ENGI E1112 Departmental Project Report: Embedded Design for an HP20b Calculator Computer Science/Computer Engineering

Aaron Burger, Isabel Baransky

May, 2012

Abstract

People with a cursory knowledge of electrical engineering are often led to believe that inside your average computer, there is a system of complicated circuits that 'do the work'. Many do not realize, that in between the input, those complicated circuits, and the output, there is an embedded system: functioning as the go-between for the voltages on the keyboard keys and the chips that handle addition, subtraction, and display.

The purpose of this project was to design that embedded system for an HP20b calculator–and over time, we did it. Carefully we progressed from displaying numbers on a screen to designing methods for the operations that make up the roots of arithmetic. By the end, our calculator could perform in two modes of operation, Reverse Polish Notation (postfix) i.e. 3 4 + » 7 and more standard infix notation i.e. 3 + 4 = 7. It was a long way from performing all the functions of the HP20b, but it was definitely on the right track.

1 Introduction

Complete with interest rate functions, Reverse Polish Notation input, and in-built functionality to handle bonds, depreciation, and cash flows, the HP20b was a calculator clearly designed for business. [3] And from 2008 until the product's discontinuation in 2011, it was used that way.

But the calculator we received on the first day of class was not designed for business, or really, anything at all. The device presented had no way of even demonstrating whether it was on or off. Fortunately though, we were also presented with a Joint Test Action Group or JTAG interface. One end of the interface could connect to a computer USB port, the other end had pins that matched with those physically located on the back of the calculator. This method of 'flashing' firmware to an embedded system was standardized by the IEEE for this exact purpose: the debugging of code operating on an embedded system. [1]

This paper is about our progression though this class, slowly building higher levels of code to return function to a calculator, while learning tenets of embedded system design that will aid us in our futures as engineers.

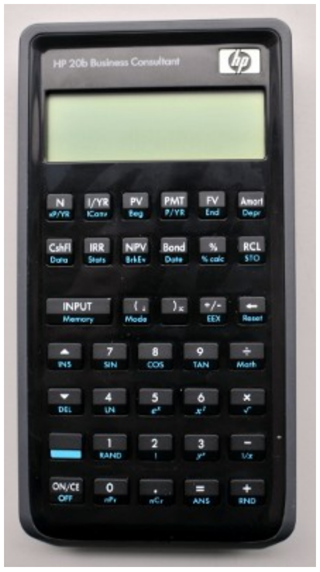

Figure 1: The HP 20b

2 User Guide

The calculator built is moderately typical for an everyday calculator. Nothing too surprising should show up, but in case you are from Mars and find yourself confused, this section will help you familiarize yourself with the calculator and how to coax it into functioning.

First, look at the calculator: the two major components are the keyboard and the display, which we'll go into more later. The HP20b can be turned on by pressing the ON/CE button the bottom left corner. After the calculator has been successfully turned on, the real fun can begin.

2.1 Displaying Numbers

Pressing any numerical key on the keypad should display that number on the right hand side of the display (the display stars numbers on the right and moves left-ward, typical of most calculators). For instance, you want to display the number 2.

1. Press the ON/CE button
2. Press the keypad with the number 2 written on it

2.2 Displaying Larger Numbers

The calculator works similar to physically writing a number on a scrap of paper. You start with the highest order number (ie the number 1 in 12,345) and continue to the right by powers of ten. If you want to display the number 253

1. Press the ON/CE button
2. Press the keypad with the number 2 written on it
3. Press the keypad with the number 5 written on it
4. Press the keypad with the number 3 written on it

2.3 Positive and Negative

Although the default setting is positive, a simple press of the +/- button will switch the sign of your number. If you want to display the number -2

1. Press the ON/CE button
2. Press the keypad with the number 2 written on it
3. Press the +/- button

2.4 Reverse Polish Notation Calculation

Our calculator can handle arithmetic functions using reverse polish notation. This way of doing calculations is not as intuitive. If you want to carry out the equation 3 + 4

1. Press the ON/CE button
2. Press the keypad with the number 3 written on it
3. Press INPUT
4. Press the keypad with the number 4 written on it
5. Press +

