Programming Book for CDP1802 Microprocessor Kit

Rev 1.0 written by Wichit Sirichote, October 27, 2018

Program 1 Q LED blinking
Program 2 GPIO1 LED
Program 3 LOAD IMMEDIATE 8-bit to D register
Program 4 Run with BREAK POINT
Program 5 16-bit Binary Addition
Program 6 EF Flag testing
Program 7 Timer interrupt
Program 8 SCANNING DISPLAY
Program 9 Using LCD display
Program 10 UART communication

The source code was assembled with A18 assembler format. For keypad monitor, we can enter hex code to the memory and test run directly.

With terminal connected at 9600 8n1 asynchronous mode, key LOAD and DUMP can be used to load Intel HEX file and display memory contents easily. For terminal connected, 1ms delay for character and line will be needed.

Program 1 Q LED blinking

```
F001              DELAY: EQU 0F001H

8000                   ORG 8000H

8000  F8 F0 B3 F8       LOAD R3, DELAY
8004  01 A3

8006  7B          LOOP:  SEQ
8007  D3                 SEP R3
8008  7A                 REQ
8009  D3                 SEP R3
800A  30 06              BR LOOP
```

The subroutine DELAY is located at F001. We use R3 as the pointer. Main program is forever loop running. SEQ makes Q LED lit then call subroutine delay with SEP R3 instruction. REQ, clears Q Flip-flop, Q LED will be turned off. BR LOOP, branch back to LOOP location at 8006.

Enter the hex code from 8000 to 800B. Press PC, then GO.

What is happening?

LOAD R3, DELAY is a built-in macro of the A18 assembler. It produces the 1802 instructions as follows:

```
F8 F0          LDI HIGH(DELAY)
B3             PHI R3
F8             LDI LOW(DELAY)
A3             PLO R3
```

Can we change blinking rate? How?

Here is the list of delay subroutine.

```
F000   D0          RET_DELAY: SEP R0

F001   F8 64       DELAY:   LDI 100
F003   A6                   PLO R6

F004   F8 00       DELAY1:  LDI 0
F006   A7                   PLO R7
F007   27          DELAY2:  DEC R7
F008   87                   GLO R7
F009   3A 07                BNZ DELAY2

F00B   26                   DEC R6
F00C   86                   GLO R6
F00D   3A 04                BNZ DELAY1

F00F   30 00                BR RET_DELAY
```

The delay subroutine is automatically loaded on RESET. We can use it by simple calling with SEP register easily.

The subroutine use R6 and R7 making simple delay by counting inner loop with R7 and outer loop by R6.

To change blinking rate, we can modify the byte at F002.

Try change it from 64 to 20. After changed, press PC then GO. (If you press RESET, 64 will be reloaded)

What is happening?

Program 2 GPIO1 LED

GPIO1 LED is the 8-bit binary display LED. It is another useful device for program debugging. We can display the contents of CPU registers by writing it to GPIO1 location directly.

```
F001              DELAY: EQU 0F001H
7000              GPIO1: EQU 7000H

8000                   ORG 8000H

8000   F8 F0 B3 F8        LOAD R3, DELAY
8004   01 A3
8006   F8 70 B4 F8        LOAD R4, GPIO1
800A   00 A4

800C   15          LOOP:  INC R5     ; INCREMENT R5
800D   85                  GLO R5     ; GET LOW BYTE OF R5
800E   54                  STR R4     ; WRITE D TO GPIO1
800F   D3                  SEP R3     ; CALL DELAY
8010   30 0C               BR LOOP    ; JUMP TO LOOP
```

The delay subroutine pointer is loaded to R3 and the location of gpio1 is loaded to R4.

Main loop will increment R5. The low byte of R5 is copied to D register. Then write D register to gpio1 LED pointed to by R4.

Delay makes slow counting. Then repeat the loop.

Enter hex code from 8000 to 8011, press PC then GO.

What is happening? Can you change running rate faster? How?

Program 3 LOAD IMMEDIATE 8-bit to D register

```
7000              GPIO1: EQU 7000H

8000                     ORG 8000H

8000  F8 70 B4 F8        LOAD R4, GPIO1
8004  00 A4

8006  F8 1F              LDI 1FH
8008  54                 STR R4
8009  00                 IDL
```

D is the accumulator register. Most instructions use D to pass or to compute the 8-bit data.

