ENGI E1112 Departmental Project Report: Computer Science/Computer Engineering

Kenneth A. Hungria - kah2204
Albert Q. Cui - aqc2109
Dan R. Schlosser - drs2161

December, 2012

Abstract

This paper documents our work on restoring calculator functionality to the HP 20B. We implemented an RPN calculator, coding in C, in order to learn more about developing on embedded systems, to improve on our collaborative skills, and to get a glimpse of life as a programmer/ computer engineer. Our implementation was done in four parts. First, we developed functions to display integers on the calculator's LCD screen. Second, we enabled keyboard input. Part three brought the two parts together, displaying input while having the calculator understand which buttons were numbers and which were operations. Finally, part 4 implemented calculator functionality. Along the way, we learned how an RPN calculator works, how a keyboard takes input, and much more. Ultimately, we were successful in our implementation, albeit efficiency could be improved upon. However, as the goals were met, and we felt the project managed to peak our interests in the field, we deem the lab successful. Ideally, we will learn more about embedded systems in the future so that we can better appreciate the work we did in this project.

1 Introduction

In this project, we attempted to restore basic calculator operations to the HP 20B calculator. The first section describes how to use our stack-based calculator implementation. The next section describes the hardware platform we developed on. Sections 4 and 5 go into detail on the software implementation, including

justification and rationale for how we implemented the solution. Also included are possible ways we could have improved on our solution. Section 6 talks about what we learned from this project, and finally section 7 contains our criticisms of the class.

2 User Guide

This calculator implementation supports addition, subtraction, and multiplication of integer values, using the stack-based RPN standard. Integers are entered using the keys numbered 0-9, and the sign may be toggled at any time using the `[+/-]` key. Numbers can next be added to the stack using the `[INPUT]` key. For example, 25 and -128 may be added to the stack in the following manner:

```
[2], [5], [INPUT], [1], [+/-], [2], [8], [INPUT]
```

Operations pressed are applied to the top two elements on the stack, and the resultant is placed back on top of the stack. For convenience, pressing an operand also adds the number currently being entered to the stack. For example, -5(13-4) can be calculated in the following manner:

```
[1], [3], [INPUT], [4], [-], [+/-], [5], [*]
```

Additionally, operands may be pressed without a number in between, and will act on the top of the stack as expected. 3-(4*5) can be calculated in the following manner:

```
[3], [INPUT], [4], [INPUT], 5, [*], [-]
```

Also, the `[<-]` key may be used to erase one digit of the current entry. For example, 1435 may be placed on the stack in the following manner:

```
[8], [<-], ..., [<-], [1], [7], [<-], [4], [3], [5], [INPUT]
```

At all times, the screen displays either the current number being entered, or the number on top of the stack.

3 The Platform

This project was developed on the HP 20B Business Consultant Financial Calculator hardware platform.

3.1 The Processor

The HP 20B contains an ATMEL 30MHz, low-power AT91SAM7L128 ARM7 chip [1]. The chip has 128K of flash memory. A system controller allows software to control the processor's use of the many peripherals connectable to the chip. For the purposes of this project, three peripherals were of interest: the LCD screen, the keyboard, and the JTAG connector.

3.2 The LCD Display

The calculator has a 2 line alphanumeric LDC screen with adjustable contrast. The first line is an 8 character scrolling display with 11 indicators (BEG, 360, RAD, etc.). The second line consists of a 12-digit display with 3 smaller digits for exponent indication. The second line also has indicators for negative numbers and exponents.

Provided to us was a library of functions pertaining to the LCD display. Two of the important functions are as follows:

- LCD_INIT: Enables the LCD peripheral
- LCD_PUT_CHAR7: "Display an ASCII character in the specified column on the 7-segment display" (This refers to the main part of the second line of the LCD display, allowing 12 possible digits to be displayed.)
- LCD_PRINT7: "Display a string on the 7-segment display starting from the leftmost column"

3.3 The Keyboard

The HP 20B keyboard consists of 37 keys. 36 of these keys are connected in a matrix of rows and columns of wires while the On/Off button is wired separately.

During the project, we took apart several keyboards to better understand how a keyboard works. In a typical keyboard, under the plastic keyboard are three sheets of plastic. The bottom and top sheets have wires that form a circuit. The middle sheet has holes where each key is positioned. The middle sheet with the holes prevents the two layers from touching when no key is pressed. When a button is pressed, the top sheet is pressed into the bottom sheet, shorting the circuit and changing the voltage of the circuit.

