ENGI E1112 Departmental Project Report: Computer Science/Computer Engineering

Phillip Godzin, Joseph Thompson, Ashley Kling

December 2012

Abstract

This lab report is on our assignment of reprogramming a HP-20b calculator. The code was written in C programming language on a Mac Pro and uploaded to the device. The operating system was wiped and only had some lower level programming before we started working on it.

The calculator now can turn on, display digits, and implements the basic operations of multiplication, division, subtraction, addition and negation.

The calculator uses Reverse Polish Notation for its computations, which was also utilized on the original calculator software. Reverse Polish Notation is slightly more efficient for computations than our standard system. Our calculator displays numbers justified to the right.

The keyboard of the HP-20b is organized into rows and columns. When a key is pressed, the program stores the associated operation or number. Many of the more complex operations on the calculator were rendered useless for our project. The calculator can hold up to 4 separate numbers in its stack. If overflowed, it will return the max value for int. Underflow will simply cause nothing to be returned. Following is a more in depth look at how we designed the software for the calculator.

1 Introduction

The calculator we reprogrammed is a HP-20b calculator. It is a fairly recent product from the long line of HP business calculators, and like its ancestors uses Reverse Polish Notation. You can find one of these calculators for around 30-40 dollars on Amazon.com.[1] Unfortunately, many of the reviewers were not terribly enthusiastic about the HP-20b, in particular they did not like the build quality of the hardware. A more in depth look at the hardware aspect of the calculator will be in section 3.

The calculator has some of the basic features we can expect from a digital calculator. It can add, subtract, divide, multiply, and negate. Division, however, works the way integer division works in C. The more complicated functions that the calculator had before it was wiped have been neglected. A look at the software side will continue in sections 4 and 5.

Many interesting lessons can be gleaned from this project. An example of this is the idea of coding something so you can understand and utilize it practically later. Section 6 will have more on what we learned. We also have to talk about our assessments at some point, so section 7 will have our criticism of the course.

2 User Guide

The calculator as mentioned before use Reverse Polish Notation. Reverse Polish Notation is named such because it is roughly the reverse of Polish Notation. Polish Notation was invented by the mathematician Jan Lukasiewicz. [2] Polish Notation separates the operations form the operands and places them on the left. Reverse Polish Notation does just the opposite by placing first the operands, then the operations to their right. This system allows for us to eliminate parentheses and the problem of them wrongly grouping operations together if inputted slightly wrong. For instance, the operation 1+2*(3+4) turns to 1,2,3,4+*+ with ',' signifying the input key.

The reprogrammed calculator is capable of displaying up to 12 digit numbers (although a 12 digit number would be over the maximum integer) with a stack size of 4. Here is a nice diagram of how the stack system works:[3]

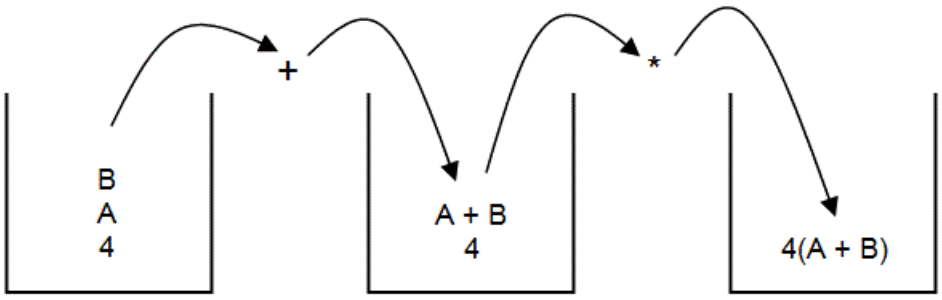

Figure 1: Stack use in RPN

In our calculator, in order to move the pointer for the stack, the input button must be pressed. For instance, to do the operation 3+4, the user must do 3 input 4 +. Note that before the operations the input button does not need to be pressed. The calculator does not support a backspace function, so the user must be careful. Negative and positive integers are supported on this calculator. In order to negate the input, one simply presses the +/- button.

