# COMSC 260

# Fall 2020

# Programming Assignment 10

# Worth 15 points (1.5% of your grade)

# DUE: Wednesday, 11/17/20 by 11:59 P.M. on Canvas

**START by downloading the 260_assign10.asm file from Canvas**

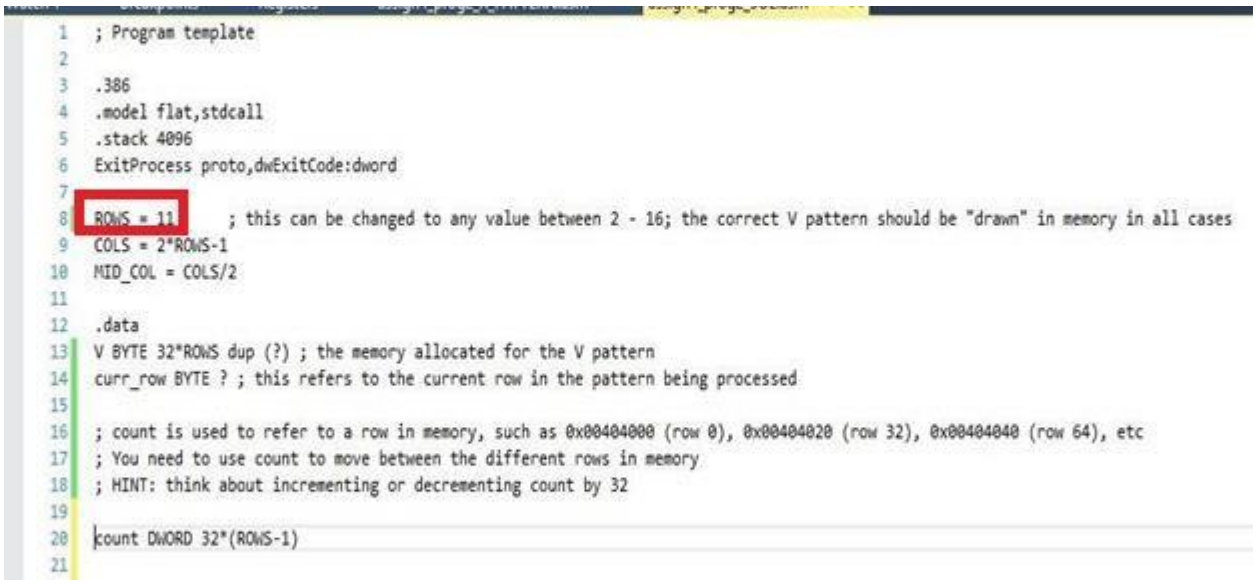
**NOTE:** Your submission for this assignment should be a single **.asm** file and a single **.pdf** file. The following naming convention should be used for naming your files: **firstname_lastname_260_assign10.asm and firstname_lastname_260_assign10.pdf**. The pdf file that you submit should contain the screenshots of your sample runs of the program (see below). For example, if your first name is "James" and your last name is "Smith", then your files should be named James_Smith_260_assign10.asm and James_Smith_260_assign10.pdf.

**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!**

**SAMPLE RUNS (worth 7.5% of your programming assignment grade):** You should submit screenshots of at least **five (5)** different sample runs of your program. Each sample run needs to use different inputs for the ROWS constant, and your sample runs should **NOT** be the same as the sample runs that are used in this write-up for the assignment.

