

ENGI E1112 Computer Science Computer Engineering Lab Report: Developing A Fully Functional HP20b Calculator

Jay Shim, SonYon Song

May 2012

Abstract

Embedded systems are often used for devices with specific functional restraints. HP 20b calculators are such devices. In addition to the hardware and mechanical parts, HP 20b calculators are equipped with embedded systems which contain firmware stored in Flash memory chips. The firmware must support real-time computing constraints, use algorithms that are fast enough to meet strict time constraints, and make the most efficient use of limited memory space. The goal of the lab project was to use this embedded programming in C to create a firmware for the HP 20b calculator.

The lab was broken down into four parts, with each lab adding a specific functionality to the HP 20b calculator. The aim of the first lab was to successfully display a predetermined number (negatives, zeros, as well as positives). The objective of the second lab was to determine and display which key was pressed. The third lab used the code from the second lab to continuously display numbers until an operation was pressed. The final output would be the number entered along with the symbol for the operation pressed. The fourth and final lab used the Reverse Polish Notation (RPN) method that allows the user to perform simple algorithms on the HP 20b calculator.

1 Introduction

Manufactured by Hewlett Packard, the HP 20b Business Consultant was created with business professionals and students in mind [5]. But the stylish HP 20b became nothing but a brick when all of its functionalities were stripped away by

Professor Stephen A. Edwards. Thanks to Professor Edwards, we were left with a non-functional calculator and a JTAG Platform. From scratch, we successfully built a fully functional RPN calculator. This paper describes how the calculator works and discusses the hidden components of the calculator that truly make it run.

2 User Guide

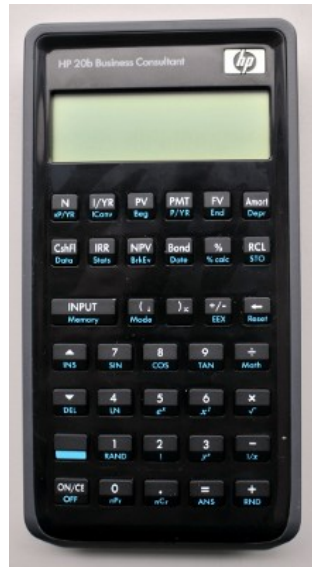




Figure 1: The HP 20b

2.1 Turning HP 20b ON/OFF

To turn the HP 20b on, press . To turn it off, press .

2.2 Entering Numbers

The user can enter digit by digit using the keys labeled 0 through 9. The user can enter up to twelve digits. In order to negate the number entered, the user can press

. To delete the most recent digit entered, the user can press .

2.3 RPN Method Computation



The RPN method utilizes () just as a normal arithmetic calculator does. The only difference is that it uses the idea of stacks to store entries [2]. The following examples demonstrates how the stack works. (NOTE: An error occurs when dividing and restarts the program. Also, in the actual HP 20b calculator, the stack is vertical, not horizontal as depicted in the figures. The stack expands up and down, and has multiple "levels" with lowest level at the bottom. We decided to portray the stack as horizontal in order to depict the integer array we used in our code.)

Example 1: 12×3


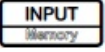

First, enter 1, then enter 2, so that the number 12 is displayed on the LCD display screen. Now, this number will only be stored into the calculator after the user presses . This will put the number 12 into the stack. Next, enter 3. This time, the user does not have to press . We made our HP 20b so that pressing an operand automatically stores whatever number on the LCD into the stack. Lastly, press . (Figure 2)



Figure 2: Example 1 of RPN.

Example 1 demonstrates how to perform simple sequential calculations on the HP 20b calculator using the RPN method. However, what about expressions that are more complex? What if the correct calculation of the expression requires following the order of operations? How would you use the RPN method to compute product of sums, sums of products, etc.? In such cases, the order of operation is crucial. In order to correctly use the RPN method for more complex arithmetic, the user should find the closest set of parentheses

Example 2: $((2 + 3) \times 5) \times 6$

Start at the inner most pair of parenthesis. Thus, first compute 2+3 by entering


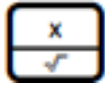

2, then 3, then pressing . Next, enter 5, then press  in order to carry out the operation contained by the next closest set of parenthesis. In this example, those are the outer set of parentheses. Lastly, enter 6, then press  to complete the outer most operation. (Figure 3)



Figure 3: Example 2 of RPN.

