Angelica Semenec

CmpE 127 Lab

Lab 6
System Integration: Calculator

Abstract:

The purpose of this lab is to integrate the functionality of the keypad, LCD and SRAM chips into an application. The application is a calculator that can add and subtract numbers entered via the keypad and output the equation as well as the answer onto the LCD.

Functionality

This lab integrates the functionality of the keypad, LCD and SRAM chips into a calculator application. When the program is run, the user will be prompted by the user-interface to either enter a math equation, read from SRAM, or quit the program.

When the user wants to do a mathematical operation, the keypad will be activated. The user can then enter in an equation that handles multiple addition and subtraction operations. The keypad runs through a continuous cycle to check for button presses and will only proceed to the next step when the user enters ‘=’. When the user presses a key, the key will be output to the console and then the data is stored in the SRAM starting at address 0 and incrementing in units of 1. After the user enters ‘=’, the program will proceed to calculate the result of the equation.

To calculate the total value, the program reads the data stored in the SRAM. The program stores the value in an integer array that will be used for the LCD. When it encounters an operation symbol, it will follow the previous operation symbol recorded and then update the new operation. When the SRAM reaches the end of the equation, it will finish the mathematical operation that was stored and output the total to the screen.

The program then parses the total value. If the value is negative, the negative symbol will be put on the LCD stack and the value will be made positive so the numbers can be parsed. Each number is then parsed and entered into the LCD stack. For example, 123 become 1, 2, 3. The LCD stack is then read through and each value is output to the LCD. The final display will show the mathematical equation entered by the user and the total value.

Once the mathematical operation is finished, the program returns to the initial prompt requesting user operation.

When the user reads from memory, they can enter in the address they wish to read from. The program will then send the address to the SRAM and output the stored data to the console. If the user had recently done a mathematical operation, the data that had been stored sequentially starting at address 0 can be read to the console.

Design

1602 16x2 LCD Instruction Set

Instruction	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears display, resets cursor
Return Home	0	0	0	0	0	0	0	0	1	-	Resets cursor
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Assign cursor direction, enable display shift
Display On/Off	0	0	0	0	0	0	1	D	C	B	Set Display(D) Cursor(C) Blinking(B)
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display shift
Function Set	0	0	0	0	1	DL	N	F	-	-	DL: 8/4-bit N: 2-line or 1-line F: 5x10 or 5x8 dots
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address
Read busy flag/ address counter	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Read flag and counter
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM

‘245 (Octal Bus Transceiver With 3-State Outputs)

Inputs		**Operation**
OE’	**DIR**	
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation

H = High Voltage, L = Low Voltage, X = Immaterial

‘74 (Dual D-Type Positive Edge Triggered Flip-Flop)

Inputs				Outputs	
Pre’	Clr’	Clk	D	Q	Q’
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H^	H^
H	H	^	H	H	L
H	H	^	L	L	H
H	H	L	X	Q0	Q0’

H = High Voltage, L = Low Voltage, X = Immaterial, Q0 = Hold, ^ = Rising Clock Edge

‘4002 (4-Input Nor Gate)

Input				Output
A	B	C	D	Y
L	L	L	L	H
H	X	X	X	L
X	H	X	X	L
X	X	H	X	L

X	X	X	H	L

H = High Voltage, L = Low Voltage, X = Immaterial

‘244 (Octal Buffer and Line Drivers) Voltage Table

Inputs		**Output**
G’	**A**	**Y**
L	L	L
L	H	H
H	X	Z

H = High voltage, L = Low voltage, X = Immaterial, Z = High impedence

IDT6116SA (SRAM) Voltage Table

Mode	**CS’**	**OE’**	**WE’**	**I/O**
Standby	H	X	X	High-Z
Read	L	L	H	Data_{out}
Read	L	H	H	High-Z
Write	L	X	L	Data_{in}

H = High voltage, L = Low voltage, X = Immaterial

‘373 Voltage Table

D_n	**LE**	**OE’**	**O_n**
H	H	L	H
L	H	L	L
X	L	L	Q_0
X	X	H	Z*

H = High voltage, L = Low voltage, X = immaterial, Z = High impedence*

‘641 Voltage Table

Control Inputs		**Operation**
G’	**DIR**	**‘641**
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation

H = High voltage, L = Low voltage, X = immaterial

‘08 (2-input AND gate) Voltage Table

Inputs		**Output**
A	**B**	**Y**
L	X	L
X	L	L
H	H	H

H = High voltage, L = Low voltage

‘04 (Hex Inverter) Voltage Table

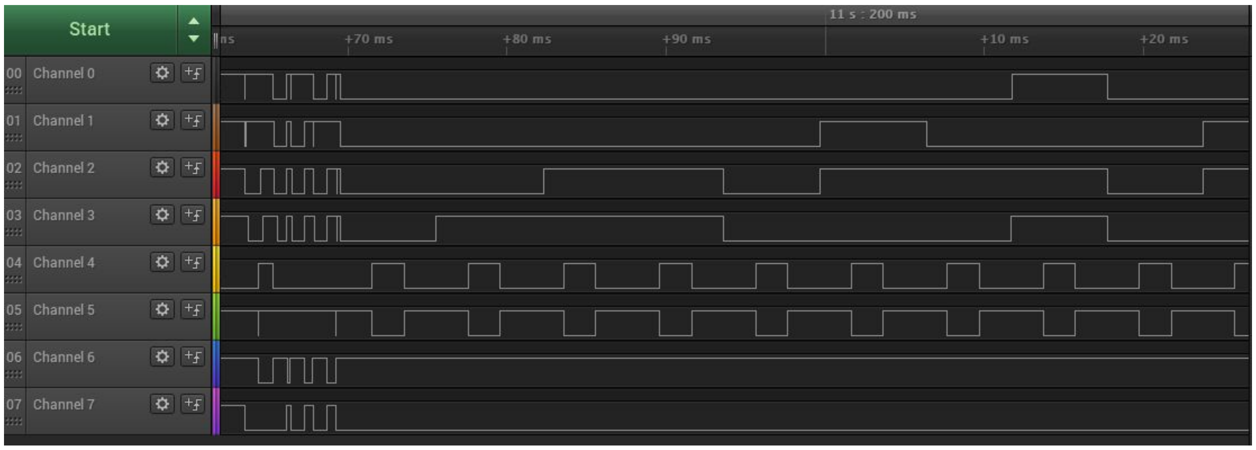
Input	**Output**
A	**Y**
L	H
H	L

H = High voltage, L = Low voltage

‘11 (3-Input AND gate) Voltage Table

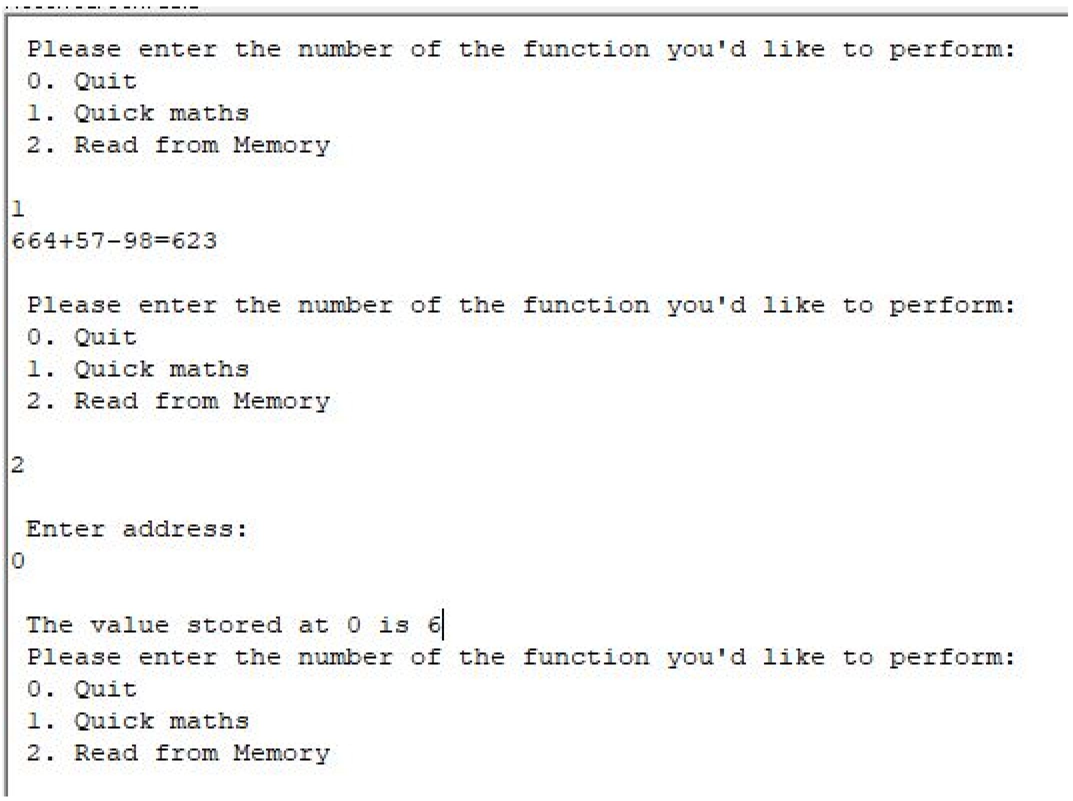
Inputs			Output
A	**B**	**C**	**Y**
H	H	H	H
L	X	X	L
X	L	X	L
X	X	L	L

