Northwestern University
EECS 205 Winter 2016
Assignment 1: Stars

Due: Thursday January 14, 2016 at 11:50pm (submit via Canvas)

This quarter you will implement a single player action-oriented video game in IA32 assembly language. The programming assignments will require you to build small models of the game including graphics, game play, and sound which will all be put together by the end of the quarter. This and all the remaining assignments are to be completed independently.

Because the programming assignments build on each other, you will need completed code from the previous assignment before beginning the subsequent one. To avoid having you fall into a hole if you fail to complete an assignment, the course staff will work with you to get you up to speed, but it is your responsibility to contact us if you are having difficulty. **We will not hand out solutions for any of the assignments!**

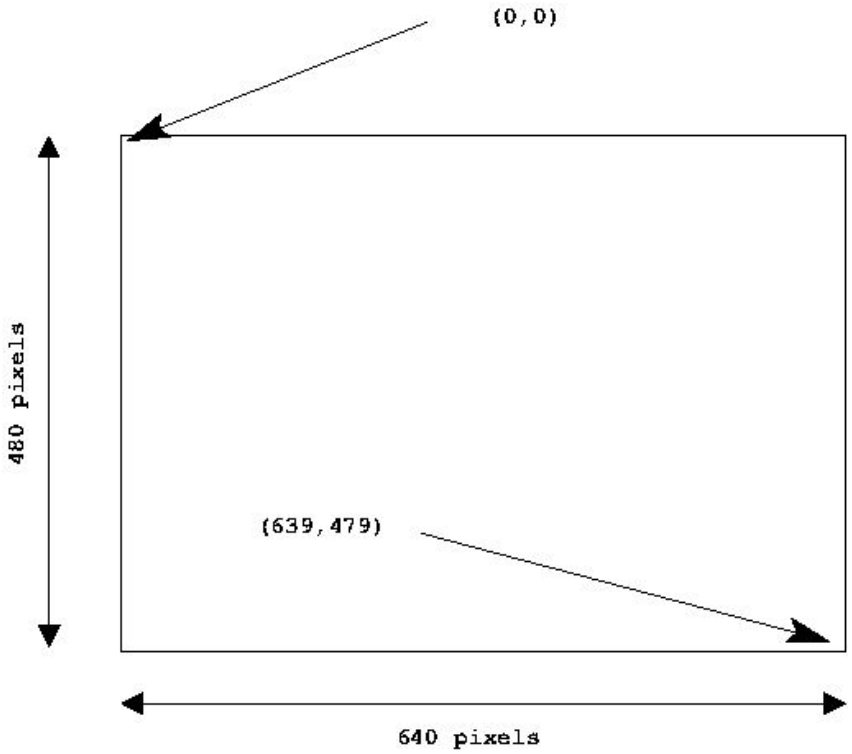
For the first three assignments you will be implementing fairly generic game components. For the remaining two assignments you will be able to make several important design choices and give your game some personality. The assignments will become progressively more challenging, but we believe that everyone is capable of finishing this quarter with working games. The first assignment is meant to introduce you to the development tools and compilation process. It will be uncharacteristically easy. The remaining assignments will be considerably more difficult. For this assignment you will create a starfield, the background for your game. This has a few potential uses going forward: (i) field of play for game set in space (ii) night time background in a platform game. You will also implement a math routine that will prove useful later on in the quarter.

Part One: Starfield

The first part of this assignment is super easy. You will be drawing some stars on a black background. You only need to understand the screen coordinate system and how to use the `invoke` directive -- that's it!

In this class, the graphics environment has a display size of 640x480 pixels. The upper left corner of the display is coordinate (0,0). The lower right corner is (639,479). Note: This orientation differs from the conventional Cartesian coordinates that you probably first encountered in your high school geometry class. You will draw at least a sixteen stars on the background in any pattern you wish, using the `DrawStar` library function that we will provide. Be creative and have fun. You do NOT need to use loops or data structures. You can just draw the stars at hardcoded locations. This assignment is meant to be easy! Just make sure that you

draw at least sixteen stars. You will place your code in a procedure called `DrawStarField`. We've provided a template for you. Just insert your code in the designated location.



`DrawStarfield proto`
Input Parameters: none
Description: Draws a series of stars on the screen. You will implement this procedure.

Part Two: Fixed Point Math

The first part of the assignment was concerned with drawing on the screen using regular integer values. For the second part of the assignment we will introduce from funky math.

