Programming Book for 6809 Microprocessor Kit

Wichit Sirichote, wichit.sirichote@gmail.com

Rev1 July 2017

Contents

Lab 1 Load and Store instruction

```
Line addr hex code            Label   Instruction

0001 0200                             org $200
0002
0003 0200 86 aa               main    lda #$aa
0004 0202 b7 80 00                    sta $8000
0005 0205 3f                          swi
0006
0007                                   end
0008
```

The accumulator is loaded with 8-bit data, AA. Then write it to gpio1 LED at location 8000. SWI is software interrupt. It makes CPU to return to monitor and save CPU registers to user registers.

Enter the hex code to memory and run it with key GO.

What is happening?

Can you change the value from AA to 55? how?

Test run with many hex byte and see the display on GPIO1 LED. It shows in BINARY number!

Lab 2 Store byte to memory

```
0003 0200 86 aa              main    lda #$aa
0004
0005 0202 b7 80 00                   sta $8000
0006 0205 97 40                      sta $40
0007 0207 b7 60 00                   sta $6000
0008
0009 020a 3f                         swi
```

The accumulator is loaded with 8-bit data, AA. Then write it to memory at location 8000, 40 and 6000.

Writing accumulator to absolute address 8000 is Extended addressing mode. Hex code has three bytes, B7, 80, 00. The location 8000 is GPIO1 LED. It is write only.

At line 6, similar one but now write it to page zero address, at 40. This is Direct addressing mode. Hex code has only two bytes, 97, 40.

And last store, to address 6000.

Before test this program, check the contents and write down.

Enter the hex code to memory and run it with key GO.

What is happening?

Location	Before	After
40		
6000		

Try change the location and load value.

Lab 3 Store byte to memory using Index register

```
0200                          org $200

0200 86 aa                   main    lda #$aa

0202 8e 10 00                        ldx #$1000 base addr

0205 a7 84                           sta ,x

0207 a7 01                           sta 1,x

0209 a7 02                           sta 2,x

020b a7 03                           sta 3,x

020d a7 1f                           sta -1,x

020f a7 88 7f                        sta $7f,x

0212 a7 89 10 00                     sta $1000,x

0216 3f                              swi

                              end
```

The accumulator is loaded with 8-bit data, AA. Then write it to memory at location pointed to using Index addressing mode with X register.

X register is loaded with base address, 1000.

Compute the effective address for each instructions, write down.

Instructions	Effective address
sta ,x	1000
sta 1,x	
sta 2,x	
sta 3,x	
sta -1,x	
sta $7f,x	
sta $1000,x	

Enter the code and write down the memory contents before and after running.

Instructions	Effective address	Memory contents before	Memory contents after
sta ,x	1000		
sta 1,x			
sta 2,x			
sta 3,x			
sta -1,x			
sta $7f,x			
sta $1000,x			

Can you change the index register from X to Y register? How?

Lab 4 Using register for offset byte

```
0003 0200                           org $200
0004
0005 0200 86 05                    main     lda #5
0006 0202 c6 55                             ldb #$55
0007
0008 0204 8e 10 00                          ldx #$1000
0009
0010 0207 a7 85                             sta b,x
0011
0012 0209 3f                                swi
0013
```

The accumulator is loaded with 8-bit data, 05. Then write it to memory pointed to using Indexed addressing mode. Now, the contents of register B is used as the offset byte.

The effective address will be X+b.

Write down the effective address and its contents before and after.

Enter the hex code to memory and run it with key GO.

Instructions	Effective address	Memory contents before	Memory contents after
sta b,x			

Lab 5 Indirection addressing mode

```
                               * Indirection

     0200                       org $200

     0200 a6 9f c0 00          main    lda [$c000]
     0204 b7 80 00                     sta $8000

     0207 3f                           swi

                                end

```

The accumulator is loaded with 8-bit data pointed to using LOCATION that stored in address C000. Then write it to GPIO1 LED.

What is the location be loaded?

Enter the hex code to memory and run it with key GO.

What is happening?

Modify the contents of the location to be loaded. Test run again.

What is happening?

