ENGI E1112 Departmental Project Report: Computer Science/Computer Engineering

Ankita Gore, Christina Huang, Shikhar Kumar

December 16, 2011

Abstract

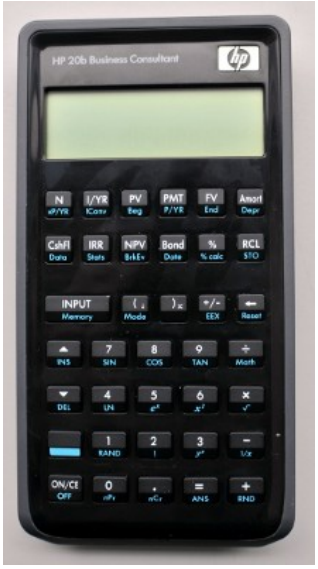
The overarching goal of the project was to learn to write new firmware for an HP 20b calculator. We were introduced to embedded programming, which is coding software for something that does not, and should not, appear to be a computer in the traditional sense, yet is one at its core. In addition, the project was meant to let us experience the challenges of designing systems ranging from traditional electrical issues like power consumption to high-level computer science problems like efficient algorithm design, and in turn, learn some standard solutions.

The project was made up of four labs, each of which allowed us to learn the intricacies of writing new firmware and build upon knowledge and code from the previous lab. The objective of the first lab was to write code that would implement a scrolling display on the HP 20b calculator. The objective of the second lab was to write code that would let the user enter and display numbers. The objective of the third lab was to write code that would read the keyboard on the HP 20b and display which key was pressed. The objective of the final lab was to write code that would allow the HP 20b to make calculations through the RPN method.

1 Introduction

The HP 20b Business Consultant is a financial calculator published in 2008 by Hewlett-Packard [3]. It is an open source calculator, since HP has made its hardware and software public. In this project, we programmed the calculator from scratch to work like an RPN calculator. The following document guides the user through the functionality of the calculator as well as the hardware and the software that make it possible.

2 User Guide



2.1 ON/OFF Key

To turn on the HP 20b calculator, press the ON/OFF key in the bottom left hand of the calculator. To turn it off, press the key above it and then the ON/OFF key.

2.2 Display

The calculator has a two-line display screen, but for the computations that we have programmed it to do, only the bottom line on the display screen will be used. The bottom line will display numbers as they are pressed.

2.3 INPUT Key

The INPUT key is used to enter numbers into stacks for the calculator's RPN mode.

2.4 Number Entry

The +, -, x and / (division) keys along the right side of the calculator keyboard perform the operations they indicate.

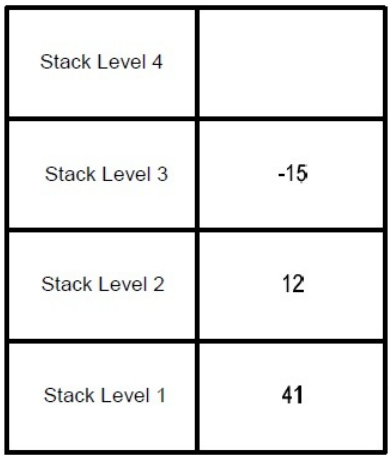
Numbers are entered using the 0 - 9 keys. To enter a number, press the number digits consecutively, with a maximum limit of 12 digits.

To delete the most recent number entry, press the backspace key (indicated by left-pointing arrow. The backspace key is located right above the / (division) key along the right side of the calculator keyboard. Each press of the backspace key erases the last digit or symbol that you entered.

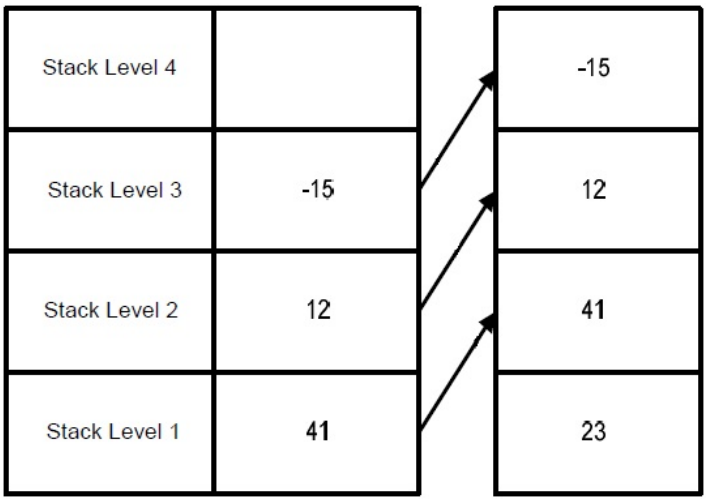
To change the sign of the number entered, press the +/- sign located next to the backspace key.

