CSE 331 PROJECT 1 REPORT

ABDULLAH ENES PATIR

200104004005

```
            .data
msg1:       .asciiz "Enter the number of rows: "
msg2:       .asciiz "Enter the number of columns: "
msg3:       .asciiz "Enter elements of the 2D array as if it's a 1D array: "
msg4:       .asciiz "Number of Seconds: "
msg5:       .asciiz "Step "
grid1:      .space  1024
grid2:      .space  1024
newline:    .asciiz "\n"
```

- I defined messages to be displayed at data section. Grid1 is used for the input that the user will enter in type of space. Grid2 is in the same type, but it is not entered by the user. Both grid1 and grid2 are string. Meaning input grid is entered as a string in one line from the user.

For example, if you want to enter a grid (3 by 3) like this:

```
...
.O.
...
```

You need to enter this as an input: ‘....O....’

Or a grid like this (6 by 7):

```
.......
...O...
....O..
.......
OO.....
OO.....
```

You need to enter this as an input: ‘..........O.......O.........OO.....OO.....’

- In these lines the inputs are entered by the user that are grid1, number of rows, columns, and seconds.

```
Main:
    # display msg1
    la $a0, msg1
    li $v0, 4
    syscall

    # get user input for the number of rows $t0 = number of rows
    li $v0, 5
    syscall
    move $t0, $v0

    # display msg2
    la $a0, msg2
    li $v0, 4
    syscall

    # get user input for the number of columns $t1 = number of columns
    li $v0, 5
    syscall
    move $t1, $v0

    # store length of grid in $t2 = r * c
    mul $t2, $t1, $t0

    # display msg6
    la $a0, msg4
    li $v0, 4
```

```

    # get user input for the number of seconds $s6 = n
    li $v0, 5
    syscall
    move $s6, $v0

    # display msg3
    la $a0, msg3
    li $v0, 4
    syscall

    # get user input for grid1
    li $v0, 8
    la $a0, grid1
    li $a1, 1024
    syscall
```

• At the right, there is the algorithm implemented in C and Mips. Firstly, loop counter and step counter are assigned in $a2 and $a3 and number of steps and grid1 which is entered by the user is getting printed. If the loop counter is equal to number of steps, then branch to exit. Number of steps printed again and grid2 is filled with bombs ‘O’. In printRes subroutine grid2 is printed which full of ‘O’s. In detonateBombs subroutine the ‘O’s that matches in grid1 and in grid2 are blows up and turns itself, left, right, bottom and top into ‘.’ in grid2. At line 68 and 69, loop counter + 1 is stored in $s7 to check if the iteration is at the last one. If so, then branch to cond1. In cond1, number of steps % 2 is checked if equal to zero. If so, then end the program otherwise do as if it was done in any other iterations. In line 70, grid2 is copied into grid1 because grid1 should be changed for the loop to continue. After that some print statements done and loop counter is incremented.

(Note: I removed the comments to fit the image)

```
58  li $a2, 1                       # index = 1
59  li $a3, 1                       # a = 1
60  jal printSteps
61  jal printGrid
62  main_loop:
63      beq $a2,$s6,main_loop_exit
64      jal printSteps
65      jal plantBombs
66      jal printRes
67      jal detonateBombs
68      addi $s7,$a2,1
69      beq $s7, $s6, cond1
70      jal strcpy
71      jal printSteps
72      jal printRes
73      add $a2,$a2,2
74      j main_loop
75  cond1:
76      rem $t3, $s6, 2
77      beqz $t3, main_loop_exit
78      jal strcpy
79      jal printSteps
80      jal printRes
81      li $v0, 10
82      syscall
83  main_loop_exit:
84      li $v0, 10
```

```c
int i = 1;
int a = 2;
printf("Step 1:\n");
printGrid(grid, r, c);
while (i < n)
{
    printf("\nStep %d\n", a++);
    createGrid(r, c, grid, result);

    if (!(n % 2 == 0 && i + 1 == n))
    {
        strcpy(grid, result);
        printf("\nStep %d\n", a++);
        printGrid(result, r, c);
    }

    i += 2;
}
```