On the software side, to see which key is pressed, we set all the column voltages to high while setting one to low. When a button is pressed in the column set to low, the row that that button belongs to will read high because of the completed circuit. This is discussed further in section 5.

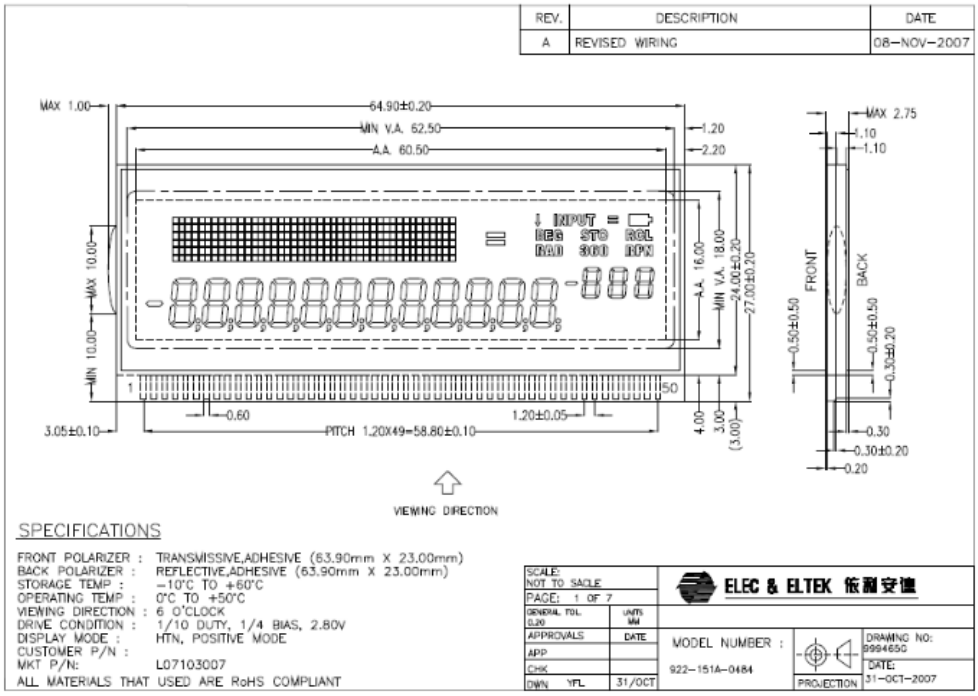

Figure 1: The lcd schematic [2]

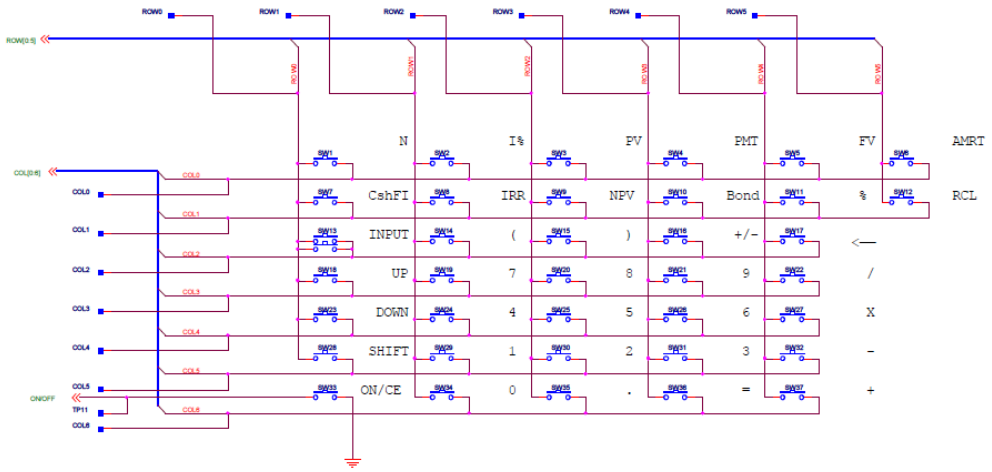

Figure 2: The keyboard schematic [2]

- KEYBOARD_INIT: Enables the keyboard peripheral
- KEYBOARD_COLUMN_HIGH, KEYBOARD_COLUMN_LOW: Sets the specified column's voltage high or low
- LCD_PRINT7: Reads if the voltage of the row is high or low

4 Software Architecture

The calculator implementation was done in four parts. The first was getting the calculator to display a given integer number, including negatives. The second part consisted of reading input from the calculator's keyboard and displaying that input in a meaningful way. The third part worked on getting the calculator to take an input and operation pair, along with enabling the backspace key. This was the first step to implementing the stack based calculator. Finally, the last part implemented operations and actual calculations.