2.5 Computation in RPN

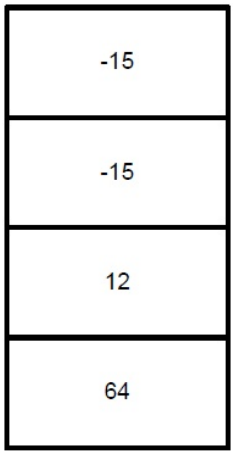
This is an RPN calculator. RPN works by placing numbers in storage registers called stacks. In our lab 4, the RPN stack has 10 levels. The levels are stacked on top of one another as shown in the following figure.



The initial stack contains 3 numbers: -15, 12 and 41. Numbers are added to the stack by entering the digits into the calculator display and pressing the INPUT key. When a number is added, it gets added to the bottom of the stack. The figure below shows how the stack moves when a new number 23 is added.



For any kind of computation (+, -, x and /), first enter all the numbers into the calculator by using the INPUT key to enter each number into the stack. Then press the desired operation key. The operation is performed with the most recent two numbers that were added to the stack (in stack levels 1 and 2). Below is how the stack looks when the + key is pressed.



When the two numbers are added, they are removed from the stack and the rest of the numbers move down the stack. Then the result of the addition is entered back into the stack. For subtraction, the number in stack level 1 is subtracted from the number in stack level 2 (level 2 - level 1). So, in this case, if the - key were pressed, the computation would be 41-23 = 18. For division, the number in stack level 2 is divided by the number in stack level 1 (level 2/level 1). So, if the / key were pressed, the computation would be 41/23. Unfortunately, division does not work at the moment because the calculator has not been programmed to handle decimals.

If another operation key is pressed after this, it will again use the numbers from stack level 2 and stack level 1 (the result of the previous operation). Operation keys can be pressed consecutively until there is only one number left in the stack. After this, no operation key will work unless more numbers are added to the stack.

NOTE: The above description of the calculator's display and keyboard only included what was relevant for our project. For a comprehensive description, refer to the Online Manual [5].

3 Social Implications

In this project, we programmed the HP 20b calculator to work as a basic calculator. The calculator that we have designed has great social implications. It makes every

one's lives easier by speeding up calculations in everyday life. When people go to a restaurant and have to tip, they can pull out their calculators and determine the right amount to pay. Students can use calculators in school to perform complicated calculations quicker, so that they can spend time understanding the concepts rather than performing tedious calculations. Calculators help to save time. Saving time allows people to focus on more beneficial activities such as volunteering to relieve poverty in Africa, getting clean water to people in villages, prevent World War III, etc. In the end, calculators help save the world.

4 The Platform

In order to implement our project, the HP 20b calculator, we used a hardware platform which required three main components: 1) the HP 20b Business Consultant Financial Calculator itself, 2) a 16-pin JTAG header, and 3) a JTAG dongle (USB adapter) connected to a 20-pin ribbon connector cable [1, 2]. In order to communicate with the calculator's processor through its built-in JTAG port, Professor Edwards had already soldered the JTAG header onto the calculator's circuit board to bring out the JTAG signals. The 20-pin connector cable, which was connected to one side of the JTAG dongle, was plugged onto the calculator's JTAG header, with the red wire on the left. The four pins on the right side of the ribbon connector were to remain unconnected, since they were negligible for this project. The images below show the JTAG header and how the hardware works.

