ECE 109 Program 3: `worm.asm`

Spring 2023

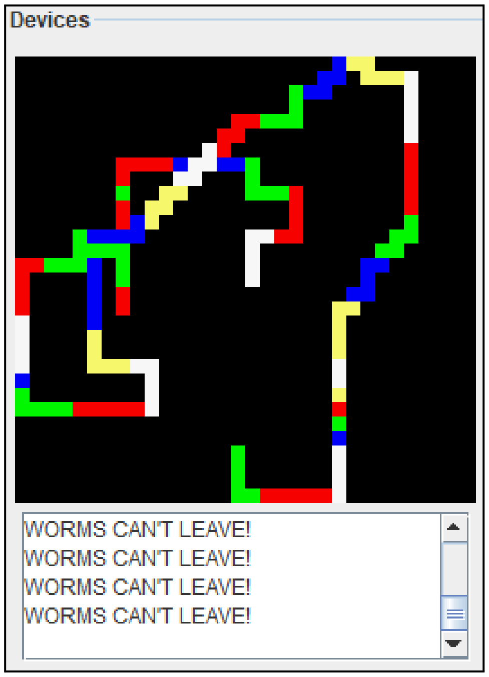
This programming assignment must be completed individually. <u>Do not share your code with or receive code from any other student.</u> The only people who may see your code are the instructor and the ECE 109 TAs.

Evidence of copying or other unauthorized collaboration will be investigated as a potential academic integrity violation. **The minimum penalty for cheating on a programming assignment is a grade of -100 on the assignment.** If you are tempted to copy because you're running late, or don't know what you're doing, you will be better off missing the assignment and taking a zero. Providing your code to someone is cheating, just as much as copying someone else's work.

DO NOT copy code from the Internet, or use programs found online or in textbooks as a "starting point" for your code. Your job is to design and write this program from scratch, on your own. Evidence of using external code from any source will be investigated as a potential academic integrity violation.

Description:

For this assignment, you will write a program that draws lines on the PennSim's graphic display, similar to an Etch A Sketch®. These lines will look like a worm on the screen. The program user will be able to "drag" a point in an up-down or left-right direction to draw a line and will be able to change the color of the line that is being drawn. The worm cannot leave the screen and the worm changes colors as it moves around the screen.



```
WORMS CAN'T LEAVE!
WORMS CAN'T LEAVE!
WORMS CAN'T LEAVE!
WORMS CAN'T LEAVE!
```

The learning objectives for this assignment are:

- Use load and store instructions to manipulate the contents of memory,
- Use I/O routines to allow a user to interact with the program,
- Use subroutine calls to modularize code.

Program Specifications:

The program must start at address x3000.

The program will manipulate the location of a point on the screen, which we will call the Pen. The Pen is a wide-tip, it covers 4 by 4 pixels and appears as a box. The Pen has a color and a location. The screen pixel at the Pen's current location will take on the Pen's color. When the Pen moves, the pixels at the previous location retain its color. In other words, moving the

Pen will draw a wide line of a particular color. An example is shown above where the Pen starts white in the center of the screen and ends as green near the middle bottom of the screen.

The PennSim graphics display (the "screen") is 128 by 124 pixels. We use an (x, y) coordinate system to describe a location on the screen. Location (0, 0) is the top left corner. The x coordinate increases as we move to the right and the y coordinate increases as we move down. In other words, (1, 0) is one pixel to the right of (0, 0) and location (0,1) is one pixel below (0, 0). Location (127, 123) is the bottom right corner of the screen. Our Pen moves 4 pixels at a time, in any direction.

When the program begins, the Pen location must be set to (64, 60) and the color must be white.

If the user attempts to move the Pen past any edge of the screen, the pen will not move. When this happens, the string "WORMS CAN'T LEAVE!" is printed to the console followed by one linefeed. Additionally, every five Pen moves the worm color changes automatically, it will change to the next color in the list from its current color: white, red, green, blue, yellow, white, and so on. The user can manually change the color using r/g/b/y/space. If the user changes the color from green to yellow, then the next color on move 5 is white and not blue. Changing the color with r/g/b/y/space does not reset the move counter. When the user manually changes the Pen color, the current location of the Pen is immediately changed color before anything else is done.

The user interacts with the program using one-character commands. The commands are typed on the keyboard, but not echoed on the console display. Only the boundary string will be printed to the console during program execution. The program will wait for a keystroke, perform the corresponding command, and repeat. If the keystroke does not correspond to a legal command, it will have no effect.