Following these steps should give you 7. RPN works by inputting the numbers in the equation first in the order desired and then inputting the manipulation methods in the order you want them to be carried out. On the inside, the numbers are placed in a stack, and the operations act on the top two elements in the stack. If you want to carry out the equation 3 + 4 + 7

1. Press the ON/CE button
2. Press the keypad with the number 3 written on it
3. Press INPUT
4. Press the keypad with the number 4 written on it
5. Press INPUT
6. Press the keypad with the number 7 written on it
7. Press +
8. Press +

Following these steps should give you 14. This also works for other operations, such as division, multiplication, and subtraction. However, note that the terms are grouped. Order of operations is not respected using this method–it is on a first come first serve basis. If you want to carry out the equation (3 + 4) x 7

1. Press the ON/CE button
2. Press the keypad with the number 7 written on it
3. Press INPUT
4. Press the keypad with the number 3 written on it
5. Press INPUT
6. Press the keypad with the number 4 written on it
7. Press +
8. Press X

The answer the calculator should give you is 49.

2.5 Using the Inflix Mode

In the infix mode, this HP20b behaves more like a traditional calculator. It is capable of performing a sum of products with traditional order of operations. Here is a demonstration of 2 + 5 x 8.

1. Press the ON/CE button
2. Press the keypad with the number 2 written on it
3. Press +
4. Press the keypad with the number 5 written on it
5. Press X
6. Press the keypad with the number 8 written on it
7. Press =

The answer the calculator should give you is 42.

2.6 Reset the Calculator

There is a reset function that has been programmed into the actual calculator. How to input and reset 45

1. Press the ON/CE button
2. Press the keypad with the number 4 written on it
3. Press the keypad with the number 5 written on it
4. Press the ON/CE button again

3 Social Implications

Providing basic code for the HP20b calculator serves not only an intellectual goal, but also a social, humanitarian goal with its applications. While the specific work that we did does not serve the human world at large in any meaningful way, this type of work is one of the most beneficial constructs of the computer age. Calculators, and thus their embedded systems, are found in equipment that keeps planes in the air, operate respiratory and dialysis machines, and control our infrastructure for long distance communications. These are functions that require the constant precision that human beings just don't have. With an embedded system, we can place our trust in a operation that will always perform as expected, so long as it was coded for correctly.

4 The Platform

The HP20b consists of a processor, a keyboard, an LCD display, and various other components for input, output and processing. As with most calculators, the purpose of the keyboard is to detect a user's interaction with the device, in the form of pressing a button. The processor uses instructions embedded into memory in order to process these inputs, then displays the results of those instructions on the LCD screen. Other components include the JTAG pins which allow us to flash new firmware, and internal components such as memory that we use for stack storage.

4.1 The Processor

It's not hard to argue that the most important component of the HP20b calculator is its Atmel AT91SAM7L128 processor. Like many electronics, the name doesn't do much to convey what the functionality, but that wont stop me from abbreviating it to SAM7L. It is part of Atmel's AT91SAM series of chips, which are all built around an ARM processor core ("AT" is for Atmel; "SAM" is "smart ARM core;" 91 may be arbitrary). The 7L series of microcontrollers are designed for low power (hence the L), allowing them to run off 3 volt watch batteries. The final 128 is a reminder that it includes 128K of flash program memory which we use to contain the active firmware and the stack (when we're using RPN mode).

Figure 4 shows a block diagram of the SAM7L chip. It's doesn't appear trivial, but processors in general can all essentially be reduced to a similar form. The processor stands surrounded by memory and a wide variety of peripherals, most of which we will not employ. Present here though, is the system controller, which, through software, controls the clock and power supply of each peripheral. This makes it possible to save energy by not powering on unneeded peripherals, but can also make a peripheral appear not to work if the system does not turn on its power.

