BENHA UNIVERSITY

BENHA FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRICAL ENGINEERING

Calculator using Embedded C

Submitted By:

Ahmed Mohsen Abouelyazed

Ahmed Lotfy El deep

Ibrahim Soliman Ibrahim

Ramadan Mahmoud Mohammed

Shaker Mohammed Shaker

Ahmed Ehab Mahmoud

Felopater nagy shokry

Power & Control Engineering, Department of Electrical Engineering,

Benha Faculty of Engineering, Benha University

Supervised By:

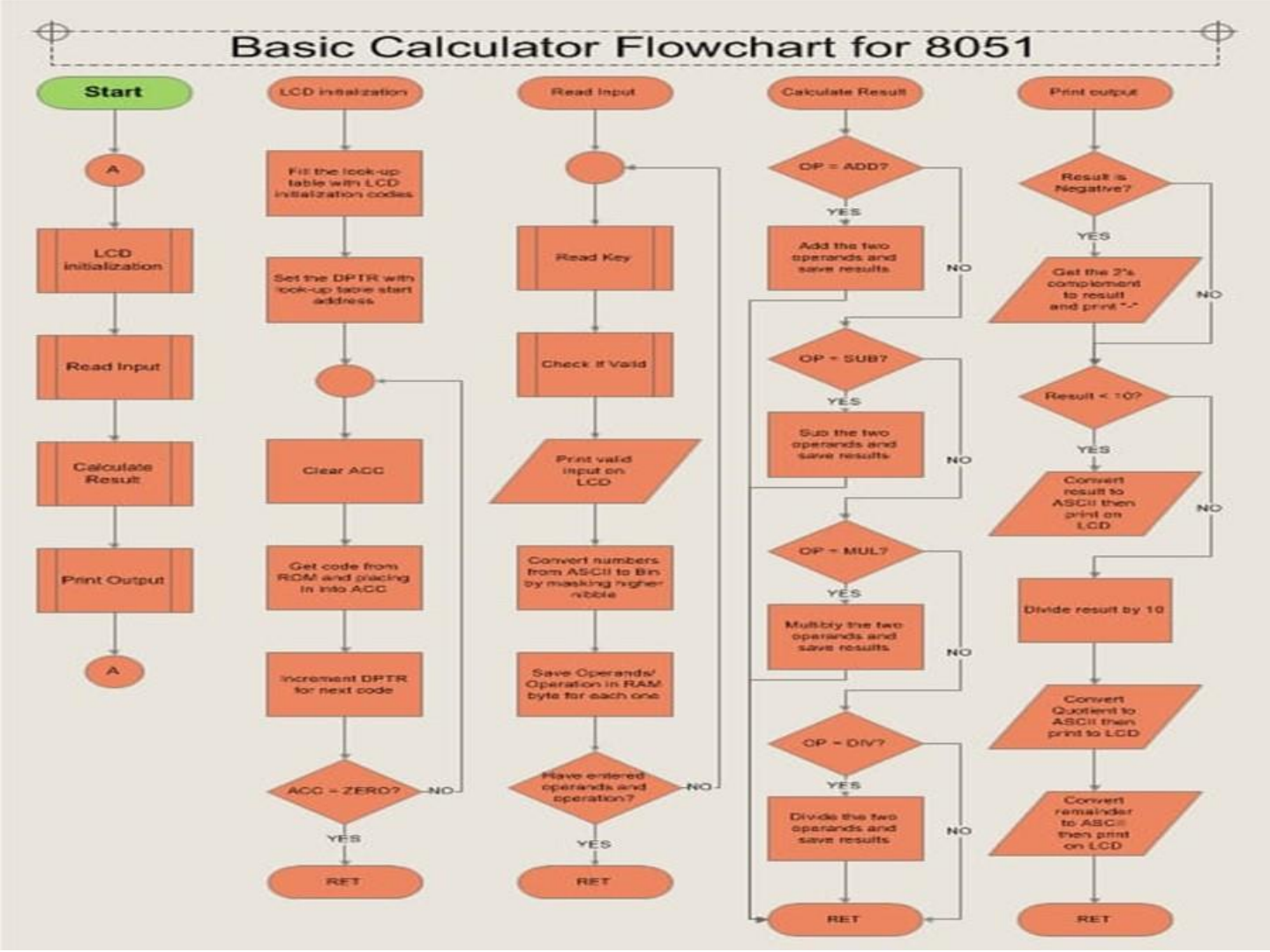
Dr. Ahmed El- Awamry

Department of Electrical Engineering, Benha Faculty of Engineering, Benha University, Benha

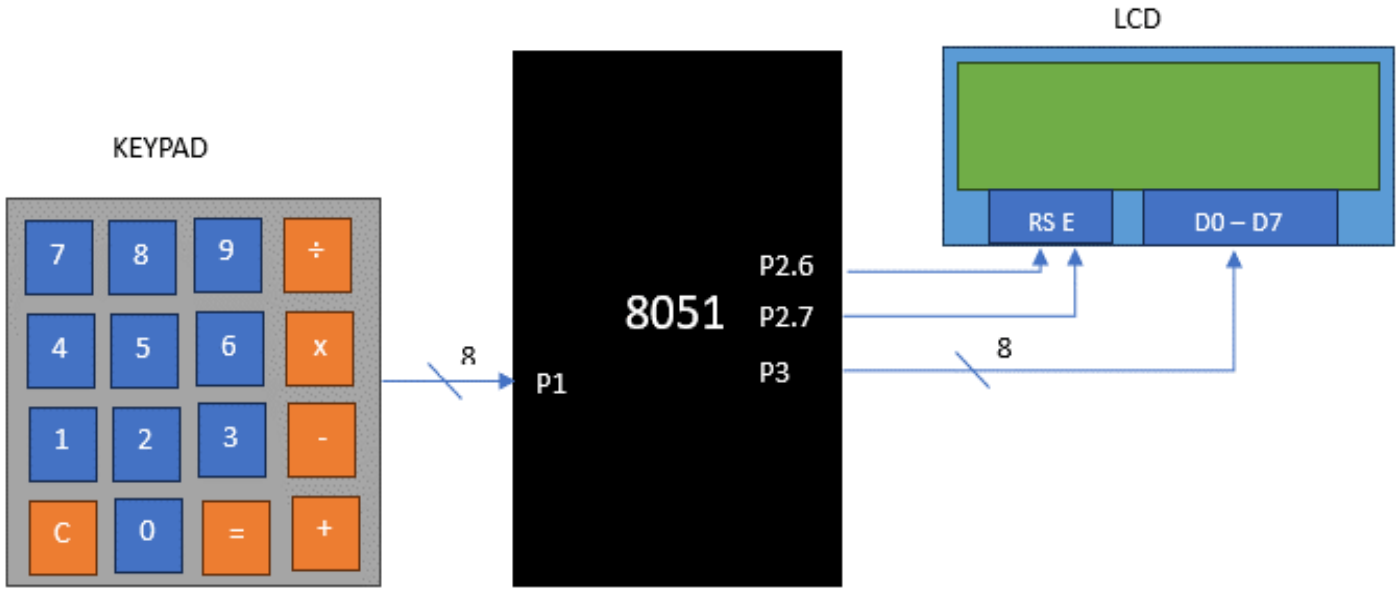
I. Abstract

Assembly code written from scratch to implement a basic calculator that performs the four basic operations: addition, subtraction, multiplication and division. An Initial message appears at the beginning to confirm to the user that our calculator functions efficiently. An operand-length limit is set to be 2 digits with the possibility to enter only one digit when desired. A remainder up to 4 digits after decimal point are provided in division operation result. There is also "ANSWER" feature which saves the result of current operation to be used in the next operation. Various error detection checkpoints to catch errors are added in order to avoid faults and wrong outputs. And also, the output is displayed up to 4 digits.