LDI 1FH, load immediate 8-bit data to the D register. We use GPIO1 LED to display register D contents. Thus STR R4 instruction will store D register to the location of gpio1 LED.

IDL is IDLE instruction. It will wait for interrupt or DMA operation.

Enter the code from 8000 to 8009. Press PC then GO.

What is happening?

Can you change R4 to R5? How?

Try modify load value from 1F to others, within the range of 00-FF.

Program 4 Running with BREAK POINT

```
 2756              BREAK:   EQU 2756H

8000                       ORG 8000H

8000   F8 1F               LDI 1FH
8002   C0 27 56            LBR BREAK
```

The monitor program provides subroutine to save CPU registers to USER registers. We can use it by LONG BRANCH instruction to BREAK location.

We can examine after program execution with key REG and hex key 0-F.

Enter the code from 8000 to 8004. Press PC then GO.

Press REG, 0 to see register D content.

The method can be applied to R3 to R15 as well.

(User code running will use R0 for program counter, R1 for interrupt vector, and R2 for STACK pointer)

LBR BREAK is available only for saving CPU registers to user registers after user program has been executed.

Key GO will not load user registers to CPU registers!

Program 5 16-bit Binary Addition

```
2756              BREAK:   EQU 2756H

8000                       ORG 8000H

8000   F8 12 B3 F8         LOAD R3,1234H ; R3 = 1234H
8004   34 A3
8006   83                  GLO  R3
8007   FC 34                ADI  34H
8009   A3                   PLO  R3
800A   93                   GHI  R3
800B   7C 12                ADCI 12H
800D   B3                   PHI R3 ; R3 = R3+1234H

800E   C0 27 56              LBR BREAK
```

R3 is loaded with 16-bit constant, 1234H. Suppose we want to add a number 1234H to R3 and put result to R3. The low byte of R3, 34H is first added with 34H, then save the result to the low byte of R3.

The high byte now is then added using ADCI 12H, add with carry flag, DF. And put result back to high byte of R3. LBR BREAK returns control to monitor program and save the R3 to USER register R3.

Enter the code, test run it and examine the result in R3 using key REG, 3.

What is the result of addition?

You can compare with hand calculation.

Program 6 EF flag Testing

```
F001              DELAY: EQU 0F001H
7000              GPIO1:   EQU 7000H

8000                    ORG 8000H

8000   F8 F0 B3 F8        LOAD R3, DELAY
8004   01 A3
8006   F8 70 B4 F8        LOAD R4, GPIO1
800A   00 A4

800C   F8 00               LDI 0
800E   A5                  PLO R5
800F   54                  STR R4

8010   34 10        LOOP:  B1 $   ; PRESSED
8012   D3                  SEP R3  ; DEBOUNCE

8013   3C 13               BN1 $    ; NO PRESSED
8015   D3                  SEP R3  ; DEBOUNCE

8016   15                  INC R5  ; DO TASK
8017   85                  GLO R5
8018   54                  STR R4

8019   30 10               BR LOOP ; REPEAT
```

CDP1802 chip has 4 external flag pins i.e. EF1, EF2, EF3 and EF4. The kit provides EF1 and EF2 on the keyboard. All of these bits are pull-up to logic HIGH. When pressed, the logic will be LOW.

Test program will test EF1 pin. However the internal circuit of the CDP1802 uses negative logic.

We can use short branch instructions B1 and BN1 for testing it.

B1 tests for if EF1=1 (for negative logic EF1 pin must be LOW).
BN1 tests for if EF1=0 (EF1 pin must be HIGH).

Below shows the datasheet descriptions, and instruction hex codes, 34 and 3C.

SHORT BRANCH IF EF1 = 1 ($\overline{EF1} = V_{SS}$)	B1	34
SHORT BRANCH IF EF1 = 0 ($\overline{EF1} = V_{CC}$)	BN1	3C

Main program first tests EF1 pin with B1 instruction.

If key was pressed, keep reading EF1. If key released, then call delay to prevent key bounce.

BN1 instruction will test EF1 if no pressed, keep reading EF1. If key was pressed, debounce, then increment R5, write R5 to gpio1 LED.

Enter the code, test run it.

What is happening?

Modify the code, use EF2 instead? How?