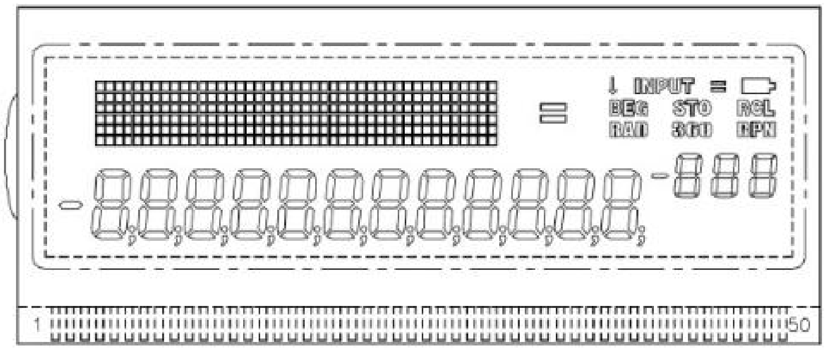

Figure 2: The HP 20b SDK LCD Screen

4.2 *The LCD Display*

The LCD display that was available consisted of 6x42 individual bits, then a row of 12 7-display modules, with 3 smaller 7-display modules. When the HP20b is functioning as advertised, the 6x42 'screen' of bits displays the operation being conducted, and the large 7-displays show the numerical components and result. The smaller 7-displays show the exponent in play, if there is one, and are most convincingly used with scientific notation. [2]

lcd_put_char7 was the primary function we used to write to the LCD screen. The function took a single character and a location from 0 to 14 as an argument, and displayed as close to that character as possible in the selected 7-display, going from left to right (and including the smaller 3).

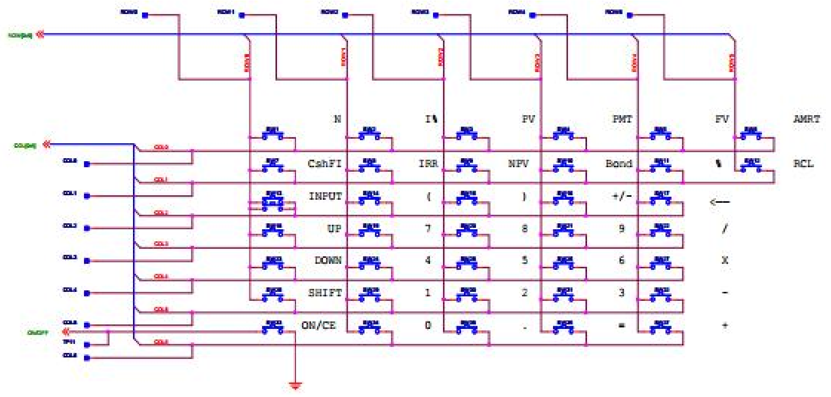

Figure 3: The HP 20b Keyboard Circuit Diagram

4.3 The Keyboard

The keyboard of the HP20b is essentially a table of buttons each rigged to an individual switch underneath. The buttons are placed on a material with slight suction, that in addition to conducting electricity, also functions as a spring to return the button to a raised position after being pressed. The grid above describes what's occurring electrically while we're busy pressing those buttons. If I were to press the 7 button, I would close a circuit connected to the column of the 7. If the column were turned on at that time, and I were to check the row of the 7, that is the only condition under which the electricity would flow through that wire and could be detected by the processor. This reduces what could be rows x columns number of sensors down to just the number of rows–very much streamlining the process.

5 Software Architecture

Good code is meant to be streamlined. Slowly building on itself in layers, each one working effectively with its predetermined use. And to an extent, the code for this project accomplishes that. The method at the center of everything is *myFavoriteNumber()*. Taking an unsigned integer and a boolean, this method displays to the screen that number, using the boolean as a modifier to decide whether to declare it positive or negative. From there our keyboard module senses human interface with the calculator, and our main method uses the combination of sensory input and display to perform calculations. Though all pieces were developed at different times, they now work together seamlessly to produce the desired result.

