DESIGNING AN ASTEROIDS VIDEO GAME:

PURPOSE: This lab uses subroutines, keyboard polled i/o, sophisticated program control and is a time intensive program.

WARNING: DO NOT PROCRASTINATE ON THIS LAB. This assignment has been broken into two parts. Beware: it is a relatively large program to write (ours is about 120 lines of assembly code).

BACKGROUND INFORMATION:
In lab #1 you used polled i/o. In this lab, you will use polled i/o to receive input from the keyboard through a simulated 6850 ACIA. For the purposes of this lab, the 6850 is controlled using the two standard interface registers:

(1) terminal status register (TSR) at $10040–READ ONLY;
This register shows the status of the keyboard interface. Bit 0 is set by the hardware, thus, you CANNOT write to this register. When bit 0 =1 a character is in the keyboard data register. Bit 0 resets to 0 immediately after the keyboard data register is read. Note that real i/o takes time and bit 0 will not change immediately in your program.

(2) keyboard data register (KDR) at $10042–READ ONLY;
This register contains the ASCII coded character input from the keyboard.

The terminal status register, at address $10040, is a byte-sized register whose bit zero is set when a character has been typed on the keyboard. The keyboard data register, located at $10042, is a byte sized register that contains the ASCII value of the character that was typed. To make your program run efficiently, you will be putting many operations inside your polling loop.

The goal of this lab assignment is to write a video game program that produces the illusion of animation on the debugger screen. The game will simulate a spaceship flying through a cave, with an asteroid bouncing through the cave toward the spaceship. The spaceship has the capability to fire a bullet at the asteroid. This type of application (a time intensive program doing computer graphics or animation) is one of the few remaining applications where asssembly language has a distinct advantage over other programming languages.

Any time a computer produces animation on the CRT, it is actually showing still images quickly in succession to simulate motion. For instance, in order to simulate the motion of the asteroid through the cave, your program should erase the asteroid in its present position in the debugger screen, update the location of the asteroid, and re-draw it in its new position. Erasing the asteroid is accomplished by printing a space directly on top of it. If this is done quickly enough, it appears to the viewer as though the asteroid has moved.

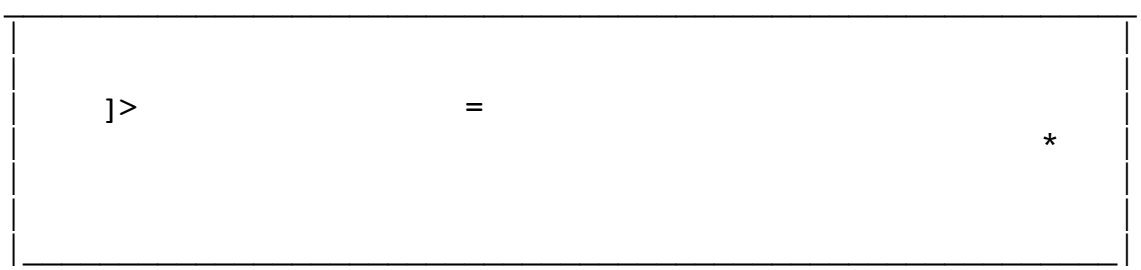
Your program will have to control the animation of three separate objects:

1) The spaceship - This will be able to move up and down in a single column on the debugger screen, and will be controlled using the 'a' and 'z' keys. If the user presses the 'a' key, the spaceship should move up, and if he/she presses the'z' key, the spaceship should move down.

2) The asteroid - This will move from the right side of the debugger window to the left side, toward the spaceship. It will move in a diagonal line, bouncing off the top and bottom of the debugger display window. If it passes the spaceship, it should be erased and a new asteroid should begin moving from the right side of the screen.

3) The bullet - This should move from the nose of the ship toward the right side of the debugger window in a straight line. Only one bullet should be allowed to fire at a time, and when it reaches the right side of the debugger screen, the user should be allowed to fire another bullet.

In this video game, it is not critical to keep score. The following is an example of how the debugger screen might look while your game is running:



The object on the left side of the screen is the spaceship. It only moves up and down. The '=' represents the bullet. It moves from the tip of the spaceship to the right of the debugger window. The '*' is the asteroid, which bounces diagonally toward the spaceship.

LAB ASSIGNMENT:

PART I:
Your program should put the ship on the screen and allow it to move up and down in the debugger window. The program should respond to the following keys:

```
'a': move the spaceship up;
'z' move the spaceship down;
'q' quit the program.
```

PART II:
Your program should be modified to make the asteroid appear and move on the screen. Only one asteroid must appear on the screen. When it passes the ship, it should disappear and a new asteroid should appear at the right side of the debugger window.

It should also be modified to allow firing a bullet. When the user presses the space bar, the bullet should begin traveling from the front of the spaceship toward the right side of the screen. If the bullet and the asteroid collide, both should disappear from the screen and a new asteroid should begin moving from the right side of the screen. It is not critical for you to keep track of a score.

Please Note: The debugger command line to run this lab is as follows:

```
db68k -c game.com <your_prog_name>
```

The lab writeup should follow the normal format and include answers to the following questions.

1. Give an overview of your program structure using a flowchart and/or pseudocode.
2. What operations did you perform using subroutines, and how did you pass parameters to them ?
3. How did you make the asteroid bounce off the top and bottom of the debugger window ?
4. How did you check to see if the bullet and the asteroid collided ?
5. How would you modify your program to allow mutliple bullets to be fired ? You do not need to give code or pseudocode, just mention the additional veriables that would need to be processed by the program.
6. How did you write your polling loop so that the asteroid and the bullet would move while waiting for keyboard input ?

Emulation Code Use
The following is an explanation of some of the parts of the emulation code written for the db68k debugger. In order to make use of these added routines, the first line of your program must be:

```
include  display.s
```

followed by

```
include  io.s
```

In addition, you should run the debugger using the following command:

```
db68k -c game.com <your_file_name>
```

You also need iodorm.com, acia.com and extra.com in your directory as well as display.s and io.s. These files are in /home/courses/282/pub directory.

Using the debugger display macro
The display macro will plot any character inside of the display window at a specified cursor location. To use the macro, do the following:

1. Put the X-coordinate into data register D0
2. Put the Y-coordinate into data register D1
3. Put the ASCII value of the character into data register D2.
4. JSR display

Example:

```
include display.s
MOVE.W  #4,D0
MOVE.W  #3,D1
MOVE.W  #40,D2
JSR     display
```

The above code segment will print the character with ASCII code $40 (@) in the fourth column, third row of the display window.

Using Polled I/O to Input From the Keyboard
The user interfaces to the routines through two byte locations in memory. One location ($10040) is the I/O status register and the other location ($10042) is the I/O receive register. When the user is expecting a byte from the keyboard, he/she should poll the status register. As long as the status register is 0, no byte has been input. As soon as it becomes a 1, there is a byte ready in the receive register. The user should then read the byte from the receive register, which automatically clears the status register.