H = High voltage, L = Low voltage, X = Immaterial



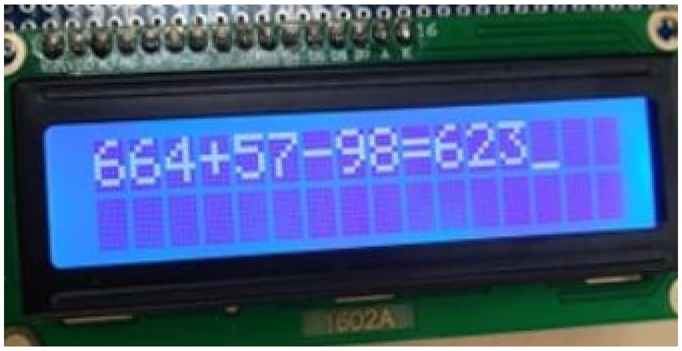
Output waveform from “Quick Maths” Operation. Chanell 0 through 3 are connected to D_7 through D_4 respectively on the LCD. Channel 4 is connected to the enable signal of the LCD. Channel 5 is connected to the enable signal for the keypad. Channel 6 is connected to the SRAM chip select for the lower half of memory and Channel 7 is connected to is connected to the output enable of the ‘245 that controls the signals going into D_7 through D_4 and RS of the LCD.

The above waveform shows when the process finishes the keypad routine. After the keypad is finished taking user input, the program reads from each address in SRAM until the final address that had been written into is read from. Each value is stored into a stack in the software and the total is calculated. After the total is calculated, the result is parsed and added to the stack in software. The stack is then read to the LCD. Channel 4 displays the pulsing of the enable signal for the LCD, which sends two nibbles of signals for each letter.