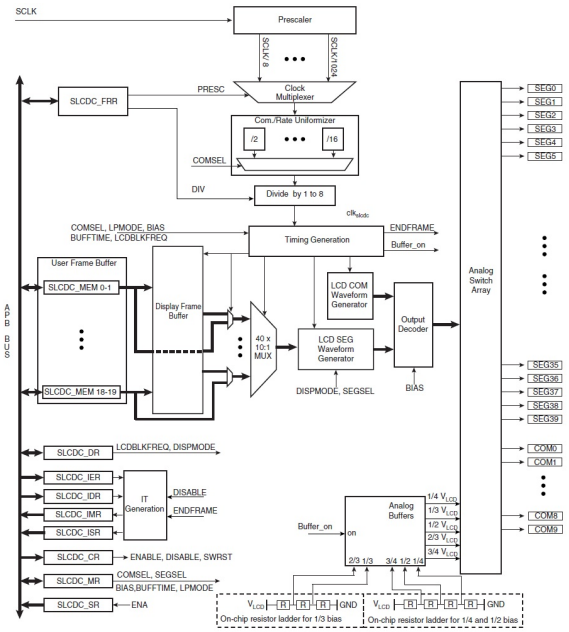
4.1 The Processor

The HP 20b is essentially a keyboard and liquid crystal display (LCD) connected to an Atmel AT91SAM7L128 processor, which will hereafter be referred to as SAM7L. The SAM7L is part of Atmel's AT91SAM series of chips, which are all built around an ARM processor core ("AT" is for Atmel; "SAM" refers to "smart ARM core;" 91 seemsto be arbitrary). The 7L series of microcontrollers are designed for low power (hence the L), and the final 128 indicates that it includes 128K of flash program memory.

Figure 1 shows a block diagram of the SAM7L chip. It is basically a single standard processor surrounded by memory and various peripherals such as a system (clock) controller and an LCD controller, but most of which were not used in this project. The system controller can control the clock and power supply of each peripheral through software. This makes it possible not only to save energy by not powering on unnecessary peripherals, but also to make a peripheral appear to not work if the user forgets to turn on its power.

4.2 The LCD Display

The LCD controller generates the complex AC waveforms necessary to drive the calculator's elaborate LCD display. To software, the LCD appears as a series of memory locations whose bits control individual LCD segments. The following figure shows a block diagram of the LCD Macrocell [5].