3 Social Implications

A calculator is an extremely convenient tool that one can use for finance, physics, real estate, and even household arithmetic purposes. It promotes productivity and efficiency by performing computations for the user. In addition, the calculator is relatively lightweight and portable, allowing it to be brought anywhere it may be needed. If one programs the calculator using RPN, which is somewhat easier to do, it can lower production costs, which can help it be mass produced in developing countries to increase productivity in schools, businesses, among other places. The Amtel processor also utilizes a very small amount of energy, so it can be used for an average of 9 month before the battery dies, which is ideal in places where power is scarce [3].

4 The Platform

In order to reprogram the HP 20b calculator, we required a hardware platform that consisted of the calculator itself, a 16-pin JTAG header, and a JTAG dongle, or USB adapter that was connected to a 20-pin ribbon connector cable. The JTAG header was soldered onto the calculator's circuit board in order to communicate to the processor through the JTAG port. The 20-pin connector cable, which was on one side of the JTAG dongle, was plugged into the JTAG header. Note that the red wire was on the left and that the four pins on the right side of the ribbon connector were not plugged in as they were irrelevant [1] [3].

4.1 The Processor

In addition to a keyboard and a liquid crystal display, or LCD, the HP 20b also has an Atmel AT91SAM7L128 processor, which is also known as SAM7L. This processor is one of Amtel's AT91SAM series of chips, which are all based on an ARM processor core. With respect to the model name, "AT" is for Atmel, "SAM" is "smart ARM core," 91 appears to be arbitrary. The 7L series of microcontrollers only consume a small amount to power, and the final 128 is a reference to the fact that it includes 128K of flash program memory [3].

Figure 4 shows a block diagram of the SAM7L processor. It is basically a single standard processor surrounded by memory and many different kinds of peripherals. Most of them will not be used for this project. The system controller manages the clock and power supply of each peripheral utilizing software. Thus energy can be saved if a peripheral is turned off, but it will not operate if it does happen to be off.