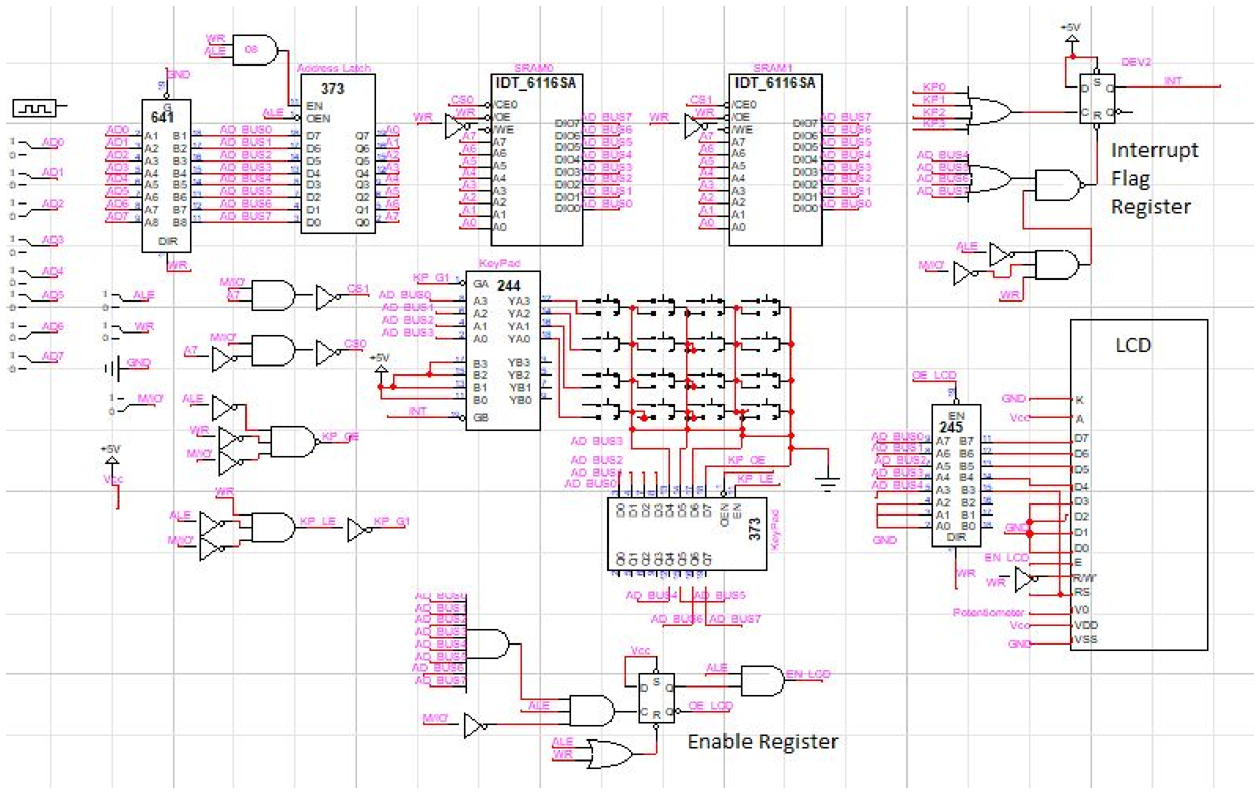
```
 Please enter the number of the function you'd like to perform:
 0. Quit
 1. Quick maths
 2. Read from Memory

1
664+57-98=623

 Please enter the number of the function you'd like to perform:
 0. Quit
 1. Quick maths
 2. Read from Memory

2

 Enter address:
0

 The value stored at 0 is 6
 Please enter the number of the function you'd like to perform:
 0. Quit
 1. Quick maths
 2. Read from Memory
```

Console output. Displays user selecting the "Quick Maths" option and entering in an equation. The user then selects to read from memory and reads the contents stored at address 0. The data output is 6, corresponding to the first value entered by the user during the Quick Maths calculation.



LCD output from the above Quick Maths operation. The equation and result are output to the screen.

Schematics



Lab 5 Circuit Schematics

When the user chooses to do a “Quick Maths” operation, the keypad is enabled. The program will send addresses 0001, 0010, 0100 and 1000 to the keypad looking for a keypress. If it detects a keypress, the LCD is disabled and the SRAM is enabled starting at address 0 and increasing sequentially. The value corresponding to the character from the keypad is then written into the SRAM and the keypad is re-enabled. The program will continue through this process until the user enters ‘=’.

Once the user enters ‘=’, the program will read from the SRAM starting at address 0. Each value is stored in a software stack until an operational character (+, -), is encountered and then it will perform a mathematical operation on the current value until the final address that had been written to is reached.

After the total has been calculated and the result is added to the stack of characters to be output to the LCD, the LCD is enabled. The LCD is sent the address FFh and the control signals for IO, and ALE. This stores the chip select in the enable register for the LCD. The inverted output is sent to the output enable of the ‘245 connected that is connected to the LCD. The direct output is ANDed with ALE so that the enable signal going into the LCD can be pulsed to allow the LCD to pick up nibbles of signals because it is operating in 4-bit mode. After the LCD has been selected, it is then fed each of the characters stored in the character stack. After each write, the characters are displayed to the LCD.

