Lab 2: Inputs, Outputs, and Time Using Optrex DmcSeries LCD Display

Irina Tolkova # 1240609
Samuel Scherer # 1435365
Tremaine Ng # 1433161

Overview

In this lab, the Optrex DmcSeries 2 line, 16 char/line LCD display connected to a Beaglebone Black board was used to display characters fed into it through user input. The input was given by way of a FIFO pipe program. This document and the programs outlines usage of 4 bit operation mode. Both 4 and 8 bit modes are capable of the same functionality, but 4 bit operation sends two nibbles of data for a total of 8 bits of information. Jitter was also explored by creating and using a bit-flipping program heartbeat.c with an oscilloscope to see imperfections in waveform period.

Hardware Setup

Board to Device Connections

The Beaglebone Black board and the LCD screen were connected using a breadboard and jumper wires as shown in figure 1 below.

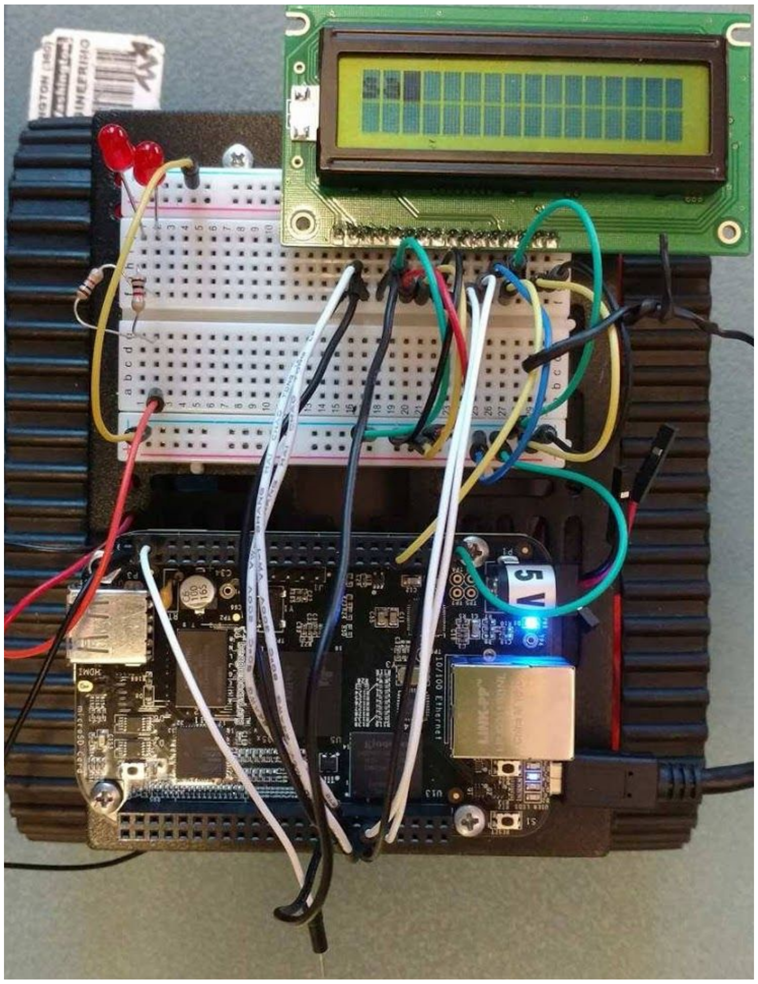


Figure 1: Beaglebone Black connected to LCD display

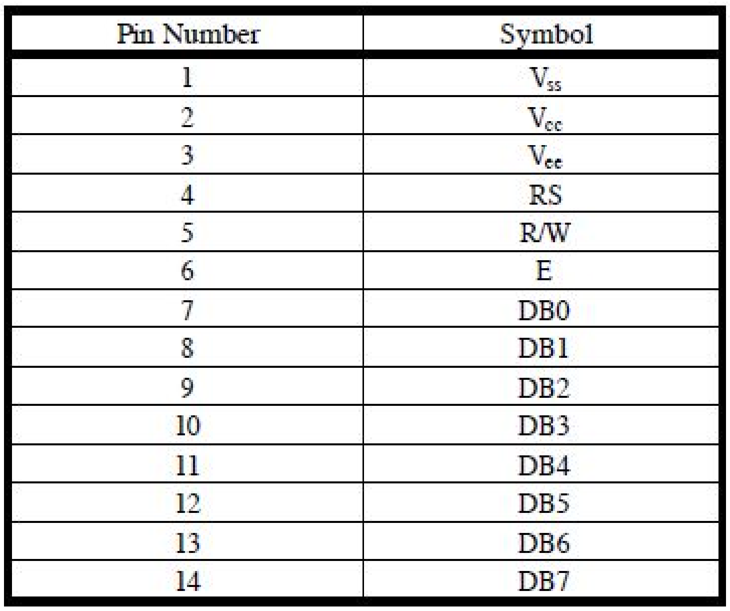
The Beaglebone interfaces with the LCD display using its GPIO pins. Following this is a list of the numbers of all GPIO pins used and their functions:

GPIO# - Function
- 69 - RS (register select)
- 45 - RW (read/write)
- 44 - Enable
- 26 - DB7 (data bus 7)
- 47 - DB6 (data bus 6)
- 46 - DB5 (data bus 5)
- 27 - DB4 (data bus 4)

More information on these functions can be found in the section below.

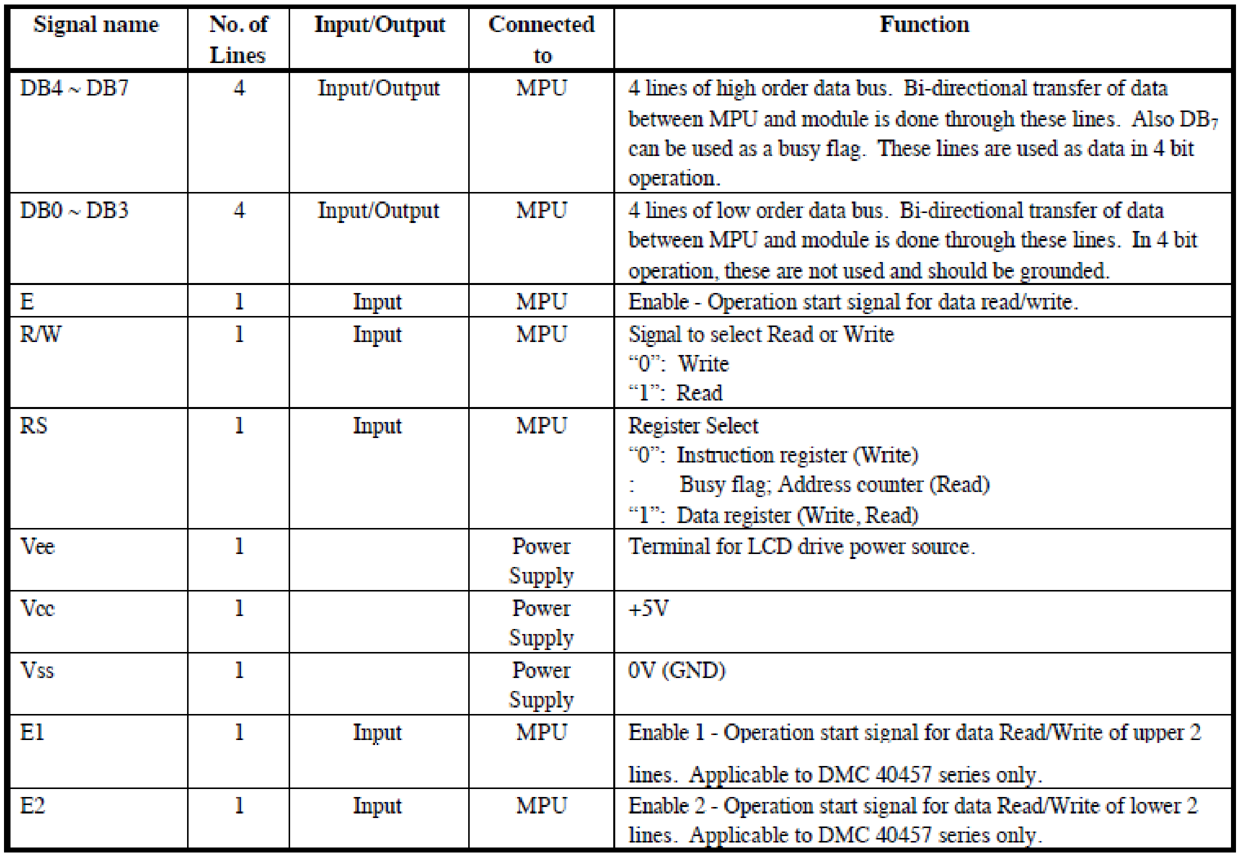
Device Pins

The Optrex DmcSeries LCD display used has a total of 16 pins, but only 14 of the pins' functions are known from documentation. The exact assignments are described in table 1.1 and table 1.2 below. These are taken from the DmcSeries LCD Module user manual. The device's pins function at 0v low and +5v high, but functioned at the Beaglebone Black's +3.3v high logic level.



