COMP1521 17s2

Computer System Fundamentals

Assignment 1

Game of Life

Draft

Objectives

- to give you experience writing MIPS assembly code
- to give you experience with data and control structures in MIPS

Admin

Marks	7(towards total course mark) (Note that the marks below add up to XX; they will be scaled to a mark out of 8)
Group?	This assignment is completed **individually**
Due	by 11:59:59pm on Sunday 10th September
Submit	`give cs1521 ass1 life.s` or via Webcms
Late Penalty	0.08 marks per hour late (approx 1.9 marks per day) off the ceiling (e.g. if you are 36 hours late, your maximum possible mark is 4.1/7)

Background

Conway's Game of Life is a mathematical zero-player game whose history and rules are described in Wikipedia. It takes place on an infinite grid of cells, where each cell is either alive or dead. Each cell has eight neighbours: left, right, above, below, and diagonal. On each turn, the whole grid is examined and the content of each cell is changed according to the following rules:

- Any live cell with fewer than two live neighbours dies, as if caused by underpopulation.
- Any live cell with two or three live neighbours lives on to the next generation.
- Any live cell with more than three live neighbours dies, as if by overpopulation.
- Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

In this assignment, you will implement a version of Game of Life in MIPS assembler. It is different from the "standard" version in having a finite board; in this case 10 cells by 10 cells. This means that some cells (on the edges) will hvae fewer than eight neighbours.

The initial board state is hard-wired into the program, but you can (and should) try variations of the initial state by changing the 0's and 1's in the `.byte` directives in the code.

Exercise

The following C code is a working version of the Game of Life. It is not written in a style we would normally use, but is intended to be a little closer to how you would render it in MIPS. You can use it as the basis of your design.

```
#include <stdio.h>
#include <stdlib.h>
```

```

#define N 10

char board[N][N] = {
   {0,0,0,0,0,0,0,0,0,0},
   {0,0,0,0,0,0,0,0,0,0},
   {0,0,0,1,0,0,0,0,0,0},
   {0,0,1,0,1,0,0,0,0,0},
   {0,0,0,0,1,0,0,0,0,0},
   {0,0,0,0,1,1,1,0,0,0},
   {0,0,0,1,0,0,1,0,0,0},
   {0,0,1,0,0,0,0,0,0,0},
   {0,0,0,0,0,0,0,0,0,0},
   {0,0,0,0,0,0,0,0,0,0},
};
char newboard[N][N];

int neighbours(int, int);
void copyBackAndShow();

int main(void)
{
    putchar('>');
    while (getchar() == '\n') {
      for (int i = 0; i < N; i++) {
         for (int j = 0; j < N; j++) {
            int nn = neighbours(i,j);
            if (board[i][j] == 1) {
               if (nn < 2)
                  newboard[i][j] = 0;
               else if (nn ==2 || nn == 3)
                  newboard[i][j] = 1;
               else
                  newboard[i][j] = 0;
            }
            else if (nn == 3)
               newboard[i][j] = 1;
            else
               newboard[i][j] = 0;
         }
      }
      copyBackAndShow();
       putchar('>');
   }
   return 0;
}

int neighbours(int i, int j)
{
    int nn = 0;
   for (int x = -1; x <= 1; x++) {
      for (int y = -1; y <= 1; y++) {
         if (i+x < 0 || i+x > N-1) continue;
         if (j+y < 0 || j+y > N-1) continue;
         if (x == 0 && y == 0) continue;
         if (board[i+x][j+y] == 1) nn++;
        }
    }
   return nn;
}

void copyBackAndShow()
{
   for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
         board[i][j] = newboard[i][j];
         if (board[i][j] == 0)
            putchar(' ');
         else
            putchar('*');
      }
      putchar('\n');
   }
}

```

A template with an initial board state is given below. Save this into the file `life.s` and work from there.

```
# life.s ... Game of Life on a 10x10 grid
#
# Written by <<YOU>>, August 2017

   .data

board:
   .byte 1, 0, 0, 0, 0, 0, 0, 0, 0, 0
   .byte 1, 1, 0, 0, 0, 0, 0, 0, 0, 0
   .byte 0, 0, 0, 1, 0, 0, 0, 0, 0, 0
   .byte 0, 0, 1, 0, 1, 0, 0, 0, 0, 0
   .byte 0, 0, 0, 0, 1, 0, 0, 0, 0, 0
   .byte 0, 0, 0, 0, 1, 1, 1, 0, 0, 0
   .byte 0, 0, 0, 1, 0, 0, 1, 0, 0, 0
   .byte 0, 0, 1, 0, 0, 0, 0, 0, 0, 0
   .byte 0, 0, 1, 0, 0, 0, 0, 0, 0, 0
   .byte 0, 0, 1, 0, 0, 0, 0, 0, 0, 0

newBoard: .space 100
main_ret: .space 4

   .text
   .globl main
main:
   sw   $ra, main_ret

# Your main program code goes here

```

```
end_main:
   lw   $ra, main_ret
   jr   $ra


# The other functions go here

```

You *must* implement the three functions given in the C code: `main()`, `neighbours()`, and `copyBackAndShow()`. You do not need to implement stack-based function calls, but can if you wish.

Things you should not do:

- use a cross-compiler to convert the above C code to MIPS assembler
- copy Victor Torres' or anyone else's solution from an online source like GitHub

The whole point of this exercise is for **you** to better understand how assembly programs are built and how they work. The above two strategies ruin the point of this assignment.

Extensions

(Worth kudos, but no marks)

- allow larger boards, read the initial board setup from a file
- run the game continuously and use additional characters to provide animation

Have fun, *jas*