Parts List

1. 4 x 4 Keypad
2. 1602 LCD
3. IDT6116SA: CMOS Static RAM 16K (2K x 8-bit)
4. CD74LS4002: Dual 4-Input NOR Gate
5. SN74LS04: Hex Inverter
6. SN74LS08: Quadruple 2-Input Positive-AND Gate
7. SN74LS11: Triple 3-Input Positive-AND Gate
8. SN74LS74: Dual D-Type Positive Edge-Tripped Flip-Flops
9. SN74LS244: Octal Buffers and Line Drivers with 3-State Outputs
10. SN74LS245: Octal Bus Transceiver with 3-State Outputs
11. SN74HC373: Octal D-Type Transparent Latch
12. SN74LS641: Octal Bus Transceiver
13. 20-pin wire-wrapping DIP socket
14. 14-pin wire-wrapping DIP socket
15. 30 gauge wire
16. 40-pin ribbon cable
17. 1 Kohm resistors
18. SJTwo board
19. PCB

Algorithms

‘244 GA’:

GA’ = (WR * (M/IO’)’ * ALE’)’

‘373 LE:

(Latch Enable for the keypad)

LE = WR * (M/IO’)’ * ALE’

‘373 OE’:

(Output Enable for the keypad)

OE’ = (WR’ * OE’ * ALE’)’

‘74 CLK:

(Clock signal for Interrupt flag)

CLK = KP0 + KP1 + KP2 + KP3

KP is the output address of the keypad before the ‘373

‘74 CLR’:

(Clear signal for interrupt flag)

CLR’ = [(‘373)(Q0 + Q1 + Q2 + Q3)*(ALE’ * (M/IO’)’ * WR’)]’

INT:

(Interrupt signal)

INT = (‘74) Q

LCD EN:

EN = (‘74)Q * ALE’

LCD RS:

RS = AD_4

LCD R/W’

R/W’ = W’

‘245 DIR:

DIR = W

‘245 OE’:

OE’ = (‘74)Q’ * ALE’

Demo

The .hex file is generated through the WSL using the command “make build”. Once the .hex file is generated, it can be uploaded to the board via the hyperload tool. Hercules is then used to interact with the board via the interface.

A user-interface is generated and the user is prompted to select 1 of three different options. Option 0 quits the program. Option 1 allows the user to enter in a mathematical equation via keypad and output the equation and answer to the LCD. Option 2 allows the user to read data stored in the SRAM at a specified location.

When the user chooses to do a mathematical calculation, the user can enter in an equation that adds and subtracts. Each button pressed is stored sequentially into SRAM starting from address 0. The equation is displayed to the screen as the user types it. When the user enters '=', the program will calculate the final total. The program will then output the results to the LCD.

When the user chooses to read from SRAM, they can enter in an address. When the user enters in the addresses that the mathematical function used to store data during the keypad activation, the output is printed to the console and will display the data from the last mathematical operation that was performed.