5 Software Details

5.1 Lab 1: Getting Started: Hello World

The first, displayInt, takes a parameter num, an integer, adding a negative sign if necessary. The first part of the solution checks if the number is negative. If it is,

```
#include "AT91SAM7L128.h"
#include "lcd.h"

void displayInt(int num);
void clearScreen();

void displayInt(int num) {
    clearScreen();

    int index = 11, isNeg = 0;
    if (num < 0)
        isNeg = 1;
        num *= -1

    if(num == 0)  //special case, avoid infinite loop
        lcd_put_char7('0', index);

    //Displays characters from right to left
    while (num > 0) {
        char c = (char)(num \% 10 + '0'); // integer -> char
        lcd_put_char7(c, index); //displays the char
        index--;
        num /= 10;
    }

    if(isNeg == 1)
        lcd_put_char7('-', index);  //show the '-'
}

void clearScreen(){ //clears the screen
    int x;
    for(x = 0;x<12;x++){
        lcd_put_char7('␣',x);
    }
}
```

Figure 3: Our solution to lab 1

we store that to make sure we add a negative sign on the LCD screen. We then make the number positive.

Our design choice was to make the number display from the right side of the calculator. In the heart of our solution, we have a while loop for when num is greater than zero. Inside the while loop, we essentially split the number into digits by taking off a digit one at a time with each iteration of the while loop. This is done by using mod 10 to the number. We then convert the integer to a char by adding 48 to get the correct ASCII value. We need to convert to a char because we then call lcd_put_char7 to actually print the digit, starting at index = 11, the right most column of the display. Index is decremented and the number is divided by 10 all to prepare for the next digit. When all the digits of the number have been displayed, if the digit was negative, we add a negative sign in front of the last index of the digit, again by calling lcd_put_char7.

We have a special case for if the number is zero. The while loop had to be for when num > 0 because 0 is a digit and at the point where num < 10 and we divide by 10, the program would keep adding zeros to the display. So for the special case where the number is zero, we just print zero. Note, this means that numbers with leading zeros won't get those displayed. clearScreen clears the screen for the next call to displayInt. We found this was necessarily because when a displayInt is called with a number already on the screen that's larger than the new number, the extra digits remain on the screen. Therefore, the first part of displayInt calls clearScreen. clearScreen prints spaces for all the columns, giving a cleared screen.

5.2 Lab 2: Scanning the Keyboard

In this part of the lab, we were assigned to have the calculator respond to a button press by creating a function called keyboard_key(). We were given certain other functions to use such as extern void keyboard_column_high(int column), extern void keyboard_column_low(int column), extern void keyboard_init(void) and extern int keyboard_row_read(int row). The keyboard_key() function works by first setting a column to low and the rest to high. Then it checks each row if it is high or low. The reason why the program reads if(!keyboard_row_read(row)) is because we match the low column with the low row instead of high column and high row. Since we made the rest high instead of the column pressed, we don't want the row that is high, rather the row that is low. We then ask it to return a char from an array of chars that we created. We use the col and row numbers to get the specific char. This function returns that char back to the main method as keyVal. The main method clears the screen of any previous input and prints the char in the first index of the lcd using the lcd_put_char7(keyVal,0). We assigned a specific

```
for (;;) {
      char keyVal = keyboard_key();
      if(keyVal != 0) {
          lcd_clear();
          lcd_put_char7(keyVal,0);
      }

const char arr[NUM_COLUMNS][NUM_ROWS] = { // Char for each button
    {'N', 'I', 'P', 'M', 'F', 'A'},
    {'C', 'R', 'V', 'B', '\%', 'L'},
    {'U', '(', ')', '_', '<', 0},
    {'^', '7', '8', '9', '/', 0},
    {'~', '4', '5', '6', '*', 0},
    {'S', '1', '2', '3', '-', 0},
    {0, '0', '.', '=', '+', 0}};

char keyboard_key() {
    int col;
    int row;
    int notCol = 0;

    for(col = 0; col <= 6; col++) {  // For each column
        keyboard_column_low(col);  //set column low

        for(notCol = 0; notCol <=6; notCol++) {
            if(notCol != col)
                keyboard_column_high(notCol); //set all other columns high
        }
        for(row=0;row<=5;row++) { // A row high => button pressed
            if(!keyboard_row_read(row)) {
                return arr[col][row]; // Returns the assigned char
            }
        }
    }
}
```

Figure 4: Selections from our solution for lab 2

char to each button on the calculator so that we may make sure that our program worked correctly. For example, if we pressed 7, the calculator should print 7, and if we pressed the up arrow, the calculator would attempt to print out the ^ symbol. The reason the main method would know to go into the keyboard_key() function is when a button is pressed, keyVal no longer is the char '0'. The only button this function doesn't work for is the on button because we decided to give it a '0' char character which does not allow it to be printed. In one run of the program, we were able to get the results we wanted from each individual button, so the program doesn't seem to crash when multiple inputs are given.

