ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E
MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	8
EXPERIMENT DATE	:	11.01.2021
LAB SESSION	:	MONDAY - 13.30
GROUP NO	:	G10

GROUP MEMBERS:

150180103	:	EGEMEN GÜLSERLİLER
150180028	:	ABDÜLKADİR PAZAR
150180716	:	AHMET SELÇUK TUNÇER

AUTUMN 2020

Contents

1 INTRODUCTION

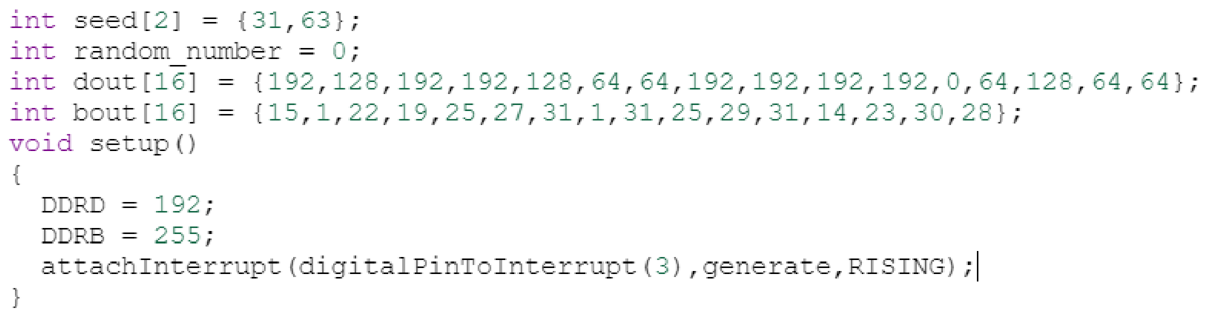
In this experiment, a random number generator is implemented with C language. In the first part, random numbers are generated according to Additive Lagged Fibonacci Generator approach and displayed on 7-segment display. In the second part, Random Number Generator method from Part 1 and and dynamically memory allocation methods are used.

2 MATERIALS AND METHODS

This experiment is conducted via using Arduino Uno Board and 1 seven segment display. This board is programmed using Tinkercad IDE according to desired tasks on the experiment handout.

2.1 Part 1

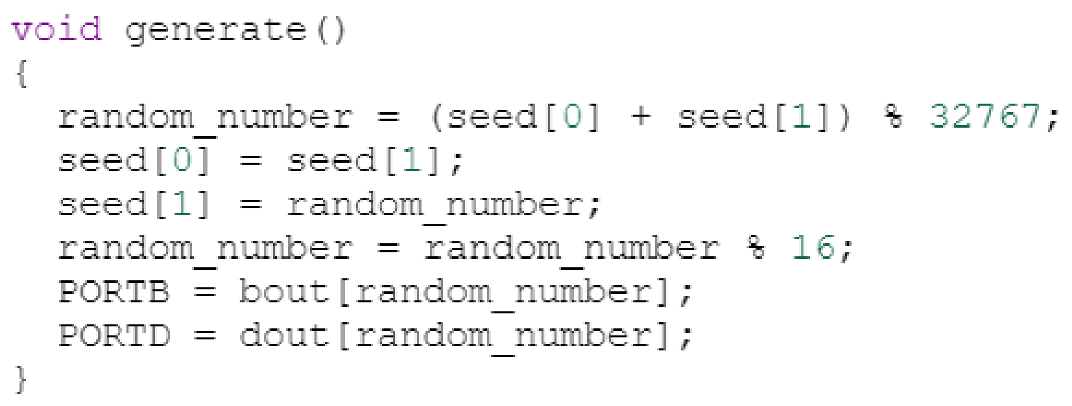
In the first part of the experiment, a random number generator that uses Additive Lagged Fibonacci Generator(ALFG) approach is designed. As shown in the Figure 1, at first a seed array with elements 31 and 63 and random_number is defined in order to implement ALFG method. Also k value is set to 2 for this experiment. After that dout, bout arrays are declared for 7-segment display operations. Data directions are set as needed and pin 3 is used for interrupts.



```
int seed[2] = {31,63};
int random_number = 0;
int dout[16] = {192,128,192,192,128,64,64,192,192,192,192,0,64,128,64,64};
int bout[16] = {15,1,22,19,25,27,31,1,31,25,29,31,14,23,30,28};
void setup()
{
  DDRD = 192;
  DDRB = 255;
  attachInterrupt(digitalPinToInterrupt(3),generate,RISING);
}
```

Figure 1: Code for part 1

As shown in the figure 2, generate() function is used to generate random numbers. As ALFG algorithm suggest desired sequence $S_n = S_{n-j} + S_{n-k}(mod m)$, after proper operations the random number is displayed on the 7-segment display.Mod 16 required to keep the value in range between 0-16.