Source Code

main.cpp

```
//Necessary libraries
#include <project_config.hpp>

#include <cstdint>
#include <iterator>
#include <stdlib.h>
#include <string>

#include <math.h>

#include "L2_HAL/displays/led/onboard_led.hpp"
#include "utility/log.hpp"
#include "utility/time.hpp"

//Bus class
class Bus
{
   public:
   enum class ControlType
   {
       kMemory = 0,
       kIO
   };
   Bus()
   {
       Write.SetAsOutput();
       ALE.SetAsOutput();
       M_IO.SetAsOutput();

       Status = 0;

       ad[0].GetPin().SetAsOpenDrain();
       ad[1].GetPin().SetAsOpenDrain();
       ad[2].GetPin().SetAsOpenDrain();
```

```
        ad[3].GetPin().SetAsOpenDrain();
        ad[4].GetPin().SetAsOpenDrain();
        ad[5].GetPin().SetAsOpenDrain();
        ad[6].GetPin().SetAsOpenDrain();
        ad[7].GetPin().SetAsOpenDrain();

    }

//Function writes to SRAM
    void WritetoMem(int address, int data)
    {
        sendAddress(address);
        Status = 0;
        writeData(data);
        Status = 0;

        return;
    }

//Function reads from SRAM
    int ReadfromMem(int address)
    {
        sendAddress(address);
        Status = 0;
        int data = readData();
        Status = 0;

        return data;
    }

//Function moves cursor to lower line of LCD
    void move_cursor()
    {
        ad[4].SetLow();
        Set_Low();
        ad[0].SetHigh();
        ad[1].SetHigh();
        pulse_enable();
        Set_Low();
        pulse_enable();
        ad[4].SetHigh();
    }

//Function clears LCD display and resets cursor
    void clear_display()
    {
        Set_Low();
        ad[4].SetLow();
        pulse_enable();
        ad[3].SetHigh();
        pulse_enable();
        return;
    }
```

```
//Function writes character to LCD
   void write_char(char symbol)
   {
       int ascii = int(symbol);
       int binary[8];
       ad[4].SetHigh();
       for (int i = 0; i < 8; i ++)
       {
           if((ascii & int(pow(2,i))) > 0)
           {
               binary[i] = 1;
           }
           else
           {
               binary[i] = 0;
           }
       }
       //Set upper 4 bits
       int count = 3;
       for(int i = 4; i < 8; i ++)
       {
           if(binary[i] == 0)
           {
               ad[count].SetLow();
           }
           else
           {
               ad[count].SetHigh();
           }
           count --;
       }
       pulse_enable();
       //Set lower 4 bits
       count = 3;
       for(int i = 0; i < 4; i ++)
       {
           if(binary[i] == 0)
           {
               ad[count].SetLow();
           }
           else
           {
               ad[count].SetHigh();
           }
           count --;
       }
       pulse_enable();
       inc_cursor();
   }

//Function increments LCD cursor
   void inc_cursor()
   {
       Set_Low();
       ad[4].SetLow();
       pulse_enable();
       ad[1].SetHigh();
       ad[2].SetHigh();
       pulse_enable();
   }

//Function sets LCD address bits to low
   void Set_Low()
   {
       ad[0].SetLow();
       ad[1].SetLow();
       ad[2].SetLow();
       ad[3].SetLow();
   }

//Function pulses the enable signal
   void pulse_enable()
   {
       Delay(1);
       ALE.SetLow();
       Delay(1);
       ALE.SetHigh();
       Delay(1);
   }

//Calculator main function
   void calculate()
   {
       int calc = 0;
       int memory_location = 0;

       //Function checks keypad until '=' is pressed
       while(calc != -1)
       {
           calc = calc_kp(memory_location);
           if (calc == 1)
           {
               memory_location ++;
           }
       }

       int total = 0;
       int numbers[32];
       char symbol[32];
       int num_top = -1;
       int operation = 1;
       int location_count = 0;
       int data;

      //Reads from SRAM
```

```
       //Stores numerical data or performs mathematical operation
        while (location_count <= memory_location)
        {
            if (location_count != memory_location)
            {
                data = ReadfromMem(location_count);
                symbol[location_count] = num_to_char(data);
            }
            else
            {
                symbol[location_count] = '=';
            }


            //not + or -
            if (data != 43 && data != 45 && location_count != memory_location)
            {
                num_top ++;
                numbers[num_top] = data;
            }
            else if (data == 43 || data == 45 || location_count == memory_location)
            {
                int index = num_top;
                int temp_val = 0;
                for(int i = 0; i <= index; i++)
                {
                    temp_val = temp_val + (numbers[num_top] * int(pow(10, i)));
                    num_top --;
                }
                if (operation == 1)
                {
                    total = total + temp_val;