Program 7 Timer interrupt

```
7000              GPIO1: EQU 7000H
FFFF              TICK: EQU 0FFFFH

8000                    ORG 8000H

8000  F8 80 B1 F8       LOAD R1, INT_VECTOR
8004  18 A1
8006  F8 70 B4 F8       LOAD R4, GPIO1
800A  00 A4
800C  F8 FF B6 F8       LOAD R6, TICK
8010  FF A6

8012  E0                SEX R0
8013  70                RET    ; ENABLE INTERRUPT
8014  00                DB 0

8015  30 15             BR $


8017              EXIT_INT:
8017  70                  RET    ; R2 = R2 +1, P = R0

8018              INT_VECTOR:       ; P = R1, X =R2

8018  22              DEC R2
8019  78              SAV     ; SAVE T, P=0

801A  06              LDN R6
801B  FC 01           ADI 1
801D  56              STR R6
801E  FB 0A           XRI 10
8020  3A 26           BNZ NEXT_INC
```

```
8022  56            STR R6
8023  15            INC R5
8024  85            GLO R5
8025  54            STR R4

8026  C4      NEXT_INC: NOP

8027  30 17         BR EXIT_INT
```

This program demonstrates the use of 10ms tick for testing the CDP1802 interrupt. SW1 when set to 10ms tick, the tick signal will tie to pin 36, INTRPT.

Main code initializes R1 for interrupt vector, R4 for gpio1 LED, and R6 for tick variable. Then enable interrupt, and repeat jump at location 8015.

The interrupt vector is located at 8018. When interrupt occurs, R2 the stack pointer must be decrement then save the X and P bits to stack memory.

Tick variable then was tested, if equal 10 (10x10ms) then clear it. Increment R5, then write the low of R5 to gpio1 LED.

On return, R2 will be incremented by 1, and program counter will be reloaded with P bits, back to R0.

Enter the code and test run with SW1 set to 10ms tick.

What is happening? Can you change the counting rate to 1Hz? How?

Program 8 SCANNING DISPLAY

```
7000              GPIO1: EQU 7000H
7102              SEGMENT: EQU 7102H
7101              DIGIT: EQU 7101H


8000                    ORG 8000H

8000  F8 80 B3 F8  START: LOAD R3, DELAY
8004  34 A3
8006  F8 71 B4 F8        LOAD R4, SEGMENT
800A  02 A4
800C  F8 71 B5 F8        LOAD R5, DIGIT
8010  01 A5


8012  F8 80 BA F8  MAIN:  LOAD R10, BUFFER
8016  3D AA
8018  F8 80 BB F8        LOAD R11, SCAN_DIGIT
801C  43 AB

801E  F8 06             LDI 6
8020  A8               PLO R8    ; LOOP COUNT

8021  0B          LOOP:  LDN R11   ; GET DIGIT CONTROL
8022  FB FF              XRI 0FFH  ; complement it
8024  55               STR R5    ; WRITE TO DIGIT

8025  0A               LDN R10   ; GET BYTE FROM BUFFER

8026  54               STR R4    ; WRITE TO SEGMENT
8027  D3               SEP R3    ; DELAY
8028  F8 00            LDI 0     ; TURN OFF DISPLAY
```

```
802A  54              STR R4

802B  1A              INC R10   ; NEXT BUFFER
802C  1B              INC R11   ; NEXT DIGIT

802D  28              DEC R8
802E  88              GLO R8
802F  3A 21           BNZ LOOP  ; UNTIL 6 DIGITS

8031  30 12           BR MAIN


8033  D0         RET_DELAY: SEP R0

8034  F8 01       DELAY: LDI 1
8036  A7                 PLO R7

8037  27          DELAY1: DEC R7
8038  87                 GLO R7
8039  3A 37               BNZ DELAY1

803B  30 33                BR RET_DELAY

803D  37 8F 85 85  BUFFER: DB 37H, 8FH, 85H, 85H,
.                                    0A3H, 0 ; HELLO
8041  A3 00

8043  20 10 08 04  SCAN_DIGIT: DB 20H,10H,8,4,2,1
8047  02 01
```

Kit has 6-digit seven segment display. The segments for all digits are connected together. To display for 6 digits, display scanning will be needed. Each digit is controlled by common pin which is active LOW. Segment is active HIGH.

Display buffer located at 803D-8042 stores display pattern, “HELLO”.

Digit control bytes are stored at 8043-8048.