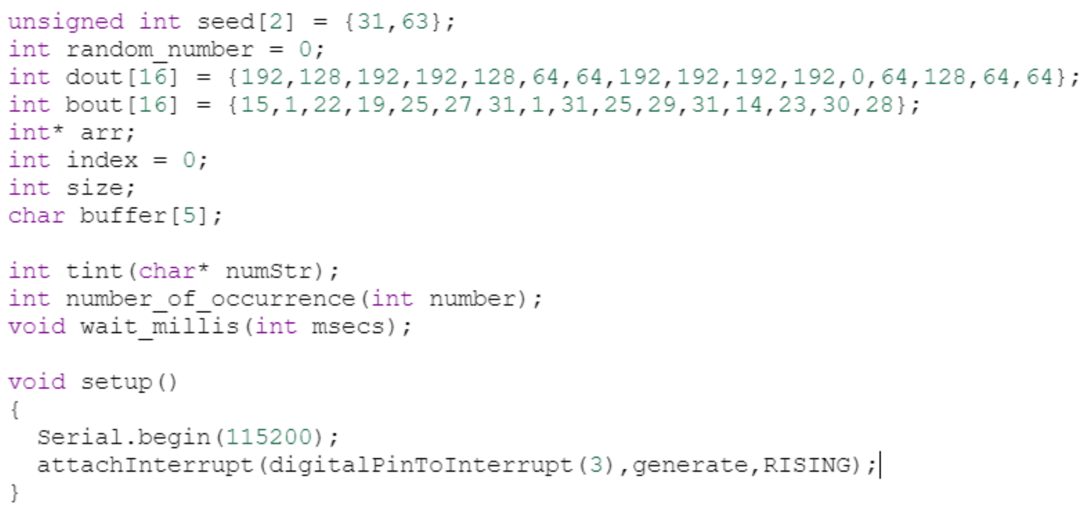
```
void generate()
{
  random_number = (seed[0] + seed[1]) % 32767;
  seed[0] = seed[1];
  seed[1] = random_number;
  random_number = random_number % 16;
  PORTB = bout[random_number];
  PORTD = dout[random_number];
}
```

Figure 2: Code for part 1

2.2 Part 2

In this part we used Random Number Generator(RNG) implemented in Part 1 and by using user interrupt to allocate memory to create an array with size of given m value and generated m random value.

As shown in Figure 3, at first we declared necessary functions tint(), wait_millis(), number_of_occurence() and variables and arrays for further use.



```
unsigned int seed[2] = {31,63};
int random_number = 0;
int dout[16] = {192,128,192,192,128,64,64,192,192,192,192,0,64,128,64,64};
int bout[16] = {15,1,22,19,25,27,31,1,31,25,29,31,14,23,30,28};
int* arr;
int index = 0;
int size;
char buffer[5];

int tint(char* numStr);
int number_of_occurrence(int number);
void wait_millis(int msecs);

void setup()
{
  Serial.begin(115200);
  attachInterrupt(digitalPinToInterrupt(3),generate,RISING);
}
```

Figure 3: Code for part 2

As shown in Figure 4, generate() function is used for taking user input, generating random values and displaying them on the serial monitor. At first, user is informed with a prompt and user input read to the buffer array. Then buffer value turned to the integer value and memory is allocated for an array of correct size and array is filled with random values using while loop. Random values created with ALFG approach. Finally, how many times each number was produced is calculated with number_of_occurrence() function and displayed on the serial monitor.

