امیرحسین احمدی ۹۷۵۲۲۲۹۲ – تمرین ششم

۱. برای این سوال ابتدا پین های C و D را خروجی قرار میدهیم که در C، دهگان عدد و در D یکان آن قرار خواهد گرفت.

```
LDI     R16,0xFF
OUT     DDRC,R16        ;PORTC as output
OUT     DDRD,R16        ;PORTD as output
```

در ادامه رجیسترهای R16, R17, R18, R19 را مقدار دهی اولیه میکنیم که به ترتیب از چپ، یکان عدد فعلی فیبوناچی، دهگان آن، یکان عدد قبلی فیبو ناچی و دهگان آن هستند. و بعد از آن به تابع دیسپلی رفته تا عددمان را درخروجی چاپ کنیم.

```
RESET:
        LDI     R16,1           ;Yekan Fibo(t)
        LDI     R17,0           ;Dahgan Fibo(t)
        LDI     R18,0           ;Yekan Fibo(t - 1)
        LDI     R19,0           ;Dahgan Fibo(t - 1)
        RJMP    DISPLAY
```

در تابع دیسپلی رجیستر R17 را در خروجی C و R16 را در خروجی D قرار میدهیم و تابع DELAY_DISPLAY را صدا زده که با انجام سه فور تو در تو کمی دیلی بین خروجی ها بیندازد تا اعداد به راحتی دیده شوند.

```
DISPLAY:
        OUT     PORTC,R17
        OUT     PORTD,R16

        CALL    DELAY_DISP

        RJMP    BEGIN

DELAY_DISP:
        LDI     R21,0x02
L4:     LDI     R22,0xFF
L5:     LDI     R23,0xFF
L6:     DEC     R23
        BRNE    L6
        DEC     R22
        BRNE    L5
        DEC     R21
        BRNE    L4
        RET
```

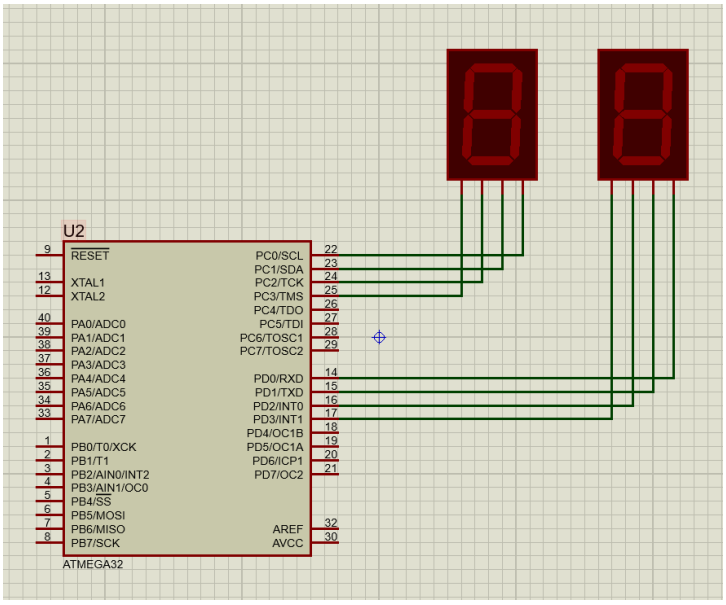
در ادامه به BEGIN رفته تا الگوریتم شروع شود. در ابتدا R18 و R19 را در R26 و R27 ریخته و R16 و R17 را در آن ها نگه میداریم تا بتوانیم در مرحله بعد فیبوناچی این مرحله را داشته باشیم. سپس با صدا زدن تابع DECREMENT روند آپدیت کردن R16 و R17 را شروع میکنیم.

```
BEGIN:
        MOV     R26, R18
        MOV     R27, R19
        MOV     R18, R16
        MOV     R19, R17

        CALL    DECREMENT
```

حال به این صورت عمل میکنیم که در هر مرحله یکی از R26 و R27 کم کرده (تابع DECREMENT) و به R16 و R17 اضافه میکنیم (تابع INCREMENT) که در نهایت Fibo(t) جدید برابر با $\mathrm{Fibo}(t-1) + \mathrm{Fibo}(t-2)$ شود و آن را نمایش دهیم. در هر مرحله از DECREMENT اگر مقدار برابر با صفر بود الگوریتم تمام شده و باید آن را نمایش دهیم و در غیر این صورت به INCREMENT برویم. در INCREMENT اگر عدد ما بیشتر از ۹۹ شد یعنی دارد سه رقمی میشود، از آن رو تابع RESET را صدا میزنیم تا مقادیر از اول شروع شوند.

```
INCREMENT:
        CPI     R16,9
        BRLO    INC_YEKAN
        CPI     R17,9
        BRLO    INC_DAHGAN
        RJMP    RESET

DECREMENT:
        CPI     R26,1
        BRSH    DEC_YEKAN
        CPI     R27,1
        BRSH    DEC_DAHGAN
        RJMP    DISPLAY
```