3 The Platform

3.1 The Processor

The calculator use an Atmel AT91SAM7L128. The chip has a 32 bit processor and can operate a 36 MHz. It has 128KB of flash memory. A 40 segment LCD controller is also part of the chip.[4]

The processor has three basic functions: It can add, subtract, multiply, and divide; it can write and move memory; and it can use logic to make decisions and act on them. [5] With these basic tools, it can perform far more advanced calculations and commands using the assembly language. The large amount of data that is flowing through the system are coordinated by the peripheral data controller. There are also controllers for the LCD and the voltage supply. [6]

3.2 The LCD Display

The LCD display uses crystals in a semi liquid state that reacts when a voltage passes through them. The material when activated creates a darker area that light cannot pass through. This is how we create numbers and letters. This LCD uses a seven segment system to display numbers, which means there are 7 lines in the most complicated number (8) and two in the least (1).[**LCD**]

The LCD system has some dedicated functions for its use. We use lcd-init to turn on and clear the screen. Lcd-print7 tells the LCD to print whatever is in the brackets. Lcd-put-char displays a character on the screen.

3.3 The Keyboard

The keyboard has to grids, one of rows and one of columns. They are separated by a non-conductive material so the two aren't accidentally shorted together. With these two we can understand exactly which key is being pressed. When a button is pressed, conductive contacts from the two layers touch and create a short, putting a uniform voltage across both.The on/off button is hard wired into the calculator, while the other button's functions must be coded in.

4 Software Architecture

The goal of this project was to create several individual pieces of code that at the end would come together to form a properly functioning basic RPN calculator.

In the first lab, we design a method to display the numbers entered by the user. We use the lcd-put-char7 method throughout our code to display user input as it is received. In the second lab, we designed a method that determines what key on the calculator has been pressed using the grid-like design of the keypad.

We then stored the numbers and operations entered by the user in the third lab in a struct. This struct was used in the fourth and final lab to

populate a stack of numbers and perform operations on the stack, eventually displaying the proper result.

5 Software Details

Below is the code and explanation for the labs done this semester.

5.1 Lab 1- Displaying

We first began our code to display numbers on the screen by clearing the screen of any preexisting values. To do so, we loop through every index on the display and place an space character.

```
void clearScreen(){
    int i;
    for(i = 0; i < 12; i++){
        lcd_put_char7(' ', i); // places a space (' ') at the index
    }
}
```

Since calculators generally display values right-justified, we decided to do that for our calculator as well. The screen on the HP-20b can display 12 digits, so numbers are displayed from end to front beginning at the 11th index. To do so, we display the remainder of the number when divided by 10, which will return the last digit, divide the number by 10, and repeat until the single digit case is dealt with, at which point the number divided by 10 will be 0.

The numbers are converted into chars before being displayed by having 48 added to them (the ASCII value of the character 0). If the original number is negative, a negative sign is placed at the next index immediately to the left of the first digit in the number.

```
void display(int num)
{
   clearScreen(); //gets rid of any current values on the screen
   int temp = num;
   const int ASCII = 48; // the ASCII value of 0 to be able to use
       numbers as chars
   int i = 0; // used for index
   int remainder = 0; // holds a single digit
   // If the number is 0, print it out and exit
   if (temp == 0){
       lcd_put_char7('0', 11);
       return;
   }
    // Turns a negative number into a positive number
   if (num < 0)
       num = -num;
   while(num!=0)
   {
           remainder = num % 10; // the last digit of num
           lcd_put_char7(remainder + ASCII, 11-i); // places digit in
               rightmost available index
          num = num/10; // Divides number by 10 for the next
              iteration
         i++;
   }
  // If the original number is negative, place a minus sign at the
      index immediately to the left of the first digit
   if (temp < 0)
       lcd_put_char7('-', 11-i);
}
```

5.2 Lab 2: Scanning the Keyboard

Since the keyboard is laid out in a convenient grid-like way, we represented it with a 2D-array of ints. Operations are defined as integer constants.

```
#define X 99 // Nothing important is pressed
#define EQUALS 11
#define INPUT 16
#define NEGATE 19
#define RETURN 20
#define DIVIDE 15
#define MULTIPLY 14
#define SUBTRACT 13
#define PLUS 12
#define NOTHING 98

//2D matrix representing the rows and columns of the keyboard
int const key[7][6] = {
   {X,X,X,X,X,X},
   {X,X,X,X,X,X},
   {INPUT, X, X, NEGATE, RETURN, X},
   {X, 7, 8, 9, DIVIDE, X},
   {X, 4, 5, 6, MULTIPLY, X},
   {X, 1, 2, 3, SUBTRACT, X},
   {X, 0, X, EQUALS, PLUS, X}
   };
```

To determine what is currently being pressed, we first set all of the columns to high, and then the column we want to test to low. We continue by looping through all the rows, and if it is pressed, the button at the intersection of the row and column is being pressed, and it is returned.

```
int keyboard_key()
{
   int c = 0;
   int r = 0;

   for(c; c<7; c++)
   {
       r = 0;
       keyboard_column_low(c);
       for(r; r<6; r++)
       {
           if(!keyboard_row_read(r))
           {
               return key[c][r];
           }
       }
       keyboard_column_high(c);
   }
   return NOTHING; // Nothing pressed
}
```

To test our code, a button had to be held when the program when run, and the pressed number or number associated with the operation was displayed on the screen. If nothing was pressed the number 98 was displayed.