Pin Number	Symbol
1	V_{ss}
2	V_{cc}
3	V_{ee}
4	RS
5	R/W
6	E
7	DB0
8	DB1
9	DB2
10	DB3
11	DB4
12	DB5
13	DB6
14	DB7

Table 1.1: Optrex DmcSeries Pin Labelling



Signal name	No. of Lines	Input/Output	Connected to	Function
DB4 ~ DB7	4	Input/Output	MPU	4 lines of high order data bus. Bi-directional transfer of data between MPU and module is done through these lines. Also DB_7 can be used as a busy flag. These lines are used as data in 4 bit operation.
DB0 ~ DB3	4	Input/Output	MPU	4 lines of low order data bus. Bi-directional transfer of data between MPU and module is done through these lines. In 4 bit operation, these are not used and should be grounded.
E	1	Input	MPU	Enable - Operation start signal for data read/write.
R/W	1	Input	MPU	Signal to select Read or Write "0": Write "1": Read
RS	1	Input	MPU	Register Select "0": Instruction register (Write) : Busy flag; Address counter (Read) "1": Data register (Write, Read)
Vee	1		Power Supply	Terminal for LCD drive power source.
Vcc	1		Power Supply	+5V
Vss	1		Power Supply	0V (GND)
E1	1	Input	MPU	Enable 1 - Operation start signal for data Read/Write of upper 2 lines. Applicable to DMC 40457 series only.
E2	1	Input	MPU	Enable 2 - Operation start signal for data Read/Write of lower 2 lines. Applicable to DMC 40457 series only.

Table 1.2: Optrex DmcSeries Pin Functions

There are 3 different pin categories consisting of power, control, and data bus.

The power supply Vcc and Vss pins expect +5V, and the third power pin Vee controls screen contrast, used to compensate for the effect of temperature on the LCD display. Connecting this to 0v gnd yields sufficient performance at room temperature.

The control pins RS (register select) and R/W (read/write) are function select pins, while a third EN (enable) pin is flipped to send the current data bus and function select values for internal operation. The RS selects between the device's DD (display data) or CG (character generator) rams. DD is used for writing to screen while CG allows the user to store their own preset characters. These 2 function select pins are used slightly differently for writing to DD ram in 4 bit operation mode.

The data bus pins send information about the character to print to screen or encode into CG memory. When writing to screen, the information is used to specify the location in a character table. The address of permanent CG ram characters corresponds to their ASCII encoding. When used to write to CG memory, the data bus pins are used to write the pixels of the character one line at at time.

In 8 bit mode, all 8 data bus pins are used to carry one instruction. In 4 bit mode, only the highest indexed pins (DB7-DB4) are used to carry instructions across two enable cycles. The highest order nibble is sent first, e.g. in printing the ASCII character “L” represented by 0100 1100, the pins would first be set to 0100, then 1100.

Initialization

To begin the using the board after power is connected it first needs to be initialized. During initialization, the the screen is cleared and operation parameters set, e.g. cursor blinking, move after print. In 4 bit operation, the process might be as follows:

1. Set to 4 bit data length (all instructions after this point send two nibbles)
2. Set to 4 bit data length again, select number of lines and pixel size of characters
3. Set cursor move direction and cursor/display shift mode
4. Clear display
5. Turn on display and set cursor display

This is not necessarily the only order that these functions can be performed. The display can be reinitialized without power cycling first to no ill effect. More information on the encodings for different initializing functions can be found in table A in the appendix.

Software Implementation

In general, our system depends on two programs: “lcdScreen.c”, which controls the pins necessary for the LCD screen, and “write_to_screen.c”, which is a command-line interface for user input. The former performs a 4-bit software initialization of the screen, and manages the display of characters as well as screen reset and cursor movement commands.

