the 16 locations in line 1
LCD should be configure before it is used. The following subroutine can configure the LCD.

CONFIGLCD: MOVB #$FF,DDRK; configure port K for output

    LDY #$10
    JSR delayby1ms

    LDAA #$28 ; set 4 bit data LCD - two line display - 5x8 font
    JSR CMD2LCD

    LDAA #$0E ; turn on display, turn on cursor , turn off blinking
    JSR CMD2LCD

    LDAA #$01 ; clear display screen and return to home position
    JSR CMD2LCD

    LDAA #$06 ; move cursor to right (entry mode set instruction)
    JSR CMD2LCD

    LDAA #$80 ; start writing at display 1 in first line
    JSR CMD2LCD

    LDY #$02
    JSR delayby1ms
    RTS
Write a program to display the following on the LCD:

ECE 3120 is easy
We like it!

```
ABSENTRY Entry
INCLUDE 'mc9s12dp256.inc'

ORG $1000
MSG1 dc.b "ECE 3120 is easy",0
MSG2 dc.b "We like it!",0

ORG $2000
Entry:
    LDS #$4000    ; Initialize stack
    JSR CONFIGLCD ; configure the LCD
    LDX #MSG1 ; display message 1
    JSR PUTSLCD
    LDAA #$C0       ; new line
    JSR CMD2LCD
    LDX #MSG2 ; display message 2
    JSR PUTSLCD
AGAIN: BRA AGAIN
```
PUTSLCD: ; this subroutine displays the characters
; starting from address pointed by X until it find 0

    PSHA
NEXT:  LDAA 1,X+
       BEQ DONE
       JSR PUTCLCD
       BRA NEXT
DONE:  PULA
       rts

Do not forget to add the following subroutines:-
         CMD2LCD, PUTCLCD, CONFIGLCD, and delayby1ms
Thank You!

Questions

Mohamed Mahmoud