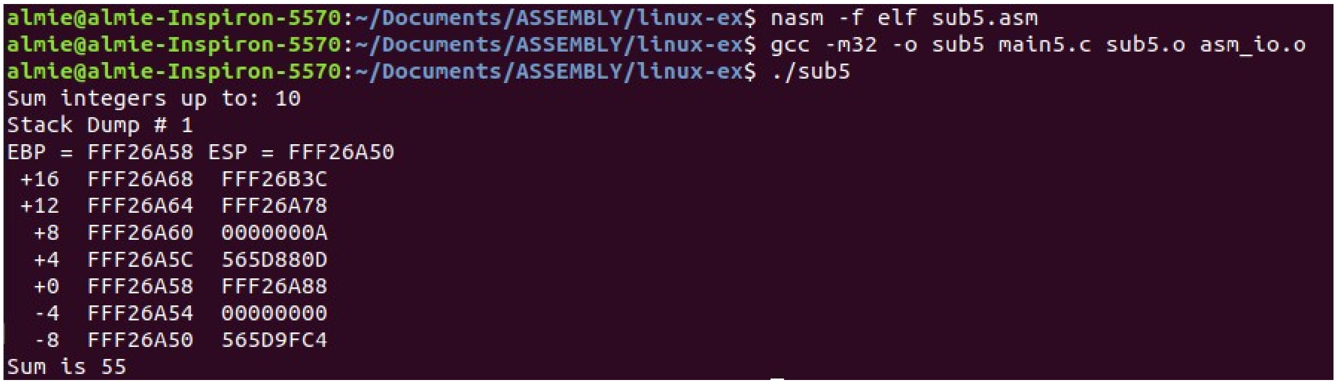
Laboratory Exercise #8

Reading

- Read Section 4.7 of Paul Carter's PC Assembly Book

Practice Exercise:

- Execute “sub5.asm” and interface it with “main5.c”.



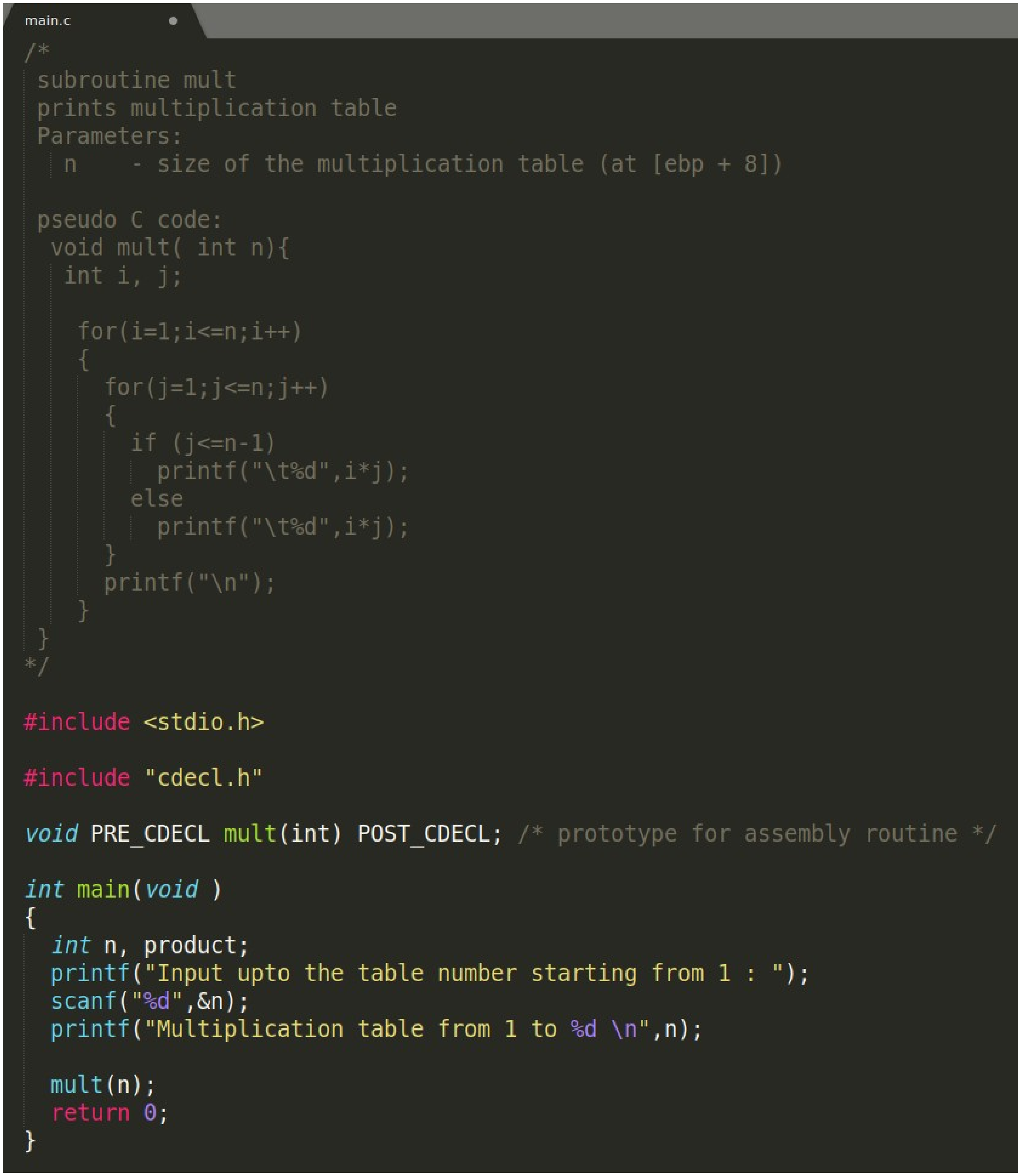
```
almie@almie-Inspiron-5570:~/Documents/ASSEMBLY/linux-ex$ nasm -f elf sub5.asm
almie@almie-Inspiron-5570:~/Documents/ASSEMBLY/linux-ex$ gcc -m32 -o sub5 main5.c sub5.o asm_io.o
almie@almie-Inspiron-5570:~/Documents/ASSEMBLY/linux-ex$ ./sub5
Sum integers up to: 10
Stack Dump # 1
EBP = FFF26A58 ESP = FFF26A50
 +16  FFF26A68  FFF26B3C
 +12  FFF26A64  FFF26A78
  +8  FFF26A60  0000000A
  +4  FFF26A5C  565D880D
  +0  FFF26A58  FFF26A88
  -4  FFF26A54  00000000
  -8  FFF26A50  565D9FC4
Sum is 55
```