برای مدار این سوال نیز، فایل hex به دست آمده از کد را به روی یک قطعه ATMEGA32 آپلود کرده و خروجی پایه های C را به یک سون سگمنت و خروجی پایه های D را به دیگری وصل میکنیم. از آنجایی که دو عدد کمتر از ۱۰ هستند، حداکثر چهار پایه اول هر کدام مقدار دارند و نیاز به وصل کردن بقیه پایه ها نیست.



۲. برای این سوال باید یک ماشین حساب با اعمال اصلی و رعایت الویت پیاده سازی میکردیم. برای این کار در ابتدا یک تابع calculate تعریف کرده که با گرفتن دو عدد و عملگر مورد نظر به صورت کاراکتر، نتیجه را برای ما خروجی دهد.

```
float calculate(float num1, float num2, unsigned char operation) {
    float result;
    switch (operation)
        {
            case KEYPAD_ADDITION:
                result = num1 + num2;
                break;
            case KEYPAD_MULTIPLICATION:
                result = num1 * num2;
                break;
            case KEYPAD_DIVISION:
                result = num1 / num2;
                break;
            case KEYPAD_MINUS:
                result = num1 - num2;
                break;
            default: break;
        }
    return result;
}
```

همچنین یک تابع keypad_scan که در تمرینات پیاده سازی شده بود، ولی با این تفاوت که با وارد کردن هر دکمه در ماشین حساب، کاراکتر آن را خروجی بدهد. در زیر این قسمت مربوط به کلید های سطر اول ماشین حساب را مشاهده میکنید.

```
unsigned char keypad_scan() {
    unsigned char result = 255;

    ////////////////////// ROW1 //////////////////////
    KEYPAD_R1 = 1;
    KEYPAD_R2 = 0;
    KEYPAD_R3 = 0;
    KEYPAD_R4 = 0;

    delay_ms(5);
    if (KEYPAD_C1)
        result = KEYPAD_NUM7;
    else if (KEYPAD_C2)
        result = KEYPAD_NUM8;
    else if (KEYPAD_C3)
        result = KEYPAD_NUM9;
    else if (KEYPAD_C4)
        result = KEYPAD_DIVISION;
```

برای توضیح الگوریتم سوال، معادله زیر را در نظر بگیرید.

$$\text{number}[0] \pm number[1] \times\div number[2]$$

ما در هر مرحله از خواندن ورودی مقداری که تا الآن داریم را به صورت معادله بالا نگه میداریم. در ابتدا که چیزی وارد نشده عدد ما برابر با صفر است و به صورت زیر نگه داشته میشود.

$$0 + 1 * 0$$

سپس به این صورت به پیش میرویم که کاربر اگر عددی اضافه کرد، number[2] را آپدیت میکنیم. به این صورت که مقدار آن را ضرب در ۱۰ کرده و عدد را به آن اضافه میکنیم.

```
} else {
    if (equal_enter == 1) {
        lcd_clear();
    }
    equal_enter = 0;
    operation_enter = 0;
    lcd_putchar(key_res);

    number[2] *= 10;
    number[2] += key_res - 48;

    //lcd_clear();
    //sprintf(buffer, "%.2f %c %.
    //lcd_puts(buffer);
}
```

اگر کاربر یکی از عملگرهای + یا = را وارد کرد کل عدد را در number[0] جمع کرده و علامت بعد از آن را چیزی که کاربر وارد کرده قرار میدهیم. دلیل این کار این است که عملگر های + و − اولویت پایینی دارند و در بقیه عملیات های آینده تاثیری ندارند.

```
} else {
    number[1] = calculate(number[1], number[2], operation[1]);
    number[2] = 0;
    number[0] = calculate(number[0], number[1], operation[0]);
    number[1] = 1;
    operation[0] = key_res;
    operation[1] = KEYPAD_MULTIPLICATION;
}
```

