Poznan University of Technology
Faculty of Electronics and Telecommunications

8051 MICROCONTROLLER

Liquid Crystal Display LCD

Poznań 2014

1. Alphanumeric Display – Introduction; LCD Example

As discussed in the previous classes, a seven-segment display (simple-in-management) can only display a few numbers or letters, which significantly limits the area of applications of that electronic module. Much effort has been put into supplying the user with a more flexible electronic tool that will enable the display of various alphanumeric signs. In that light, Liquid Crystal Displays (LCDs) have superseded the seven-segment modules (and also other types of displays) since they enable the display of a quite large number of various signs, including self-defined characters. Thus, LCDs are widely used in different forms in every-day consumer electronic devices.

In the considered LCD module that will be used in laboratories, each character is defined on a field of the size of 5×7 pixels, thus – as it has been said – various signs understandable to humans can be used. Such great possibilities came with the price of a much more complicated handling when compared to the seven-segment display. Thus, LCDs are typically equipped with dedicated specialized processors (so-called display drivers) applied only for display handling purposes. In such a case, the microcontroller does not determine how to display a particular character on the display (which points shall be lit and which shall left blank, where to put a blinking or non-blinking caret, when to finish a line and move the cursor/caret to a new one, etc.), since this is the task of the driver. The microcontroller just delivers the instructions and data to be displayed to the driver, and... "forgets" about them. One of the most popular examples of the display driver family is the module by Hitachi denoted as HD 44780.

In the context of the 8051 microcontroller, different signalling has been used, thus the application of dedicated GAL (Generic Array Logic) elements was required in order to configure the cooperation between the 8051 microcontroller and the display driver.

The display driver HD 44780 is logically connected in such a way that it uses four consecutive addresses in the microcontroller address space, starting from address FF80H. Each of those addresses plays a separate part:

- FF80H (80H) – this memory cell (address) is used when the microcontroller wants to send (save) an instruction to the controller (e.g., initialize the LCD),
- FF81H (81H) – this memory cell (address) is used when the microcontroller wants to send (save) data to the controller (e.g., to be displayed),
- FF82H (82H) – this memory cell (address) is used when the microcontroller wants to read the controller status,
- FF83H (83H) – this memory cell (address) is used when the microcontroller wants to read controller data.

NOTE: The above addresses are also available in the alphabetic form, i.e., FF80H is equivalent to LCDWC (LCD Write Code), FF81H – LCDWD (LCD Write Data), FF82H - LCDRC (LCD Read Code), and FF82H – LCDRD (LCD Read Data).

Let us note that after receiving a new instruction or data, the LCD driver has to execute this request (manage the received byte of data/instructions). This operation needs some time during which the display-driver is busy and cannot receive new requests beside the "status read" one from the address FF82H.

In consequence, before sending a new request to the display driver, the user (developer) needs to verify if the driver is not busy by reading the aforementioned display status. The seventh bit of the received byte denotes the Busy Flag – it is set to 1 when the driver is busy and cannot accept new requests. This behavior of the driver is illustrated in the following Example 1 (available on the computers in laboratory – PLEASE HAVE A LOOK BEFORE THE NEXT CLASSES).

```
;************************************************
LCD
Example 1          Displaying the character
;************************************************

        LJMP    START
        ORG     100H
START:

        LCALL LCD_CLR

        MOV     R0,#LCDWD       ; set the address used for displaying data on the LCD
        MOV     R1,#LCDRC       ; set the address used for checking the LCD status

LOOP:
        LCALL WAIT_KEY      ;get the key
        ADD     A,#30H          ;change the key code
        MOV     R2,A            ;on the LCD-compatible key code

BUSY:
        MOVX  A,@R1             ;read the LCD status
        JB      ACC.7,BUSY      ;wait until Busy Flag is 0

        MOV     A,R2            ;send the data to be displayed to the LCD
        MOVX  @R0,A             ;

        SJMP    LOOP
```

One can notice that each key detected within the WAIT_KEY procedure is transcoded (from the representation used inside the DSM-51 system and the one applied inside the HD 44780 module) and sent to the display driver. Before sending the data byte, the microcontroller checks the Busy Flag and waits (inside the loop) until it is equal to zero. Then, it sends a new character to the driver.