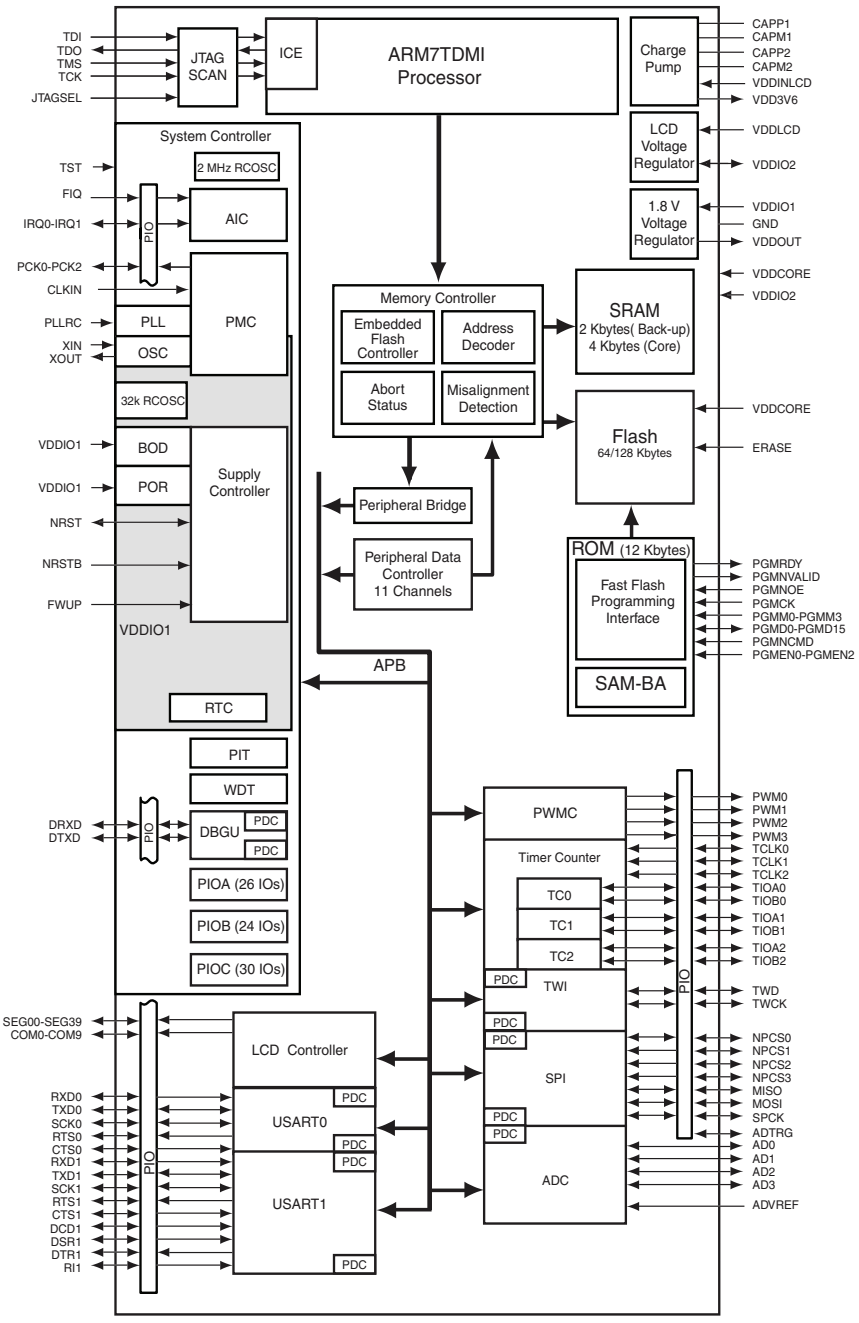


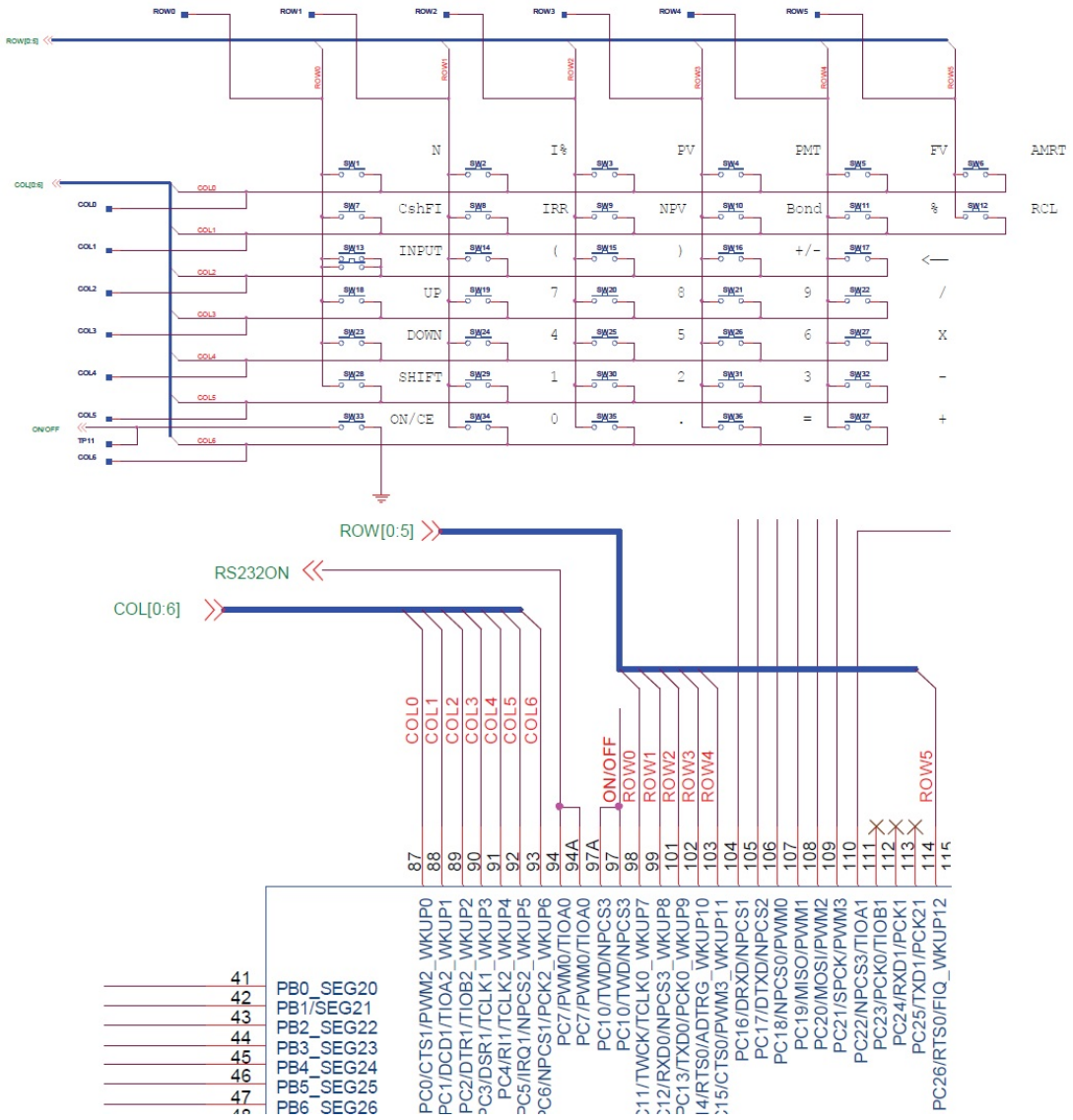
Figure 1: A block diagram of the AT91SAM7L microcontroller that is at the core of the HP 20b

Below are some library functions concerning the LCD display.