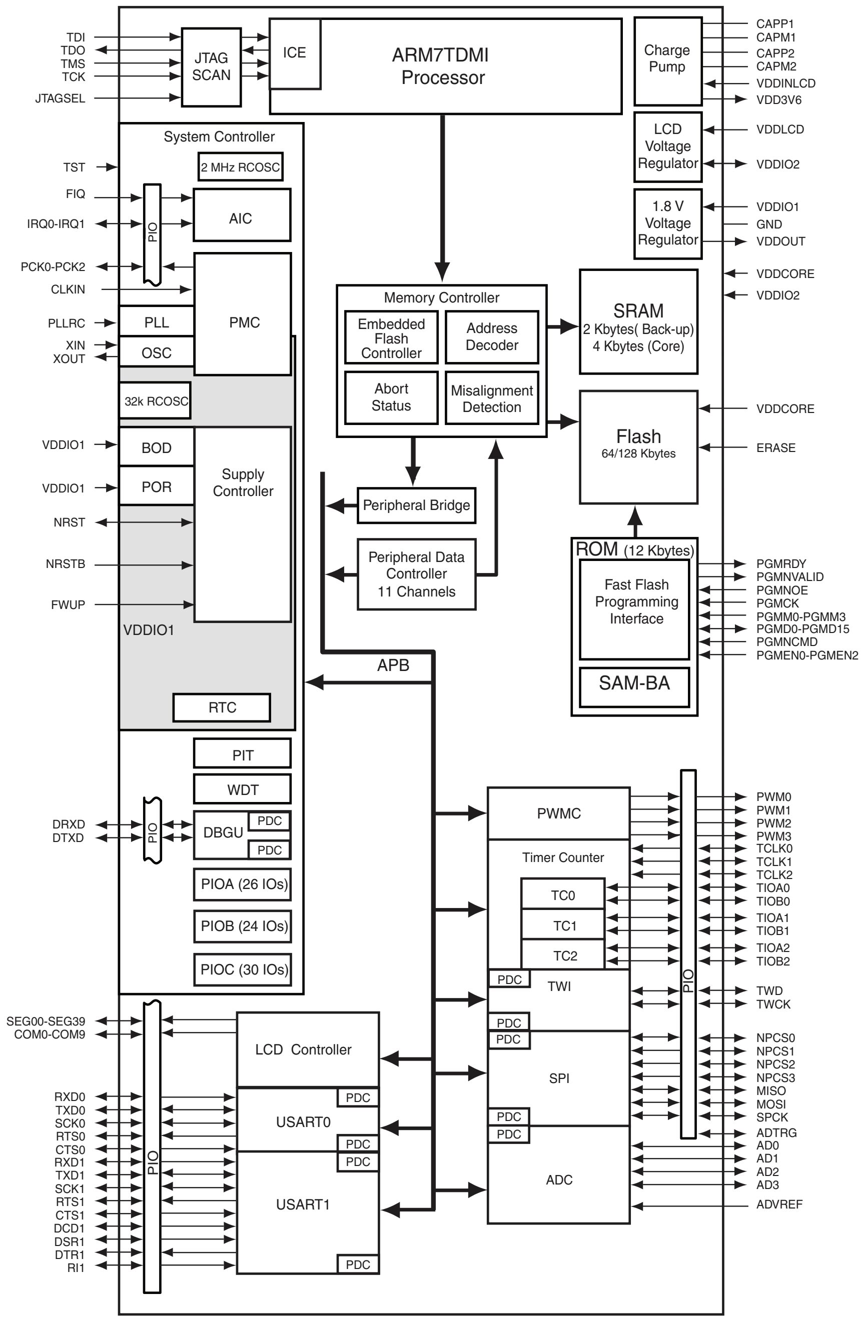

Figure 4: A block diagram of the AT91SAM7L microcontroller that is at the heart of the HP 20b

6 Software Details

6.1 Lab 1: A Scrolling Display

Initially, we have the calculator do the most basic function: display a number. Without this crucial ability, a calculator is rendered useless. We start off by calling the function myFavoriteNumber with the input of int x (what is input into the keyboard)

```
int myFavoriteNumber(int x)
```

It is important to understand how a calculator works and closely study things that might otherwise be written off as common sense. For instance, in a normal text editor, characters will be displayed from left to right. In the calculator we are working with, the text is first displayed at the rightmost position and moves left. Although this observation seems trivial, it is important to account for.

