

# COMSC 260

Fall 2020

## Programming Assignment 9

Worth 15 points (1.5% of your grade)

**DUE: Tuesday, 11/3/20 by 11:59 P.M. on Canvas**

**START by downloading the 260\_assign9.asm file from Canvas**

**NOTE:** Your submission for this assignment should be a single **.asm** file. The following naming convention should be used for naming your file:

**firstname\_lastname\_260\_assign9.asm**. For example, if your first name is “James” and your last name is “Smith”, then your file should be named James\_Smith\_260\_assign9.asm

**COMMENTS (worth 7.5% of your programming assignment grade):** Your program should have at least **ten (10)** different detailed comments explaining the different parts of your program. Each individual comment should be, at a minimum, a short sentence explaining a particular part of your code. You should make each comment as detailed as necessary to fully explain your code. You should also number each of your comments (i.e., comment 1, comment 2, etc.). **NOTE: My comments do NOT count towards the ten comments!**

You will not be submitting sample runs for this assignment, as the program is always run with fixed values (i.e. 7 for DAYS, 24 for HOURS, 60 for MINUTES, and 60 for SECONDS).

For this programming assignment you will be writing an assembly program that will use **nested looping** to write all of the seconds in a week one-by-one to a location in memory. To start with, look at the following code from the **date\_time.cpp** C++ program:

```
#include <iostream>
using namespace std;

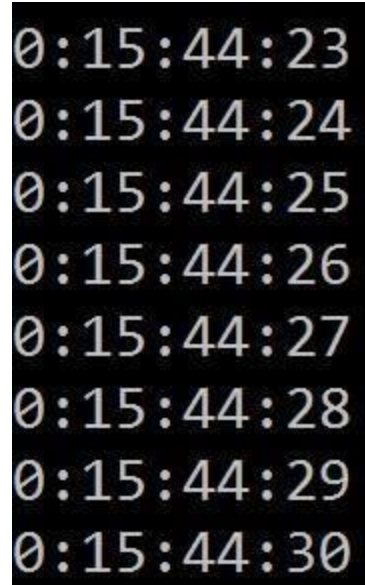
int main()
{
    int count=0;

    for(int day=0; day<7; day++)
        for(int hour=0; hour<24; hour++)
            for(int min=0; min<60; min++)
                for(int sec=0; sec<60; sec++)
                {
                    cout<<day<<":"<<hour<<":"<<min<<":"<<sec<<endl;
                    count++;
                }

    cout<<"count: "<<count<<endl<<endl;

    system("PAUSE");
    return 0;
}
```

This program will display all of the seconds in a week in the following format: day:hour::minute:second. For example, here are some lines of output in the program:



0:15:44:23  
0:15:44:24  
0:15:44:25  
0:15:44:26  
0:15:44:27  
0:15:44:28  
0:15:44:29  
0:15:44:30

There are a total of  $7 * 24 * 60 * 60 = \mathbf{604,800}$  iterations:

0:0:0:0 (day 0, hour 0, minute 0, second 0) – this is the first date/time value

0:0:0:1 (day 0, hour 0, minute 0, second 1)

.....

....

0:0:1:0 (day 0, hour 0, minute 1, second 0)

...

....

0:1:0:0 (day 0, hour 1, minute 0, second 0)

...

...

1:0:0:0 (day 1, hour 0, minute 0, second 0)

...

...

6:23:59:59 (day 6, hour 23, minute 59, second 59) – this is the final date/time value

```
6:23:59:50
6:23:59:51
6:23:59:52
6:23:59:53
6:23:59:54
6:23:59:55
6:23:59:56
6:23:59:57
6:23:59:58
6:23:59:59
count: 604800

Press any key to continue . . .
```

You will be writing a similar program to this in assembly language. In the **260\_assign9.asm** file you are given the following data:

```
DAYS = 7      ; constant representing the number of days in a week
HOURS = 24    ; constant representing the number of hours in a day
MINUTES = 60  ; constant representing the number of minutes in an hour
SECONDS = 60  ; constant representing the number of seconds in a minute

.data

date_time DWORD ? ; byte 0000 stores the DAY, byte 0001 stores the HOUR, byte 0002 stores the MINUTE, and byte 0003 stores the SECOND
num_of_iters DWORD 0 ; counts the total number of iterations
day BYTE DAYS ; used to restore cl to the current day value
hour BYTE HOURS ; used to restore cl to the current hour value
minute BYTE MINUTES ; used to restore cl to the current minute value
```