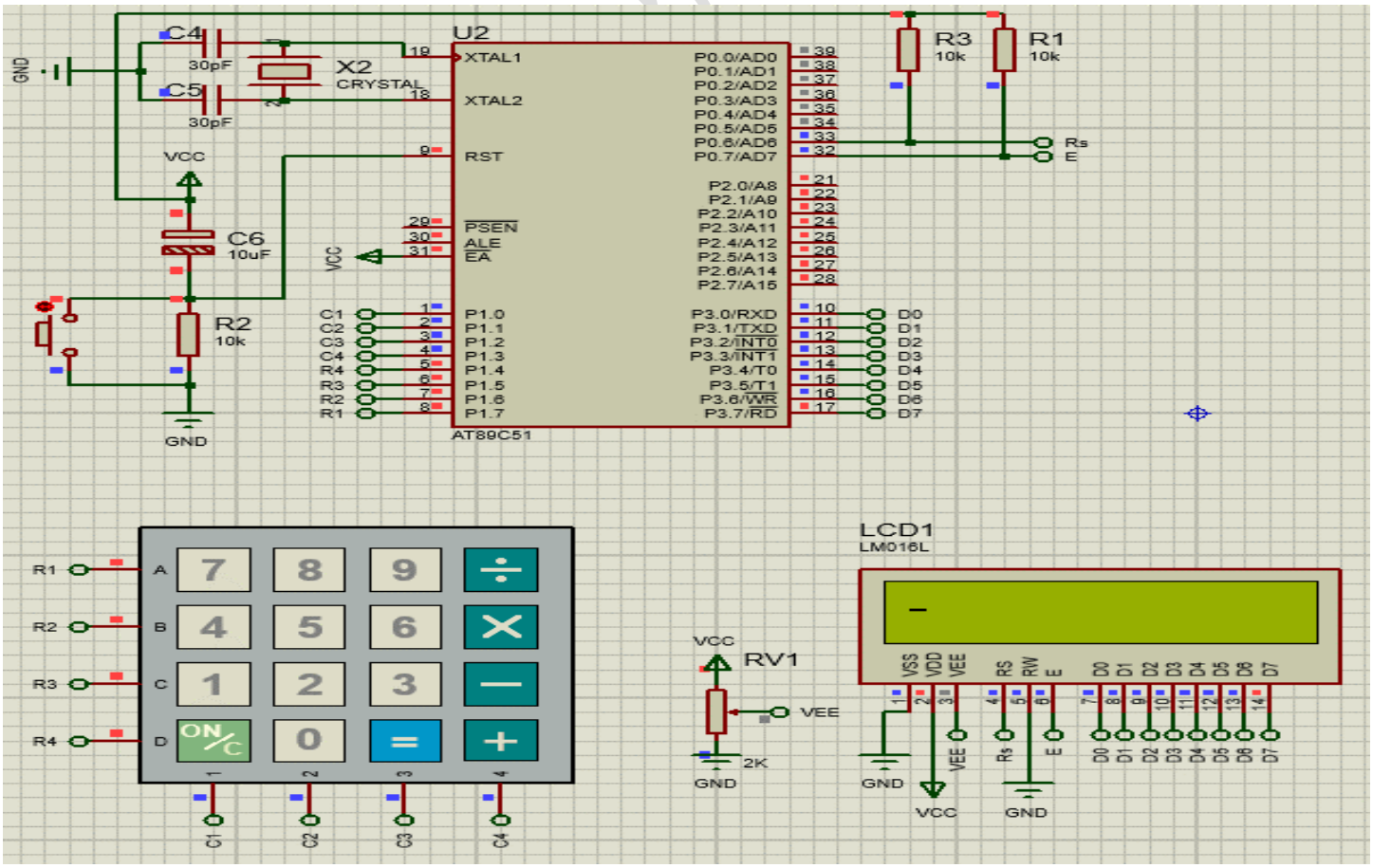
II. Flow chart



III. Simplified Block diagram



IV. Detailed Circuit



V. Assembly Code

```
#include <reg51.h>
#define max 9
#define LCD_PORT P3
sbit rs=P0^6;
sbit e=P0^7;
#define keyport P1
unsigned char colloc, rowloc;
unsigned char keypad[4][4];
void lcd_command(unsigned char x);
void lcd_init();
void lcd_data(unsigned char x);
void lcd_delete();
void delay(unsigned char x);
unsigned char key_detect();
char scr;

//*****************************************************************************

typedef struct Stack
{
char top;
signed int items[max];
} stack;

void reset(stack *a)
{
        a->top=-1;
}
void push(int pushed,stack *a)
{
 a->items[++a->top]=pushed;
}
signed int pop(stack *a)
{
        return a->items[a->top--];
}

signed int peek(stack a)
{
        return a.items[a.top];
}
signed int underpeek(stack a)
{
        return a.items[a.top-1];
}
signed int returnintresult(stack a)
{
        return a.items[0];
}

signed int returnfloatresult(stack a)
```

```
{
        return a.items[2];
}
//*****************************************************************************************************
char Priority(char val)
{
   if(val=='+'||val=='-')
      return 1;
   else if(val=='*'||val=='/')
      return 2;
   else
      return 0;
}

/*****************************************************************************************************
*****************************************************************************************************/
void main()
{
        unsigned char x;
        signed int digits [19];
        char digit_counter;
        char numalpha ;
        stack alpha;
        char evaluation;
        long firstoperator;
        long secoundoperator;
        signed int firstfloat;
        signed int secoundfloat;
        signed int resultfloat;
        signed int resultint;
        signed int ans_int;
        signed int ans_float;

        start:
        scr=0;
        digit_counter=0;
        x='0';
        evaluation=0;
        numalpha = 3;
        reset(&alpha);
        P3=0xff;
        lcd_init();
        x=key_detect();
        ans_label:
        while (x!='=')
        {
                while(x=='.')
        {
        lcd_data(x);
        digits[digit_counter++]=x;
        numalpha=1;
        x=key_detect();
        }
        lcd_data(x);
```

```
if (x>='0'&&x<='9')
{       x=x-0x30;
        if (numalpha==0)
        digits[digit_counter-1]=digits[digit_counter-1]*10+x;
        else
        digits[digit_counter++]=x;
        numalpha=0;
}
else
{
if(numalpha!=0)//if operation pressed at the bggenning we push 0
digits[digit_counter++]=0;

if(Priority(x)>Priority(peek(alpha)))
push(x,&alpha);
else
{
digits[digit_counter++]=pop(&alpha);
push(x,&alpha);
}
numalpha=1;
}
x=key_detect();
}

while(alpha.top!=-1)
{
digits[digit_counter++]=pop(&alpha);
}
digits[digit_counter]='$';
lcd_data(x);
reset(&alpha);



while(digits[evaluation]!='$')
{
        if (digits[evaluation]=='+'||digits[evaluation]=='-'||digits[evaluation]=='*'||digits[evaluation]=='/')
        {
        if (underpeek(alpha)=='.')
        {
        secoundfloat=pop(&alpha);
        if (secoundfloat/10==0)
        secoundfloat*=10;
        pop(&alpha);
        secoundoperator=pop(&alpha);
        }
        else
        {
        secoundoperator=pop(&alpha);
        secoundfloat=0;
        }
        if (underpeek(alpha)=='.')
        {
```

```
		firstfloat=pop(&alpha);
		if (firstfloat/10==0)
		firstfloat*=10;
		pop(&alpha);
		firstoperator=pop(&alpha);
}
else
{
firstoperator=pop(&alpha);
firstfloat=0;
}


switch (digits[evaluation])
{
		case '+':
		resultfloat=firstfloat+secoundfloat;
		resultint=firstoperator+secoundoperator+(resultfloat/100);
		resultfloat=resultfloat-((resultfloat/100)*100);
		push(resultint,&alpha);
		push('.',&alpha);
		push(resultfloat,&alpha);
		break;

		case '-':
		if(firstfloat<secoundfloat)
		{
			firstfloat+=100;
			firstoperator--;
		}
		resultfloat=firstfloat-secoundfloat;
		resultint=firstoperator-secoundoperator;
		push(resultint,&alpha);
		push('.',&alpha);
		push(resultfloat,&alpha);
		break;

		case '*':
		firstoperator=firstoperator*100+firstfloat;
		secoundoperator=secoundoperator*100+secoundfloat;
		resultfloat=firstoperator*secoundoperator%10000;
		resultint=firstoperator*secoundoperator/10000;
		push(resultint,&alpha);
		push('.',&alpha);
		push(resultfloat,&alpha);
		break;
		case '/':
		{
		firstoperator=firstoperator*100+firstfloat;
		secoundoperator=secoundoperator*100+secoundfloat;
		resultfloat=((firstoperator%secoundoperator)*100)/secoundoperator;
		resultint=firstoperator/secoundoperator;
		push(resultint,&alpha);
		push('.',&alpha);
```