در صورتی که کاربر عملگر های ضرب و تقسیم را استفاده کرد، کل مقدار number[0] و number[1] را در number[2] میریزیم و اوپریشت بعد از آن را برابر عملگر وارد شده قرار میدهیم. دلیل این کار این است که ضرب و تقسیم اولویت بالاتری دارند و نباید با اعداد جمع یا تفریق شده قبل یکی شوند.

```
if (key_res == KEYPAD_DIVISION || key_res == KEYPAD_MULTIPLICATION) {
    number[1] = calculate(number[1], number[2], operation[1]);
    number[2] = 0;
    operation[1] = key_res;
```

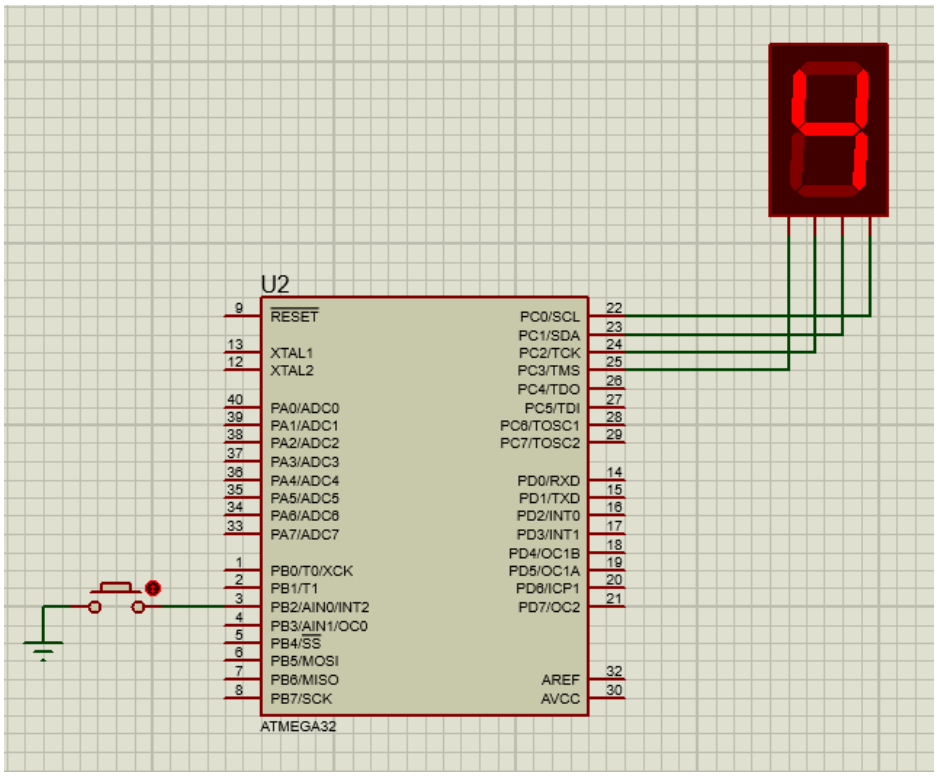
اگر کاربر مساوی را وارد کرد معادله ذخیره شده را محاسبه کرده و number و operation را ریست میکنیم.

```
} else if (key_res == KEYPAD_EQUAL) {
    if (operation_enter != 1 && equal_enter != 1) {
        number[1] = calculate(number[1], number[2], operation[1]);
        number[2] = 0;
        number[0] = calculate(number[0], number[1], operation[0]);
        number[1] = 0;

        lcd_gotoxy(0, 1);
        sprintf(buffer, "%.2f", number[0]);
        lcd_puts(buffer);

        number[0] = 0;
        number[1] = 1;
        number[2] = 0;
        operation[0] = KEYPAD_ADDITION;
        operation[1] = KEYPAD_MULTIPLICATION;
        operation_enter = 0;
        equal_enter = 1;
    }
```

اگر هم دکمه AC را زد، ال سی دی را پاک کرده و متغیر ها را ریست میکنیم.

```
if (key_res == KEYPAD_AC) {
    lcd_clear();

    number[0] = 0;
    number[1] = 1;
    number[2] = 0;
    operation[0] = KEYPAD_ADDITION;
    operation[1] = KEYPAD_MULTIPLICATION;
    operation_enter = 0;
    equal_enter = 1;
```

۳. برای مدار این سوال یک سون سگمت تکی را به پورت های کم ارزش c وصل کرده که بتوانیم یک تا نه را نمایش دهیم و یک کلید برای اینتراپت دو قرار داده که در صورت فشردن یک بتوانیم از تابع اینتراپت استفاده کنیم.