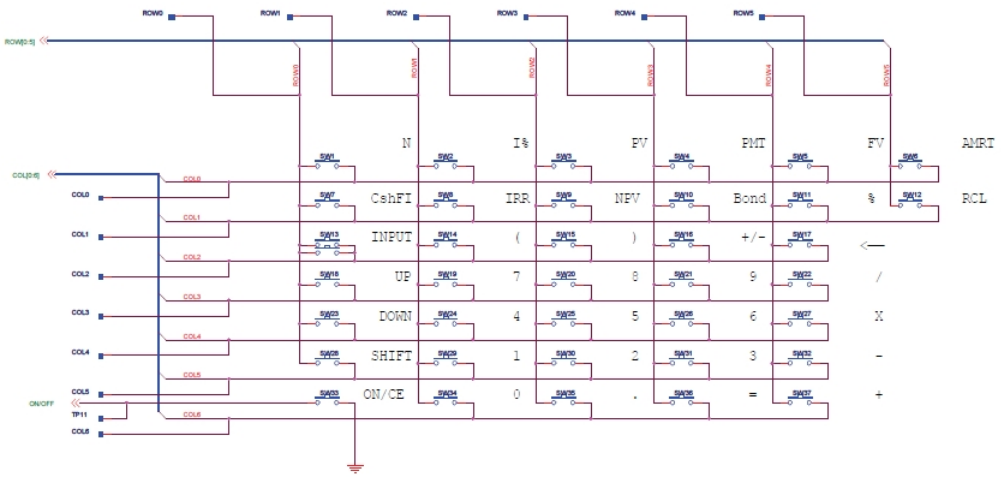

Figure 2: Schematic of the keyboard

5.3 Lab 3: Entering and Displaying Numbers and Operations

For data storage, we used a struct that holds a character for an operation and an integer to hold the number entered by the user. Just a number can be entered and stored (number + INPUT pressed), a number and operation (number and operation pressed without INPUT), or just an operation (operation). If just one is pressed, either the maximum value that can be stored in ints goes into the struct, or the input operation goes into the struct. Upon display, "MAX" will be shown on the screen rather than the actual max int value.

Numbers are displayed onto the screen as the user enters them, but the user is not allowed to enter a number that is greater than the max value for int. The +/- key can be pressed as many times as the user wants, and it will simply toggle the sign of the number.

```
void keyboard_get_entry(struct entry *result)
    {
        int num_pressed = 0;
        int pos = 1; //1 if number is positive, -1 if neg
        result->operation = ' ';
        result->number = 0;
        int keyPressed; //Stores the current key being pressed
        while(((*result).operation == ' '))
        {
            keyPressed = keyboard_key();
            if(keyPressed == NEGATE) //toggle sign of the number
                pos *= -1;
            if(keyPressed >= 0 && keyPressed < 10 && (*result).number
                < INT_MAX / 10) //valid number is being entered
            {
                result->number = (*result).number * 10 + keyPressed;
                num_pressed = 1; // a number has been pressed
            }
            else if (keyPressed >= INPUT && keyPressed <= DIVIDE)
                //operation
            {
                result->operation = keyPressed;
                if(num_pressed == 0) //no number has been pressed
                    result->number = INT_MAX;
                else
                    result->number = (*result).number * pos;
            }
            if((*result).number != INT_MAX && num_pressed==1)
                lcd_print_int((*result).number);
            else if ((*result).number == INT_MAX){
                lcd_put_char7('M',9);
                lcd_put_char7('A',10);
                lcd_put_char7('X',11);
            }

        }
}
```

5.4 Lab 4: An RPN Calculator

paragraphThe previous code we wrote to determine what has been pressed and storing it finally came together in this lab. To make our calculator actually work properly, we keep storing user input into an array that emulates what a stack does. Our calculator will handle a maximum of four operations. Our keyboard-get-entry method allows the user to simply press a number and input, so the array is populated by these entered numbers.