                }
                else if (operation == 2)
                {
                    total = total - temp_val;
                }
                if (data == 43)
                {
                    operation = 1;
                }
                else if (data == 45)
                {
                    operation = 2;
                }
            }
            location_count ++;
        }

        printf("%d\n", total);

        //Checks if result is negative
        //Sets up negative result for parsing
        if (total < 0)
        {
            symbol[location_count] = '-';
            location_count ++;
            total = total * -1;
        }

        char num[32];
        int count = -1;
        int value = total;
        int x;
        //Parses number into individual numbers
        while (value != 0)
        {
            count ++;
            value = total / int(pow(10, count));
            x = value % 10;
            num[count] = num_to_char(x);
        }

        //Adds parsed value to stack for LCD
        while (count != 0)
        {
            count --;
            symbol[location_count] = num[count];
            location_count ++;
        }

        //Sets up LCD signals
        LCD_call();

        count = 0;

        //Clears the display and outputs the math equation and answer
        clear_display();
        for(int i = 0; i < 2; i ++)
        {
            while (count != location_count)
            {
                if (count == 16)
                {
                    move_cursor();
                }
                write_char(symbol[count]);
                count ++;
            }
        }

        //Disables LCD
        Write.SetLow();
        ALE.SetLow();
        return;
    }
```

```
//Handles keypad input
   int calc_kp(int memory_location)
   {
       setupIO();

       int address = 0;
       int state = 0;
       char c;
       for (int i = 0; i < 4; i ++)
       {
           setIOaddress(state);
           address = checkKeyPress(state);
           if (address != 0)
           {
               c = getChar(address);
               printf("%c", c);
               Delay(100);
               int num = char_to_num(c);
               Write.SetLow();
               Delay(1);
               //Stores keypad info in SRAM
               WritetoMem(memory_location, num);
               if (c == '=')
               {
                   return -1;
               }
               else
               {
                   return 1;
               }
           }
           state = nextState(state);
       }
       return 0;
   }

//Changes number to character
   char num_to_char(int num)
   {
       switch(num)
       {
           case 0: return '0';
           case 1: return '1';
           case 2: return '2';
           case 3: return '3';
           case 4: return '4';
           case 5: return '5';
           case 6: return '6';
           case 7: return '7';
           case 8: return '8';
           case 9: return '9';
           case 43: return '+';
```

```
            case 45: return '-';
        }
        return -1;
    }

//Changes character to number
    int char_to_num(char c)
    {
        switch(c)
        {
            case '0': return 0;
            case '1': return 1;
            case '2': return 2;
            case '3': return 3;
            case '4': return 4;
            case '5': return 5;
            case '6': return 6;
            case '7': return 7;
            case '8': return 8;
            case '9': return 9;
            case '+': return 43;
            case '-': return 45;
        }
        return -1;
    }

//Gets character from keypad address
    char getChar(int address)
    {
        char key;
        switch (address)
        {
            case 17: key = '1'; break;
            case 18: key = '4'; break;
            case 20: key = '7'; break;
            case 24: key = '*'; break;
            case 33: key = '2'; break;
            case 34: key = '5'; break;
            case 36: key = '8'; break;
            case 40: key = '0'; break;
            case 65: key = '3'; break;
            case 66: key = '6'; break;
            case 68: key = '9'; break;
            //#
            case 72: key = '='; break;
            //A
            case 129: key = '+'; break;
            //B
            case 130: key = '-'; break;
            case 132: key = 'C'; break;
            case 136: key = 'D';
        }
       return key;
```

```
   }

   //Sets up keypad signals
   void setupIO(void)
   {
       ALE.SetLow();
       M_IO.SetLow();
       Write.SetHigh();

       ad[0].SetAsOutput();
       ad[1].SetAsOutput();
       ad[2].SetAsOutput();
       ad[3].SetAsOutput();

       ad[4].SetAsInput();
       ad[5].SetAsInput();
       ad[6].SetAsInput();
       ad[7].SetAsInput();
   }

//Sets keypad address