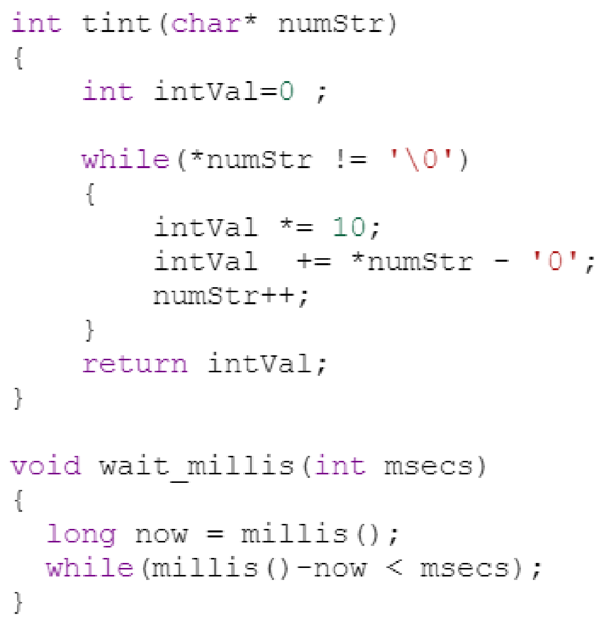
```
void generate()
{
  sei();
  int i = 0;
  Serial.println("Please enter a m value:");
  while(1)
  {
     if(Serial.available())
     {
       wait_millis(10);
       while(Serial.available() > 0)
       {
         buffer[i] = Serial.read();
         i++;
       }
       break;
     }
  }
  size = tint(buffer);
  arr = (int*) malloc(sizeof(int) * size);
  while(index < size)
  {
    random_number = (seed[0] + seed[1]) % 32767;
    seed[0] = seed[1];
    seed[1] = random_number;
    random_number = random_number % 8;
    arr[index] = random_number;
    index++;
  }
```

Figure 4: First part of generate() function

As shown in the Figure 5, number_of_occurrrence() function is used to calculate how many times were produced from which number in a for loop. As seen in Figure 6, tint() function is used to convert char value to int. wait_millis() function is used to creating delays.

```
int number_of_occurrence(int number)
{
  int count = 0;
  for(int i = 0; i < size; i++)
  {
    if(arr[i] == number) count++;
  }
  return count;
}
```

Figure 5: number_of_occurrrence() function



```
int tint(char* numStr)
{
    int intVal=0 ;

    while(*numStr != '\0')
    {
        intVal *= 10;
        intVal  += *numStr - '0';
        numStr++;
    }
    return intVal;
}

void wait_millis(int msecs)
{
  long now = millis();
  while(millis()-now < msecs);
}
```

Figure 6: tint() and wait_millis() function

3 RESULTS

3.1 Part 1

In the first part of the experiment random values has generated with ALFG approach and displayed on the 7-segment display.

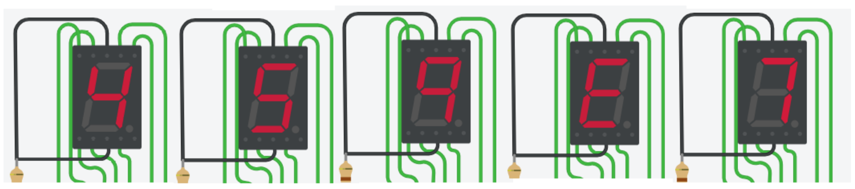
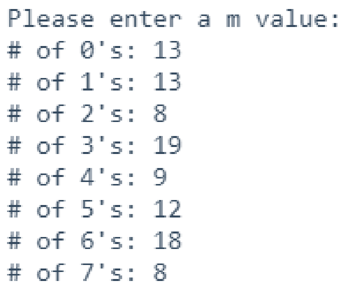


Figure 7: Random value sequence for Part 1

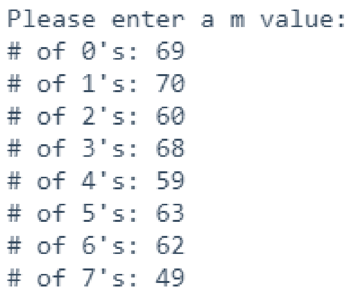
3.2 Part 2

In the second part of the experiment, m value has taken from user and m times random value created and distribution of values depending on generation count has displayed on Serial Monitor.



```
Please enter a m value:
# of 0's: 13
# of 1's: 13
# of 2's: 8
# of 3's: 19
# of 4's: 9
# of 5's: 12
# of 6's: 18
# of 7's: 8
```

Figure 8: Output of program when m=100



```
Please enter a m value:
# of 0's: 69
# of 1's: 70
# of 2's: 60
# of 3's: 68
# of 4's: 59
# of 5's: 63
# of 6's: 62
# of 7's: 49
```

Figure 9: Output of program when m=500

4 DISCUSSION

At the end of the experiment, desired tasks are performed successfully. Since ALFG algorithm was easy enough to understand easily team quickly finished the first part. And in the second part, since we used generator from first part we only need to handle memory allocation and displaying method, but they also did not take too much time.

5 CONCLUSION

This experiment was quite straightforward one. Since we learned how to handle interrupt handling and how 7-segment displays work in the previous experiment we did not faced any difficulties. At the end of the experiment we finished all 8 experiments and at the end of the term, we have gained a lot of experience and theoretical information about microcomputer topic.