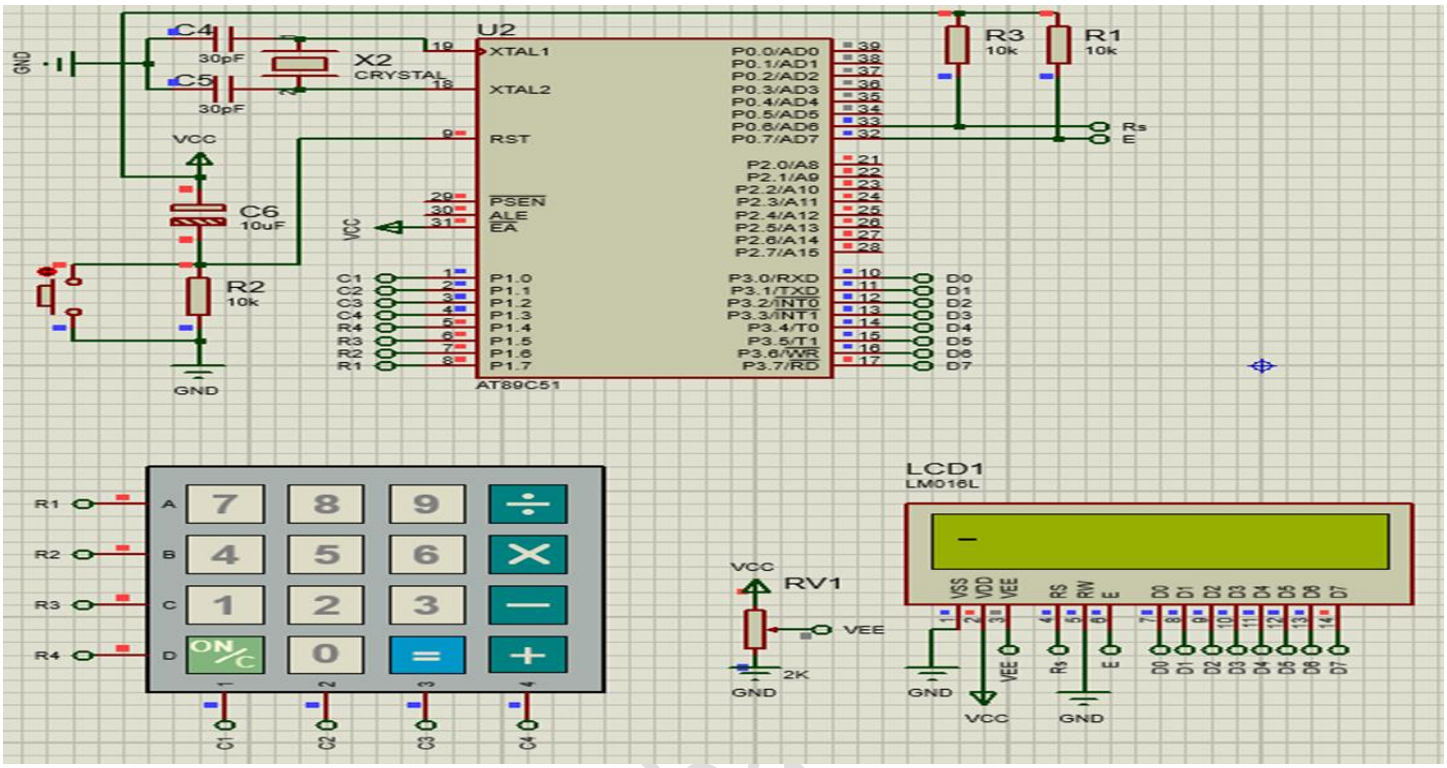
```
                    push(resultfloat,&alpha);
                    }
                    }}
else
push(digits[evaluation],&alpha);
evaluation++;
}


resultint=returnintresult(alpha);
ans_int=resultint;
resultfloat=returnfloatresult(alpha);
ans_float=resultfloat;
if(resultint<0)
{
lcd_data('-');
resultint=0x10000-resultint;
}

reset(&alpha);
while(resultint!=0)
{
push(resultint%10,&alpha);
resultint/=10;
}
if (alpha.top==-1)
                    push(0,&alpha);
while(alpha.top!=-1) //to display integer nums
{
lcd_data(pop(&alpha)+0x30);
}


lcd_data('.');

while(resultfloat!=0)
{
push(resultfloat%10,&alpha);
resultfloat=resultfloat/10;
}
if (alpha.top<1)
                    push(0,&alpha);

while(alpha.top!=-1) //to display integer nums
{
lcd_data(pop(&alpha)+0x30);
}




x=key_detect();
if(x=='/'||x=='*'||x=='+'||x=='-')
{
```

```
        reset(&alpha);
        lcd_init();
        lcd_data('A');
        lcd_data('N');
        lcd_data('S');
        digit_counter=0;
        digits[digit_counter++]=ans_int;
        digits[digit_counter++]='.';
        digits[digit_counter++]=ans_float;
        numalpha=0;
        scr=4;
        evaluation=0;

        goto ans_label;

        }
        else
        goto start;


}
/*****************************************************************************************************
*****************************************************************************************************/
void delay(unsigned char itime)
{
unsigned char j;
{
for (j=0;j<itime;j++);
}
}

//LCD INTERACING:
void lcd_command(unsigned char x)
{
LCD_PORT=x;
rs=0;
e=1;
x=0;
e=0;
delay(255);

}

void lcd_init()
{
lcd_command(0x38); //8bits & 2lines
lcd_command(0x0e); // display on cursor blinking
lcd_command(0x01); // clear screen
lcd_command(0x80); //force curcor to 1st line
}

void lcd_data(unsigned char x)
{
scr++;
```

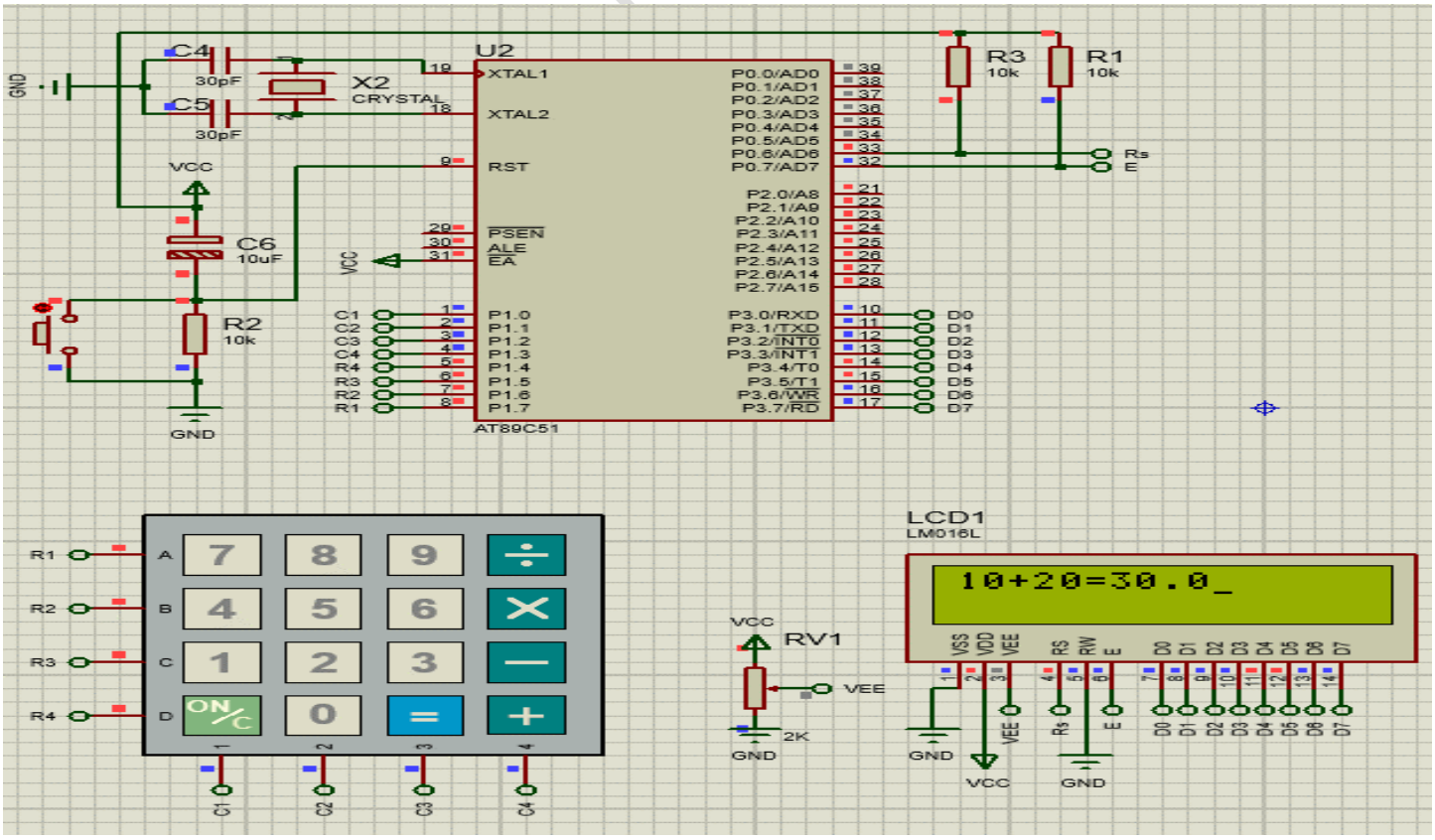
```
if (scr>15)
        lcd_command(0x18);
LCD_PORT=x;
rs=1;
e=1;
delay(255);
e=0;

}


unsigned char keypad[4][4] = {{'.','7','4','1'},

        {'0','8','5','2'},

        {'=','9','6','3'},

        {'+','-','*','/'} };

unsigned char key_detect()
{
        keyport=0xF0;                   /*set port direction as input-output*/
        do
        {
                keyport = 0xF0;
                colloc = keyport;
                colloc&= 0xF0;   /* mask port for column read only */
        }while(colloc != 0xF0);         /* read status of column */

        do
        {
                do
                {
                        delay(20);      /* 20ms key debounce time */
                        colloc = (keyport & 0xF0);          /* read status of column */
                }while(colloc == 0xF0);   /* check for any key press */

                delay(1);
                colloc = (keyport & 0xF0);
        }while(colloc == 0xF0);

        while(1)
        {
                /* now check for rows */
                keyport= 0xFE;                                                  /* check
for pressed key in 1st row */
                colloc = (keyport & 0xF0);
                if(colloc != 0xF0)
                {
                        rowloc = 0;
                        break;
                }
```

```
                keyport = 0xFD;                                                 /* check for pressed key in
2nd row */
                colloc = (keyport & 0xF0);
                if(colloc != 0xF0)
                {
                        rowloc = 1;
                        break;
                }

        keyport = 0xFB;                 /* check for pressed key in 3rd row */
        colloc = (keyport & 0xF0);
        if(colloc != 0xF0)
        {
                rowloc = 2;
                break;
        }

        keyport = 0xF7;                 /* check for pressed key in 4th row */
        colloc = (keyport & 0xF0);
        if(colloc != 0xF0)
        {
                rowloc = 3;
                break;
        }
}

        if(colloc == 0xE0)
        {
                return(keypad[rowloc][0]);
        }
        else if(colloc == 0xD0)
        {
                return(keypad[rowloc][1]);
        }
        else if(colloc == 0xB0)
        {
                return(keypad[rowloc][2]);
        }
        else
        {
                return(keypad[rowloc][3]);
        }
}
```