## The date\_time variable should store the 604,800 values one-by-one,

starting with 00 00 00 00 and ending with 06 17 3b 3b (which is 06 23 59 59 in decimal):

Address: 0x00404000  
0x00404000 00 00 00 00 (the first value stored in **date\_time**)

Address: 0x00404000  
0x00404000 00 00 00 01 (the second value stored in date\_time) .... (several iterations later)

Address: 0x00404000  
0x00404000 00 00 01 00 (day 00, hour 00, minute 01, second 00)

## PROGRAM OUTPUT 1

... (several iterations later)


Address: 0x00404000  
0x00404000 06 17 3b 3b (this is equivalent to 06 23 59 59 in decimal)

Address	Value
0x00404000 (relative address 0000)	06h
0x00404001 (relative address 0001)	17h
0x00404002 relative address 0002)	3Bh
0x00404003 (relative address 0003)	3Bh

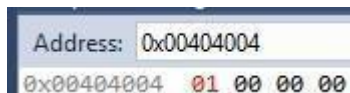
Notice that the final value stored in the `date_time` variable (which starts at address `0x00404000`) is `06 17 3B 3B`, where `06` is the day, `17` is the hour, and `3B` is the minute and the second.

This is because `17 hex = 23 decimal`, and `3B hex = 59 decimal`.

The `num_of_iters` variable (which starts at address `0x00404004`) should be incremented each time the body of the **innermost for loop** (i.e., the loop that iterates through the **seconds**) is executed. The `num_of_iters` variable will start with the value `00 00 00 00`, and after the program finishes running the final value stored in `num_of_iters` should be `80 3A 09 00` (which is equivalent to `604,800` in decimal):



Address: `0x00404004`  
`0x00404004 00 00 00 00` (the first value stored in **num\_of\_iters**)



Address: `0x00404004`  
`0x00404004 01 00 00 00` (the second value stored in `num_of_iters`)

.....

..... (several iterations later)



Address: `0x00404004`  
`0x00404004 45 00 00 00` (where `45 hex = 69 decimal`)

....

..... (several iterations later)

## PROGRAM OUTPUT 2

When the program finishes running, the final value stored in `num_of_iters` should be :

Address: 0x00404004  
0x00404004 80 3a 09 00

Address	Value
0x00404004 (relative address 0004)	80h
0x00404005 (relative address 0005)	3Ah
0x00404006 (relative address 0006)	09h
0x00404007 (relative address 0007)	00h

Where 00093A80 hex = 604,800 decimal

**Your assembly program should mimic the structure of the C++ program.** In

particular, you should have **FOUR (4)** nested loops in your assembly program:

- One loop that iterates through the days
- One loop that iterates through the hours
- One loop that iterates through the minutes
- One loop that iterates through the seconds

You can use any of the instructions we have learned about so far\*:

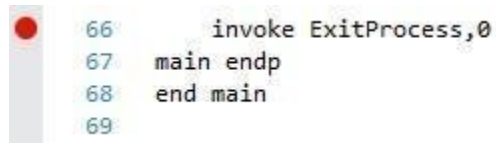
mov  
add  
sub

inc  
dec  
movzx  
movsx  
xchg  
neg

jmp  
loop

\*Think about which instructions would make sense to use

**NOTE:** You probably don't want to manually iterate through the 604,800 iterations! Fortunately, you don't have to do this manually. In Visual Studio you can set a break point before the invoke ExitProcess, 0:



The image shows a snippet of assembly code in a text editor. A red circle, representing a break point, is placed in the left margin next to line 66. The code is as follows:

```
66      invoke ExitProcess,0
67  main endp
68  end main
69
```

Just click in the grey area to the left of the invoke ExitProcess, 0 to set a break point. Then, while you are stepped into the program, from the **debug menu** you can click on step out to take you to the end of the program. If you do this and you have implemented all of the logic correctly, you should see the value

06 17 3B 3B in **date\_time** and the value 80 3A 09 00 in **num\_of\_iters**.