Scanning is made with ACTIVE the digit control, then write the pattern to the segment, and delay, then turn segment off.

Repeat until 6 digits. The display will show still pattern, “HELLO”.

Enter hex code 8000-8048. Press PC, GO.

What is happening?

Can you change to another word? How?

Program 9 Using LCD display

```
7200              LCD_CWR: EQU  7200H
7201              LCD_DWR: EQU  7201H
7202              LCD_CRD:  EQU  7202H
7203              LCD_DRD:  EQU  7203H

8000                    ORG 8000H
8000   30 0A              BR MAIN

                ; LCD DRIVERS

8002   D0          RET_LCD1: SEP R0

8003   05          LCD_READY: LDN R5
8004   FA 80                ANI 80H
8006   3A 03               BNZ  LCD_READY

8008   30 02                BR RET_LCD1

800A   F8 72 B4 F8   MAIN:  LOAD R4, LCD_CWR
800E   00 A4

8010   F8 72 B5 F8         LOAD R5, LCD_CRD
8014   02 A5
8016   F8 72 B6 F8         LOAD R6, LCD_DWR
801A   01 A6

801C   F8 80 B3 F8         LOAD R3, LCD_READY
8020   03 A3

8022   D3                  SEP R3  ; wait until ready
8023   F8 01               LDI 1    ; clear display command
8025   54                  STR R4 ; write to command register
```

```
8026   D3          SEP R3   ; wait until ready
8027   F8 41       LDI 'A'  ; load D with ASCII ‘A’
8029   56          STR R6 ; write to data register

802A   D3          SEP R3
802B   F8 42       LDI 'B'
802D   56          STR R6

802E   00          IDL      ; stop here
```

This lab will need LCD module. Any text LCD with HD44780 compatible controller can be used.

Care should be taken, turn power off before Insert/Remove the LCD module! Ensure pin position before turn the power on!

The example was tested with standard 16x2 text LCD display.

The LCD registers are Command and Data registers. Both registers are READ and WRITE.

To use the LCD, it must be initialized, settings mode, clear memory. This was done under system monitor when RESET.

The LCD is rather slow device, so its command register then provides BUSY bit for indicating the ready for command or data reception.

R3 is pointed to subroutine to test this bit. First we send command 1 to command register to clear the display. The start location to be printed character is then at the TOP/LEFT.

Now we write the ASCII code for letter ‘A’ to data register and followed with ‘B’.

Enter the code and run it. What is happening?

Can you put your name on the LCD by entering code manually? How?

Program 10 UART communication

```
7305                UART_STATUS: EQU 7305H
7300                UART_BUFFER: EQU 7300H

8000                    ORG 8000H

8000   30 0D              BR MAIN

8002   D0          RET_COUT: SEP R0

8003   A3          COUT:   PLO R3

8004   04          COUT1:   LDN R4
8005   FA 20                ANI 20H ; test TE flag
8007   32 04                BZ COUT1
8009   83                   GLO R3
800A   55                   STR R5
800B   30 02                BR RET_COUT

800D   F8 73 B4 F8   MAIN:  LOAD R4, UART_STATUS
8011   05 A4
8013   F8 73 B5 F8         LOAD R5, UART_BUFFER
8017   00 A5
8019   F8 80 B8 F8         LOAD R8, COUT
801D   03 A8

801F   F8 41        LOOP:  LDI 'A'
8021   D8                   SEP R8
8022   30 1F                BR LOOP
```

This lab will test serial communication between the CDP1802 kit and the terminal. Communication format is asynchronous 9600 bit/s, 8 data bit no parity and one stop bit.

The onboard UART chip is INS8250. The chip was initialized by system monitor.

This lab demonstrates how to send the ASCII character from the kit to 9600 terminal. RS232 cross cable will be used to connect between the kit and terminal.

R4 is loaded with UART status register. R5 is loaded with UART Buffer register.

COUT is a subroutine that writes D register to the UART buffer. The TE bit is tested, until set. Then write D register to UART buffer. The UART hardware will convert parallel data into 10 bits serial data stream. The format includes start bit, 8-data bit and one stop bit.

Enter the code and run it. What is happening?

Can you print your name on terminal? How?

The terminal accepts control ASCII code that makes a new line. Try sending codes 0D (CR), 0A (LF) at the end of the string being sent.