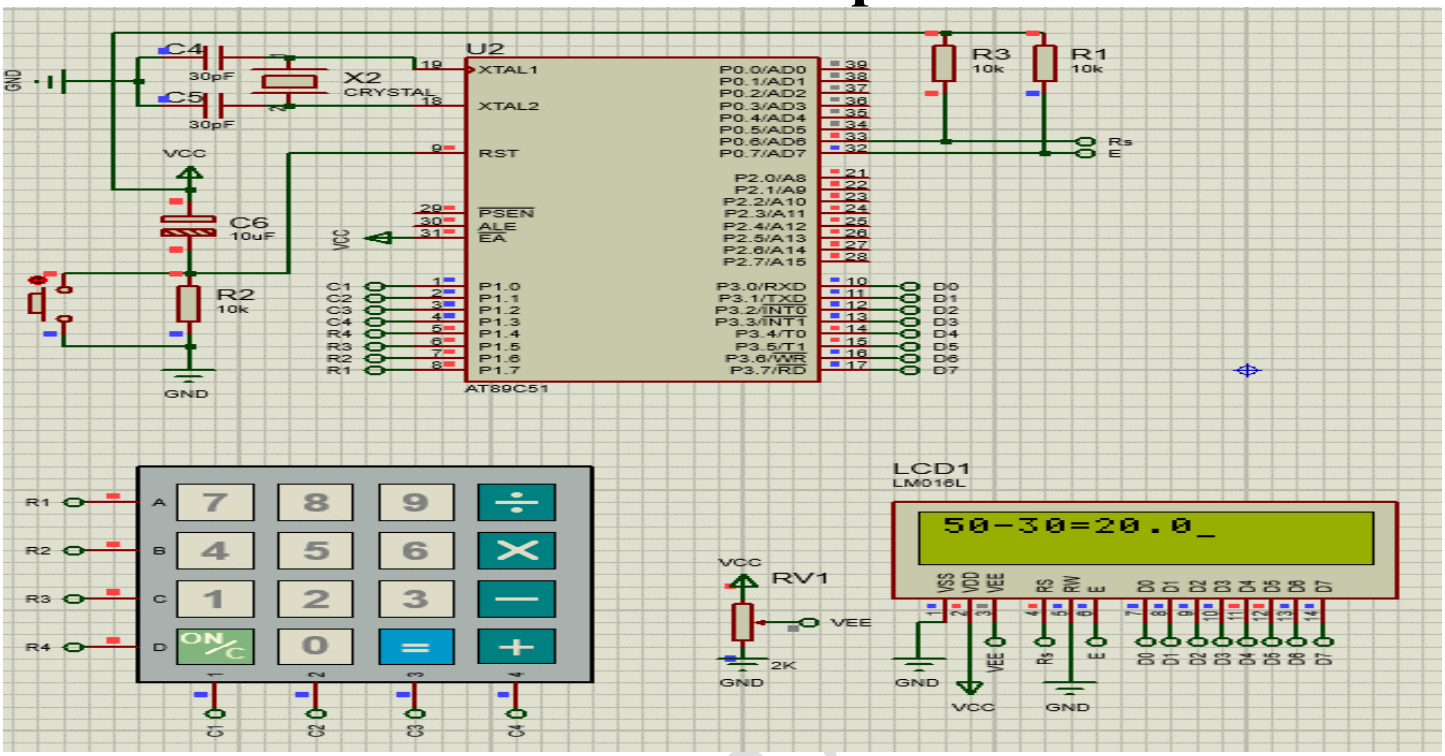
VI. PROTEUS Simulations and Results



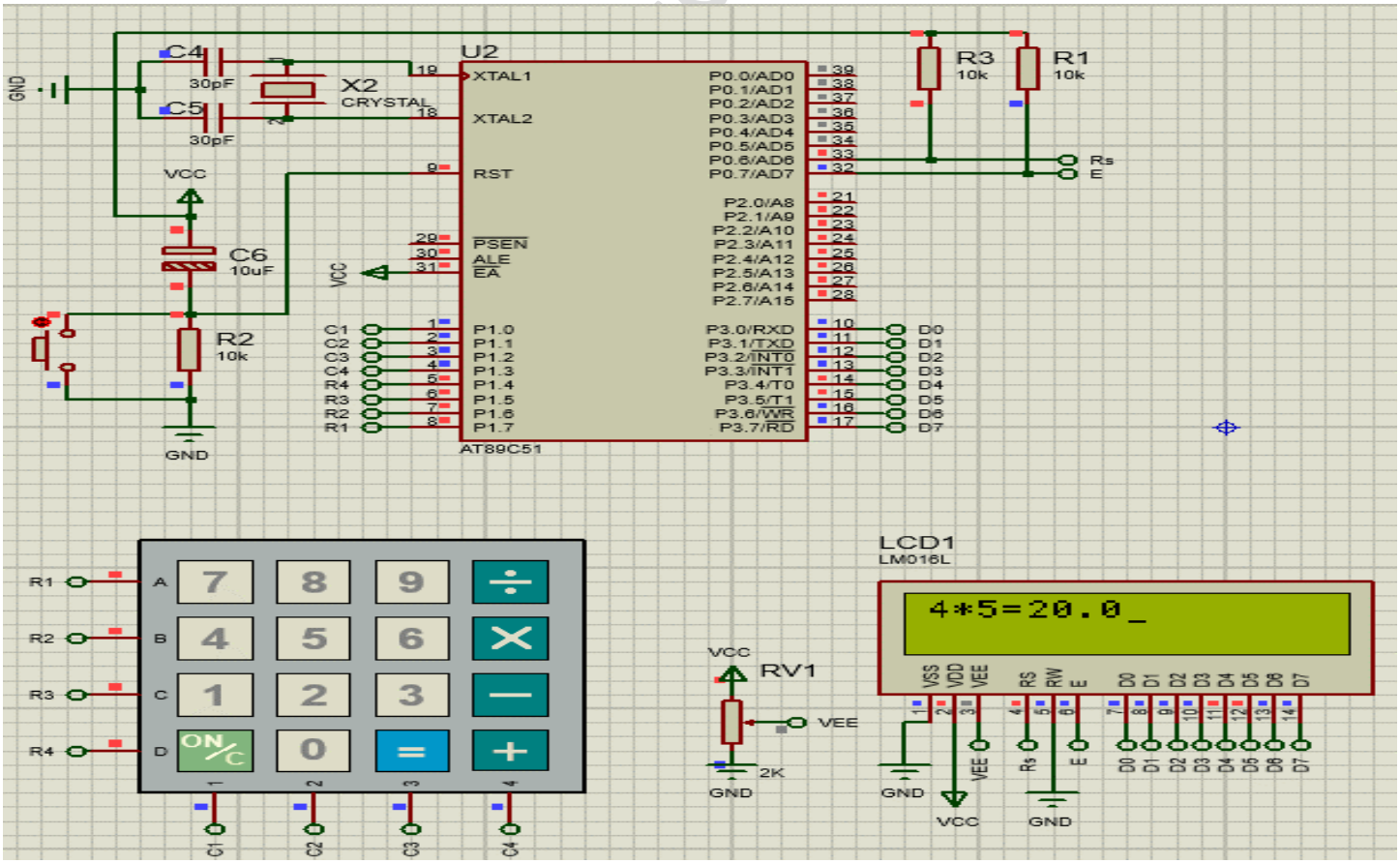
Addition Operation