In the laboratory device DSM-51, the LCD display contains two lines of 16 characters, thus after displaying the first 16 signs, the caret (cursor) disappears and the following characters are apparently not shown on the display. Yet, after typing 40 characters, the caret appears again but in the second line. This phenomena is due to the fact that the HD 44780 driver display is equipped with a *generic* display driver that can store 80 al-

phanumeric characters. Thus, LCDs of various sizes can be managed (e.g., 2 lines of 16 characters, 2×20, 2×40, 1×80). It further means that at a specific moment, only a subset of all stored characters can be displayed to the user.

An experienced Reader may notice that the aforementioned situation did not exist in the previous laboratory classes when built-in procedures (existing in EPROM) have been used. It is due to the fact that those sub-routines possess mechanisms for end-line detection - when 16 characters have been provided (or more precisely when the end of the first line has been reached) the cursor is shifted to the beginning of the bottom line. Similarly, when the end of the second line is detected, the caret is shifted to the beginning of the first line. In consequence, all characters entered by the user can be visible on the LCD.

Example 2 illustrates the way the instructions are sent to the driver in order to control its behaviour. The example is also available on the computers in the laboratory.

```
;************************************************
; LCD
; EXAMPLE 2      Steering instructions
;************************************************

        LJMP    START
        ORG     100H
START:

        MOV     R0,#LCDWC       ;set the address for instruction writing
        MOV     R1,#LCDRC       ;set the address for driver status reading

        MOV     A,#1            ;clear the LCD data
        ACALL WRITE

        MOV     A,#0FH          ;start the LCE and activate cursor
        ACALL WRITE             ;as well as its blinking

        MOV     A,#06H          ;set the direction of the cursor shifting (left, right)
        ACALL WRITE             ;

        INC     R0              ;set the address for data writing
        MOV     DPTR,#TEXT      ;text address

WRITE_TXT:
        CLR     A               ; get the next
        MOVC  A,@A+DPTR     ;text character
        JZ      TEXT_END        ;if byte is zero, the text has is finished

        ACALL WRITE             ;write data on LCD
        INC     DPTR            ;modify the address of getting the next character
        SJMP    WRITE_TXT       ; and get this character

TEXT_END:
        DEC     R0                      ;address of instruction writing
        MOV     DPTR,#KEY_COD           ; address of key-code change table

LOOP:                                   ;loop for reactions on key detection
```

```
        LCALL WAIT_KEY              ;get the key

        CJNE   A,#0DH,NO_DOWN       ;is it 'v' key
DOWN:                               ;for key 'v' (down)
        MOVX  A,@R1
        JB     ACC.7,DOWN   ;wait for Busy Flag=0
        MOVX  A,@R1          ;read the address from LCD
        CPL    ACC.6         ;change between the upper and bottom lines 1< >2
        SETB   ACC.7         ; set the request tag (bit)
        ACALL WRITE          ;set new address
        SJMP   LOOP

NO_DOWN:
        MOV    R2,A          ;store the key
        MOVC   A,@A+DPTR     ;transcode keys for instructions
        JZ     WRITE_DAT     ;0 key as data

        ACALL WRITE          ;send instructions
        SJMP   LOOP

WRITE_DAT:                   ;write character on LCD
        MOV    A,R2          ;restore key
        ADD    A,#30H        ;modify as character
        INC    R0            ;set the data writing address
        ACALL WRITE          ;write the data on LCD
        DEC    R0            ;change to instruction writing address
        SJMP   LOOP

;**************************************
; Subroutine for writing data or instruction on LCD; the correct addresses stored in R0 and R1 are as-
sumed

WRITE:
        MOV    R2,A          ;store the data
BUSY:
        MOVX  A,@R1          ;read the status
        JB     ACC.7,BUSY    ; wait for Busy Flag = 0
        MOV    A,R2          ;restore data
        MOVX  @R0,A          ;send data
        RET

;**************************************
;Key code transcoding table used for converting the key number on the instructions; 0 - key is as charac-
ter.

KEY_COD:
        DB     0,0,0                ;0,1,2
        DB     0,18H,0              ;3,4,5
        DB     1CH,0,0              ;6,7,8
        DB     0,10H,14H            ;9,<,>
        DB     02H,0,06H            ;^,v,Esc
        DB     07H                  ;Enter

;**************************************
TEXT:
        DB     '"MicroMade" Systemy'
        DB     ' Mikroprocesorowe  '
        DB     ' ul. Sikorskiego 33 '
        DB     '  64-920 PILA     ',0
```