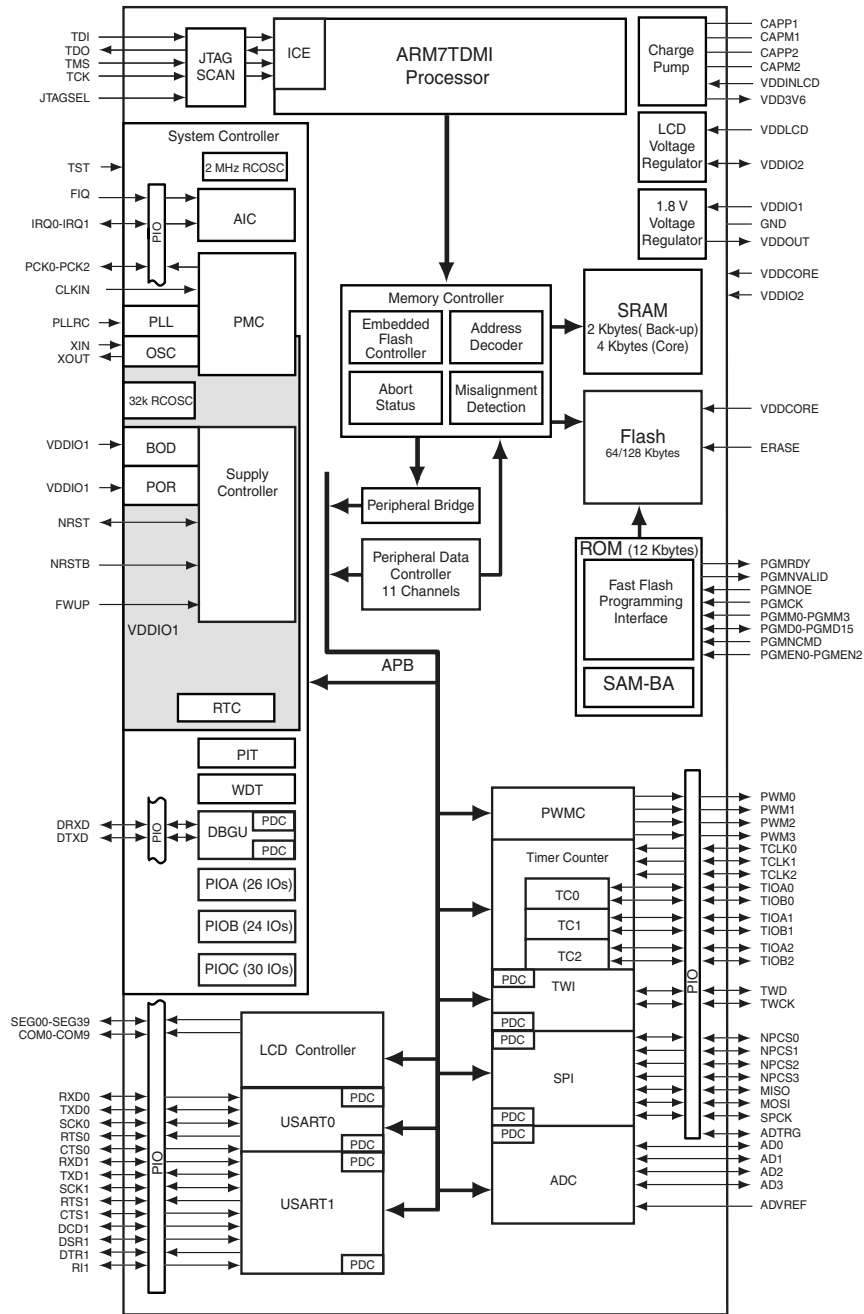


Figure 4: A block diagram of the AT91SAM7L128 micro-controller that is at the heart of the HP 20b

4.2 The LCD Display

The LCD controller creates the AC waveforms needed for the LCD display. The software interprets the LCD as a series of memory locations that control individual LCD segments. The LCD of the HP 20b consists of 15 s7 digit displays and two places that indicates a negative number: one for 12 regular digits and 3 for exponent digits [5]. There are also symbols for storage, notation, battery life, and so on (Figure 5).

lcd_init initializes the LCD display

lcd_put_char7 prints an ASCII character in a given column

lcd_print7 prints a string starting from the left most column

lcd_print_int_neg prints a right-justified integer that can be positive or negative based on a boolean

lcd_print_int prints a right-justified signed integer

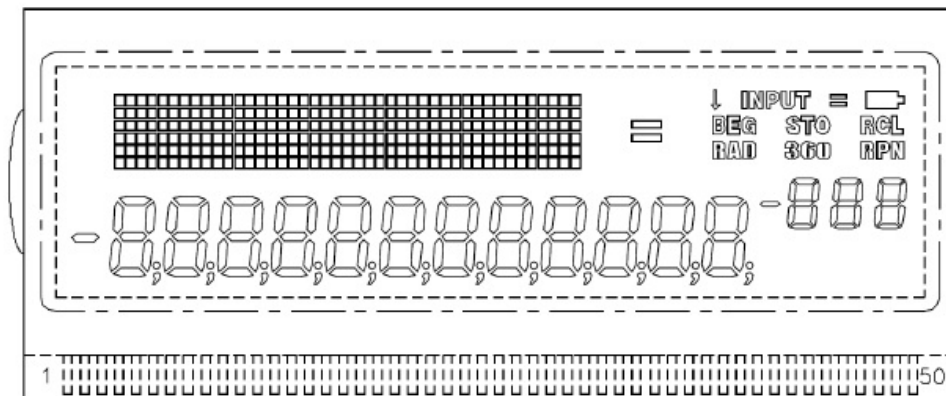


Figure 5: LCD Display

4.3 The Keyboard

Old keyboards were torn apart and 3 layers of plastic sheets were found under the keys. The outer sheets had circuit grids printed onto them while the middle sheets had holes at each pressure point from the keys. The center sheet separated the two circuits, preventing them from connecting to each other, but allowed the circuits to be completed when a key was pressed, which sent a signal to the processor. The keyboard on the HP 20b was similar to these old keyboards (Figure 6). The keyboard matrix, which is a 6x7 matrix of keys, is connected to pins on the SAM7L chip that can be driven by a parallel I/O controller, which is a peripheral that allows

software to control and read each pin [4]. Note that the HP 20b keyboard has its rows set up vertically and its columns horizontally, which is counter-intuitive. Functions were given to use to incorporate into our code related to the keyboard. `keyboard_init` set all the pins to high. `keyboard_column_high` set the pins of a certain column to high. `keyboard_column_low` does the opposite by setting a column to low. `keyboard_row_read` checks the row of low columns and returns true if a key was not pressed.