Setting Pin Values

Pin voltage levels are set in a function named setPins. This function is passed an integer value representing the value of a seven bit binary value corresponding to the pins. The values that make up this seven bit value represent, in order from least to most significant bits, pins connected to: DB4, DB5, DB6, DB7, En, RW, CS. The function works by shifting the bits right through integer p.division by 2 within a loop. A related function is named setPinsEn which sets the pins in the same manner as setPins, waits 5000 microseconds, sets the pins again with the same value +16 (flipping the enable pin’s value), waits 5000 microseconds, sets the pins one final time to the original value, (the enable pin is low now) and waits 5000 microseconds one last time. This function was used for the initialization and for printing piped characters.

User Input

We implemented a simple program “write_to_screen.c” to accept user input for text display. The user is expected to print one character at a time to the console and press the enter key, and the character is passed to the program “lcdScreen.c” and consequently displayed on the screen. The communication between programs is implemented by using a FIFO special file, which store unread input in the order it was entered. The program “write_to_screen.c” initializes a FIFO file at the location “/tmp/myfifo”, and opens it in a write-only mode; on the other end, the program “lcdScreen.c” opens the same file in read-only mode.

Representation of Characters
For an 8-bit system, characters are represented by their ASCII values, along with 3 bits corresponding to the values of the En, RW, and CS pins. For a 4-bit system, the data bus must be loaded twice, in two nibbles. To create these input values, we first read in the next character from the FIFO and cast it to an integer (the ASCII value). The higher-order four bits of this value are obtained by dividing the value by 16 (which is equivalent to shifting right by four bits). The lower-order four bits are obtained by a modulus-16 operation. In both nibbles, the enable pin is set to 0 and the CS pin is set to 1. The RW pin must be set to 1 for the first nibble and to 0 for the second nibble. To capture the value of these three pins, we add 64 and 32 to the first nibble, and we add 64 to the second nibble.

Note that since all standard keyboard characters have an ASCII value and are compatible with this board, this implementation supports lowercase and uppercase letters, numbers, spaces, and keyboard symbols.

Newline Functionality
We implemented wrap-around functionality, such that if one line becomes filled, the cursor and display moves to the start of the second line. This was implemented by tracking the number of characters printed, and executing the command for shifting to the second line once that number equals 16. Additionally, if both lines become full (the number of printed characters exceeds 32), we execute the command for a clear and reset of the display, and the cursor moves back to the start of the first line.

To correctly respond to a newline character (“\n”), we made a special conditional statement which executes upon receiving the integer ASCII code for “\n”, equal to 10. If the cursor is on the first line and the input is “\n”, the cursor and display move to the start of the second line. If the cursor is on the second line and the input is “\n”, the cursor and display move to the start of the first line, and the screen is reset.

Evaluating Jitter
To assess the amount of jitter present in our signal, we wrote a program “heartbeat.c” adapted from an LED blinker program. Similarly to the programs for blinking LEDs in Lab 1, we initialize the GPIO pin 20, set it to output, then alternate writing 0 and 1 to its respective “value” file. The frequency of the signal is defined by the amount of time spent waiting in between the write

operations. We measured the length of multiple intervals of scheduler jitter at 1 microsecond of waiting (1 MHz) as well as 0.1 microseconds of waiting (10 MHz).