برای کد ابتدا دو متغییر counter و pause را نگه میداریم که counter مقدار شمارنده و pause وضعیت شمارنده را برای ما نگه میدارد.

```
int counter = 0;
int pause = 0;
```

برای کد، ابتدا یک تابع اینتراپت مربوط به timer مینویسیم. این تابع هر یک ثانیه یک بار زده میشود. در هر بار صدا زدن، در صورت پاوز نبودن شمارنده آن را یکی زیاد کرده و باقی مانده به ۱۰ میگیریم زیرا شمارنده ما ۰ تا ۹ است.

```
// Timer1 output compare A interrupt service routine
interrupt [TIM1_COMPA] void timer1_compa_isr(void) {
    if (!pause) {
        counter++;
        counter %= 10;
    }
}
```

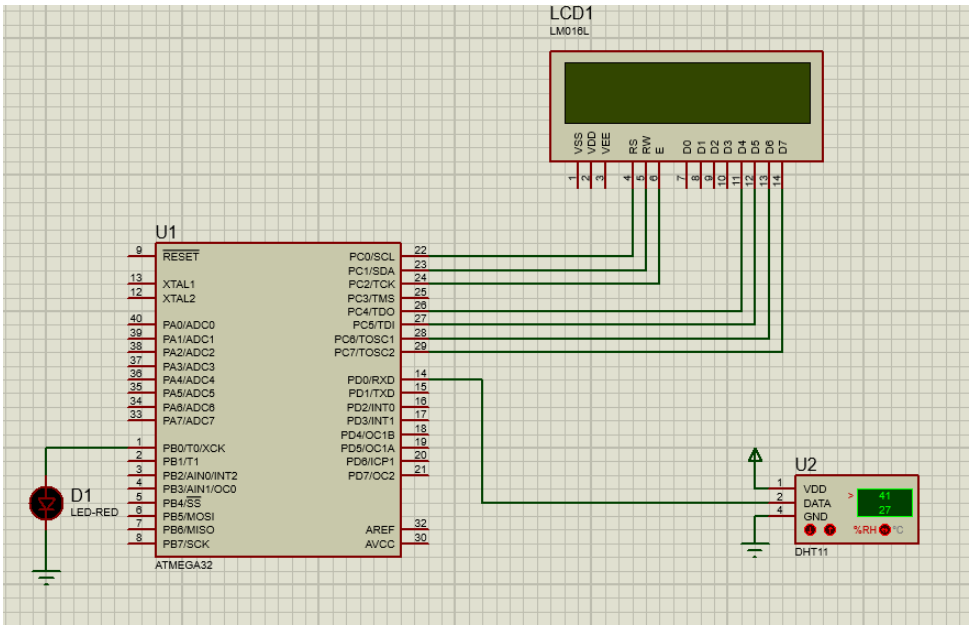
همچنین اینتراپت مربوط به int2 را پیاده سازی کرده که در هر بار صدا زدن آن، pause را تغییر وضعیت میدهیم.

```
interrupt [EXT_INT2] void ext_int2_isr(void) {
    pause ^= 1;
}
```

در تابع main نیز به غیر از تنظیمات اینتراپت های ست شده، فقط کافیست که متغییر counter را روی پورت های C تنظیم کنیم.

```
while (1) {
    PORTC = counter;
}
```

۴. برای مدار این سوال یک ال سی دی که با پورت C کار میکند، یک قطعه DHT11 که به پورت اول D متصل است و یک ال ایدی که به پورت یک B متصل است داریم.



برای کار با قطعه DHT11 لازم است که ابتدا یک پالس ریکوعست فرستاده و سپس یک ریسپانس در یافت کنیم. سپس با استفاده از تابع read_dht11 به ترتیب اعداد مربوط به رطوبت و دما را دریافت کنیم. همچنین به همراه این ورودی ها، یک ورودی checksum نیز فرستاده میشود که بتوان صحت ورودی ها را چک کرد.

```
request();
response();

hum_int = read_dht11();
hum_dec = read_dht11();
temp_int = read_dht11();
temp_dec = read_dht11();
check_sum = read_dht11();

if(hum_int + hum_dec + temp_int + temp_dec - check_sum)
    continue;
```

در ادامه بر روی LCD مقدار رطوبت داده شده را نمایش داده و در صورت بیرون بودن از رنج خواسته شده، پورت B که مربوط به LCD بود را روشن میکنیم.

```
lcd_gotoxy(0, 0);
sprintf(buffer, "%d%% ", hum_int);
lcd_puts(buffer);

if(hum_int > 40 && hum_int < 60)
    PORTB.0=0;
else
    PORTB.0=1;
```