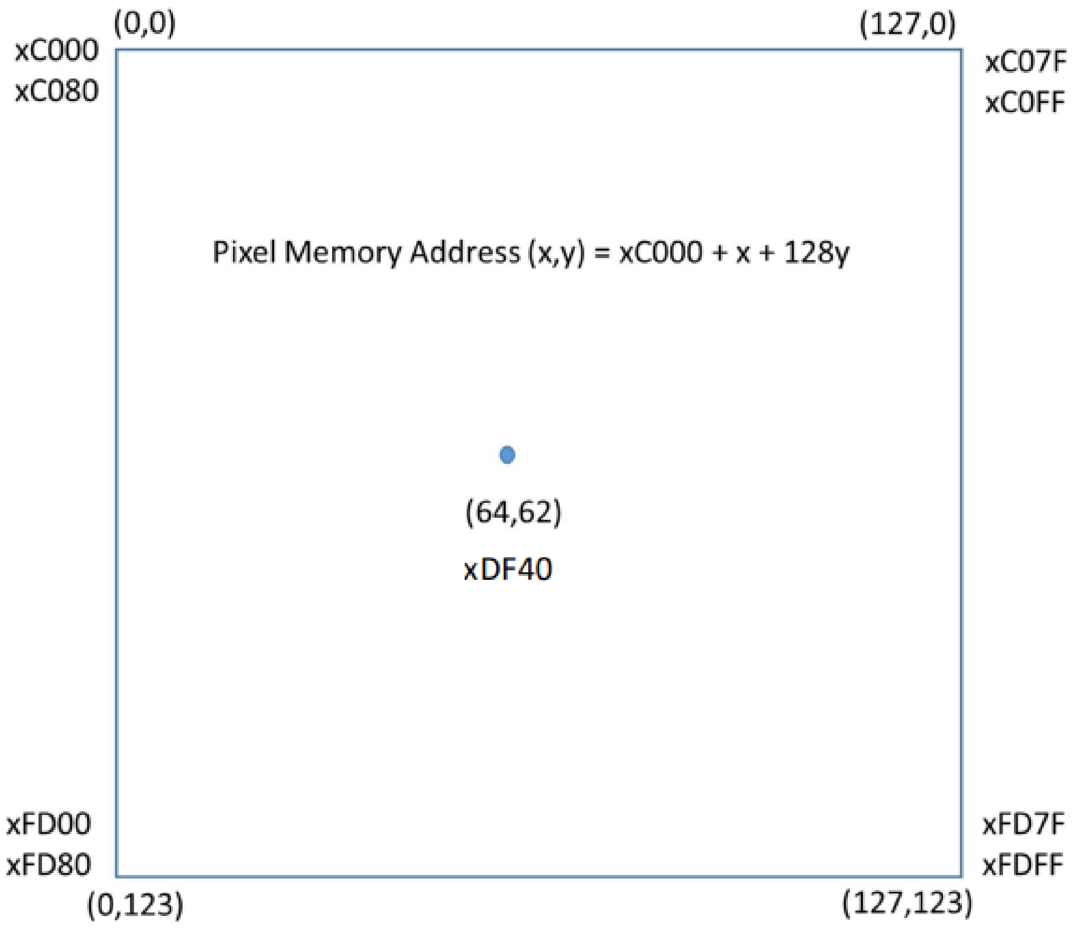
The Pen will be moved around the screen using the WASD scheme of navigation, used by various computer games. As mentioned above certain keys are used to manually change the Pen color. Pressing "enter" will reset the screen, reset the Pen to its initial location and color, and reset the move counter. As usual 'q' is used to quit the program.

The commands are detailed below:

Command Character	**Action**
w	*Move up four pixels.* Pen location changes from (x, y) to (x, y-4). If the Pen is at the top border of the screen, the command has no effect.
a	*Move left four pixels.* Pen location changes from (x, y) to (x-4, y). If the Pen is at the left border of the screen, the command has no effect.
s	*Move down four pixels.* Pen location changes from (x, y) to (x, y+4). If the Pen is at the bottom border of the screen, the command has no effect.
d	*Move right four pixels.* Pen location changes from (x, y) to (x+4, y). If the Pen is at the right border of the screen, the command has no effect.

Command Character	Action
r	Change the color to *red.*
g	Change the color to *green.*
b	Change the color to *blue.*
y	Change the color to *yellow.*
space	Change the color to *white.*
enter	*Clear the screen.* Paint all pixels black and reset the Pen to its initial state.
q	*Quit the program.* The simulated machine must stop running.

Graphics Display Details:



The PennSim graphics display is bit-mapped, meaning that each pixel has a corresponding memory location. The content of that memory location controls the color of the pixel.

Pixel Addresses

Addresses xC000 through xFDFF are assigned to the graphics display. The lower address corresponds to the top left corner (0, 0). Moving one pixel to the right adds one to the address, and it "wraps around" to the next row when it gets to the right edge. Since the display is 128 pixels wide, this means that moving down one pixel is equivalent to adding 128 to the address.

The address of point (x, y) can be calculated as: xC000 + x + 128y.

For this assignment, you will not need to calculate arbitrary pixel addresses, except to figure out where the initial location is. You will be drawing digits that are modeled as rectangles. To draw a rectangle you will need to loop over the width (row) by adding 1 to the address of the point, and then looping over the height (column) by adding a row offset to the address of the point. The offset will be 128 minus the width of the rectangle.