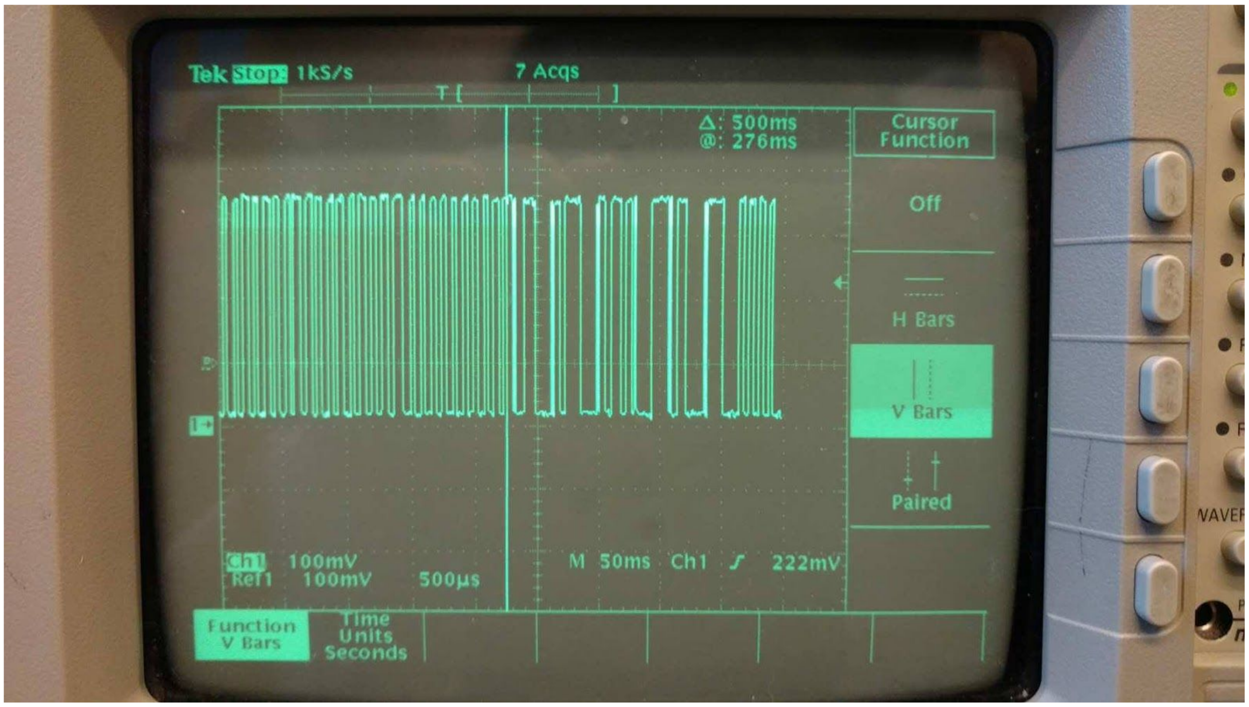


Figure 2: Oscilloscope Showing Jitter

Discussion

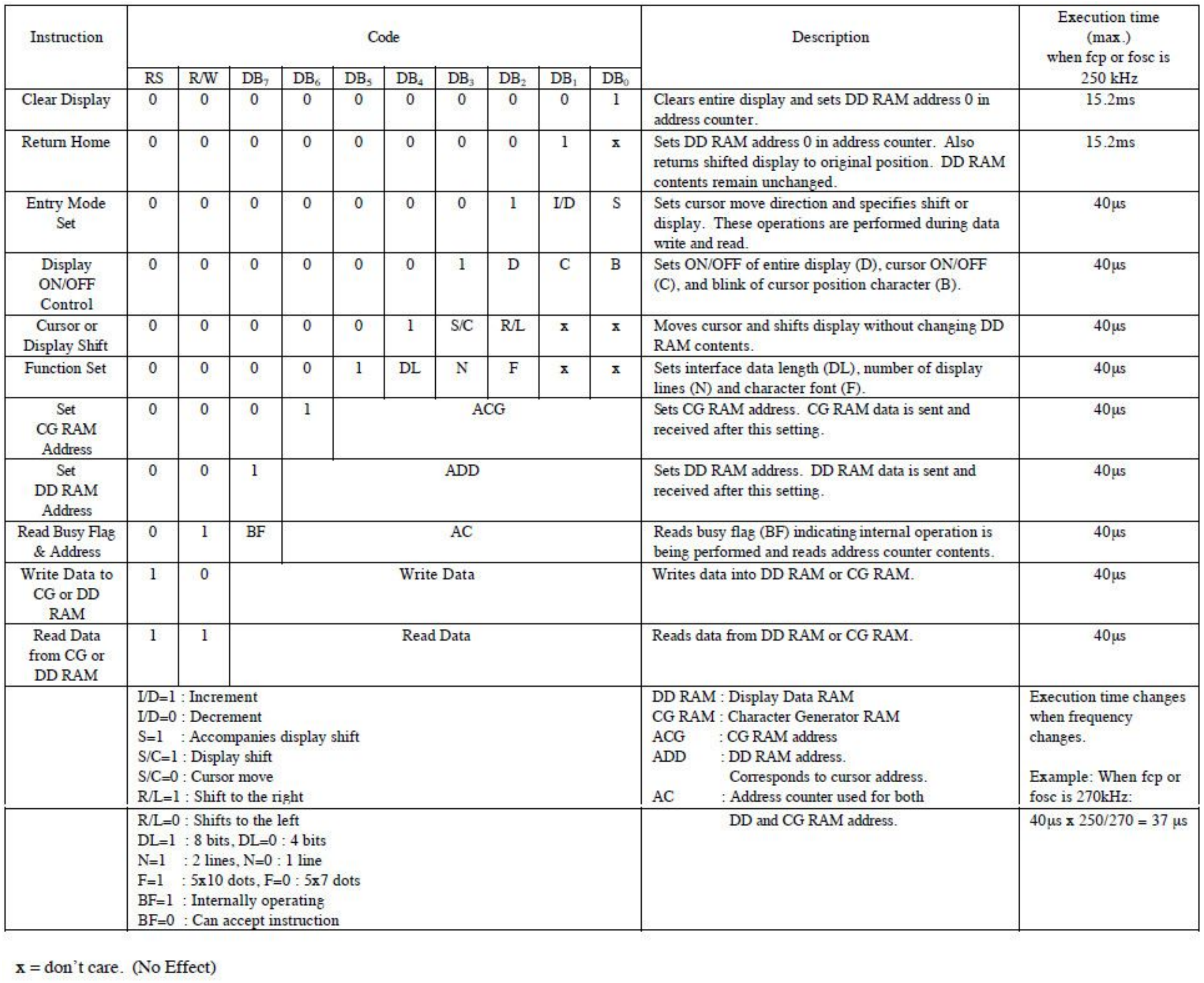
Only single characters can be sent to the display. This is because our group found that the strings were getting distorted when we tried to implement that functionality. A new line character is also supported by pressing return twice when typing input. This is accomplished by checking if the character is a return, moving to the next line if it is and on the first line, and clearing the display and moving to the first line if it is on the second line. Our design does not allow for looking at previous text once it has scrolled.