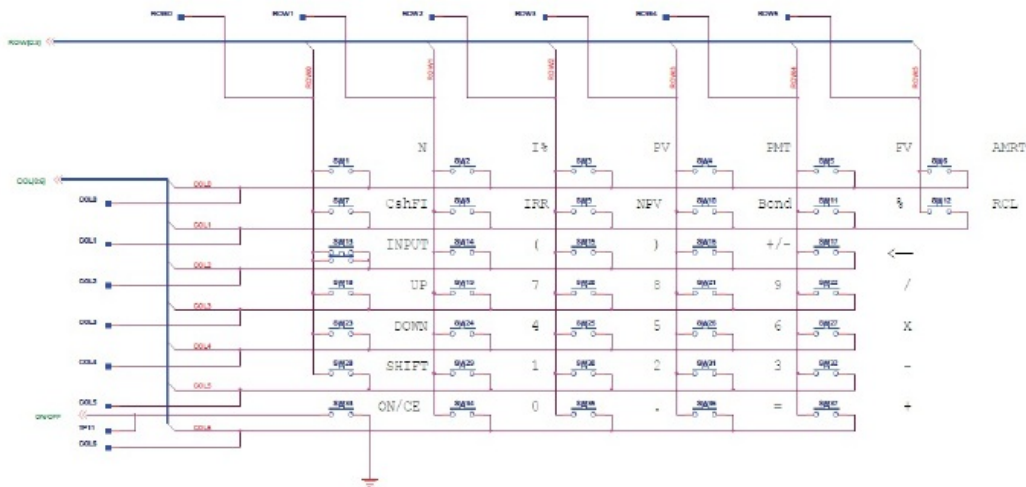


Figure 6: Keyboard

5 Software Architecture

We used Ubuntu Linux and OpenOCD, an open on-chip debugger, to create a development environment that allows communication with the SAM7L CPU. This is only possible when there is a successful connection between the JTAG and the USB port [3]. Also, in order to compile our code, we needed an ARM toolchain, which includes the C compiler, assembler, linker, and the debugger. Linux provided the GNU tool chain that we used to compile C code (i.e. `make flash`) [1]. In general, three methods interacted with each other to program our calculator to perform RPN. `keyboard_key` scans the keyboard and checks for which key was pressed. `keyboard_get_entry` shows numbers and operations on the screen using `keyboard_key`. The main method of the RPN lab allows the calculator to perform operations on numbers obtained by `keyboard_get_entry`.

6 Software Details

6.1 Lab 1: Getting Started: Hello World

For Lab 1, numbers had to be displayed on the LCD screen, whether they were positive numbers, negative numbers, or 0.

Figure 7 shows how we tried to show that on the HP 20b calculator. In the main method, the user would input the number that he or she would like to see displayed on the screen. “number” is the number to be displayed on the calculator, “digit” is the current digit that the program is looking at, and “lcdIndex” is the index on the LCD display. First, the program makes sure that the number is in range by checking that it is between, -999999999 and 999999999 because numbers outside of the range created an error. This was the range that we chose, but the actual range of usable numbers is from -int-max to int-max. The program is split into three parts depending on whether the number is positive, negative or 0. If positive, the program obtains each digit of the number by finding the remainder of the number when divided by 10 and printing the digit starting at the right most position. Then the number is actually divided by 10 and the process repeats until all of the digits have been printed. If the number is 0 then just 0 is printed. If the number is negative, then it undergoes the same process as the positive number except that each digit is negated to make it positive and a “-” is printed in the left most position on the screen. We realized that we should have made lab 1 into separate methods rather than putting everything into the main method so that the methods can be reused. Also, the code for printing the positive and negative number was very similar, so it probably could have been condensed.

6.2 Lab 2: Listening to the Keyboard

Before doing anything, the keys of the calculator were arranged in a 2d character array. The keyboard_key method (Figure 8) returns which key of the calculator was pressed. It does this by originally setting all of the columns of to calculator to high. Then, one of the columns is set to low and each row of that column is checked to see if the button was pressed. If it was pressed, it returns a 0, otherwise a 1 if the button was not pressed. Then the column is set back to high and the process repeats for the next column. the method returns the character of the button pressed.

In the main method (Figure 9), which is an infinite for loop, the program keeps checking to see if a button is pressed. If a key does happen to be pressed, the entire screen is cleared, accomplished using a for loop that incremented a counter j where a blank space was printed. The character of the key pressed is

then displayed in the right most position. If no key was pressed the calculator prints “no key pressed.”