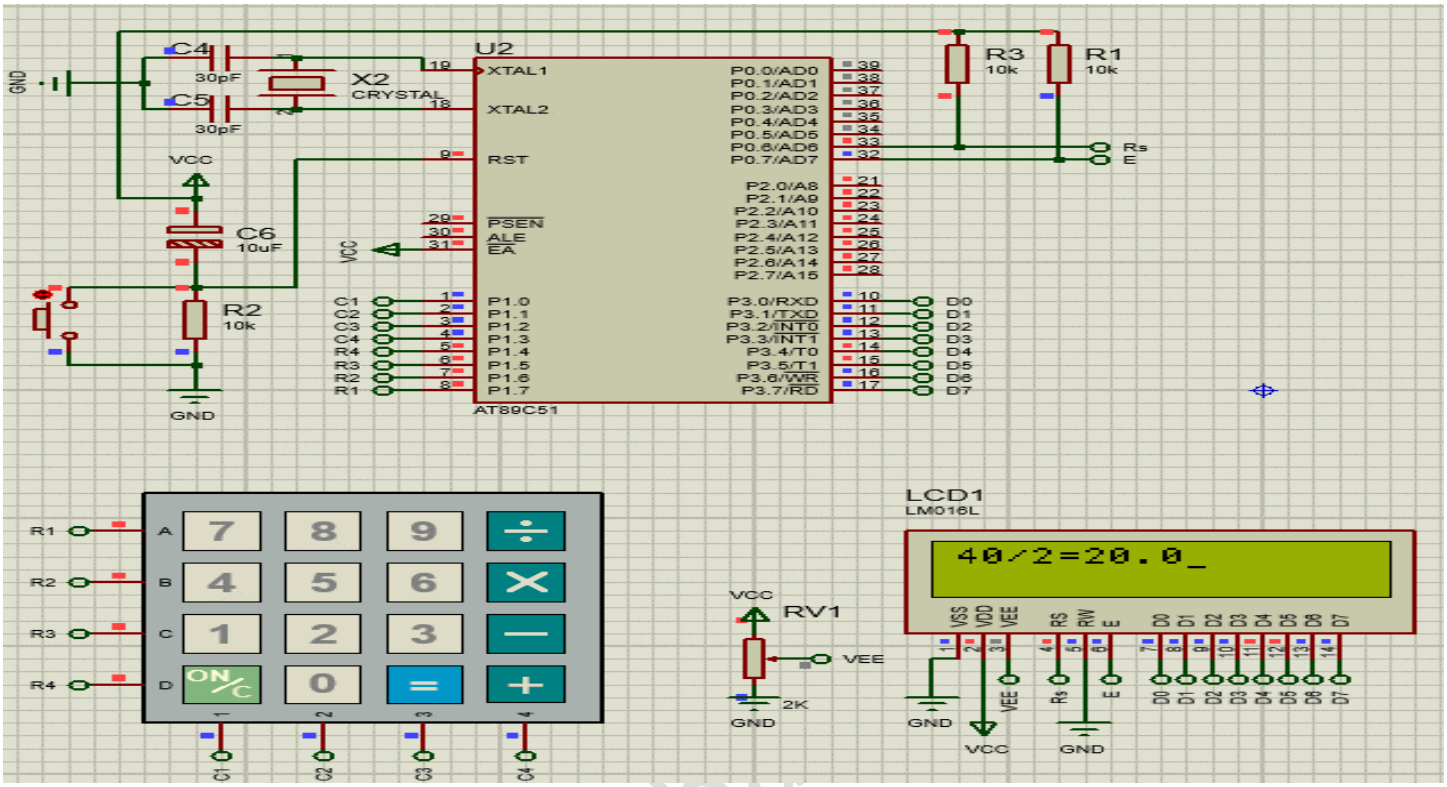
Subtraction Operation



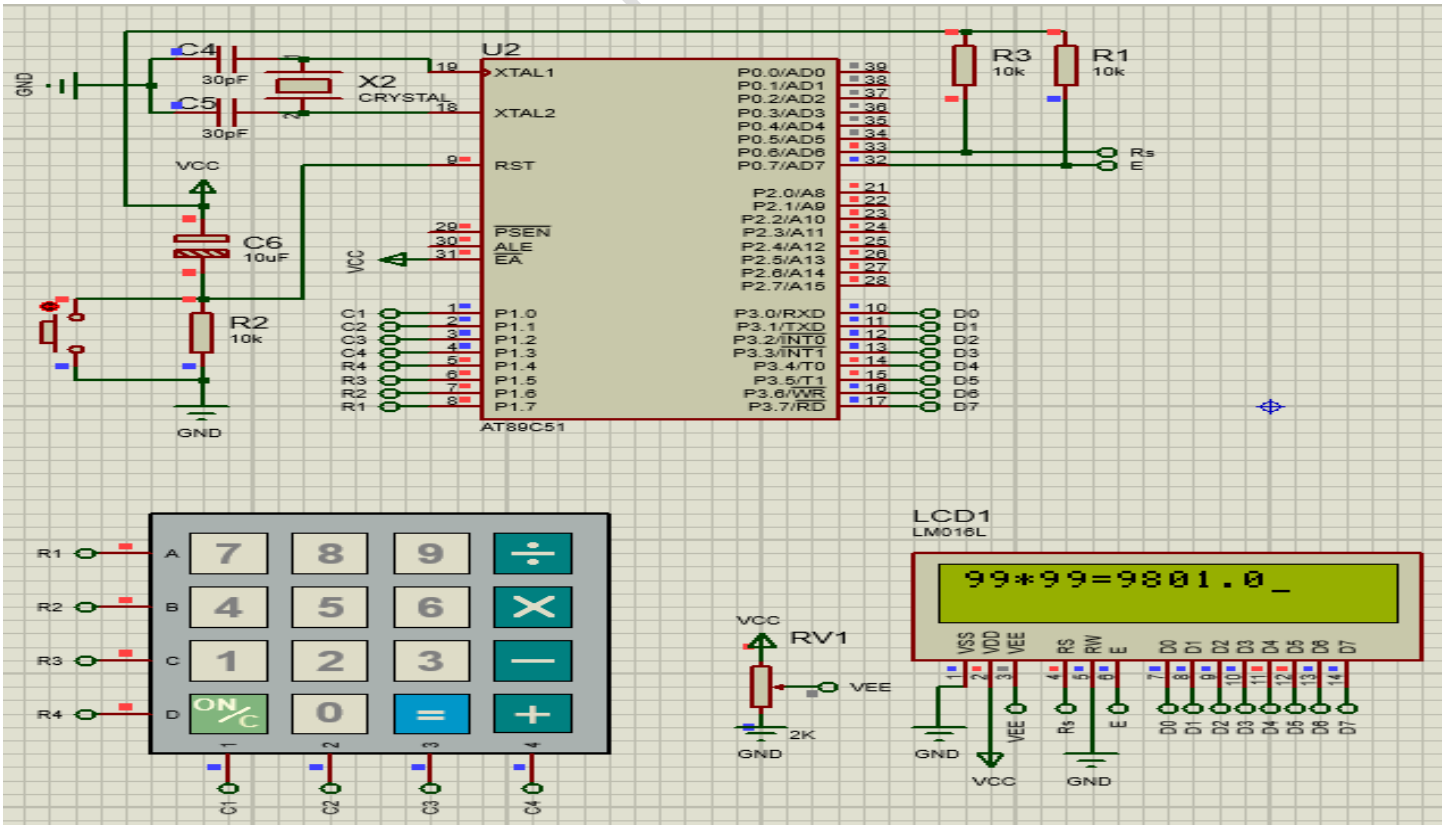
Multiplication Operation