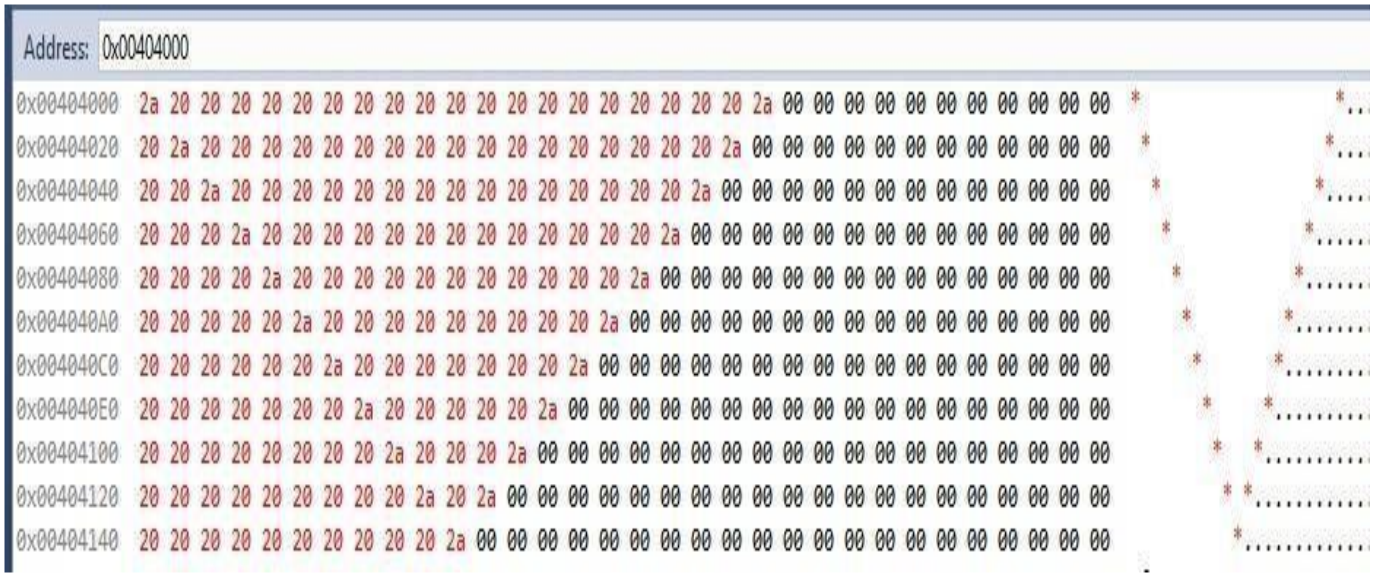
You should also number each of your sample runs (i.e., sample run 1, sample run 2, etc.). All of your sample runs should follow this format – for each individual sample run, screenshot (1) the value used in the ROWS constant and (2) the values in the V pattern (V array) at the end of the program. For example:

(1)



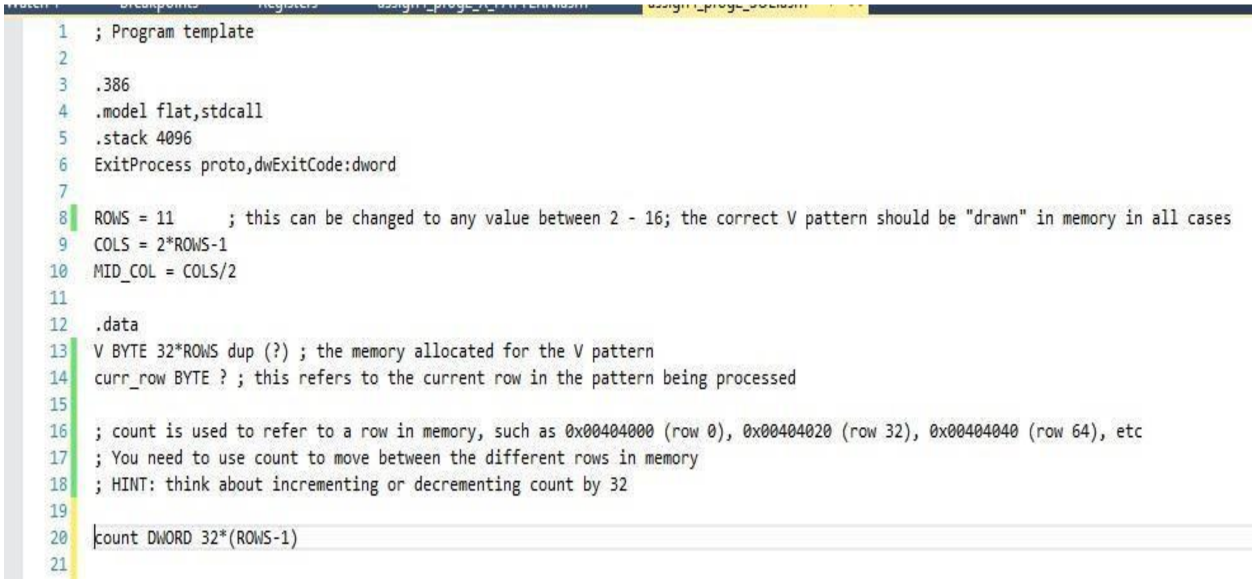
```
; Program template

.386
.model flat,stdcall
.stack 4096
ExitProcess proto,dwExitCode:dword

ROWS = 11      ; this can be changed to any value between 2 - 16; the correct V pattern should be "drawn" in memory in all cases
COLS = 2*ROWS-1
MID_COL = COLS/2

.data
V BYTE 32*ROWS dup (?) ; the memory allocated for the V pattern
curr_row BYTE ? ; this refers to the current row in the pattern being processed

; count is used to refer to a row in memory, such as 0x00404000 (row 0), 0x00404020 (row 32), 0x00404040 (row 64), etc
; You need to use count to move between the different rows in memory
; HINT: think about incrementing or decrementing count by 32

count DWORD 32*(ROWS-1)
```