5.3 Lab 3: Entering and Displaying Numbers

Our implementation of number entry involves building an integer representing the absolute value of the integer being entered (numPressed) and an indicator of sign (isNegative). As numbers are entered, numPressed is increased. isNegative may be toggled as desired, and a single dash is displayed on screen if the `[+/-]` key is pressed before any other digits. The screen is updated to show the number being entered as it is built. While keyboard_get_entry does not return any values, it edits entry, a data structure containing two fields, number and operation. Number holds INT_MAX (the largest possible integer) by default, representing the lack of a number, or the signed integer value of the number entered. Operation contains a string representing the operational button pressed by the user. Each button pressed is read into KeyVal as a character represented by keyboard_keys, where the physical placement of the key corresponds with the location in the two dimensional array:

keyboard_get_entry() should be called to retrieve an operation and optionally a number value from the user through the keyboard. A number value equal to INT_MAX should be interpreted as no number at all. This function allows for expanded operational functionality as well - It gives any buttons besides integers, `[+/-]`, and `[<-]` as an operation, allowing for expansion to currently unsupported functionality like exponents without further changes to this function.

We also implemented a method to ensure that each button press was only recorded once. To do this, we used an indicator variable released to ensure that the key press was only considered if the key had been previously read as not pressed.

5.4 Lab 4: An RPN Calculator

This lab involved implementation of a stack, in order to simulate an RPN calculator. To do this in C, we used an array implementation. We created an array of size four (the maximum number of simultaneous elements supported), and an integer

```
void keyboard_get_entry(struct entry *result) {
    int isNegative = 0;         // boolean negative indicator
    int numPressed = INT_MAX;   // |number pressed|, INT_MAX~null
    int released = 1;           // 1 if no button is being pressed
    int keyVal=-1;
    while (keyVal!='+' && keyVal!='-' &&keyVal!='/'
           && keyVal!='*' && keyVal!='\r') { // Not an operation
        keyVal = keyboard_key();  // Read from keyboard
        if (keyVal != -1 && released == 1) { // count each press once
            if(keyVal >= '0' && keyVal<='9' && (numPressed==INT_MAX ||
               numPressed+1<INT_MAX/10)){ // valid number pressed
                numPressed = (numPressed==INT_MAX)? keyVal-'0' :
                                numPressed*10+keyVal-'0'; // signed
                lcd_print_int(numPressed*((isNegative)?-1:1)); // signed
            } else if(keyVal=='~') { // The user pressed the +/- key.
                 isNegative = (isNegative)? 0:1; // flip negative
                lcd_clear();
                if(numPressed==INT_MAX)
                    lcd_put_char7('-',LCD_NUM_REAL_COLUMNS-1);
                else { lcd_print_int(numPressed*((isNegative)?-1:1)); }
            } else if(keyVal=='\b') { // The user pressed the backspace key.
                if (numPressed == INT_MAX && isNegative) {
                    isNegative = (isNegative)? 0:1; // flip negative
                    lcd_clear();
                } else if (numPressed < 10) {
                    lcd_clear();
                    if(isNegative){lcd_put_char7('-',LCD_NUM_REAL_COLUMNS-1);}
                    numPressed = INT_MAX; // Reset numPressed
                } else if (numPressed != INT_MAX) {
                    numPressed = numPressed / 10; // remove ones digit
                    lcd_print_int(numPressed*((isNegative)?-1:1)); // signed
            }} released = 0; //The user has begun holding down a key.
        } else if (keyVal == -1) { released = 1;}
    } result->operation=keyVal;
    result->number=numPressed*((isNegative)?-1:1);
}
```

Figure 5: Selections from our solution for lab 3

top representing the current location of the top of the stack. The standard pop, push, and peek methods were implemented.

When the entry structure contains a user-entered number (not INT_MAX), it is pushed to the stack. Then, operations are performed, if there are sufficient elements in the stack. Addition pushes the sum of two pops, and multiplication the product of two pops. Subtraction pushes the difference between two pops, where the integer on the top of the stack is subtracted from the integer directly beneath it. A peek (the result of the operation), is then displayed on screen. Our function currently does not safeguard against operations that would either exceed the maximum integer allowable or displayable. This is functionality that we would have liked to implement given additional time.

