Final Project Update: The Game of Life

Eric Dilmore, Andrew Julian, Corrin Thompson, and Ian Brown

October 6, 2014

1 Introduction

So far, we have decided on an interface and we have written pseudocode for each of the four sections of our project: file IO, array management, program logic, and interface output.

We will address each of these in turn.

2 Text Interface (as a group)

The grid will be 80×50 characters. Each row will end with a newline. Another line will be used to display the HUD, with stats about the current game. The stats we plan to track are the number of living cells, the number of dead cells, and the current turn. The following several lines is a sample output of one single step. (Prototype output made by Andrew.)

```
   @@ @  @@ @@@@@@@@@@           @ @            @@@@   @@@     @@   @@@     @@
@@  @         @ @        @@     @@@      @   @  @  @     @     @  @ @ @  @@  @
 @  @ @  @@ @@    @@@@           @ @            @@@@   @@@      @ @ @ @ @ @ @@ @
--------------------------------------------------------------------------------
   Number of alive cells: ????     Number of dead cells: ????     Turn: ????
```

Each step should take up 52 lines, and the program will assume that the terminal is 52 lines high.

Each step will take approximately one tenth of a second so that the user can see the results of each step. Eventually, it may be possible to speed up or slow down this rate.

3 Pseudocode

3.1 File IO (Corrin)

This opens a file for reading, reads the starting position from the file, and closes the text file. The values will either be space or @ characters and will be parsed into a boolean grid.

```
.data
# file will contain an mxn grid of @'s and spaces; ' ' = dead, @ = alive
file: .asciiz "fileName" # file to be read from
grid: .byte 0:m*n # array of values from file
# to be stored in row-major order
buffer: .space bufferSize # stores data read

.text
main:
# load array
la $a3, grid # load base address of grid into $a3
move $t0, $zero # load index of array into $t0 (begins at 0)

# open file
li    $v0, 13        # open file syscall
la    $a0, fin       # board file name
li    $a1, 0         # flags
li    $a2, 0    # open for reading
syscall                # open the file
move  $s6, $v0       # save the file descriptor

# read from file
li    $v0, 14        # read from file syscall
move  $a0, $s6       # file descriptor
la    $a1, buffer    # address of buffer to store read data
li    $a2, m*n bytes  # max buffer length
syscall             # read from file

# close file
li   $v0, 16        # close file syscall
move $a0, $s6       # file descriptor to close
syscall             # close file

# setting up array
FOR LOOP:
loops through chars in buffer and inputs them into array ($a3)
```

3.2 Array Management (Ian)

There need to be functions that check adjacent cells to tell whether they are alive or dead. These functions will do address calculations on the current address and put either a one or a zero in $v0.

```
Left Function:
calculate address to left by subtracting 1 byte from current address
load address of the cell to the left in temp register
load life value (1 for alive, 0 for dead) to the left into temp register
return life value

Left-Up Function:
calculate diagonal address by subtracting 1 byte from the current address and subtractin
load left-up address in a temp register
load life value of left-up address into a temp register
return life value

Up Function:
calculate up address by subtracting the line width from current address
load up address in a temp register
load life value of left-up address into a temp register
return life value

Right-Up Function:
calculate diagonal address by adding 1 byte to the current address and subtracting the l
load right-up address in a temp register
load life value of right-up address into a temp register
return life value

Right Function:
calculate address to right by adding 1 byte from current address
load address of the cell to the right in temp register
load life value (1 for alive, 0 for dead) to the right into temp register
return life value

Right-Down Function:
calculate diagonal address by adding 1 byte to the current address and adding the line w
load right-down address in a temp register
load life value of right-down address into a temp register
return life value

Down Function:
calculate down address by the the line width to the current address
load up address in a temp register
load life value of left-up address into a temp register
return life value
```

```
Left-Down Function:
calculate diagonal address by subtracting 1 byte from the current address and adding the
load left-down address in a temp register
load life value of left-down address into a temp register
return life value
```

3.3 Game Logic (Eric)

For every turn, the game must loop through each of the cells and check in the four cardinal directions and the four diagonals to test how many adjacent cells are alive and are dead. Then, the current cell must be turned either on or off based on the number of adjacent cells touching it.

This will be a recursively called function, with the recursion in between when the calculation is done and the grid itself is updated. The calculated value will be stored in an s register, which will be stored on the stack at the beginning of every recursive call and restored to its previous value at the end of that function.

As you recurse down, you must also put the return address on the stack in order to preserve the call stack.

```
store all used s registers to stack
check each direction
increment counter temporary register each time
store in stack:
    return address currently stored in $ra
step current address forward
recursive call
(Recurses to the end)
decrement current address
pull from stack:
    return address
update current character as alive / dead
restore s registers
```

3.4 Interface Output (Andrew)

The output function will display to the console what is described in Section 2. It will loop over every byte in the grid array and translate a boolean dead/alive value to either a space or an @ symbol.

```
Read in word from file at current position
Test to see if word is equal to zero
If so, output " ". Elsewise, output "@"
When the end of a row is reached, jump to the next line
When the 49th line is reached, output a line of "-"
For the 50th row, output
  number alive
  number dead
  turn number
```

4 Outcomes

We have not yet tested our program, as we do not have actual code yet. This should be tested as soon as we learn proper function calls.

5 Changes From Proposal

Nothing has been changed at this stage in the project.

6 Next Steps

6.1 File IO (Corrin)

Corrin will make the file input loop, looping through each character and deciding whether it is on or off. She will also test the loop using MARS.

6.2 Array Management (Ian)

Ian will translate the pseudocode into MIPS assembly code and test it with sample arrays.

6.3 Game Logic (Eric)

Eric will translate his pseudocode into MIPS assembly code. Due to its reliance on array management, he will not test the code directly. However, the recursion will be tested for resiliance.

6.4 Interface Output (Andrew)

Andrew will translate his pseudocode into MIPS assembly code and test it with sample arrays.