(2)



```
Address: 0x00404000
0x00404000  2a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00  *                   *...
0x00404020  20 2a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00   *                 *....
0x00404040  20 20 2a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00    *               *.....
0x00404060  20 20 20 2a 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00     *             *......
0x00404080  20 20 20 20 2a 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00      *           *.......
0x004040A0  20 20 20 20 20 2a 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00       *         *........
0x004040C0  20 20 20 20 20 20 2a 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00        *       *.........
0x004040E0  20 20 20 20 20 20 20 2a 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00         *     *..........
0x00404100  20 20 20 20 20 20 20 20 2a 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00          *   *...........
0x00404120  20 20 20 20 20 20 20 20 20 2a 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00           * *............
0x00404140  20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00            *.............
```

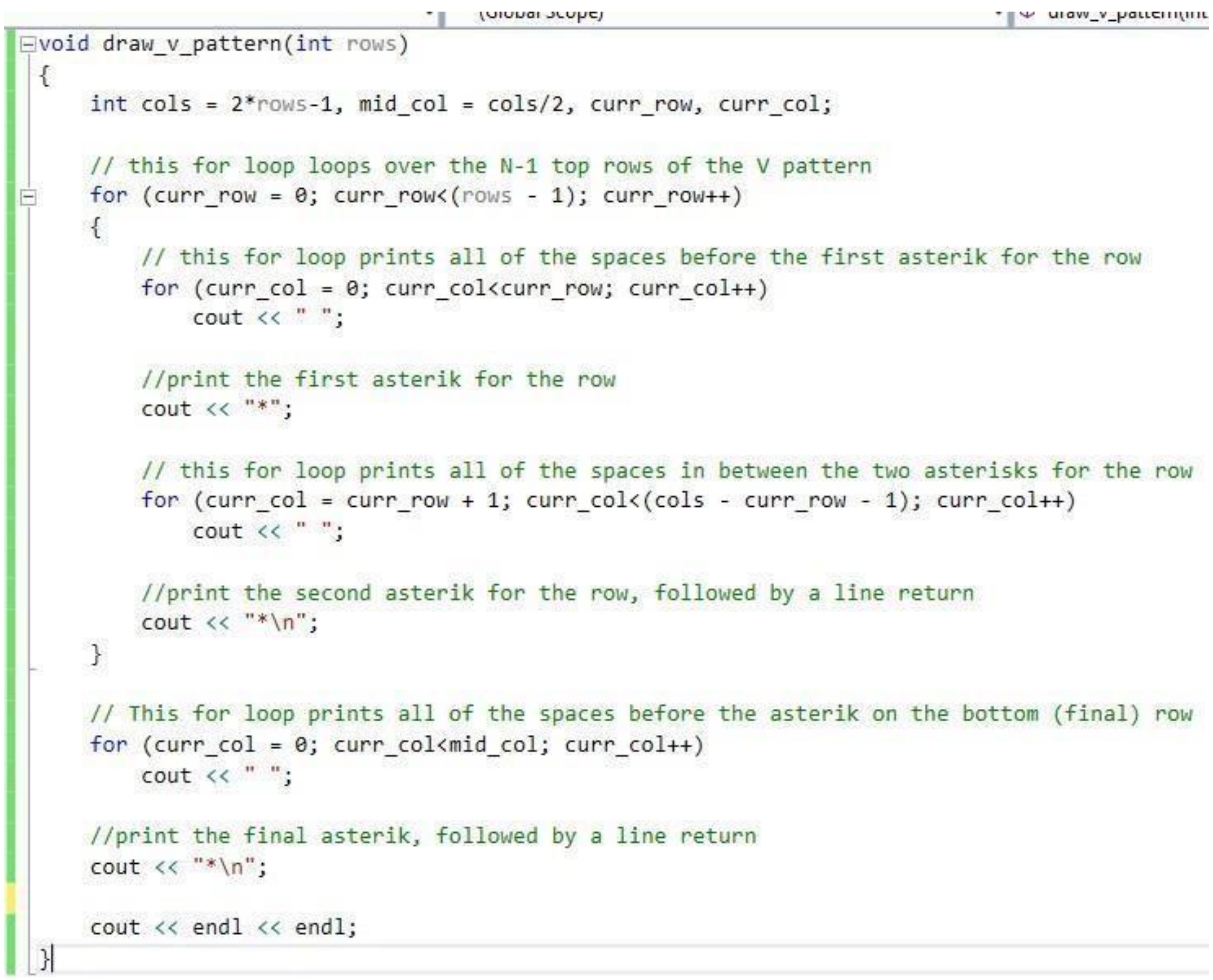
For your tenth programming assignment you will be writing an assembly program that uses a total of four loops (with some of them nested) to “draw” a V pattern in memory. In the data segment you are given the following:



```
; Program template

.386
.model flat,stdcall
.stack 4096
ExitProcess proto,dwExitCode:dword

ROWS = 11      ; this can be changed to any value between 2 - 16; the correct V pattern should be "drawn" in memory in all cases
COLS = 2*ROWS-1
MID_COL = COLS/2

.data
V BYTE 32*ROWS dup (?) ; the memory allocated for the V pattern
curr_row BYTE ? ; this refers to the current row in the pattern being processed

; count is used to refer to a row in memory, such as 0x00404000 (row 0), 0x00404020 (row 32), 0x00404040 (row 64), etc
; You need to use count to move between the different rows in memory
; HINT: think about incrementing or decrementing count by 32

count DWORD 32*(ROWS-1)