   void setIOaddress(int state)
   {
       Write.SetHigh();

       switch (state)
       {
           case 0:
               ad[0].SetHigh();
               ad[1].SetLow();
               ad[2].SetLow();
               ad[3].SetLow();
               break;
           case 1:
               ad[0].SetLow();
               ad[1].SetHigh();
               ad[2].SetLow();
               ad[3].SetLow();
               break;
           case 2:
               ad[0].SetLow();
               ad[1].SetLow();
               ad[2].SetHigh();
               ad[3].SetLow();
               break;
           case 3:
               ad[0].SetLow();
               ad[1].SetLow();
               ad[2].SetLow();
               ad[3].SetHigh();
               break;
       }
       Delay(1);
   }

//Checks for key press
   int checkKeyPress(int state)
   {
       Write.SetLow();
       Delay(1);
       for (int i = 4; i < 8; i ++)
       {
           if (ad[i].Read() == true)
           {
               return (int(pow(2,state) + int(pow(2, i) ) ) );
           }
       }
       return 0;
   }

//Updates keypad address state
   int nextState(int state)
   {
       state ++;
       if (state == 4)
           state = 0;
       return state;
   }

//Sends address to SRAM
   void sendAddress(int address)
   {
       Status = 1;

       ALE.SetHigh();
       M_IO.SetLow();
       Write.SetHigh();

       for (int i = 0; i < 8; i++)
       {
           ad[i].SetAsOutput();
           if((address & int(pow(2,i))) > 0)
           {
               ad[i].SetHigh();
           }
           else
           {
               ad[i].SetLow();
           }
       }
       return;
   }

//Writes data to SRAM
   void writeData(int data)
   {
       Status = 2;

       ALE.SetLow();
       M_IO.SetHigh();
       Write.SetHigh();

       for (int i = 0; i < 8; i++)
       {
           ad[i].SetAsOutput();
           if((data & int(pow(2,i))) > 0)
           {
               ad[i].SetHigh();
           }
           else
           {
               ad[i].SetLow();
           }
       }
       return;
   }

//Reads data from SRAM
   int readData()
   {
       Status = 3;

       Write.SetLow();
       ALE.SetLow();
       M_IO.SetHigh();

       int data = 0;

       for (int i = 0; i < 8; i ++)
       {
           ad[i].SetAsInput();
           if(ad[i].Read() == true)
           {
               data = data + int(pow(2,i));
           }
       }

       return data;
   }

//Initializes LCD for 4-bit setup
   void LCD_setup_4bit()
   {
       power_on();
       configure_4bit();
   }

//Power on LCD signals
   void power_on()
```

```
   {

       Delay(15);
       ad[4].SetLow();
       Delay(1);
       ALE.SetLow();
       Delay(5);
       ALE.SetHigh();
   }

//LCD 4-bit configuration
//Prints “QuickMaths”
       void configure_4bit()
   {
       ad[2].SetHigh();
       pulse_enable();
       pulse_enable();
       Set_Low();
       ad[0].SetHigh();
       pulse_enable();
       Set_Low();
       pulse_enable();
       ad[0].SetHigh();
       ad[1].SetHigh();
       ad[2].SetHigh();
       pulse_enable();
       Set_Low();
       pulse_enable();
       ad[1].SetHigh();
       ad[2].SetHigh();
       pulse_enable();

       char qm[] = "QuickMaths";
       for(int i = 0; i < 10; i ++)
       {
           write_char(qm[i]);
       }

       Write.SetLow();
       ALE.SetLow();
       return;
   }

//Function to set LCD enable flag
   void LCD_call()
   {
       M_IO.SetLow();
       Write.SetHigh();
       ALE.SetHigh();
       for (int i = 0; i < 8; i ++)
       {
           ad[i].SetAsOutput();
           ad[i].SetHigh();
```

```
        }
        Delay(1);
        Set_Low();
    }

    private:
    Gpio Write = Gpio(0, 17);
    Gpio ALE = Gpio(0, 22);
    Gpio M_IO = Gpio(0, 0);
    //Gpio Int = Gpio(0, 11);

    int Status;

    Gpio ad[8] = {
      Gpio(2, 2),
      Gpio(2, 5),
      Gpio(2, 7),
      Gpio(2, 9),
      Gpio(0, 15),
      Gpio(0, 18),
      Gpio(0, 1),
      Gpio(0, 10)
    };
};

int main(void)
{
    Bus obj;
    int op = -1;

    obj.LCD_call();
    obj.LCD_setup_4bit();

//User interface
    while(op != 0)
    {
        printf("\n Please enter the number of the function you'd like to perform:");
            printf("\n 0. Quit");
            printf("\n 1. Quick maths");
            printf("\n 2. Read from Memory\n");
        scanf("%d", &op);
        if (op == 1)
        {
            obj.calculate();
        }
        else if (op == 2)
        {
            int address;
            int data;
            printf("\n Enter address: ");
            scanf("%d", &address);
            data = obj.ReadfromMem(address);
            printf("\n The value stored at %d is %d", address, data);
        }
        else
        {
          printf("\n The operation entered is invalid.\n");
        }

    }
    return 0;
  }
```