lcd_init	initializes the LCD display (turns on its power supply and set various counters)
lcd_put_char7	prints an ASCII character in the given column on the 7-segment display
lcd_print7	prints a string on the 7-segment display starting from the leftmost column
lcd_print_int_neg	prints right-justified integer with an optional leading negative sign (which depends on whether a given condition is true/false)
lcd_print_int	prints an right-justified signed integer

4.3 The Keyboard

Earlier in the term, we tore apart some keyboards in order to figure out how keyboards work. We found that there were three layers of thin plastic sheets underneath the keys of the keyboard. The outer two sheets had printed matrix-like grids of circuits on them, while the center sheet was blank with cut-out holes at each circuit point. This discovery provided crucial insight into how keyboards work. We realized that the center sheet separated the circuits, keeping them from connecting; however, whenever a key pressed down, it would complete the circuits, thus giving the ability for an electrical signal to be sent to a processor.



The HP 20b's keyboard is similar to the old keyboards that we tore apart. The figure above shows the schematic for the HP 20b's keyboard, which is a standard matrix-type, consisting of row and column wires that can be shorted together by the keys. The top is the keyboard matrix itself. The bottom shows how the matrix is connected pins on the SAM7L chip that can be driven by a parallel I/O controller: a peripheral that enables software to control and read the state of each pin [4].

Two important pieces of information to note about the HP 20b's keyboard is that it counter-intuitively reads rows vertically and columns horizontally, and the ON/CE key is separate from the matrix. The below figure is a more concise table that lists the pair of pins–one for its row, the other for its column–that each key shorts when pressed.

		"rows"					
		PC11	**PC12**	**PC13**	**PC14**	**PC15**	**PC26**
"columns"	**PC0**	N	I/YR	PV	PMT	FG	Amort
	PC1	CshFl	IRR	NPV	Bond	%	RCL
	PC2	INPUT	(	)	+/–	←	
	PC3	▲	7	8	9	÷	
	PC4	▼	4	5	6	×	
	PC5	shift	1	2	3	–	
	PC6		0	.	=	+	

5 Software Architecture

Our system required two pieces of software: Ubuntu Linux and OpenOCD. OpenOCD is a software package that communicates to the SAM7L CPU through USB of the computer and JTAG of the calculator. The "OCD" in its name stands for "on-chip debugger." The lab's Ubuntu Linux workstations provided the C compiler, assembler and linker, necessary for packaging our code [1, 2].

6 Software Details

In this project, we programmed our calculator to accomplish four different tasks. Each lab built upon the previous one and became progressively more difficult. The goal of the first lab was to display a message across the LCD, while in our second lab, we displayed which key was being pressed when the user pressed a button on the calculator. The objective of our third lab, was to let the user see which numbers and operation key have been pressed so far, while lab four builds upon this functionality to create an Reverse Polish notation (RPN) calculator.

6.1 Lab 1: A Scrolling Display

Let us begin by explaining the details of Lab 1, shown in Figure 2 and Figure 3. The objective of this lab was to have a message scroll across the LCD screen. Since this was the first exposure to programming in C for any of the members in our group, we were unsure of how to approach this lab. We contemplated the use of pointers to run through a character array and append a character at a time to

the current string being displayed, but none of us were comfortable enough with pointers to write code this way. Instead, we used arrays to display the string of letters. The character array name1 stores the string we want to display, and the character array name2 is the array that is going to be displayed on the screen. We use two nested while loops to iterate once through name1 and once through name2. We constantly modify name2 so that it displays 15 letters of the display string at a time. Each time name2 is displayed, it has been shifted over by one so that the overall effect of the display is that the string to display is scrolling across the screen. Once name2 reaches the end of the string, it just displays a blank space, and this code is written in line 22. Of course, the calculator is operating at a very fast speed so if this code is run without any delay, the message will scroll across too quickly, so in 29 and 30 we have made the calculator count from 0 to 50000 without doing anything. The constant BLANKSPACES represents the number of blank spaces to output before wrapping over and reprinting the message.

The main problem with this code is that it works only for the specified string that has been inputted. In order to make this program more versatile, we should have made a function that would return the string length of the input string and also by using pointers instead of arrays, we could have made the code slightly more condensed.