In main.c:

```
int stack[6];
int stack_size = 0;
while(stack_size < 6){
      keyboard_get_entry(&entry);
      if(entry.number != INT_MAX)
      {
            stack[stack_size] = entry.number;
            stack_size++;
      }
      if(entry.operation != INPUT)
            executeOp(entry.operation, stack,
stack_size);
   }
```

Once an operation is pressed, it is executed on the two most recent numbers in the stack and the result replaces the last number. The result of each operation is displayed on the screen.

```
void executeOp(int op, int stack[], int stack_size)
{
   int num1 = stack[stack_size-2];
   int num2 = stack[stack_size-1];

   int result = 0;
   if (op == PLUS)
       result = num1+num2;
   else if (op == SUBTRACT)
       result = num1-num2;
```

```
    else if (op == MULTIPLY)
        result = num1*num2;
    else if (op == DIVIDE)
        result = num1/num2;
    stack_size--;
    stack[stack_size] = result;
    lcd_print_int(result);

}
```

6 Lessons Learned

Probably the most obvious outcome of this lab was that we learned about the C programming language. It had similarities to Java but had more lower level parts to it. Some of the syntax we thought existed in C because we knew Java turned out to be wrong. For example, C does not have booleans, and array declarations are slightly different. Dealing with structs and pointers was also new. C seemed to be fairly flexible, which allowed for more creative solutions but also could allow for us to create a really inefficient and ineffective code.

Teamwork is clearly an essential part of any group project, especially this one. In order for us to be able to get anything done, we all had to be on the same page. We also had a difference in skill levels in programming, so time had to be spent to make sure everybody understood what was happening. We learned what group member was best at doing the task at hand and letting them taking point in that section.

Another lesson we learned was how to work in a limited time frame. We had to make sure we were completing the main objective first and to not get caught up on minor features. There were many things we could have done for the calculator, but the most important thing was to make sure it could actually calculate. The time we could have spent on a backspace feature we instead spent on making sure the input button worked correctly.

Working with actual hardware was also a bit of a learning experience. Although most of the problems we had were because of coding bugs, we also had times where it was the calculator itself that was malfunctioning. Understanding where the problems came from allowed us to find a solution quickly. In more complex hardware/software interfaces this skill would be crucial.

7 Criticism of the Course

The course was a brief interesting look into programming an embedded system. Overall, it was good. During the first lab we had to figure out how parts of C work through trial and error, which at the time was kind of frustrating. Although it wasn't a huge issue, a more thorough introduction to C could help.

The labs were clear enough for us to understand what was happening, but were open enough for us to complete the tasks the way we wanted to. I think the way all the labs would work together at the end would be helpful, as we didn't always understand how what we wrote would be used later on. For example, we didn't really know what we would do with the struct in the next lab, which caused us to have to rewrite some of our code.

The code reviews were helpful and definitely helped us create a better program. Learning how to comment effectively and structuring a program so it made sense was useful and interesting. Probably the most important thing that came from this class is the ability to create something that is intelligent enough to be used as a foundation for later work. It was nice that we learned more than syntax in a computer science class.

References

[1] "HP 20b Business Consultant Financial Calculator (F2219AA)." `http://www.amazon.com/HP-Consultant-Financial-Calculator-F2219AA/dp/B001D4XAMM` **: Electronics. N.p., n.d. Web. 19 Dec. 2012.**

[2] Smith, John, and Anthony Morris, and Wilfredo Lopez. "What is RPN?" `http://www.hpmuseum.org/rpn.htm`**. N.p., n.d. Web. 19 Dec. 2012.**

[3] "Converting Polish and Infix." http://www.theteacher.info/websites/ocr/WebPages/F453_Advanced/ConvertPolish/ConvertPolish.html. **N.p., n.d. Web. 19 Dec. 2012.**

[4] "AT91SAM7L128." - Atmel Corporation.http://www.atmel.com/devices/sam7l128.aspx **N.p., n.d. Web. 21 Dec. 2012.**

[5] Brain, Marshall. "How Microprocessors Work." Marshall, Brian. http://computer.howstuffworks.com/microprocessor.htm **N.p., n.d. Web. 21 Dec. 2012.**

[6] "System Controller." Altera News.http://www.altera.com/products/ip/iup/powerpc/m-eur-system-cont.html **N.p., n.d. Web. 21 Dec. 2012.**

[7] "How LCDs Work." Tyson, Jeff. HowStuffWorks. http://www.howstuffworks.com/lcd.htm **N.p., n.d. Web. 21 Dec. 2012.**