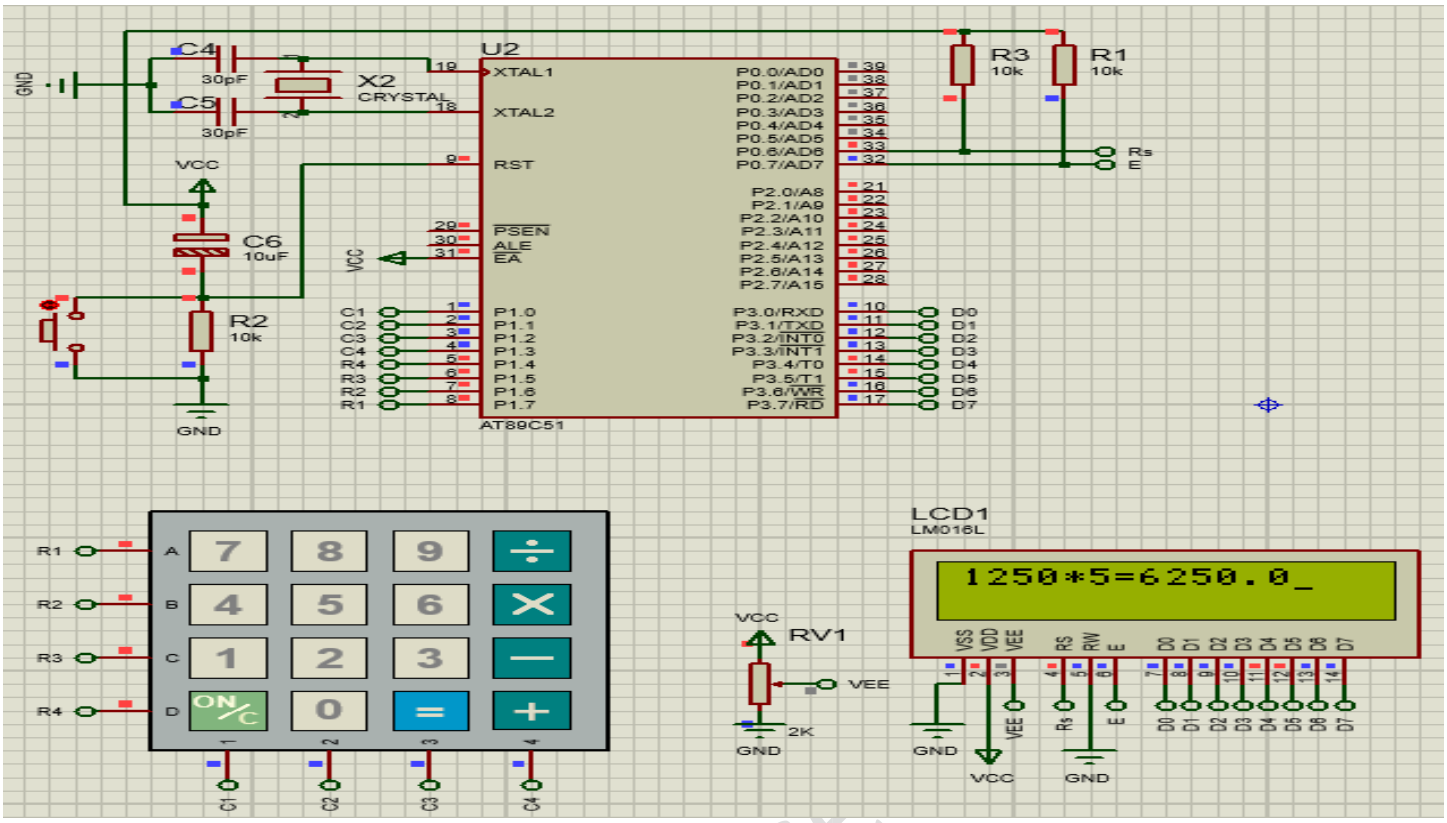


Division Operation

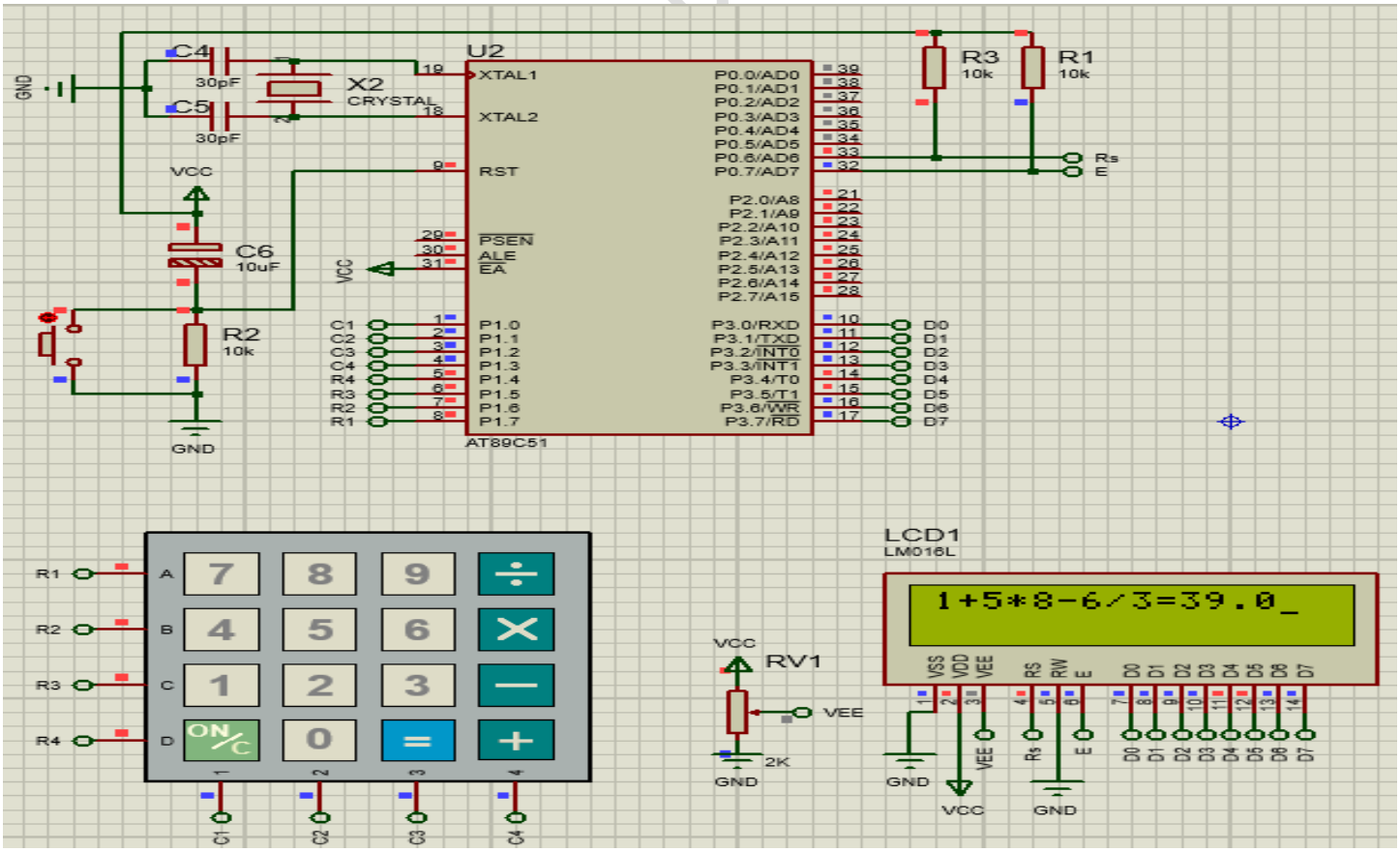


Multiple digit Inputs and Multiple digits output (feature)