Let us try to briefly analyze the provided Example. One can notice that each instruction is send to the LCD driver by means of the WRITE subroutine. Inside that procedure, the mechanism for waiting through the occupancy of the driver is employed, i.e., the microcontroller waits in the loop for the Busy Flag to be zeroed. It guarantees that each instruction will be correctly delivered to the driver. Please note that such an expected behaviour will be possible only if the R1 register contains a proper value (address). However, the WRITE subroutine is quite generic; it can be used for sending both data and instructions. What is written and where depends on the correct setup of the R0 register (the WRITE subroutine does not modify this register).

At the beginning of Example 2, three separate instructions are sent to the display-driver HD 44780:

- Instruction 01H clears the display and sets the cursor at the address 0,
- Instruction 0FH starts (activates) the display, as well as the cursor and its blinking,
- Instruction 06H defines how the cursor shifts after each displayed character.

When these initialization instructions are finished, the text (written inside the program code, starting from the TEXT etiquette) is being sent to the driver by means of the WRITE subroutine. In order to apply this procedure correctly, an appropriate address is set to the R0 register (LCDWC).

Note: for instructions, the LCDWC address is stored inside R0, it is incremented every time when data instead of instructions are to be sent; these addresses differ by one.

In the main loop called LOOP, the routine waits for the user to press a key. One can notice that the key-codes are not sent directly to the display. The program itself tries to recognize the code and initiate the action depending on the pressed key. First, the down key [↓] is detected, the handling procedure of which differs from other keys. The remaining codes are transcoded by means of the KEY_COD table, which divides the keys virtually in two groups. If for a given key, the corresponding value inside the KEY_COD table is other than zero, the real key number (stored in R2) has to be displayed after another transcoding (using, e.g., other coding tables, if necessary). The second group contains the “function keys”. For each key, various steering options have been assigned in order to present the opportunities offered by the LCD driver. So, key [4] and [6] result in rotating the whole displayed data left or right allowing for reading the whole stored text in the driver memory (i.e., the memory that contains the previously mentioned 80 characters).

Next, keys [←] and [→] shift the cursor position to the left or right (it moves across the display memory left or right, regardless of what is currently visible on the screen). Key [↑] restores the “initial settings” of the driver, i.e., the data stored at the first address in the display memory are shown at the first position of the display and the caret is set to address 0.

It is worth noticing that both lines on the driver are rotating simultaneously, but the data in both lines are independent from each other. At the same time, the cursor moves from the end of the first line to the beginning of the second line, and backwards. Let us discuss this phenomenon. The first display line contains the addresses from 00H to 27H

(this is 40 bytes), whereas the second from 40H to 67H (also 40 bytes). While rotating the display, the bytes are shifted within the addresses associated with that line; however, the cursor is automatically moved to the beginning of the second (or first) line when it reaches address 27H (or 67H).

Please observe that the addresses in the first and second lines differ by 40H (e.g., the second address in the first line is 01H, and the second address in the second line is 41H). The current address can be restored inside the code while analyzing the status byte received from the display driver. While the last bit of the status-byte is used for the detection of the Busy Flag, the bits 0 to 6 store the address. Only if the Busy Flag is zero, the address provided in the status-byte is correct.

There exists a dedicated instruction for setting the current position address. This mechanism has been used to move the cursor between the lines up and down. All that is needed is to read the value of the sixth bit in the status byte, revert it (set zero if it was one or set one if it was zero) and set the address again. Such a procedure is used when the down key is detected.

Furthermore, there are other options of displaying new data on the display, i.e., the newly sent character will always be stored in the current cursor position (address), regardless of the cursor being visible or not. In Example 2, two function keys, [Enter] and [Esc], have been used to change the way the data are displayed. When the former key has been pressed, entering new data results at the same time in the rotation of the whole display – in consequence, the cursor is at the same position all the time and is always visible. After pressing [Esc], the cursor moves to the right by one position after entering a new character and the display does not rotate. Thus, when the last visible position of the cursor is reached, the new character will be stored, but it will not be visible to the user.