- Analyze the sample code (sub5.asm and main5.c). Reflective questions:

What is the function of sub5.asm? What is the function of main5.c? Explain the output of sub5.asm implementing stack.

Problem #8.

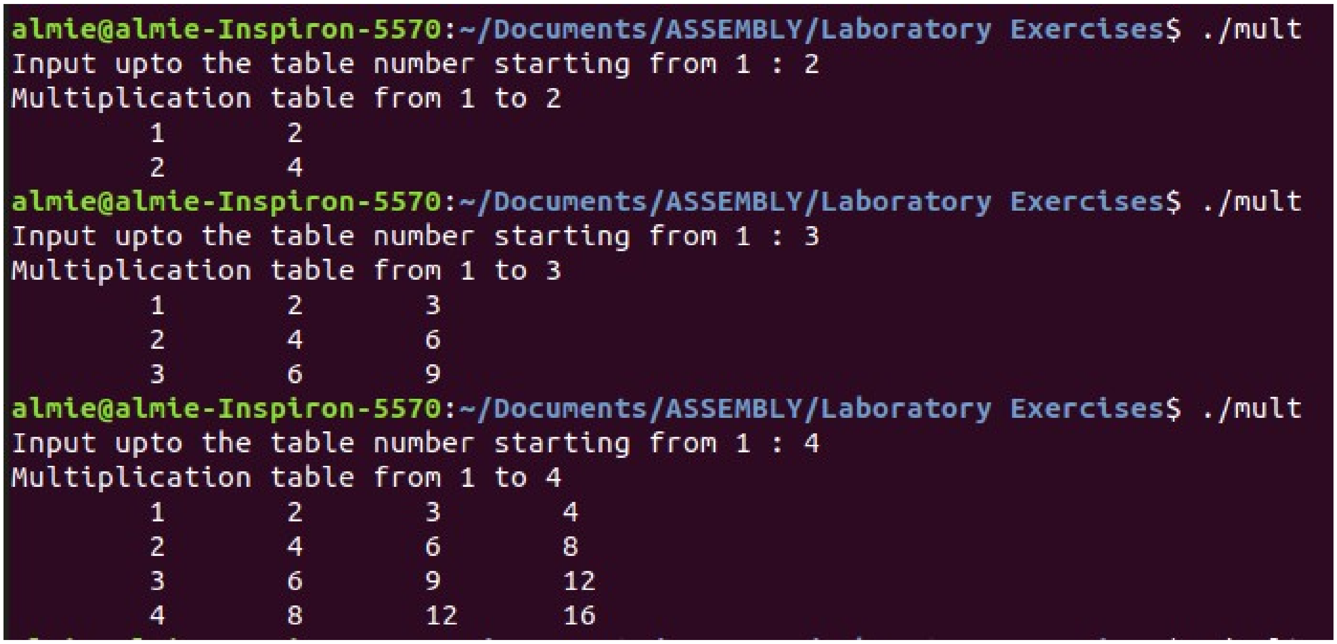
- Write an assembly program that prints the multiplication table.
- Below is the code snippet in high level language (C language) named “**main.c**”.



```c
/*
 subroutine mult
 prints multiplication table
 Parameters:
   n    - size of the multiplication table (at [ebp + 8])

 pseudo C code:
  void mult( int n){
   int i, j;

    for(i=1;i<=n;i++)
    {
      for(j=1;j<=n;j++)
      {
        if (j<=n-1)
          printf("\t%d",i*j);
        else
          printf("\t%d",i*j);
      }
      printf("\n");
    }
 }
*/

#include <stdio.h>

#include "cdecl.h"

void PRE_CDECL mult(int) POST_CDECL; /* prototype for assembly routine */

int main(void )
{
  int n, product;
  printf("Input upto the table number starting from 1 : ");
  scanf("%d",&n);
  printf("Multiplication table from 1 to %d \n",n);

  mult(n);
  return 0;
}
```

- Create a “mult.asm” (computes for each product) that interface with “main.c”.

- The output of your program is something like this:



```
almie@almie-Inspiron-5570:~/Documents/ASSEMBLY/Laboratory Exercises$ ./mult
Input upto the table number starting from 1 : 2
Multiplication table from 1 to 2
        1       2
        2       4
almie@almie-Inspiron-5570:~/Documents/ASSEMBLY/Laboratory Exercises$ ./mult
Input upto the table number starting from 1 : 3
Multiplication table from 1 to 3
        1       2       3
        2       4       6
        3       6       9
almie@almie-Inspiron-5570:~/Documents/ASSEMBLY/Laboratory Exercises$ ./mult
Input upto the table number starting from 1 : 4
Multiplication table from 1 to 4
        1       2       3       4
        2       4       6       8
        3       6       9       12
        4       8       12      16
```

- A good programming practice is to write comments on important line of codes for readability and documentation.

Note: Take a screen record of your working code and make sure to record a video explaining each line of your code as well as showing the correct output of your code. Use screen recorder application in Ubuntu (https://itsfoss.com/best-linux-screen-recorders/) or Windows (https://atomisystems.com/screencasting/record-screen-windows-10/)

Deadline :____

Rubric for Programming Exercises				
Program (50 pts)	**Excellent**	**Good**	**Fair**	**Poor**
Program Execution	Program executes correctly with no syntax or runtime errors (9-10)	Program executes with minor (easily fixed) error (4-8)	Program executes with a major (not easily fixed) error (2-3)	Program does not execute (0-1)
Correct Output	Program displays correct output with no errors (9-10)	Output has minor errors (6-8)	Output has multiple errors (3-5)	Output is incorrect (0-2)

Design of Output	Program displays more than expected (7-8)	Program displays minimally expected output (5-6)	Program does not display the required output (3-4)	Output is poorly designed (0-2)
Design of Logic	Program is logically well-designed (9-10)	Program has slight logic errors that do not significantly affect the results (6-8)	Program has significant logic errors (3-5)	Program is incorrect (0-2)
Standards	Program is stylistically well designed (6-7)	Few inappropriate design choices (i.e., poor variable names, improper indentation) (4-5)	Several inappropriate design choices (i.e., poor variable names, improper indentation) (2-3)	Program is poorly written (0-1)
Documentation	Program is well-documented (5)	Missing one required comment (4)	Missing two or more required comments (2-3)	Most or all documentation missing (0-1)