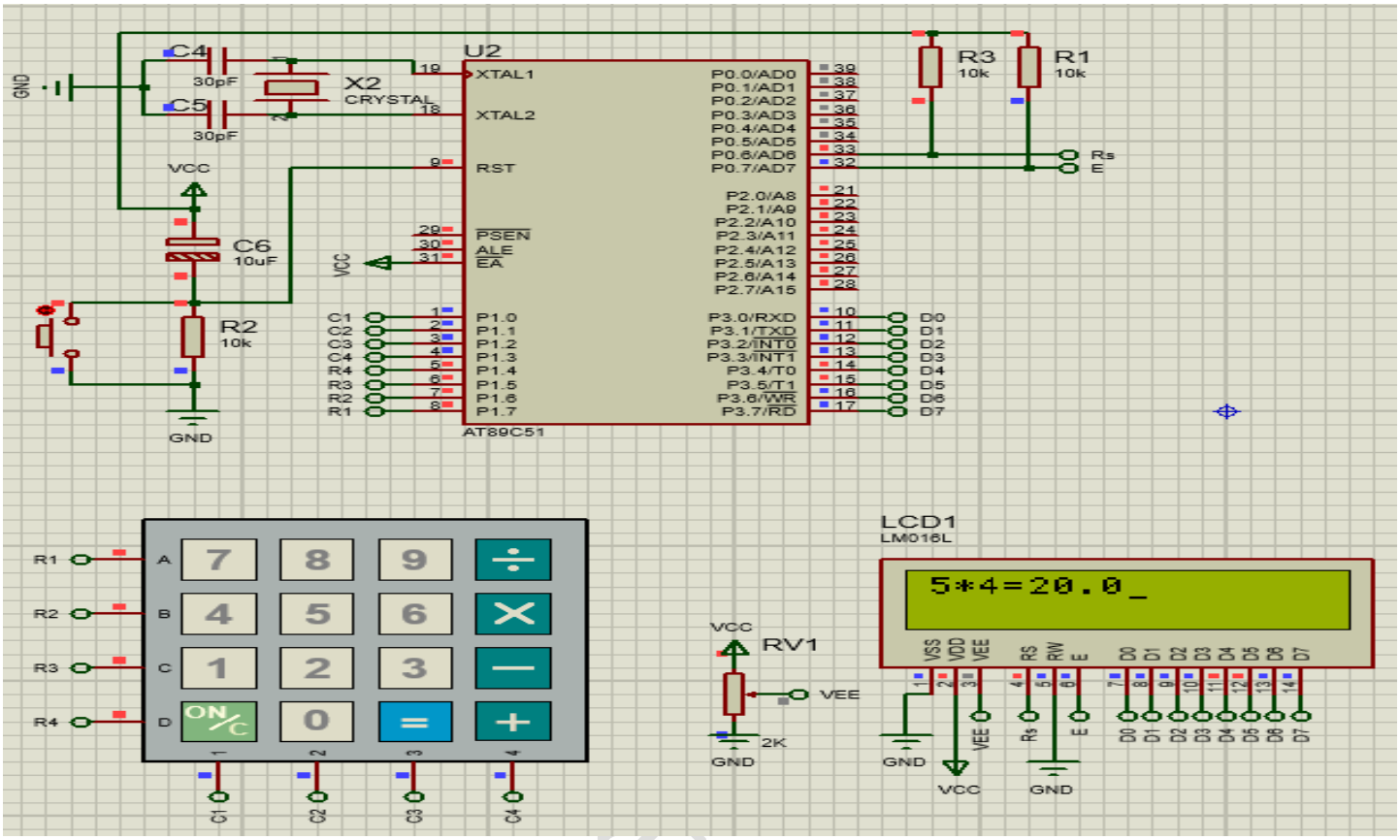


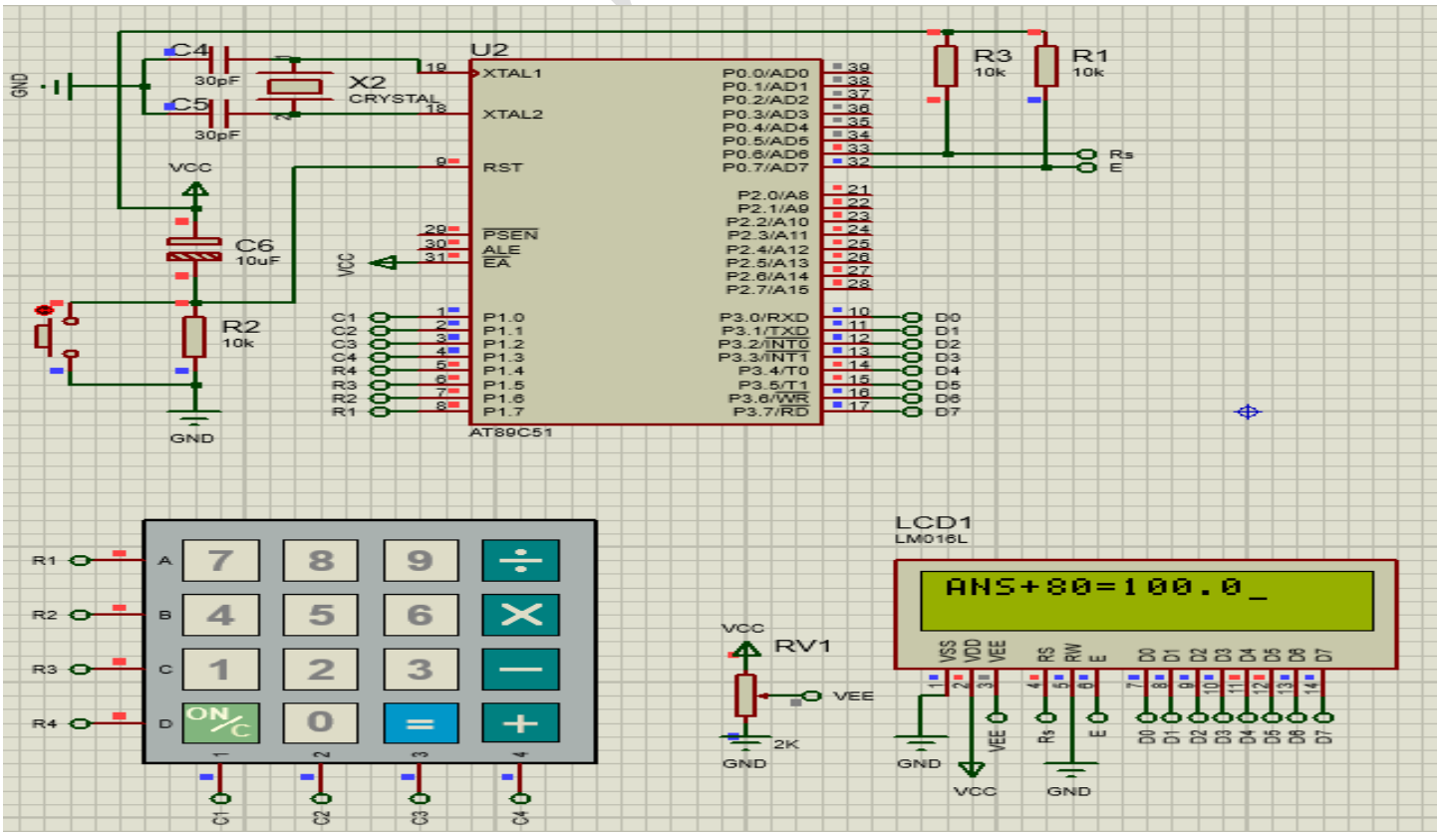


Multiple Operations at the same time (feature)