Video game programmers often try to avoid using floating-point operations. They instead try to represent numbers whenever possible using fixed point math. For most practical purposes, fixed point math can be much faster than floating point, and x86 PCs are much better equipped to handle these types of operations. So what then is fixed point? Fixed point arithmetic can be really thought of as the "poor man's way of doing fractional math". In essence, you basically use integer operations (e.g add, sub) and datatypes, but pretend that the is a "binary- point" at some fixed position in the number – hence the term fixed point. The programmer usually decides where to put the binary point. For this assignment, we will but the binary point in the middle of a 32-bit signed `DWORD` value. Consequently, the lower 16-bits of a value will hold its fractional component. The upper 16-bits will hold the sign and the non-fractional component.

Using this form of number representation, you can basically think of multiplying or dividing a given number, n by 2^16 to go back and forth between its standard integer representation and

fixed point. When you multiply fixed point numbers you should keep track of the binary point. For instance, if you multiply a regular 32-bit integer value by a fixed point value, the binary point stays in the same place as it was in the fixed point operand. (This is not surprising when you consider for example 54 × 0.02 = 1.08 in base ten.) So remember, if you need to use that result as a regular integer, you will need to truncate the fractional component (e.g. an arithmetic right shift). For this assignment, you will need to write a procedure called `AXP` which takes three fixed point values and returns a fixed point value = A * X + P. Ignore any overflow.

In general, fixed point numbers in this assignment may be positive or negative. The values in other words are signed. Note that the regular `add` and `sub` instructions work the same way for signed/unsigned numbers, but you will need to use the appropriate instructions for multiplication. Hint: To give yourself a start on this assignment, think about how you could multiply two fixed point numbers. Imagine taking a fixed point number in the `ebx` register, multiplying it by a fixed point number in `ecx` and putting the result into eax. What sequence of instructions would you use? Think about shifts...

We have provided a template for your `AXP` routine. Just fill it in. Make sure to place the return value into the `eax` register. Our library code will call AXP with various values. You don't need to call this function yourself, just implement it. Just to be clear, your star drawing using regular integer values and this second part uses only fixed point math.

```
AXP proto a:FXPT, x:FXPT, p:FXPT
```
Input Parameters: a, x, and p are all fixed point values
Description: Computes the value `a * x + p` and returns the result in the eax register as a fixed point value

Library Routines

We will provide startup code and library routines to support your module. The code that you write should call the following library routine using the **`invoke`** directive as discussed in class.

Library routine description follows:

```
DrawStar proto x:DWORD, y:DWORD
```
Input Parameters: x = x-coordinate, y = y-coordinate
Description: Draws a star on the screen at the specified location.

Getting Started

The most difficult part of this assignment might be setting up the development tools! If you purchased the Irvine book with the supplemental CD, follow the instructions for installing MASM. If you are working in the Wilkinson Cluster, those machines should have MASM and Visual C++

installed. If you are working on a NUIT machine, download MASM32 (see links in Course Documents/x86 Assembler Resources) and install it in a local temp directory. You will need to slightly modify the build process. Email the course staff if you need help.

Next download and unpack the assignment files from the courseweb page. You should only need to modify stars.asm. Fill in your own code to draw your stars. You can compile and execute your program from the command line or with the included make.bat script. To use from the command line, enter the directory where your assignment 1 files are kept and type: `make`. If there are no errors, you may run your program by typing: stars. You should also be able to click on the `make.bat` file in the Windows GUI. Pay attention to error messages that appear during compilation/linking. In particular, one common problem that you may encounter is that the linker is unable to write the executable `stars.exe` because the file is already in use -- just kill any running copies of `stars.exe`.

When the program launches, you should see a black background, any stars that you have drawn and some text that you can use to test your AXP procedure. The text presents the parameters (A, X, P) and return value of your AXP procedure. The values are printed as DWORDS in hex, but in reality you should interpret them as fixed point 16/16 (just imagine a binary point in the middle of each hex value). You can use the UP/DOWN keys to change values and TAB to select a different parameter. Thoroughly debug your procedure and when you are confident that you have something working, you can hit SPACE to see an animation that uses your procedure. To exit the program, hit the Escape key.

Submit your assignment on canvas as a single file called `stars.asm`. You should implement the **`DrawStarfield`** and **`AXP`** routines as described above. Comment your code. Make sure that any code that you write for this assignment is in this file and that it correctly compiles using the standard `make.bat` file included with this assignment. Please clearly identify your name in the comments. All programming assignments in this class are to be completed independently. No collaboration!