Programming Book for 6809 Microprocessor Kit

Wichit Sirichote, wichit.sirichote@gmail.com

Rev2.0 April, 2020

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
0005 0202 B7 80 00            STA $8000
0006
0007 0205 3F                  SWI
0008
0009                          END
```

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

```
LINE ADDR HEX CODE  LABEL   INSTRUCTION

0001 0200                    ORG $200
0002
0003 0200 86 00      MAIN   LDA #0
0004 0202 1F 8B             TFR A,DP
0005 0204 86 AA             LDA #$AA
0006
0007 0206 B7 80 00          STA $8000
0008 0209 97 40             STA $40
0009 020B B7 60 00          STA $6000
0010
0011 020E 3F                SWI
0012
0013                         END
0014
```

The accumulator is loaded with 8-bit data 00. TFR A,DP transfers the content of accumulator to DP register. This Direct Page register is used to access the zero page location.

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 8, we write it to page zero address, at 40. This is Direct addressing mode. The location to be accessed will be xx40. xx is DP register. 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, load value and DP register.

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 ADDRESS

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  BASE ADDRESS
0009
0010 0207 A7 85          STA B,X
0011
0012 0209 3F             SWI
0013
0014                     END
```

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

```
0003 0200                         ORG $200
0004
0005 0200 A6 9F C0 00  MAIN         LDA [$C000]
0006 0204 B7 80 00                  STA $8000
0007
0008 0207 3F                        SWI
0009
0010                                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 LOCATION C000 HAS 10CE

0006     *  THE BYTE AT 10CE WILL BE SENT TO GPIO1 LED

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 00 02       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
```

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 value would 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 Testing IRQ interrupt with 10ms tick

```
                  *   Testing IRQ with 10ms TICK

0200                          ORG $200

0200 86 00        MAIN       LDA #0
0202 1F 8B                   TFR A,DP   * SET PAGE ZERO
0204 86 7E                   LDA #$7E   * jump instruction
0206 B7 7F F0                STA $7FF0
0209 8E 60 00                LDX  #SERV_IRQ
020C BF 7F F1                STX $7FF1

020F 1C EF                   ANDCC #%11101111
0211 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

6011 3B           SKIP       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 program starts with setting DP register to 0 for direct addressing within zero page.

Then inserts the JUMP instruction code, 7E to location 7FF0 and the address of service routine 6000 to 7FF1, 7FF2.

Clear the I flag, to enable IRQ interrupt.

And wait for interrupt with BRA *, instruction.

The service routine is located at 6000. The zero page byte 0 will be counting. When it reaches 100 or 1000ms, clear it to zero. The zero page byte 1 will be incremented and sent to gpio1 LED. We will see binary counting at 1s or 1Hz rate.

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

700E             TICK    EQU $700E

0200                      ORG $200

0200 86 00       MAIN    LDA #0
0202 1F 8B               TFR A,DP     SET PAGE 0
0204 3C EF               CWAI #%11101111 ENABLE IRQ

                * BELOW CODE 10mS

0206 B6 70 0E   LOOP     LDA TICK
0209 81 64                CMPA #100
020B 26 0D                BNE SKIP
020D 7F 70 0E            CLR TICK

                * BELOW CODE 1000mS

0210 96 00               LDA 0
0212 8B 01               ADDA #1
0214 19                  DAA
0215 97 00               STA 0
0217 B7 80 00            STA $8000

021A 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 C228          INIT_LCD  EQU $C228
0005 C2F5          PUTCH_LCD EQU $C2F5
0006
0007 0200                    ORG $200
0008
0009 0200 BD C2 28  MAIN JSR INIT_LCD
0010
0011 0203 C6 36              LDB #$36
0012 0205 34 06              PSHS D
0013 0207 BD C2 F5           JSR PUTCH_LCD
0014 020A 32 62              LEAS 2,S
0015 020C 3F                 SWI
0016
0017                         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 12 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 print more letters? How?

Lab 11 Display message on LCD display

```
                *  CALLING MONITOR C FUNCTION
                *  DISPLAY TEXT ON LCD

     0200                 ORG $200

     C228       INIT_LCD  EQU $C228
     C26B       PSTRING   EQU $C26B

     0200 BD C2 28  MAIN JSR INIT_LCD

     0203 CC 02 0E       LDD #TEXT1
     0206 34 06          PSHS D

     0208 BD C2 6B       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

C228          INIT_LCD  EQU $C228
C26B          PSTRING   EQU $C26B
C197          GOTO_XY   EQU $C197

0200                     ORG $200
0200 BD C2 28 MAIN      JSR INIT_LCD

0203 CC 02 25           LDD #TEXT1
0206 34 06              PSHS D

0208 BD C2 6B            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 97            JSR GOTO_XY

0218 32 64               LEAS 4,S

021A CC 02 35            LDD #TEXT2
021D 34 06              PSHS D

021F BD C2 6B            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
0234 00                          FCB 0
0235 45 6E 74 65 72 20 TEXT2 FCC "ENTER CODE IN HEX"
```

```
     63 6F 64 65 20 69
     6E 20 48 45 58

0039 0246 00                    FCB 0
0040
0041                             END
```

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 are 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

     DFDF           PUTCHAR     EQU $DFDF
     E041           PUTS        EQU $E041
     DF73           INITACIA    EQU $DF73

     0200                          ORG $200

     0200 BD DF 73 MAIN         JSR INITACIA

                       LOOP
     0203 C6 41                   LDB #'A'
     0205 34 06                   PSHS D

     0207 BD DF DF               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. Therminal speed is 19,200 bit/s.

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

DFDF           PUTCHAR     EQU $DFDF
E041           PUTS        EQU $E041
DF73           INITACIA    EQU $DF73
E0EB           NEWLINE     EQU $E0EB

0200                         ORG $200

0200 BD DF 73 MAIN         JSR INITACIA

                  LOOP
0203 CC 02 12              LDD #TEXT3
0206 34 06                 PSHS D

0208 BD E0 41             JSR PUTS

020B 32 62                LEAS 2,S

020D BD E0 EB             JSR NEWLINE

0210 20 F1               BRA LOOP

0212 48 65 6C 6C 6F 20TEXT3FCC "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 E041. 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

```
                *   CALLING MONITOR C FUNCTION
                *   SIMPLE ONE SECOND WAIT FUNCTION

DFDF            PUTCHAR     EQU $DFDF
E041            PUTS        EQU $E041
DF73            INITACIA    EQU $DF73
E0EB            NEWLINE     EQU $E0EB

E750            WAIT1S      EQU $E750

0200                       ORG $200

0200 BD DF 73 MAIN         JSR INITACIA

                   LOOP
0203 CC 02 15              LDD #TEXT3
0206 34 06                 PSHS D

0208 BD E0 41               JSR PUTS

020B 32 62                  LEAS 2,S

020D BD E0 EB               JSR NEWLINE
0210 BD E7 50               JSR WAIT1S

0213 20 EE                  BRA LOOP

0215 48 65 6C 6C 6F 20 TEXT3 FCC "HELLO PRINTING 1S"
     70 72 69 6E 74 69
     6E 67 20 31 73

0226 00                      FCB 0

                              END
```

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

The monitor function wait1s() located at E750 uses 10ms tick for counting 100 times to 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?