Fall 2003 **CMSC 311: Project 2** Kruskal and Scolnik

Due Friday, November 21, 11PM.

This project is based on the maze program of Project 1. You are to write a MIPS assembler program to determine the distance from START to FINISH in a maze. The maze is on an $m \times 32$ grid (where $m \leq 1024$). Some grid points are walls, which are represented by a '1'. The other grid points are open space that can be moved onto, which are represented by a '0'. Distance is measured by moving one grid space north, south, east, or west. But you are not allowed to move into (or through) a wall. You can assume there is a path from START to FINISH.

INPUT:

- SIZE: A positive integer m.
- MAZE: An $m \times 32$ table of bits, where a '1' represents wall and a '0' represents open space. The bits in each row will be stored in a base 10 unsigned number.
- START: An $m \times 32$ table of bits, where a '1' represents the start location and all other bits are '0'. It is represented the same way as MAZE.
- FINISH: An $m \times 32$ table of bits, where a '1' represents the finish location and all other bits are '0'. It is represented the same way as MAZE.

(MAZE, START, and FINISH will each use m words of memory.)

ALGORITHM:

1. {Produce the complement of MAZE; this is the OPEN space that can be moved onto.}
   OPEN $\leftarrow$ not MAZE.
2. NOW $\leftarrow$ START; COUNTER $\leftarrow$ 0;
3. {Check if arrived at FINISH.}
   If NOW & FINISH $\neq$ 0 then print COUNTER and exit;
4. Increment COUNTER;
5. {Find new locations you can reach: For every '1' bit in NOW copy it north, south, east, and west into NEW_NOW. Do this using bit operations on words.)
6. {Make sure you do not walk into a wall.}
   NOW $\leftarrow$ NEW_NOW & OPEN;
7. Go to step 3;

EXTRA CREDIT: For extra credit write your program for general $m \times n$ mazes (for $m, n \leq 1024$ and $32|n$).

NOTES AND HINTS:

1. Some students chose not to follow the algorithm in Project 1. This project will be MUCH easier if you follow the algorithm.

2. It may help to use C as pseudocode for implementing the algorithm. However, in order to program effectively in any new language, you must be able to think in that language. Style counts: Any code which appears to have been mechanically disassembled from a C implementation will lose credit. There are examples of translating C language constructs into MIPS in the lecture notes on the class web page, in the textbook, and in Charles Lin's 311 notes

   (http://www.cs.umd.edu/class/spring2003/cmsc311/Notes/index.html).

3. Comments are a major element of any assember program. They are crucial in explaining to a grader or anyone else what is going on. They are also a big help in maintaining your own sanity in the debugging process. It is not unreasonable to have at least one comment for every line of assembler code. Comments WILL be included in the grading process. At the beginning of your program, you should list the registers associated with the important variables.

4. See the submit instructions on the web page. You MUST submit your program from your 311 account to be sure it can be identified. One student submitted Project 1 from a 330 account.

Sample input files will be provided in the posting account directory

ss311001/Projects/P2.