We tell the calculator to display the interger at position 11

```
int position = 11;
```

Therefore, our first bit of code established the position of the number that will be displayed. We declared the variable as "position" and give it a value of 11. Because the screen has 12 places, we are putting the number at the rightmost position, or 11 (0 counts as the first position, so 11 is the twelfth position). Now we start dealing with the actual number that will be displayed. Before we go into a logarithmic approach, we create exceptions for numbers that are difficult to deal with: the number zero and negative numbers.

```
if (x == 0) {
  lcd_put_char7(48, position);
  return 0;
}
```

Simply, if the code detects a zero, the calculator displays the number zero (in ACII, the number zero is formatted as 48) in position (set to 11). The loop then returns the value zero and we move on.

```
if (x < 0) {
  lcd_put_char7('-', 0);
  x = -x;
}
```

If the number is not zero and instead it is negative, the calculator deals with that using this if loop. We tell the calculator to display a negative sign at the first position (position 0), as the calculators of this type traditionally do. Then the number is treated as positive and is returned.

```
while (x != 0) {
    char d = (x%10 + 48);
    lcd_put_char7(d, position);
    x /= 10;
    position -= 1;
}

return (12-position);
```

If the number is not zero, the while loops is entered. A character d is established as the remainder of x divided by ten (so 58 would be 8). The character is displayed at the current position, and then the entire number is divided by ten (so 58 would be 5). The position has 1 subtracted from it and the process repeats. So 5 divided by ten remainder would be 5, and it would be input at position 10. Because a calculator works right to left, this would display 58. 5/10 would end up being zero, and because x would now equal zero, the while loop would terminate.

6.2 Lab 2+3: Scanning the Keyboard and Entering and Displaying Numbers

When we first scan the keyboard is when we move from being merely computer scientists, to being computer engineers. Through the method described in the keyboard section above, we use a sort of 'grid-search' method in the code where we turn on columns of the keyboard, checking to see if any of the rows respond. In this way we can pinpoint what button, if any is being pressed down, and once we have that, we're computer scientists again as we return that location to the main function, encoded as a two digit number.

With that location, we first check if we're in the grid that we 'know' how to use. For the purposes of this calculator, this grid is the numbers themselves and the operations we know how to apply (addition, subtraction, multiplication and equality). In this case, all we have to work with are numbers, so, limiting the number to the size of an integer, we multiply the number we have by 10, add the new number, and display the result. We combined these labs as, after finishing the second lab, our group went straight into the third, even before the specifications for it were available, seeing it as the next logical step. Unfortunately, we did not make any attempt to separate our code for them, and therefore we present them simultaneously.

6.3 Lab 4: An RPN Calculator

Before beginning this lab, we made several integrity changes to our architecture. For starters, we had the myFavoriteNumber function (that you should remember from earlier) taking an integer before. Now it takes an unsigned integer and a boolean. This is almost exclusively to allow for the use of '-0', which can be displayed on most calculators innately.

We also modified our sensory methods of the last lab. The keyboard function now returns a character instead of an encoded position.

As for making the calculator though, we first placed the operation sensing code in an infinite loop to allow for a steady stream of numbers and appropriate operations. Then we simply divided up our methods for dealing with different operators. If an 'input' was entered, we placed its operator on top of the stack. If an arithmetic operator was entered, we would apply it to its number, and the other number highest on the stack. If there was no other number on the stack, we would return an error. This might sound too simple, and if you think that, you've also discovered the reason why the early calculators used this method so often. RPN is an extraordinarily easy method to turn user inputs and operators into the results of mathematical calculations as it does not have any form of operator precedence.

When this was complete, our group was proud of themselves, but we decided to take the step further, and create a calculator that we wouldn't need extra instruction to learn how to use: an infix interpreter.

6.4 Lab 5: Sum of Products

Though we never accomplished the processing of parenthesis, by the time we were through with this lab, our calculator could produce the sum (and difference) of products. In order to do this, we had to overhaul our entire main processing method that we used above for RPN. Instead, we had a loop, but one that would look specifically for addition and subtraction to apply. If it encountered multiplication, it would enter an inner loop that would only be broken if it encountered an addition or subtraction operator.