Change C000 to D000 and test run again.

Lab 6 Branch and Long Branch

```
0001                            *  branch and long branch
0002
0003 0200                        org $200
0004
0005 0200 b6 c0 00              main     lda $c000
0006
0007 0203 b7 80 00                       sta $8000
0008
0009 0206 20 07                          bra  load_b
0010
0011 0208 12                             nop
0012 0209 12                             nop
0013
0014 020a 16 xx xx                       lbra load_b
0015
0016 020d 12                             nop
0017 020e 12                             nop
0018
0019 020f f6 c0 01              load_b   ldb $c001
0020
0021 0212 1e 89                          exg a,b
0022
0023
0024
0025 0214 3f                             swi
0026
0027                             end
0028
```

6809 uses relative addressing for branch and long branch instructions.

The current PC register will be added with OFFSET byte.

Kit has CAL key for hex number calculation.

The first branch at line 9 has OFFSET byte 07.

We can compute it with,

OFFSET = destination – current PC.

For example, at line 6, bra load_b

OFFSET = 20f – 208

Using CAL key is easy. Press CAL, enter destination, then key – then current PC, the GO.

Try compute the 16-bit offset byte for long branch instruction at line 14.

What is value will be?

Lab 7 Simple delay using X register

```
0001                *  simple delay with x register
0002
0003 0200                      org $200
0004
0005 0200 86 01                main     lda #1
0006
0007 0202 b7 80 00             loop     sta $8000
0008 0205 49                            rola
0009
0010 0206 8d 02                         bsr delay
0011
0012 0208 20 f8                         bra loop
0013
0014 020a 8e 30 00             delay    ldx #$3000
0015 020d 30 1f                delay1   leax -1,x
0016 020f 26 fc                         bne delay1
0017 0211 39                            rts
0018
0019
0020                            end
0021
```

Simple delay using register counting down is a common useful subroutine for program testing.

We use X register loaded with initial 16-bit value. Decrements it, until ZERO flag set.

Main code is a loop running with bit rotation. We can see the bit rotation on GPIO1 LED easily.

Forever loop running is done by bra loop instruction.

Can you compute the OFFSET byte of this instruction? How F8 comes? Why?

The subroutine that counts X register uses leax -1,x instruction.

The instruction, leax -1,x will decrements the x register by 1.

Thus the number of repeating at line 15 will be $3000 or 12288 rounds.

Enter the hex code and test run.

What is happening?

Can you change the speed of rotation? How?

Can you change x register to y register? How?

Lab 8 Using IRQ interrupt with 10ms tick

```
                             *  using 10ms tick

0200                          org $200

0200 86 7e                   main    lda #$7e
0202 b7 7f f0                        sta $7ff0
0205 8e 60 00                        ldx  #serv_irq
0208 bf 7f f1                         stx $7ff1

020b 1c ef                           andcc #%11101111
020d 20 fe                           bra *

                     * IRQ interrupt service routine

6000                                 org $6000

6000 0c 00                   serv_irq inc 0
6002 96 00                            lda 0

6004 81 64                            cmpa #100
6006 26 09                            bne skip
6008 0f 00                            clr 0

600a 0c 01                            inc 1
600c 96 01                            lda 1
600e b7 80 00                         sta $8000
                             skip
6011 3b                               rti

                              end

```

Instead of making the delay by counting the X or Y register, we can use interrupt with 10ms tick generator.

The 6809 IRQ vector is stored in monitor ROM at location FFF8, FFF9

The monitor program of the 6809 kit has relocated the IRQ vector to RAM location at 7FF0. When triggered by IRQ the CPU will jump to location 7FF0.

The test code inserts the JUMP instruction to the service routine.

Then clear the I flag, to enable IRQ interrupt.

And wait with BRA *, instruction.

Enter the code and change SW1 to 10ms tick position.

What is happening at GPIO1 LED?

Can you change the counting rate from 1Hz to 10Hz? How?