Pixel Color

As mentioned above, the value of the pixel address determines the color of the pixel. The 16 bits contain 5 bits for each RGB component of color: bits [14:10] for red, [9:5] for green, and [4:0] for blue. Bit 15 is ignored. The higher value of a component, the more of that color is present. The table below gives some color values (in hex) that you could use for this program.

Color	Value
Red	x7C00
Green	x03E0
Blue	x001F
Yellow	x7FED
White	x7FFF
Black	x0000

Miscellaneous

For more explanation about the PennSim display, see the PennSim reference manual.

Hints and Suggestions:

- **SAVE R7 AND OTHER REGISTERS BEFORE A SUBROUTINE AND RESTORE R7 AND OTHER REGISTERS AFTER A SUBROUTINE.**
- As always, **design before you code**! Draw a flowchart to organize and document your thinking before you start writing the program. Remember the 80/20 Ratio!!
- Write one section of the program at a time. Then TEST it. Then add the second part. And so on.

- You are required to use five or more subroutines, implement one subroutine at a time, make sure it works and then move on to the next. **If you change a subroutine, always retest it to make sure you did not break it.**
- Use GETC to read a character from the keyboard. Since we're using the TRAP instruction (for GETC, etc.), don't put any useful values in R7.
- There are several different ways to solve this problem. Some ways are easier than others, but the important thing is to solve the problem. The TAs may give you some hints, but you should figure out on your own how to solve this problem.
- *Test your program with a wide variety of inputs*. Test corner cases, such as boundary checks after a reset.
- Use the PennSim simulator and assembler. You must load the file **p3os.obj** into PennSim alongside loading your user code. There are other simulators and assemblers out there, but there are some differences. Your program will be graded using PennSim, and no other tools will be used or considered.
- **Work incrementally!** Implement one thing at a time. When you have a new working piece completed, save and backup a copy of your program. This way you can restore back to a working copy if you need to.
- Recommended order of work:
    1. Get your main loop working with the 'q' input to quit the program.
    2. Write and test a subroutine that draws the Pen at a user specified location with a specified color.
    3. Write and test a subroutine that can move the Pen 4 pixels in any direction. Implement inputs w/s/a/d in the main loop.
    4. Write and test a subroutine that can change the Pen's color. Implement inputs r/b/g/y/space in the main loop.
    5. Write and test a subroutine that clears the screen and resets the Pen. You will need to reset the Pen's location and color.
    6. Write and test a subroutine that can check for the screen boundaries. This subroutine will need to be tested in conjunction with the subroutine that moves the Pen. Once completed add this subroutine call to the w/s/a/d keypresses.
    7. Write and test a subroutine that can check a move counter and change the Pen color appropriately. Once completed add this subroutine call to the w/s/a/d keypresses. Edit the clear screen subroutine to also clear the move counter.
- Note: You can re-use subroutines and call subroutines from other subroutines. For example, the subroutine in step 7 might call the same subroutine that keypresses r/g/b/y/space use for changing the pen color.

Administrative Information:

Updates or clarifications on Moodle:

Any corrections or clarifications to this program spec will be posted on Moodle, using the discussion forum. It is important that you read these postings, so that your program will match the updated specification. (I strongly recommend that you "subscribe" to that forum, so that you get an email for every posting.)

What to turn in:

Assembly Language Source file – it must be named **worm.asm**. Submit via **Moodle**. DO NOT submit .obj or .sym files. Do not submit a .txt file, a .doc file, or anything like that. It must be a simple text file with the proper .asm extension. If we are unable to open or interpret your file, you will get a zero for the assignment (even if it has the right name!).

The program will be graded by one of the TAs, and your grade will be posted in the Moodle gradebook. You will also get back some information about what you got wrong, and how points were deducted (if any).

Grading criteria:

5 pts: File submitted with the proper name, **worm.asm**. These are FREE POINTS!

10 pts: **The program is complete.** There is code that makes a reasonable attempt to perform every necessary part of the program's function (print the prompt, read the input, print the result). **Assembles with no warnings and no errors using PennSim.** Programs that do not assemble will not be graded any further. For warnings, points will be deducted, but the program will be tested for correctness.

10 pts: **Proper coding style, comments, and header.** Use indentation to easily distinguish labels from opcodes. Leave whitespace between sections of code. Use *useful* comments and *meaningful* labels to make your code more readable. Your file **must** include a header (comments) that includes your name, submission date, and a description of the program. Don't cut-and-paste the description from the program spec – that's plagiarism. Describe the program in your own words.

75 pts: The program **performs all functions correctly:**

(30 pts)	WASD movement (7.5 pts each - 2.5 pts for boundary, 5 pts for working).
(20 pts)	Can manually change to each color (4 pts for each color).
(5 pts)	Program contains five or more subroutines.
(5 pts)	Can clear the display and reset the Pen.
(5 pts)	Proper string is printed when the worm hits the screen boundary.
(5 pts)	The worm color changes every five Pen moves.
(5 pts)	Input 'q' halts program.