6.3 Lab 3: Entering and Displaying Numbers

Figure 10 shows our code for lab 3. The `keyboard_get_entry` method is an infinite while loop. If a number is pressed, it is considered to be a digit and is added to the input with the other digits being shifted one index to the left, essentially by multiplying the input by 10 and adding the digit. Sign button changes the sign of variable “sign” when pressed, and this determines the sign of the input. Backspace removes the most recently added digit by dividing by 10. A blank screen is displayed if the input happens to be 0. Note that the key pressed is determined using the `keyboard_key` function from lab 2. If an operation is pressed other than a number, backspace, or the sign button, the for loop is broken. The values of the variables `operation` and `number` are based on the last operation pressed and the last input variable in the calculator. The sign of the number variable depends on the sign of “sign.” If no number was pressed before an operation, a predetermined `max_integer` is displayed.

The main method (Figure 11) is also an infinite while loop and it gets the entry of the keyboard using the function `keyboard_get_entry`. It gets updated as the user presses numbers, sign, and backspace. If an operation is pressed, it is displayed onto the screen. The while loop continues to listen for the next number input.

6.4 Lab 4: An RPN Calculator

In lab 4, shown in Figure 12, we recreate the calculator that uses RPN, described in Section 2. An integer array is used as the stack of numbers. The `calculate` method is used to perform operations on the top two numbers on the stack. It can perform addition, subtraction, multiplication, and division. The program reports an error if the index of the stack ever goes below 0. After undergoing the operation, the top most number is set to 0 and the result of the operation is displayed.

The main method (Figure 13) consists of an infinite while loop. Using the `keyboard_get_entry` function from lab 3, the screen displays the number that the user inputs, which is the number that is in the current index of that stack. If “input” is pressed, then the index of the stack increases and a new number can be entered. If +, -, *, or / is pressed, the top two numbers undergo the respective operation using the aforementioned `calculate` function. More than 1 operation can be pressed subsequently if more than 2 numbers are in the stack. If not an error message is displayed.

7 Lessons Learned

Through this project we learned embedded programming in C. We learned how to think through loops and algorithms in writing our programs rather than just use brute force. Also, we learned the efficiency of putting it on paper before starting our code. We grew accustomed to using variable names in a way that will help guide someone through the code without any confusion, and using comments that are not superfluous but instead help others understand what we did.

8 Criticism of the Course

Sometimes calculators would die and the program would not run, but this was only a minor inconvenience and the equipment was otherwise fine. Other than changing the battery once in a while there were no major problems with the calculators. Also, we enjoyed that we were able to follow our own approach in writing the code for each lab. Code reviews provided insightful comments and suggestions, and showed that everyone's code was different despite what looked like rather specific instructions

However, depending on one's experience with C, it was difficult to get started. Lab 1 was given to us, but the basics of C and the syntax were not explained, which made it difficult for people with no experience with C. A lecture on pointers was given later on, but it would be more helpful if we were given a brief crash course of C before starting Lab 1.

References

- [1] Hp-20b repurposing project. Online http://www.wiki4hp.com/doku.php?id=20b:repurposing_project.
- [2] Hp-20b online manual. Online http://h10010.www1.hp.com/wwpc/pscmisc/vac/us/product_pdfs/HP_20b_Online_Manual.pdf.
- [3] Engi 1112 lab 1. Online <http://www.cs.columbia.edu/~sedwards/classes/2012/gateway-spring/hello.pdf>.
- [4] Engi 1112 lab 2. Online <http://www.cs.columbia.edu/~sedwards/classes/2012/gateway-spring/keyboard.pdf>.
- [5] Hp-20b product description. Online <http://h10010.www1.hp.com/wwpc/us/en/sm/WF05a/215348-215348-64232-20036-215349-3732534.html?dnr=1>.

```

int main()
{
int number = 999 ;
int digit;
int lcdIndex = 11;
if (number > 999999999 || number < -999999999) {
    lcd_put_char7('O', 0);
    lcd_put_char7('V', 1);
    lcd_put_char7('E', 2);
    lcd_put_char7('R', 3);
    lcd_put_char7('F', 4);
    lcd_put_char7('L', 5);
    lcd_put_char7('O', 6);
    lcd_put_char7('W', 7);
} else {
    if (number > 0) {
        while (number > 0) {
            digit = number % 10;
            number = number / 10;
            lcd_put_char7(digit+48, lcdIndex);
            lcdIndex--;
        }
    } else if (number == 0) {
        lcd_put_char7(48, lcdIndex);
    } else if (number < 0) {
        while (number < 0) {
            digit = number % 10;
            digit = digit*-1;
            number = number / 10;
            lcd_put_char7(digit+48, lcdIndex);
            lcdIndex--;
        }
        if (number == 0) {
            lcd_put_char7(45, 0);
        }
    }
}
return 0;
}