Lab 9 Running code using 10ms tick

```
                    *   running code using 10ms tick

de94                          beep    equ $de94
700e                          tick    equ $700e

0200                           org $200

0200 3c ef                    main    cwai #%11101111

                              * below code 10ms

0202 b6 70 0e                 loop lda tick
0205 81 64                     cmpa #100
0207 26 0d                     bne skip
0209 7f 70 0e                  clr tick

                              * below code 1000ms

020c 96 00                     lda 0
020e 8b 01                     adda #1
0210 19                        daa
0211 97 00                     sta 0
0213 b7 80 00                  sta $8000

0216 20 ea                    skip    bra loop

                               end
```

Variable tick is 8-bit memory location at 700E. By default, the monitor program prepares the service routine for IRQ after CPU was RESET.

If we enable the IRQ flag, I, the CPU will enter IRQ service routine every 10ms (with SW1 set to 10ms position). The service routine will increment tick variable every 10ms.

Above code demonstrates how to read the tick variable to provide 10ms and 1000ms time slot for a given task running.

Enter the code and test run it. What is happening?

Lab 10 Calling monitor c function

```
0001                    *  calling monitor c function
0002                    *  display ascii letter on lcd
0003
0004 0200                        org $200
0005
0006
0007 c23a                       init_lcd equ $c23a
0008 c307                       putch_lcd equ $c307
0009
0010 0200 bd c2 3a              main    jsr init_lcd
0011
0012 0203 c6 36                         ldb #$36
0013 0205 34 06                         pshs d
0014 0207 bd c3 07                     jsr putch_lcd
0015 020a 32 62                        leas 2,s
0016 020c 3f                           swi
0017
0018                             end
```

The monitor program listing provides symbol for reference. We can call the function in c written directly.

This program demonstrates calling the LCD display driver.

For this LAB, we will need text LCD installed.

For c function without parameter passing, we can use JSR instruction and the location of that function directly.

The example one is jsr init_lcd, jump to subroutine to initialize the LCD module.

For the function that needs char size input parameter, register B will be used to pass value. Passing is done by using system stack. We see that, line 13 push register d to system stack. Then jump to subroutine putch_lcd(). When completed, the original stack will be restored with leas 2,s instruction.

Enter the code, test run it. What is happening on the LCD display?

Can you change the letter? How?

Lab 11 Display message on LCD display

```
                    *  calling monitor c function
                    *  display message on lcd

0200                          org $200

c23a                         init_lcd  equ $c23a
c27d                         pstring   equ $c27d

0200 bd c2 3a                main    jsr init_lcd

0203 cc 02 0e                        ldd #text1
0206 34 06                           pshs d

0208 bd c2 7d                        jsr pstring

020b 32 62                           leas 2,s
020d 3f                              swi

020e 48 65 6c 6c 6f 20  text1 fcc "Hello from 6809"
 66 72 6f 6d 20 36
 38 30 39

021d 00                              fcb 0

                                      end

```

We can use monitor function that displays message on the LCD, pstring function.

The input parameter is location of message. Register d is loaded with the address of text1, 020E.

Then push it on system stack, jump to subroutine pstring, then restore the original location of the system stack with leas 2, s instruction.

Enter the code, test run it. What is happening on the LCD display?

Can you change the message? How?

Lab 12 Display message on LCD two lines

```
                          *  calling monitor c function
                          *  display text on lcd
     0200                        org $200
     c23a                       init_lcd  equ $c23a
     c27d                       pstring   equ $c27d
     c1a9                       goto_xy   equ $c1a9
     0200 bd c2 3a              main    jsr init_lcd
     0203 cc 02 25                      ldd #text1
     0206 34 06                         pshs d
     0208 bd c2 7d                      jsr pstring
     020b 32 62                         leas 2,s
     020d c6 00                         ldb #$00
     020f 34 06                         pshs d
     0211 c6 01                         ldb #$01
     0213 34 06                         pshs d
     0215 bd c1 a9                      jsr goto_xy
     0218 32 64                         leas 4,s
     021a cc 02 35                      ldd #text2
     021d 34 06                         pshs d
     021f bd c2 7d                      jsr pstring
     0222 32 62                         leas 2,s
     0224 3f                            swi
     0225 48 65 6c 6c 6f 20 text1   fcc "Hello from 6809"
 66 72 6f 6d 20 36
 38 30 39
```

```
0038 0234 00                        fcb 0
0039 0235 45 6e 74 65 72 20 text2 fcc "Enter code in HEX"
 63 6f 64 65 20 69
 6e 20 48 45 58
0040 0246 00                        fcb 0
0041
0042                               end
0043
```

If we use 20x2 line LCD, we can display two lines by using function goto_xy(a,b) easily.