For instance, if we wanted to calculate 3 + 4 x 5 x 1 = , our loop would start, encounter the addition and continue, storing 3. Then it would encounter the x and enter the inner loop, waiting for more multiplication operators. Once the equal sign was found at the end, the value of the current multiplication (4 x 5 x 1 = 20) would be added to the value stored for the sum (3) and the answer 23 would be produced. There were several optimizations that could have been made to combine the idea of an inner and outer loop, but unfortunately, by the time those optimizations could have been made, it was the conclusion of our lab and our progress would remain at this point.

6.5 Debouncing

```
while(keyboard_key() != -1) {
  continue;
}
```

Those 3 lines of code, so small they're easy to miss, crop up often in the various segments of this firmware–and what it does is another very interesting exercise in computer engineering. It used to be, with older, less stable keyboards, that if you pressed a key, there would be an undesirable ripple of signal voltage that would often misreport the number of times a button had been pressed. Therefore, de-bouncers were devised in order to remove this undesirable ripple. In this program, the debouncer waits for the keyboard function to declare that no key is being pressed, before allowing the program to continue. If it were continue, it could reenter the sensory loop and detect a button pressed for a second time, even if it has been only pressed once.

7 Lessons Learned

This lab taught us the importance of efficiency. Especially in programming, countless hours can be wasted if temporary laziness is not kept in check. Keep yourself organized, plan ahead, and don't take short cuts. It is always better to do things thoroughly and for the long haul. Only focusing on the present will not only make your future code much more complicated than originally intended, but possibly cause you to completely rewrite your earlier code. This lesson doesn't only apply to this specific setting, or even only Computer Science.

For future students: do not do the bare minimum. Learn C comfortably, do not settle on your code barely working. Flaws and loopholes only come back around later.

8 Criticism of the Course

Many students, us included, found the course was far too reliant on previous knowledge of programming. As a segment of a required introductory course, our expected level of knowledge was unreasonably high. Though topics like pointers and stacks were explained, for someone who has never ran a compiler before, those explanations were lost. One suggestion would be to section off much more time to teach C to the students so that everyone can be engaged. But there might also be a better way.

If you did know how to program, this course was a splendid demonstration of the application of computer science within an embedded system. Electrical engineers and computer engineers to were inspired by this demonstration, and encouraged to take more those electives in the future. Perhaps then, the solution is to split the class into two groups, one with programming experience and one without–both with separate sets of labs. Members of both groups would accomplish personal strides without the less experienced having to face down the barrel of an arcane topic that's often difficult to understand.

References

[1]

[2] Hewlett-Packard Company. *HP20b Business Consultant Quick Start Guide*. HP, first edition, 2008.

[3] Hp 20b business consultant. Online `http://www.educalc.net/1439486.page`.

```c
int myFavoriteNumber(int x) {
    int position = 11;
    if (x == 0) {
          lcd_put_char7(48, 11);
          return 0;
    }
    if (x < 0) {
        lcd_put_char7('-', 0);
        x = -x;
    }
    while (x != 0) {
        char d = (x%10 + 48);
        lcd_put_char7(d, position);
        x /= 10; // x = x/10
        position -= 1;
    }
    return (12-position);
}
```

Figure 5: Lab 1: Display a number

```
int keyboard_key () {
    int i = 0;
    int j = 0;
    for (i=0; i<7; i++) keyboard_column_high(i);

    for (i=0; i<7; i++) {
        keyboard_column_low(i);
        for (j=0; j<6; j++) {
            if (!keyboard_row_read(j)) {
                return j*10 + i;
            }
        }
        keyboard_column_high(i);
    }
    for (i=0; i<7; i++) keyboard_column_low(i);
    return -1;
}

int main() {
...
    char A[4][4] = { {'7', '8', '9', '/'},
        {'4', '5', '6', 'X'},
        {'1', '2', '3', '-'},
        {'0', '.', '=', '+'} };
    for (;;) {
        inn = keyboard_key();
        if (inn != -1) {
            res[0] = (inn - (inn % 10))/10;
            res[1] = inn % 10;
        }
   else {
            res[0] = -1;
            res[1] = -1;
        }
```