A 4 bit instead of 8 bit data bus for the LCD screen was used because pin 9 (DB2) on the LCD screen was broken. When the LCD had power it would go to about 1.5 volts. Our group realized that the screen was broken too late to ask for a new screen so we proceeded with a 4 bit design. This design has the upside of fewer wires but downside of slower speed.

Conclusion

In all, we developed a system for writing single character input from a command line to an LCD display connected to the BeagleBone board. Further development of this project could incorporate functionality for writing a full string, rather than character-by-character input, and expand the number of possible input values to accept punctuation, newline commands, and other symbols. Allowing multiple character input in particular will allow keywords like CLEAR to be used to enact special display functions.

Appendix



Instruction	Code										Description	Execution time (max.) when fcp or fosc is 250 kHz
	RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DD RAM address 0 in address counter.	15.2ms
Return Home	0	0	0	0	0	0	0	0	1	x	Sets DD RAM address 0 in address counter. Also returns shifted display to original position. DD RAM contents remain unchanged.	15.2ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies shift or display. These operations are performed during data write and read.	40µs
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Sets ON/OFF of entire display (D), cursor ON/OFF (C), and blink of cursor position character (B).	40µs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	x	x	Moves cursor and shifts display without changing DD RAM contents.	40µs
Function Set	0	0	0	0	1	DL	N	F	x	x	Sets interface data length (DL), number of display lines (N) and character font (F).	40µs
Set CG RAM Address	0	0	0	1	ACG						Sets CG RAM address. CG RAM data is sent and received after this setting.	40µs
Set DD RAM Address	0	0	1	ADD							Sets DD RAM address. DD RAM data is sent and received after this setting.	40µs
Read Busy Flag & Address	0	1	BF	AC							Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	40µs
Write Data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	40µs
Read Data from CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	40µs
	I/D=1 : Increment I/D=0 : Decrement S=1 : Accompanies display shift S/C=1 : Display shift S/C=0 : Cursor move R/L=1 : Shift to the right										DD RAM : Display Data RAM CG RAM : Character Generator RAM ACG : CG RAM address ADD : DD RAM address. Corresponds to cursor address. AC : Address counter used for both	Execution time changes when frequency changes. Example: When fcp or fosc is 270kHz:
	R/L=0 : Shifts to the left DL=1 : 8 bits, DL=0 : 4 bits N=1 : 2 lines, N=0 : 1 line F=1 : 5x10 dots, F=0 : 5x7 dots BF=1 : Internally operating BF=0 : Can accept instruction										DD and CG RAM address.	40µs x 250/270 = 37 µs

x = don't care. (No Effect)

Table A: List of Instructions