```

Figure 7: Lab 1: Getting Started: Hello World

```

const char keyboard_layout [7][6] = {{'N', 'I', 'P', 'M', 'F', 'A'},
                                     {'C', 'R', 'N', 'B', '%', 'L'},
                                     {'T', '(', ')', 'G', '<', '_'},
                                     {'U', '7', '8', '9', '/', '_'},
                                     {'D', '4', '5', '6', '*', '_'},
                                     {'S', '1', '2', '3', '-', '_'},
                                     {'0', '0', '.', '=', '+', '_'}};

char keyboard_key()
{
    int column;
    int row;
    char button;

    for (column = 0; column <= COLUMN; column++) {
        keyboard_column_low(column);
        for (row = 0; row <= ROW; row++) {
            if(!keyboard_row_read(row)) {
                keyboard_column_high(column);
                button = keyboard_layout[column][row];
                return button;
            }
        }
        keyboard_column_high(column);
    }
    return 0;
}

```

Figure 8: Lab 2: Listening to the Keyboard (keyboard key method)

```

int main()
{
    int i;
    int j;
    for (;;) {
        char key = keyboard_key();
        if (key != 0) {
            for(j = 0; j < 14; j++) {
                lcd_put_char7('_',j);
            }
            lcd_put_char7(key,11);
        } else {
            lcd_print7("no_key_pressed");
        }
    }
    return 0;
}

```

Figure 9: Lab 2: Listening to the Keyboard (main method)

```

void keyboard_get_entry(struct entry *result) {
    int input = 0;
    int counter = 0;
    int sign = 1;
    for (;;) {
        int c = keyboard_key();
        if (c <= '9' && c >= '0') {
            input = input * 10 + (c - '0');
            counter++;
        } else if (c == '~') {
            sign *= -1;
        } else if (c == '\b') {
            input /= 10;
            if(input == 0)
                lcd_put_char7('_', 11);
        } else if (c != -1){
            result->operation = c;
            break;
        }
        while (c != -1) {
            c = keyboard_key();
        }
        if (input != 0)
            lcd_print_int_neg(sign == -1, input);
    }
    if (counter == 0) {
        result->number = INT_MAX;
    } else {
        result->number = sign*input;
    }
}

```

Figure 10: Lab 3: Entering and Displaying Number (keyboard get entry)

```

int main()
{
    struct entry entry;
    lcd_init();
    keyboard_init();
    lcd_print7("PRESS");
    while (1) {
        keyboard_get_entry(&entry);
        if (entry.operation != 0 && entry.number!=INT_MAX) {
            lcd_print_int(entry.number);
            lcd_put_char7(entry.operation, 0);
            entry.operation = 0;
        }
    }
    return 0;
}

```

Figure 11: Lab 3: Entering and Displaying Number main method

```

void calculate(char operation, int index, int *stack) {
    if (index < 0) {
        lcd_print7("ERROR");
    } else {
        int n1 = stack[index];
        int n2 = stack[index+1];
        switch (operation) {
            case '+': stack[index] = n1 + n2; break;
            case '-': stack[index] = n1 - n2; break;
            case '*': stack[index] = n1 * n2; break;
            case '/': stack[index] = n1 / n2; break;
        }
        stack[index+1] = 0;
        lcd_print_int (stack[index]);
    }
}

```

Figure 12: Lab 4: An RPN Calculator calculate method


```

int main()
{
    int stack [50];
    struct entry entry;
    int index = 0;
    while (1) {
        keyboard_get_entry(&entry);
        if (entry.operation != 0 && entry.number!=INT_MAX) {
            stack[index] = entry.number;
            lcd_print_int(entry.number);
            if (entry.operation == '\r') {
                index++;
                //if user presses an operation
            } else if (entry.operation == '+' || entry.operation == '-' ||
                entry.operation == '*' || entry.operation == '/') {
                index--;
                calculate(entry.operation, index, stack);
                index++;
            }
            //if user enters an operation without a number
        } else if (entry.operation == '+' || entry.operation == '-' ||
            entry.operation == '*' || entry.operation == '/') {
            if (index >= 2) {
                index = index - 2;
                calculate(entry.operation, index, stack);
                index++;
            } else {
                lcd_print7("ERROR");
            }
        }
    }
    return 0;
}

```

Figure 13: Lab 4: An RPN Calculator main method