```

The idea is that the number of rows can be changed to different values, and the V pattern should be “displayed” in memory correctly in all cases (see the sample runs below).

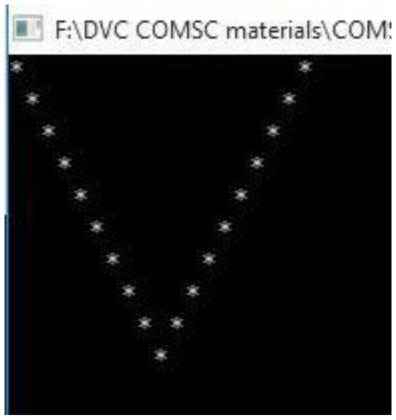
To start with, ***carefully*** study the code for the V Pattern in C++ in the v_pattern.cpp file. You should be comfortable with the C++ code before you start working on your assembly solution.



```
void draw_v_pattern(int rows)
{
    int cols = 2*rows-1, mid_col = cols/2, curr_row, curr_col;

    // this for loop loops over the N-1 top rows of the V pattern
    for (curr_row = 0; curr_row<(rows - 1); curr_row++)
    {
        // this for loop prints all of the spaces before the first asterik for the row
        for (curr_col = 0; curr_col<curr_row; curr_col++)
            cout << " ";

        //print the first asterik for the row
        cout << "*";

        // this for loop prints all of the spaces in between the two asterisks for the row
        for (curr_col = curr_row + 1; curr_col<(cols - curr_row - 1); curr_col++)
            cout << " ";

        //print the second asterik for the row, followed by a line return
        cout << "*\n";
    }

    // This for loop prints all of the spaces before the asterik on the bottom (final) row
    for (curr_col = 0; curr_col<mid_col; curr_col++)
        cout << " ";

    //print the final asterik, followed by a line return
    cout << "*\n";

    cout << endl << endl;
}
```

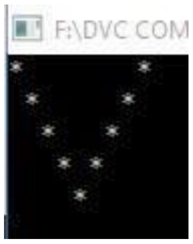
Here are some sample runs of the C++ program:

```
draw_v_pattern(10);
```