6.2 Lab 2: Scanning the Keyboard

Now let us move on to Lab 2, shown in Figure 4 and Figure 5. The objective of this lab was to notify the user when no key was being pressed, and if a key was being pressed, then the calculator should notify the user which key was being pressed. We approached this problem by creating a function keyboard_key which would return a decimal value where the first digit corresponded to the column number and the second digit corresponded to the row number. This decimal value would then be displayed in main, and if no key was being pressed, then the message "No key" would be displayed. For this program, we implemented the function keyboard_init to set all columns initially to high. Then, we used the fact that if a key was pressed, its row and column would be low, so we iterated through each of the columns and set them to row. Then we iterate through each of the columns and check if any of the rows in that column are set to low. If it set to low, we execute lines 12 and 13. Line 13 returns the two digit number with row and column information, and if none of the keys are being pressed in that specific column, the column is set back to high and the function loops through the next column. If none of the keys are being pressed, then the function returns 1 in line 18. In main, we obtain the integer returned by keyboard_key and display if it is

```
1  //Ankita Gore, Shikhar Kumar, Christina Huang
2  #include "AT91SAM7L128.h"
3  #include "lcd.h"
4
5  #define DELAY 50000  //Delay factor
6  #define STRLEN 19  //Length of display string
7  #define COLS 15 //Number of spaces allotted to display on the
     //LCD
8  #define BLANKSPACES 2 //Number of blank spaces to display
     //before wrapping around
9
10  int main()
11  {
12    *AT91C_WDTC_WDMR = AT91C_WDTC_WDDIS; //Turn off watchdog
        //timer
13     lcd_init();
14
15     char name1[] = "HELLO WORLD ABC123";  //Display string
16     char name2[COLS]; //String which will be displayed on LCD17
17     int i, j, p;
18     for(;;) {
19       for (j=0;j<STRLEN;j++) { //Iterate through display string
20          for (i=0;i<COLS;i++) { //Iterate through LCD display
21          if (i+j>COLS+BLANKSPACES) { //LCD string has reached
              //end of display string
22                    name2[i]=" ";
23          }
24          else {  //LCD string still within display string
25                    name2[i]=name1[i+j];
26          }
27          }
28      lcd_print7(name2);
29       for (p=0;p<DELAY;p++) {  //Delay
30       }
31      }
```

Figure 2: Part 1 of our solution for lab 1: the scrolling message

```
32      }
33      return 0;
34  }
```

Figure 3: Part 2 of our solution for lab 1: the scrolling message

not -1. If it is -1, that means a key is not being pressed, so we display the "No key" message.

One difficulty we faced when programming this lab was determining how to tell user which key was being pressed. In our case, we returned decimal digit with row and column number, but in retrospect we should have created a two-by-two array with all the different character buttons so that if a key was pressed, the character corresponding to the button instead of the row and column number would be returned. Another difficulty we faced was that we did not include line 16 initially, so the code would not work properly because we did not reset the current column back to high.

6.3 Lab 3: Entering and Displaying Numbers

Now let us move on to Lab 3, shown in Figure 6 and Figure 7. The objective of this lab was to let the user input a bunch of numbers on the calculator, and when the user presses either the input key or an operation key, the string of numbers which have been pressed so far will be displayed. We approached this problem by defining a list of variables in the function keyboard_get_entry, which accepts a struct with variables operation and number as a parameter. Then, the method enters an infinite loop which breaks at line 41. Each time through the loop, keyboard_key is called to determine which key is being pressed. The variable b stores the previous value of a to see ifthe key is being pressed down or not. If b stores the same value as a, that means the key has been held down because the value ofa would be reset to -1 if the user's finger had let go. If a number has been pressed in line 12, then that number is displayed and the value of result is modified. If the negative key has been pressed in line 20, then the sign value switches and a negative sign will be displayed or removed depending on whether or not the negative key has been pressed before. In line 30, if any of the operation keys or the input key has been pressed, then the operation will be displayed, the value of result and operation will be modified, and the function will break out of the infinity for loop.

One difficulty we faced when programming this lab was trying to make keyboard_key be able to display numbers as they were being pressed and still be able

```
1   //Ankita Gore, Shikhar Kumar, Christina Huang
2   #define NUMCOLS 7 //Number of columns in calculator
      //configuration
3   #define NUMROWS 6 //Number of rows in calculator
      //configuration
4
5    int keyboard_key() {
6       int r;
7       int c;
8       for (c = 0; c < NUMCOLS; c++) { //iterate through columns
9          keyboard_column_low(c);  // change current column to low
10            for (r = 0 ; r < NUMROWS ; r++) { //iterate through
                //rows in current column
11            if (!keyboard_row_read(r)) { //current row also reads
                //low
12            keyboard_column_high(c);
13            return (c*10)+r;  //return array position of key
14            }
15            }
16            keyboard_column_high(c); //reset column back to high
17      }
18      return -1;
19  }
20
21  int main() {
22
23      lcd_init();
24      keyboard_init();
25      for (;;) {
26            int x = keyboard_key();
27
28            if (x==-1) {
29            lcd_print7("No key");
30            }
31            else {
32            lcd_print7("KEY ");
```