For two parameters, a and b for position x,y, again we push twice sending both parameters. After that restore original location of system stack with leas 4,s instruction.

Enter the code, test run it. What is happening on the LCD display?

Can you change the message? How?

Lab 13 Sending ASCII character to terminal

```
                   *  calling monitor c function
                   *  display text on terminal using UART

     dff1                       putchar    equ $dff1
     e053                       puts       equ $e053
     df85                       initacia   equ $df85

     0200                        org $200

     0200 bd df 85              main    jsr initacia

                                loop
     0203 c6 41                         ldb #'A'
     0205 34 06                         pshs d

     0207 bd df f1                      jsr putchar

     020a 32 62                         leas 2,s
     020c 20 f5                         bra loop


                                        end

```

This lab will need RS232 terminal. We can use PC running terminal emulator, says VT100 for the test.

Serial wiring between 6809 kit and RS232 terminal is cross cable.

Enter the code, test run it.

What is happening on the terminal display?

Can you change the ASCII letter from A to B? How?

Lab 14 Sending message to terminal

```
                 *  calling monitor c function
                 *  display text on terminal using UART
dff1                        putchar    equ $dff1
e053                        puts       equ $e053
df85                        initacia   equ $df85
e0fd                        newline    equ $e0fd
0200                         org $200
0200 bd df 85               main    jsr initacia
                            loop
0203 cc 02 12                       ldd #text3
0206 34 06                          pshs d
0208 bd e0 53                       jsr puts
020b 32 62                          leas 2,s
020d bd e0 fd                       jsr newline
0210 20 f1                          bra loop
0212 48 65 6c 6c 6f 20 text3 fcc "Hello from 6809 kit"
     66 72 6f 6d 20 36
     38 30 39 20 6b 69
     74
0225 00                             fcb 0
                             end
```

To display message, we can use monitor function puts() at location E053. By loading the start address of the message to register d. Then put it to system stack, call the function.

Enter the code, test run it.

What is happening on the terminal display?

Can you change the message? How?

Lab 15 Sending message to terminal every one second

```
0001            *  calling monitor c function
0002            *  simple one second wait function
0003
0004 dff1                     putchar    equ $dff1
0005 e053                     puts       equ $e053
0006 df85                     initacia   equ $df85
0007 e0fd                     newline    equ $e0fd
0008
0009 e747                     wait1s     equ $e747
0010
0011 0200                      org $200
0012
0013 0200 bd df 85            main    jsr initacia
0014
0015                          loop
0016 0203 cc 02 15                    ldd #text3
0017 0206 34 06                       pshs d
0018
0019 0208 bd e0 53                   jsr puts
0020
0021 020b 32 62                      leas 2,s
0022
0023 020d bd e0 fd                   jsr newline
0024 0210 bd e7 47                   jsr wait1s
0025
0026 0213 20 ee                       bra loop
0027
0028 0215 48 65 6c 6c 6f 20  text3   fcc "Hello from 6809
kit with one second print"
          66 72 6f 6d 20 36
           38 30 39 20 6b 69
           74 20 77 69 74 68
           20 6f 6e 65 20 73
           65 63 6f 6e 64 20
           70 72 69 6e 74
0029 023e 00                          fcb 0
0030
0031                           end
```

We can add wait one second function to make printing every one second on the display easily.

The monitor function wait1s() located at E747 uses 10ms tick for counting 100 times the exit. We can use it for slow down the output display.

Function newline() will enter the new line control character.

Enter the code, test run it.

What is happening on the terminal display?

Can you change the message? How?

Can you change the interval from one second to three seconds? How?