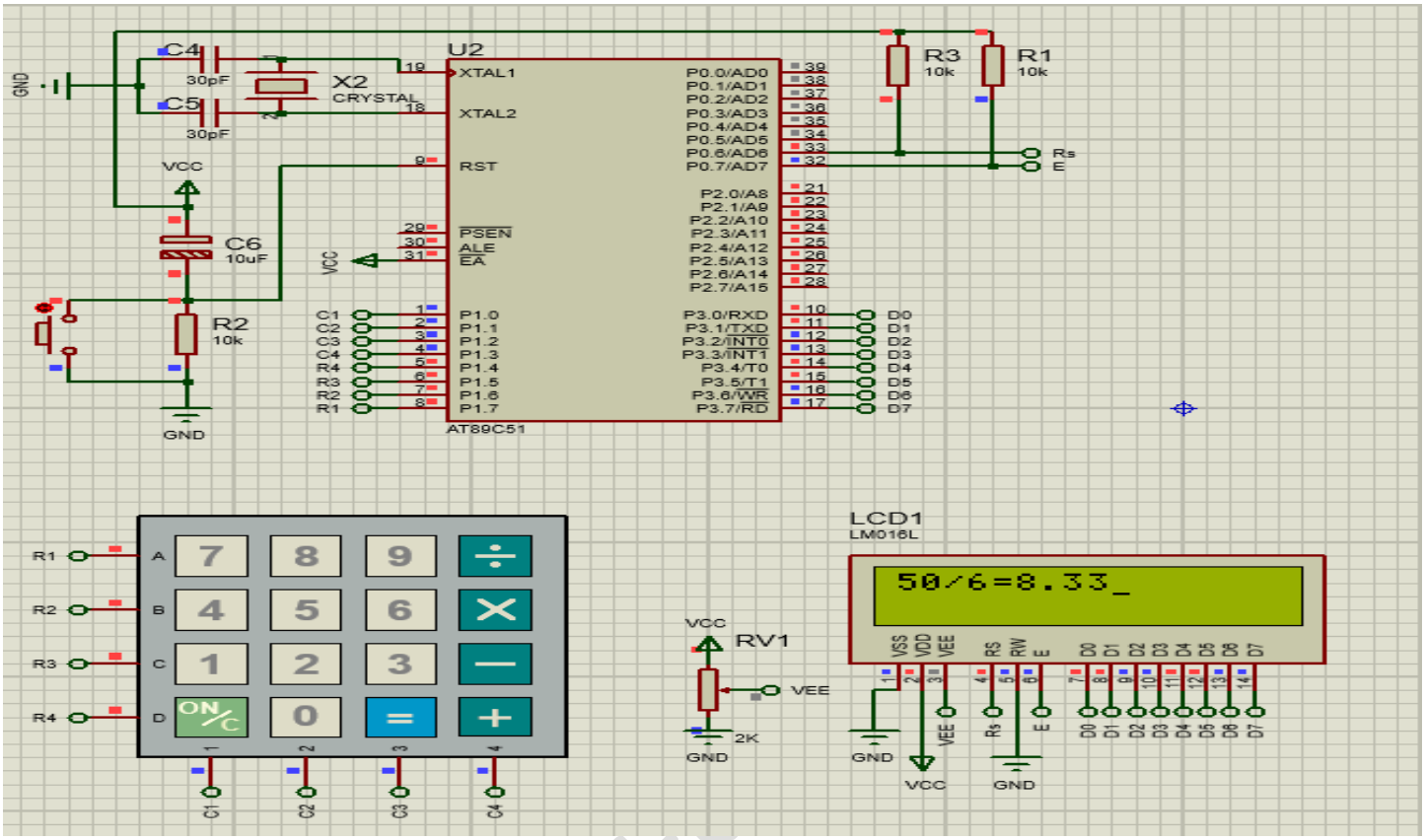


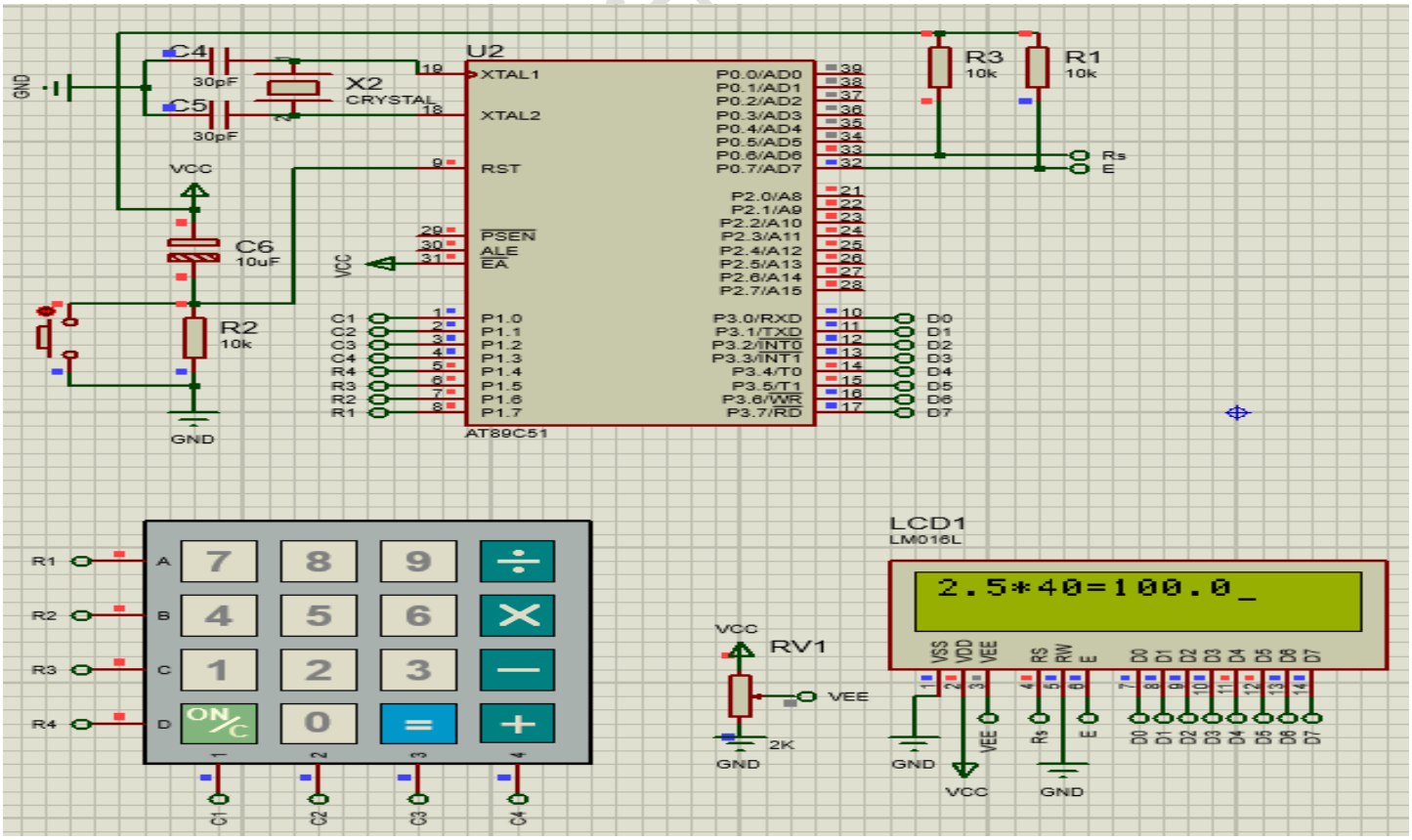
“ANS” feature


C4
30pF
C5
30pF
X2
CRYSTAL
U2
XTAL1
XTAL2
RST
PSEN
ALE
EA
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
AT89C51
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
P2.0/A8
P2.1/A9
P2.2/A10
P2.3/A11
P2.4/A12
P2.5/A13
P2.6/A14
P2.7/A15
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
R3
10k
R1
10k
Rs
E
C6
10uF
R2
10k
VCC
GND
D0
D1
D2
D3
D4
D5
D6
D7
LCD1
LM016L
5*4=20.0_
RV1
2K
VEE
VSS
VDD
RS
RW
E
R1
R2
R3
R4
A
B
C
D
7
8
9
÷
4
5
6
×
1
2
3
—
ON/C
0
=
+


C4
30pF
C5
30pF
X2
CRYSTAL
U2
XTAL1
XTAL2
RST
PSEN
ALE
EA
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
AT89C51
P0.0/AD0
P0.1/AD1
P0.2/AD2
P0.3/AD3
P0.4/AD4
P0.5/AD5
P0.6/AD6
P0.7/AD7
P2.0/A8
P2.1/A9
P2.2/A10
P2.3/A11
P2.4/A12
P2.5/A13
P2.6/A14
P2.7/A15
P3.0/RXD
P3.1/TXD
P3.2/INT0
P3.3/INT1
P3.4/T0
P3.5/T1
P3.6/WR
P3.7/RD
R3
10k
R1
10k
Rs
E
C6
10uF
R2
10k
VCC
GND
D0
D1
D2
D3
D4
D5
D6
D7
LCD1
LM016L
ANS+80=100.0_
RV1
2K
VEE
VSS
VDD
RS
RW
E
R1
R2
R3
R4
A
B
C
D
7
8
9
÷
4
5
6
×
1
2
3
—
ON/C
0
=
+

Floating Point Input or/and Output (feature)





VII. Hardware documentation