Tinkering with Tinker

Task 1

Modify the assembler that you wrote such that it produces a program in the same structure as discussed in the lecture about the program abstraction. All code should be placed in an independent segment, and all data are to be placed in an independent segment. Additionally, the assembler should pre-pend the segments with a descriptive header as follows:

```
struct tinker_file_header {
        u_int file_type;             // Currently, 0
        u_int code_seg_begin;        // Address into which the code is to be loaded in memory
        u_int code_seg_size;         // Size of the code segment
        u_int data_seg_begin;        // Address into which the data is to be loaded in memory
        u_int data_seg_size;         // Size of the data segment (could be 0)
};
```

The header should be followed by the code segment and then the data segment. You will need to modify the assembler so that it computes the labels correctly. For the purpose of this exercise, the code segment should start at address 0x2000, and the data segment should start at address 0x10000. Modify the simulator so that it can read the header and load the codes and data segments in the proper locations. Rerun your test cases from the previous two projects.

Task 2

Write in Tinker assembly three programs as follows:

- Matrix multiplication which receives the data as floating-point numbers from the input port. The input data consists of the dimension of the matrix (an integer), then the elements of first matrix row by row, and then the elements of the second matrix row by row. The matrices are square. The program should output the resulting matrix to the output port, row by row.
- Fibonacci numbers. The input consists of the number of Fibonacci numbers to be computed. The program should output the results to the output port.
- A binary search program. The input consists of the number of items, and then the rest of the items. After the input is read, the program works interactively by receiving an input value to be searched, and outputs to port 3 “found” or “not found” (use ASCII presentations and modify the simulator to use ASCII characters on port 3).

Run the programs on the simulator after assembling them in the new format.