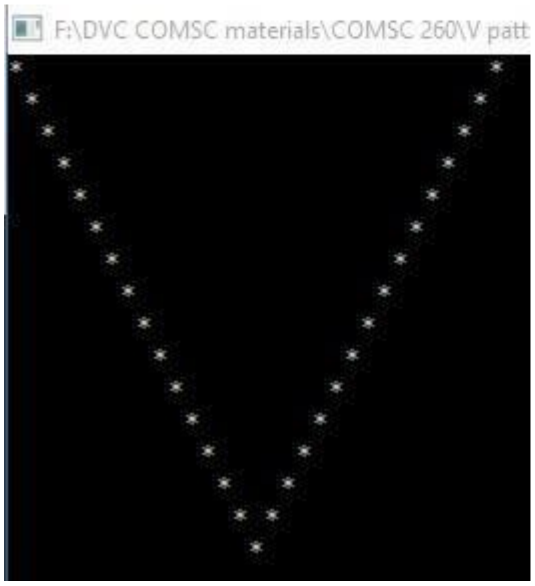
```
*                 *
 *               *
  *             *
   *           *
    *         *
     *       *
      *     *
       *   *
        * *
         *
```

```
draw_v_pattern(5);
```



```
*       *
 *     *
  *   *
   * *
    *
```

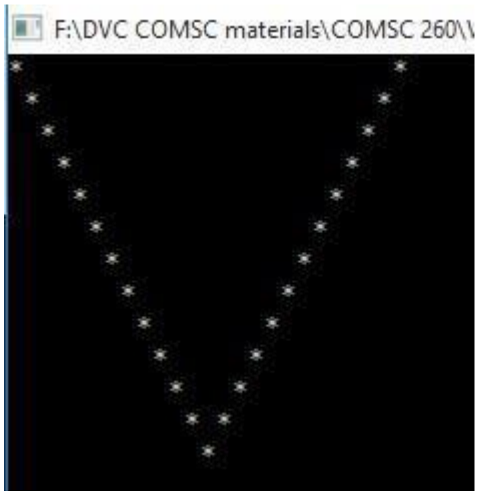
```
draw_v_pattern(16);
```



F:\DVC COMSC materials\COMSC 260\V patt

```
*                             *
 *                           *
  *                         *
   *                       *
    *                     *
     *                   *
      *                 *
       *               *
        *             *
         *           *
          *         *
           *       *
            *     *
             *   *
              * *
               *
```

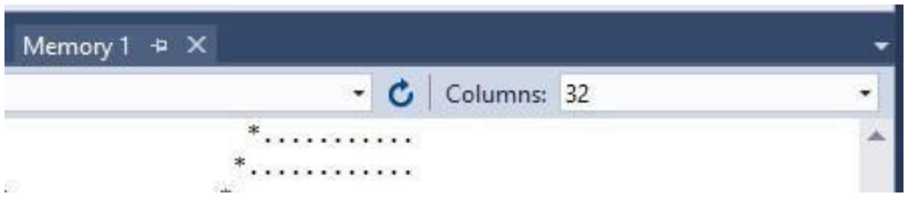
draw_v_pattern(13);



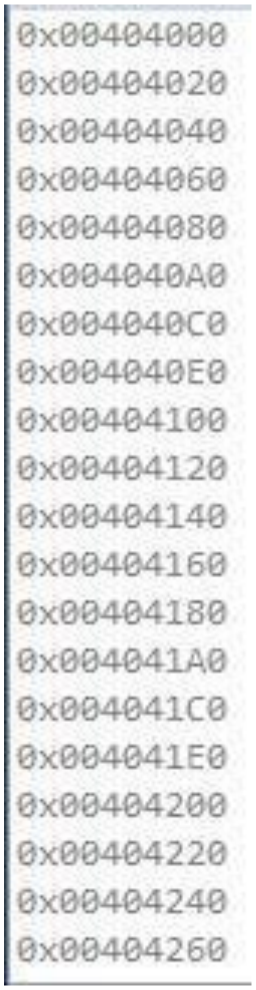
F:\DVC COMSC materials\COMSC 260\\

```
*                       *
 *                     *
  *                   *
   *                 *
    *               *
     *             *
      *           *
       *         *
        *       *
         *     *
          *   *
           * *
            *
```

You will be implementing the same program in x86 assembly language. **<u>The assembly language program should mimic the structure of the C++ program.</u>** Instead of displaying the pattern on the console screen, the pattern will be "drawn" in memory. To start with, on the memory pane in Visual Studio change the columns from **auto** to **<u>32</u>**:



Starting from address 0x00404000, your layout of memory should now look like this:



```
0x00404000
0x00404020
0x00404040
0x00404060
0x00404080
0x004040A0
0x004040C0
0x004040E0
0x00404100
0x00404120
0x00404140
0x00404160
0x00404180
0x004041A0
0x004041C0
0x004041E0
0x00404200
0x00404220
0x00404240
0x00404260
```

Just like the C++ program you should have a total of **four** loops:

(1) One outer loop to loop over the N-1 top rows
(2) One inner loop to print the number of spaces before the first asterisk in the top N-1 rows
(3) One inner loop to print the number of spaces between the first and second asterisk in the top N1 rows
(4) One loop to print the number of spaces before the asterisk in the bottom (final) row

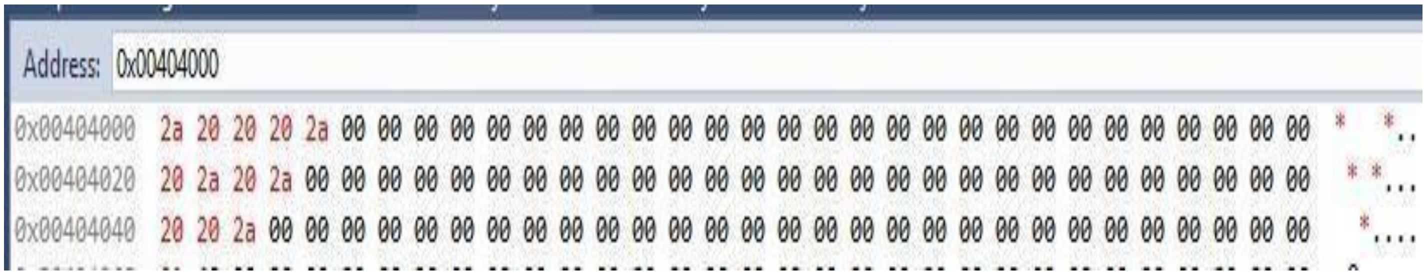
In addition to the above, you also need to:

(1) **Use the runtime stack to switch ECX back and forth between the outer loop and the inner loops**. We saw before that one way that you could save and restore ECX was by using a variable (or another register) to do so, but we can also accomplish this with the runtime stack. In this assignment you need to use the runtime stack to do this.

(2) **Implement the logic for "drawing" the V pattern in memory in a procedure (function) named draw_v_pattern (this logic should NOT be in main!)**. In main you need to call the draw_v_pattern procedure. Make sure to properly initialize the appropriate register(s) in main **BEFORE** the call to the draw_v_pattern procedure. Make sure to also return back to main when the draw_v_pattern procedure is finished!

# Sample Runs of the V pattern Assembly Program:

**Memory layout of V when rows = 3 (and cols = 5)**



```
Address: 0x00404000
0x00404000  2a 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  *   *..
0x00404020  20 2a 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   * *...
0x00404040  20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    *....
```

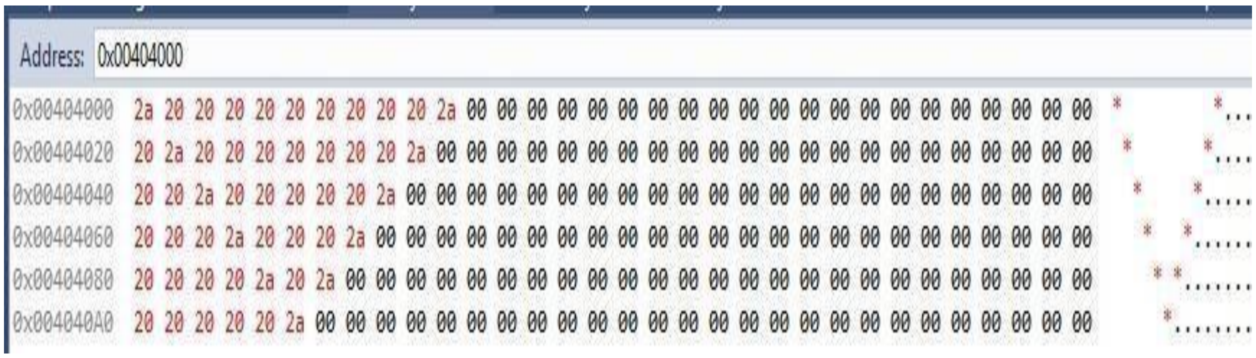
**NOTE:** 2a hex = 42 decimal, which is the ASCII code for an asterisk character (‘*’)