This function calls keyboard_get_entry() to get the user-inputed operation and number, and uses lcd_print_int() to display the top element of the stack after calculation.

6 Lessons Learned

From this lab, there was a large variety of people in this class, from those with little experience programing to those that have taken many coding classes. At first glance, the amount of coding that was given to us initially could have been a bit intimidating especially if they haven't done enough programing in C. Later through the lab though, it became clear that we only had to know what we wanted to add on rather than code a large amount than what we saw in the beginning. The lessons learned isn't just the coding aspect, but also the teamwork aspect. A large part of this lab is being able to communicate your ideas with your partners and that is a very important skill. It felt more like a job experience rather than a classroom one. It also helped in balancing the workload so that even the least skilled person there would be able to do some work and understand how the code works. This was a group project that also required some learning in C coding. It started with simple application of given functions and if statements, to the point where we had to use 2D arrays and stacks to complete the labs. These are tricks that are C specific and someone who has never coded before would have problems knowing about this until they were told. So it did expand the vocabulary and the C methods at our disposal.

For future students, the advice I would recommend they have is to definitely have a well rounded group, where there is some experience in coding within the group. The lab would go much smoother and more time would be used trying to figure out how to code the solution rather than trying to figure out what to do next. One thing I wish I were told is how little coding is really required and it is more

```
int mysize;
int stack[4];
int top;
int main() {
    top = -1;    // Index of the top of the stack. -1 => empty stack.
    mysize = 4;  // The stack itself.
    struct entry entry;
    ...
    for(;;){
        keyboard_get_entry(&entry); // Get the number and operation
        if(entry.number != INT_MAX) // Push the number, if there is one.
            push(entry.number);
        if(top >= 1){
            if(entry.operation == '+')
                push(pop()+pop());  // Place the sum on the stack
            else if(entry.operation == '-'){
                int first = pop();
                int second = pop();
                push(second - first); // Place the difference on the stack
            }else if(entry.operation == '*')
                push(pop()*pop());  // Place the product on the stack
        } lcd_print_int(peek());
}}
int push(int element){ // Place element on top of the stack
    if (top == mysize) { return INT_MAX;} //If pushing will blow the stack
    top++;
    stack[top] = element;
    return element;
}
int pop() { // Remove and return the top element of the stack
    if (top == -1) { return INT_MAX;} //If the s///tack is empty
    int element = stack[top];
    top --;
    return element;
}
int peek() { return push(pop()); } //Return the top element of the stack
```

Figure 6: Selections from our solution for lab 4

on trying to adjust the program that is already there. There are instances where labs could have been done in about 20-30 lines of code or less. That would have been useful in eliminating several ways to solve some of the problems we had.

7 Criticism of the Course

The lab as a whole was a good and interactive lab because it gave us a chance to work in a group to simulate a group project that would happen in many jobs. The coding part was interesting to some extent but not overly exciting. Since much of the program was done for us, it didn't really feel as if we were making the calculator work, it was more like we were adding a missing component. It would have been informative to learn how certain aspects of the coding worked like, for example, the function for printing on the lcd of the calculator. We just used the function because it told to us what it did. Each lab went rather smoothly in difficulty where it wasn't hard but was rather frustrating to code because of so much troubleshooting but that has to do with the skill of the group as well. The functions given to us weren't too hard to use although sometimes we would need to ask for help for the function and how to use it. A part of the program that we didn' expect to run into was sort of "malicious" code that was in our program. It probably wasn't intended to be malicious because it could have run properly if used in a specific manner, but there was a good time spent by us trying to find a problem in our code when the problem was a code written that wasn't told to us. It was probably malicious to us because we wrote the code differently than expected but it was rather interesting how we can't always trust what code is given to us, and we should check if it works to our specifications. Finally, the code reviews were helpful only to an extent. It gave everyone an idea on how others solved the problem, but that usually isn't what people would have problems with. They would have problems with the implication of the solution, which they wouldn't know how to handle.

References

[1] Hp-20b business consultant datasheet. Online `http://h20426.www2.hp.com/product/calculators/docs/learning_modules/20b/datasheet_20b.pdf`, May 2008.

[2] Hp-20b business consultant sdk. Online `http://h20000.www2.hp.com/bizsupport/TechSupport/SoftwareIndex.jsp?lang=en&cc=us&prodNameId=3732535&prodTypeId=215348&prodSeriesId=3732534&swLang=13&taskId=135&swEnvOID=54`, October 2009.