Preston Carroll & Jack Stone
Cpe 3150
May 10, 2023

Project 3
AVR Based Calculator in C

Project Summary

The final project transitions to C programming language for AVR microcontrollers. This makes some processes much easier because it is a higher-level programming language. This project also introduces serial communication to a terminal emulator and is used to display the calculations of the 4 basic mathematical functions performed on the microcontroller.

Keypad Coding

The first process of the project was to get the keypad to send the correct characters to the terminal. The keypad was connected to Port A, with 4 pins for the columns and 4 pins for the rows. By grounding all the rows and indefinitely checking the input from the columns, the column location can be determined. Then the usual debounce check can be applied to make sure it is a legitimate keypress. To determine which row the keypress is in, a low signal is sent to each row individually. Once the corresponding output is low it can be determined that is the row location of the keypress. A lookup table is then coded to represent the character location on the keypad. The code below is how this was accomplished.

```c
        _delay_ms(20);
        colloc = (Key_Pin & 0x0F);
        } while ( colloc == 0x0F);

        while(1)
        {
            Key_Prt = 0xEF;                         //Check row0
            asm("nop\n\t"
                "nop\n\t");
            colloc = ( Key_Pin & 0x0F);
            if(colloc != 0x0F)
            {
                rowloc = 0;
                break;
            }
            Key_Prt = 0xDF;                         //Check row1
            asm("nop\n\t"
                "nop\n\t");
            colloc = ( Key_Pin & 0x0F);
            if (colloc != 0x0F)
            {
                rowloc = 1;
                break;
            }
            Key_Prt = 0xBF;                         //Check row2
            asm("nop\n\t"
                "nop\n\t");
            colloc = (Key_Pin & 0x0F);
            if (colloc !=0x0F)
            {
                rowloc = 2;
                break;
            }
            Key_Prt = 0x7F;                   //check row3
            //_delay_ms(20);
            asm("nop\n\t"
                "nop\n\t");
            colloc = (Key_Pin & 0x0F);
            rowloc = 3;
            break;
        }
        if(colloc == 0x0E)
        {KeyPress = (keypad[ rowloc][0]);}
        else if(colloc == 0x0D)
        {KeyPress = (keypad[ rowloc][1]);}
        else if(colloc == 0x0B)
        {KeyPress = (keypad[ rowloc][2]);}
        else
        {KeyPress = (keypad[ rowloc][3]);}
        return KeyPress;
    }
```

Figure 1: Keypress Location

Serial communication

Once the correct character was retrieved from the array, it needs to be sent to the terminal using serial communication. One of the requirements for this is setting the baud rate. A baud rate of 9600 was desired. UBRR (ubrr) was defined at the beginning of the code using the formula below.

$$UBRR = \frac{16Mhz}{16(9600)} - 1$$

This value was then loaded into the UBRR register to set the baud rate. The next step was setting RXEN and TXEN bits high to enable transmission and reception. Then the UCSZ register is set to 3 (011) for 8-bit data transmission. To be able to send a character, the UDR register needs to be empty. Once this is checked the character can be sent to said register. A function was made that completes this process and is called any time a character is transmitted to the terminal.

```c
void USART_Init(unsigned int ubrr) {
//Set baud rate /
UBRR1H = (unsigned char) (ubrr >> 8);
UBRR1L = (unsigned char) ubrr;
UCSR1B = (1 << RXEN) | (1 << TXEN);
// Set frame format: 8data */
UCSR1C = (3 << UCSZ0);
}
```

Figure 2: USART settings

```c

void USART_send (unsigned char ch)
{
    while (! (UCSR1A & (1<<UDRE)));
    UDR1 = ch;
}
```

Figure 3:USART send function.

Calculator Specifications

The calculator operates in its own unique way. The max integer digits are limited to 4, while the max decimal digits are limited to 2. If the user tries to go over these limits an error message will be sent to the terminal. First the user is prompted to input the integer portion of the first number and then the decimal portion, the same is repeated for the second number and then an operator can be selected. The code keeps the integer and decimal portions separate to do arithmetic operations on. The characters entered from the keypad must first be converted to integers. The full integer (or decimal) entered is built by taking the previous integer, multiplying it by 10 and adding the current integer to it. Also, if the decimal entry is only 1 digit, it is multiplied by 10 so that it is treated as the correct decimal value i.e., .10 vs .01. Then to display the number a function called numout is called which converts the integer to characters which can then be sent to the terminal display.

```
int switch_num(char key, int num)
{
    num=num*10+(key-'0');
    return num;
}
```