The HD 44780 display driver can also be used for the definition of new, previously undefined characters, e.g., diacritic ones. In addition to the predefined character table, the user can define 8 new characters. This possibility has been illustrated in Example 3 below(again available on the computers in the laboratory).

```
;************************************************
; LCD
; EXAMPLE      -Definition of the new sign
;************************************************

        LJMP    START
        ORG     100H
START:

        MOV     R0,#LCDWC       ;set the address for instruction writing
        MOV     R1,#LCDRC       ;set the address for status reading

        MOV     A,#48H          ;set the Char-Generator (CG) address for the first character
        LCALL WRITE             ;

        INC     R0              ; set the address for data writing
        MOV     DPTR,#LITERA ;set in  DPTR the letter definition
        MOV     R3,#8           ; each character contains 8 bytes, counter used for that purpose
```

```
LOOP:                           ;the loop for writing the definition of new character to char generator
        CLR     A
        MOVC    A,@A+DPTR       ; read next byte
        LCALL   WRITE           ; and store it to the char generator
        INC     DPTR            ; modify the address
        DJNZ    R3,LOOP                 ; repeat for the remaining bytes (8 times together)

        DEC     R0              ;address for instruction writing

        MOV     A,#1            ; clear the display
        LCALL   WRITE

        MOV     A,#0FH          ;activate display, as well as cursor and its blinking
        LCALL   WRITE           ;

        MOV     A,#06H          ; set the direction of cursor shifting
        LCALL   WRITE           ;

        INC     R0              ; set the address for data writing
        MOV     DPTR,#TEXT      ; set the address of the text to be displayed on the LCD

WRITE_TXT:                      ; write the text on LCD
        CLR     A
        MOVC    A,@A+DPTR       ; get the character
        JZ      TEXT_END        ; if zero the text has finished
        LCALL   WRITE           ; write on the display
        INC     DPTR            ;increment the address
        SJMP    WRITE_TXT       ; and write the new character

TEXT_END:
        SJMP    $               ;finish

;***************************************
; Subroutine for writing data or instruction on LCD; the correct addresses stored in R0 and R1 are assumed

WRITE:
        MOV     R2,A            ;store the data
BUSY:
        MOVX    A,@R1           ;read the status of the driver
        JB      ACC.7,BUSY      ;wait until Busy Flag is zero
        MOV     A,R2            ; restore prestored data
        MOVX    @R0,A           ; and send the data
        RET

;***************************************
;Exemplary table that defines the Polish character 'ń'
LITERA:
        DB      00000010B
        DB      00000100B
        DB      00010110B
        DB      00011001B
        DB      00010001B
        DB      00010001B
        DB      00010001B
        DB      00000000B

;***************************************
TEXT:
```

```
        DB      'Gdansk '
        DB      'Gda',1,'sk',0
```

In Example 3, a new 8-byte-high letter „ń" has been defined – the activated pixels define the shape of the letter. Thus, through binary definition of the new letter in the code, its shape can be observed. Such letter definition can then be sent and stored in the dedicated driver memory called Char Generator. The Char Generator contains 64 bytes (starting from 00H) used for the definition of new 8 characters numbered from 0 to 7. In the above example, the "ń" character has been stored as sign 1, thus it occupies the memory from the address 08H to 0FH. In order to store the new character in the Char Generator, an appropriate address has to be set and a dedicated instruction has to be called. After that, the new character can be used inside the code. For example, the following text has been sent to the display driver: 'Gdansk Gda',1,'sk', which results in "Gdansk Gdańsk".

2. Exercise Flow

1. Analyze all examples provided in this instruction. Please execute them.
2. Write a program that will automatically rotate a pre-defined long text (50 characters including 3 new characters) allowing its easy reading. In particular:
   - When the program starts, the first 16 characters (1 to 16) should be displayed in the first line and 10 characters in the second line (41 to 50).
   - After pressing [←], the whole text should be rotated, allowing the display of hidden characters.
   - After pressing [→], the initial settings should be restored.