**NOTE:** 20 hex = 32 decimal, which is the ASCII code for a blank space character (‘ ‘)

This memory layout corresponds to the following diagram:

| | COLUMN 0 | COLUMN 1 | COLUMN 2 | COLUMN 3 | COLUMN 4 |
|---|---|---|---|---|---|
| ROW 0 | * | | | | * |
| ROW 1 | | * | | * | |
| ROW 2 | | | * | | |

**Memory layout of V when rows = 6 (and cols = 11)**



```
Address: 0x00404000

0x00404000  2a 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  *         *...
0x00404020  20 2a 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   *       *....
0x00404040  20 20 2a 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    *     *.....
0x00404060  20 20 20 2a 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00     *   *......
0x00404080  20 20 20 20 2a 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00      * *.......
0x004040A0  20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00       *........
```

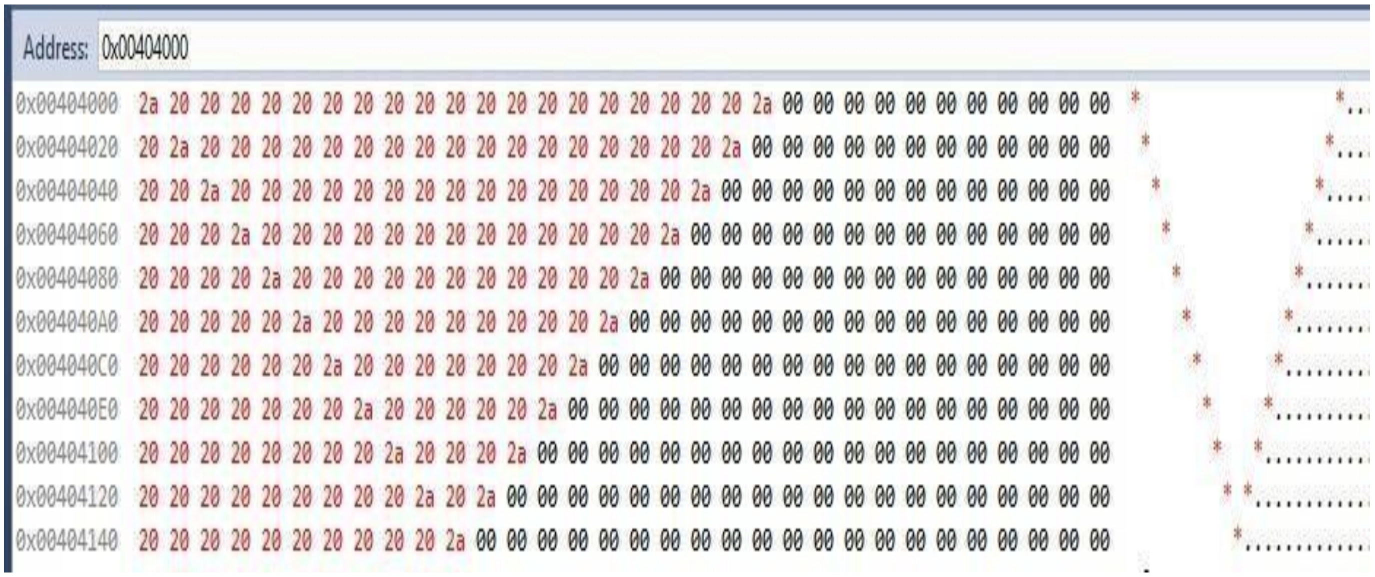
This memory layout corresponds to the following diagram:

| | **COL 0** | **COL 1** | **COL 2** | **COL 3** | **COL 4** | **COL 5** | **COL 6** | **COL 7** | **COL 8** | **COL 9** | **COL 10** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **ROW 0** | * | | | | | | | | | | * |
| **ROW 1** | | * | | | | | | | | * | |
| **ROW 2** | | | * | | | | | | * | | |
| **ROW 3** | | | | * | | | | * | | | |
| **ROW 4** | | | | | * | | * | | | | |
| **ROW 5** | | | | | | * | | | | | |