Figure 4: Switch_num function

```
void numout(int num)
{
    char NumStr[42];
    itoa(num,NumStr, 10);
    textout(NumStr);
}
```

Figure 5: numout function

Calculator Operation

Once the four values have been entered, the four mathematical functions can be performed. The addition and subtraction do the operation on the two decimal values first. By keeping track of the result of this operation it can be determined if a carry is needed in the arithmetic. If so, the carry is then added or subtracted from the arithmetic of the two whole integer values. This way keeping track of the decimal location is not necessary. The output to the terminal will be the whole result with the character "." Followed by the decimal result.

```
    switch(oper)
    {
        case'+':
        {
            Rdec=Adec+Bdec;
            while(Rdec>100)
            {
                Rdec=Rdec-100;
                carry=carry+1;
            }
            Rwhole=Awhole+Bwhole+carry;
            return 0;
        }
        case'-':
        {
            Rdec=Adec-Bdec;
            while(Rdec<0)
            {
                Rdec=Rdec+100;
                carry=carry+1;
            }
            Rwhole=Awhole-Bwhole-carry;
            return 0;
        }
```

Figure 6: Addition and subtraction

For multiplication a partial FOIL method is used for the decimal. While the decimal result is greater than 100 the carry increments by 1, and 100 is subtracted from the decimal result. The whole parts are then multiplied together and added to the carry integer.

```
case'*':
{
    Rdec=(Awhole*Adec+Awhole*Bdec+Adec*Bdec);
    if(Awhole==0 && Bwhole==0)
    {

        temprf=Rdec/100;
        Rdec=round(temprf);
    }
    while(Rdec>100)
    {
        Rdec=Rdec-100;
        carry=carry+1;
    }
    Rwhole=Awhole*Bwhole+carry;
    USART_send(0xD);
    USART_send(0xA);
    USART_send(0xD);
    USART_send(0xA);
    return 0;
}
```

Figure 7: Multiplication

Division was the most difficult operation to code for this project and it was required to combine the whole and decimal parts of the operands. This is done by multiplying the whole part by 100 and adding the decimal part. When division occurs, it rounds the result to the nearest integer to keep the result from having decimals. The decimal part can be separated by using a while loop twice to shave off the last digit. Then the complete decimal can be built by taking the last digit that was removed multiplied by 10 and added to the first digit that was removed. The split sections can then be displayed with a decimal point between on the terminal. Included is an option for users to pay a $10 monthly subscription for full operation of the division operation. This decision came from “the higher ups”. We dare not impede their quest for money.

```
case'/':
{
    tempa=Awhole*100+Adec;//combines both parts of a
    tempb=Bwhole*100+Bdec;//combines both parts of b
    remain=tempa%tempb;//Gives Remainder
    remain=remain/100;
    temprf=tempa/tempb;//divides a by b
    temprf=temprf*100;

    tempr=round(temprf);//rounds to the nearest integer, also makes the value an integer again

    if(tempr/100!=0)
    {
    while(x<2)
    {
        z=y;
        y=tempr%10;//splits last digit from integer
        tempr=tempr-y;
        tempr=tempr/10;
        x++;
        }
    }
    Rdec=y*10+z;//recombines decimal parts of the result
    Rwhole=tempr;//should be the whole number part of the result
    USART_send(0xD);
    USART_send(0xA);
    USART_send(0xD);
    USART_send(0xA);
    textout("For a premium version of division as well as other add-on features, a $10 monthly subscription can be purchased at web.mst.edu/rdua/");
    USART_send(0xD);
    USART_send(0xA);

    return 0;
```

Figure 8: Division

Conclusion

This project showed the advantages of writing code for micro-controllers in C compared to assembly. But also showed that there are some benefits to writing in assembly, and that it should not be dismissed. There are hidden easter eggs found by pressing specific numbers on the keypad that the user can try to find. The use of serial communication for this project is extremely important and is very valuable knowledge moving forward into professional sectors. Almost all technology has a digital element to it, and microcontrollers and micro-processors will inevitably show up in many fields of electronic design. Learning so much about these tools opens doorways to so many projects. This is why Arduinos are so popular now. But working for a company, it would be wasteful to buy millions of Arduinos for their products when the same could be achieved with an AVR microcontroller. Learning how to program these IC's and how they function will be a great skill when finding employment in the future.