Figure 6: Lab 2+3: From keyboard to screen

```
   if (res[1]>2 && res[1]<8 && res[0]>0 && res[0]<6 && len < 10) {
            if (pause == 1) {
                num*=10;
   num+=A[res[1]-3][res[0]-1] - '0';
                len = myFavoriteNumber(num);
                pause = 0;
            }
        }
   else if (res[1]==0 && res[0]==0) {
            for (j=0; j<12; j++)
                lcd_put_char7('␣', j);
            num = 0;
   myFavoriteNumber(num);
   len=0;
        }
   else if (pause == 0){
            pause = 1;
        }
   }
...
}
```

Figure 7: Lab 2+3: From keyboard to screen

```
for (;;) {
        keyboard_get_entry(&beta);
        if (beta.operation == 'q') {
            opp = &op[0];
        }
        else if (beta.operation == '\r') {
            *opp = beta.number;
            opp++;
            while(keyboard_key() != -1) {
                continue;
            }
        }
        else if (beta.operation == '+' || beta.operation == '-'
|| beta.operation == '*')
{
            if (beta.newNum == 1)
                *opp = beta.number;
            else
                opp--;

            if (opp > &op[0]) {
                if (beta.operation == '+')
                    *(opp-1) = *(opp-1) + *opp;
                else if (beta.operation == '-')
                    *(opp-1) = *(opp-1) - *opp;
                else if (beta.operation == '*')
                    *(opp-1) = *(opp-1) * *opp;
                myFavoriteNumber(*(opp-1) < 0 ? -*(opp-1) :
*(opp-1), *(opp-1) < 0);
            }
            else {
                lcd_put_char7('r', 1);
                if (beta.newNum == 0)
                    opp++;
            }

            while(keyboard_key() != -1) {
                continue;
            }
        }
    }
```

Figure 8: Lab 4: RPN Calculator

```
    for (;;) {
        else if (beta.operation == '+' || beta.operation == '-') {
            if (beta.newNum == 1) {
                *opp = beta.number;
                if (opp == &op[0]) {
                    opp++;
                }
                else if (opp == &op[1]) {
                    *(opp-1) += *opp * xSign;
                    myFavoriteNumber(*(opp-1) < 0 ? -*(opp-1) :
*(opp-1), *(opp-1) < 0);
                }
                xSign = (beta.operation == '-' ? -1 : 1);
            }
            while(keyboard_key() != -1) {
                continue;
            }
        }
        else if (beta.operation == '*') {
            do {
                if (beta.newNum == 1) {
                    pHold *= beta.number;
                    myFavoriteNumber(pHold < 0 ? -pHold : pHold, pHold < 0);
                    keyboard_get_entry(&beta);

                    while(keyboard_key() != -1) {
                        continue;
                    }
                }
                else {
                    keyboard_get_entry(&beta);
                }
            } while(beta.operation == '*');
```

Figure 9: Lab 5: Sum of Products

```
            if (beta.operation == '+' || beta.operation == '-')
xSign = (beta.operation == '-' ? -1 : 1);

            if (opp == &op[0]) {
                *opp = pHold * beta.number;

                if (beta.operation == '=') {
                    myFavoriteNumber(*opp < 0 ? -*opp :
*opp, *opp < 0);
                    opp = &op[0];
                }

                else {
                    opp++;
                }
            }
            else if (opp == &op[1]) {
                *(opp-1) += pHold * beta.number * xSign;
                myFavoriteNumber(*(opp-1) < 0 ? -*(opp-1) :
*(opp-1), *(opp-1) < 0);
            }

            pHold = 1;
        }

        else if (beta.operation == '=') {
            if (beta.newNum == 1) {
                *opp = beta.number;
                if (opp == &op[1]) {
                    if (tOp == '-' || tOp == '+') {
                        *(opp-1) += *opp * xSign;
                        myFavoriteNumber(*(opp-1) < 0 ? -*(opp-1) :
*(opp-1), *(opp-1) < 0);
                        opp = &op[0];
                    }
                }
            }
```

Figure 10: Lab 5: Sum of Products