**Top N-1 rows, each with two asteriks**

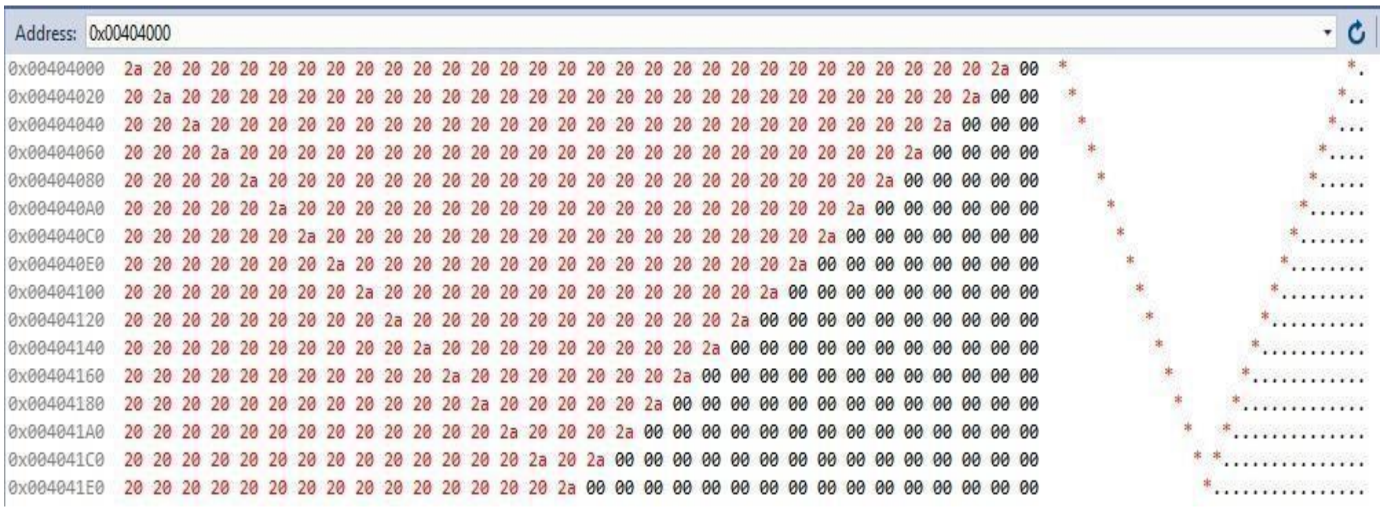
**Bottom row with only one asterik**

**Memory layout of V when rows = 11 (and cols = 21)**



```
Address: 0x00404000

0x00404000  2a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00  *                   *..
0x00404020  20 2a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00   *                 *...
0x00404040  20 20 2a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00    *               *....
0x00404060  20 20 20 2a 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00     *             *.....
0x00404080  20 20 20 20 2a 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00      *           *......
0x004040A0  20 20 20 20 20 2a 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00       *         *.......
0x004040C0  20 20 20 20 20 20 2a 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00        *       *........
0x004040E0  20 20 20 20 20 20 20 2a 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00         *     *.........
0x00404100  20 20 20 20 20 20 20 20 2a 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00          *   *..........
0x00404120  20 20 20 20 20 20 20 20 20 2a 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00           * *...........
0x00404140  20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00            *............
```

**Memory layout of V when rows = 16 (and cols = 31)***



```
Address: 0x00404000

0x00404000  2a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00  *                             *.
0x00404020  20 2a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00   *                           *..
0x00404040  20 20 2a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00    *                         *...
0x00404060  20 20 20 2a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00     *                       *....
0x00404080  20 20 20 20 2a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00      *                     *.....
0x004040A0  20 20 20 20 20 2a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00       *                   *......
0x004040C0  20 20 20 20 20 20 2a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00        *                 *.......
0x004040E0  20 20 20 20 20 20 20 2a 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00         *               *........
0x00404100  20 20 20 20 20 20 20 20 2a 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00          *             *.........
0x00404120  20 20 20 20 20 20 20 20 20 2a 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00           *           *..........
0x00404140  20 20 20 20 20 20 20 20 20 20 2a 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00            *         *...........
0x00404160  20 20 20 20 20 20 20 20 20 20 20 2a 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00             *       *............
0x00404180  20 20 20 20 20 20 20 20 20 20 20 20 2a 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00              *     *.............
0x004041A0  20 20 20 20 20 20 20 20 20 20 20 20 20 2a 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00               *   *..............
0x004041C0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00                * *...............
0x004041E0  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2a 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00                 *................
```

*This is the largest number of rows that can be handled when the number of columns is set to 32 in Visual Studio