Figure 4: Part 1 of our solution for lab 2: scanning the keyboard

```
33              lcd_put_char7('0'+(x/10), 4); //place 1st digit of
                  //keyboard_key in 4th spot
34              lcd_put_char7('0'+(x%10), 5); //place 2nd digit of
                  //keyboard_key in 5th spot
35              }
36      }
37      return 0;
38  }
```

Figure 5: Part 2 of our solution for lab 2: scanning the keyboard

to return the correct result and operation. The way the code is written currently, the numbers are being displayed as they are pressed, and the resulting number and operations are being stored, but nothing is being done with result.number and result.operation. They are not being returned by the function so their values are lost once the function is done running. In retrospect, we should have returned the result and operation so that the function would be more versatile and these values could be used in the main method. Also, the way this program is written, the main method does not do anything while all the functionality of the calculator occurs in keyboard.c, which is not very good programming practice.

6.4 Lab 4: An RPN Calculator

Finally, we would like to talk about Lab 4, shown in Figure 8 and Figure 9. For this lab, we were supposed to design a Reverse Polish calculator (RPN calculator). This kind of calculator allows you to keep on adding new numbers to a data structure called a stack, and when any of the operation keys were entered, the two most recent entries in the stack will be combined together by the operation. Line 16 puts a zero in the first position of the calculator to tell the user that the program is running. Then, keyboard_get_entry is run to update the struct entry. If the struct has a value not equal to int_max, then we add that value to the stack in lines 22 to 25. In lines 27 to 42, an operation key has been pressed so we apply that operation to the previous two values in the stack and update the value of numOfTermsInStack. Then, for each operation,we update the number of terms in the stack. In line 44, we test for the error case where an input greater than INT_MAX or less than -1*INT_MAXis entered, and in line 49 we print the value of the integer if the integer is a valid input.

For this code, the divide function does not work properly because the C pro-

```
1    // Ankita Gore, Shikhar Kumar, Christina Huang
2    void keyboard_get_entry(struct entry *result) {
3       result->number = 0;  // entry.number
4       int a = -1;  // nothing is being pressed
5       int numOfKeysPressed = 0;
6       int sign = 1; // sign is positive
7       int position = 1;
8
9       for (;;) {
10         int b = a; // b stores previous value of a
11         a = keyboard_key();
12         if (a>='0' && a<='9' && b==1) { // a is an integer and
             //a has not been held down
13         lcd_put_char7(a, position); // displays key pressed in
             //appropriate position
14         position++;
15         numOfKeysPressed++;
16         int num_to_add = a- '0'; // converts key pressed to
             //int and adds to entry number
17         result -> number = (result -> number)*10 + num_to_add;
18         }
19
20         if (a=='~' && b==1) { // negative button has been
             //pressed
21         if (sign==1) {
22         lcd_put_char7('-', 0);  // put negative sign
23         }
24         else {
25         lcd_put_char7('␣', 0); // get rid of negative sign
26         }
27           sign = sign*-1;  // switch sign of variable 'sign'
28     }
29
30     if (a=='/' || a=='*' || a=='+' || a=='-' || a=='\r') { //
         //operation/input key pressed
31            if(numOfKeysPressed == 0) {
```

Figure 6: Part 1 of our solution for lab 3: entering and displaying numbers

```
32            result-> number = INT_MAX; // return INT_MAX if no
                //key pressed
33     }
34     if (a=='/' || a=='*' || a=='+' || a=='-') {
35        lcd_put_char7(a, position);
36     }
37     result->operation = a;
38     if (sign==-1) {
39            result->number*=-1;  // Switch sign of result
40     }
41     return; // if operation/input pressed, break out of infinite
         //for-loop
42   }
43  }
44 }
```

Figure 7: Part 2 of our solution for lab 3: entering and displaying numbers

gramming code that has been inputted into the code does not allow for division by variables. Also, another problem we faced was that when a value greater than int_max was entered, the number would wrap around to a very negative number which was less than int_max, so if we tried to add or multiply by a very big number, the result would become negative. Also, because we ran out of time, we did not account for the fact that the user might try to input more terms into the stack than the stack size, which would lead to errors.

7 Lessons Learned

We often wasted a lot of time thinking that our program was buggy, when it wasn't. We advise future students to make sure that the power connector is plugged into the calculator, that the 20-pin ribbon connector cable is connected to the 16-pin JTAG header connector with the red wires facing the same direction, that the USB cable is plugged into the computer, and that the calculator is turned on. We also advise for future students to bring a pencil and paper to scribble their thought processes on. Furthermore, we advise that they take their time in order to fully comprehend what their codes do.

We gained a deeper understanding of C programming and programming in general, due to both the code review and different labs.

```
1   //Ankita Gore, Shikhar Kumar, Christina Huang
2
3   #include "AT91SAM7L128.h"
4   #include "lcd.h"
5   #include "keyboard.h"
6
7   int main() {
8
9     struct entry entry;
10   // Disable the watchdog timer
11   *AT91C_WDTC_WDMR = AT91C_WDTC_WDDIS;
12
13    lcd_init();
14    keyboard_init();
15    int stack[10];            //Declare stack
16    int numOfTermsInStack = 0; // Keeps track of how many terms
        //in track
17    lcd_put_char7('0',11); //Start off by displaying 0
18
19    for(;;) {
20        keyboard_get_entry(&entry);
21
22        if(entry.number != INT_MAX) { //Add value to stack if
            //it is valid entry number
23        numOfTermsInStack++;
24        stack[numOfTermsInStack] = entry.number;
25        }
26
27        if(entry.operation == '+' && numOfTermsInStack > 1) {
            //add two top values of stack if more than two values
            //exist in stack
28        stack[numOfTermsInStack - 1] = stack[numOfTermsInStack - 1]
            +stack[numOfTermsInStack];
29        numOfTermsInStack--; //number of terms in stack decreases
            //by one
30       }
```

Figure 8: Part 1 of our solution for lab 4: the RPN calculator

```
31        else if (entry.operation == '-' && numOfTermsInStack > 1) {
            //add two top values of stack if more than two values exist
            // in stack
32        stack[numOfTermsInStack - 1] = stack[numOfTermsInStack - 1]
           -stack[numOfTermsInStack];
33        numOfTermsInStack--;
34        }
35        else if (entry.operation == '*' && numOfTermsInStack > 1) {
            //multiply two top values in stack
36        stack[numOfTermsInStack - 1] = stack[numOfTermsInStack - 1]
            *stack[numOfTermsInStack];
37        numOfTermsInStack--;
38        }
39        else if (entry.operation == '/' && numOfTermsInStack > 1) {
            //divide two top values in stack
40        stack[numOfTermsInStack - 1] = stack[numOfTermsInStack - 1]
            /stack[numOfTermsInStack];
41        numOfTermsInStack--;
42        }
43
44        if(stack[numOfTermsInStack] > INT_MAX
45              || stack[numOfTermsInStack] < INT_MAX*-1) {
                 //print error if input value too big
46        lcd_print7("ERROR!                    ");
47        numOfTermsInStack = 0;
48        }
49        else {  //print value in stack
50        lcd_print_int(stack[numOfTermsInStack]);
51        }
52
53     }
54     return 0;
55  }
```

Figure 9: Part 2 of our solution for lab 4: the RPN calculator

8 Criticism of the Course

The lab rooms were much too small for a group as big as ours. There often were not enough chairs, and when there were, the space was very cramped. Also, there would often be graduate or upperclassman students that we would have to kick out of our stations before we could get to work.

The individual labs were just the right difficulty, especially with help and guidance from Professor Edwards and Yoonji. Since the labs basically built upon each other, we had a pretty clear idea about how the pieces would fit together as we progressed through each lab. The code reviews were generally helpful. They allowed us to compare how differently (sometimes vastly) other groups went about in order to code the same thing. What helped the most was that the code reviews showed us what each group, including our own, did right, wrong, or inefficiently.

A more in-depth "crash course" to C would have helped, though we did not have enough time to fully cover it at the beginning of class. None of the members in our group had any existing knowledge about C programming. In particular, for the one member of the group who had never done any programming, it was very discouraging at the beginning to have no idea what was going on.

References

[1] Lab 1. Online http://www.cs.columbia.edu/~sedwards/classes/2011/gateway-fall/hello.pdf.

[2] Hp-20b repurposing project. Online http://www.wiki4hp.com/doku.php?id=20b:repurposing_project.

[3] Hp-20b. Online http://en.wikipedia.org/wiki/HP_20b.

[4] Lab 2. Online http://www.cs.columbia.edu/~sedwards/classes/2011/gateway-fall/keyboard.pdf.

[5] Hp-20b business consultant financial calculator manual. Online http://h10010.www1.hp.com/wwpc/pscmisc/vac/us/product_pdfs/HP_20b_Online_Manual.pdf.