• At the right, the same algorithm is implemented in C and Mips. $s0 and $s1 are used to point to grid1 and grid2. $ t3 is used as loop counter. If loop counter is equal to rows X columns then branch to exit. Load byte from grid1 in $t4 and load $t5 with char 'O' if they match branch to case0 otherwise increment counter and point to next elements in grid1 and grid2. In line 140 and 141, the byte at grid2 is turned into '.' but the left right top and bottom remains unchanged, so we need to change them. From line 143 to 147 I made some computations because in C I used comparisons below:

`(i >= c), (i < r * c - c), (i % c > 0),`
`((i + 1) % c != 0)`

From line 149 to 153, is implemented for condition where `(i >= c)`. I used blt because it the opposite of '>=' meaning if the condition i >= c is not true then it will branch to case2. In line 150-151-152-153 '.' byte is loaded into $t6 and it is loaded into $s1's current byte using sb instruction. I used sub in order to point to element at grid2[i-c] and right after storing I used add to neutralize the subtraction operation.

Case2 are very similar to logic above. If `(i < r * c - c)` is true it does as follow otherwise branches to case3. Different from above it stores the '.' byte at grid2[i+c] .

Case 3 checks if `(i % c > 0)` condition is met. If true it stores the '.' byte at grid2[i-1].

Case 4 checks if `((i + 1) % c != 0)`condition is met. If true it stores the '.' byte at grid2[i-1].

Case 5 is the case where the program will end up if it is entered in Case0 to increment loop by 1 and point to next elements in grid1 and grid2.

From line 134 to 137, incrementing and pointing to next is done for the iterations that did not get into case0.

(Note: I removed the comments to fit the image)

```
126 detonateBombs:
127     la $s0, grid1
128     la $s1, grid2
129     li $t3, 0
130     Loop:
131         beq $t3, $t2, Exit
132         lb $t4, 0($s0)
133         li $t5, 'O'
134         beq $t4, $t5,Case0
135         add $s0,$s0,1
136         add $s1,$s1,1
137         add $t3,$t3,1
138         j Loop
139     Case0:
140         li $t6, '.'
141         sb $t6,0($s1)
142         sub $t7, $t2,$t1
143         div $t3,$t1
144         mfhi $s2
145         addi $t6,$t3,1
146         div $t6,$t1
147         mfhi $s3
149         blt  $t3,$t1,Case2
150         li $t6, '.'
151         sub $s1, $s1, $t1
152         sb  $t6, 0($s1)
153         add $s1,$s1,$t1
154
155         Case2:
156         bge  $t3,$t7,Case3
157         li $t6, '.'
158             add $s1,$s1,$t1
159             sb $t6,0($s1)
160             sub $s1, $s1, $t1
161
162         Case3:
163             ble  $s2,$zero,Case4
164             li $t6, '.'
165             sub $s1, $s1, 1
166             sb  $t6, 0($s1)
167             add $s1, $s1, 1
168
169         Case4:
170             beqz $s3,Case5
171             li $t6, '.'
172             add $s1, $s1, 1
173             sb  $t6, 0($s1)
174             sub $s1, $s1, 1
175
176         Case5:
177             add $s0,$s0,1
178         add $s1,$s1,1
179         add $t3,$t3,1
180         j Loop
181     Exit:
182         jr $ra
```

```c
void createGrid(int r, int c,
 char gridAtPreviousStep[r * c],
 char gridAtNextStep[r * c])
{

    for (int i = 0; i < r * c; i++)
    {
        char currentCell = gridAtPreviousStep[i];
        if (currentCell == 'O')
        {
            gridAtNextStep[i] = '.';

            if (i >= c)
            {
                gridAtNextStep[i - c] = '.';
            }
            if (i < r * c - c)
            {
                gridAtNextStep[i + c] = '.';
            }
            if (i % c > 0)
            {
                gridAtNextStep[i - 1] = '.';
            }
            if ((i + 1) % c != 0)
            {
                gridAtNextStep[i + 1] = '.';
            }
        }
    }
}
```

- In plantBombs subroutine, $t4 used as pointer for grid2, $t3 is used as a variable to hold char 'O', $t5 is used as loop counter. I used store byte instruction to fill grid2 at its first index at each iteration with char 'O' and each iteration loop counter incremented by 1 and pointer points to next element. If loop counter is equal to $t2 that holds rows X columns, then branch to exit.

(Note: I removed the comments to fit the image)

```
plantBombs:
    li $t3, 'O'
    la $t4, grid2
    li $t5, 0
    fill:
        beq $t5,$t2,exit
        sb $t3,0($t4)
        addi $t4, $t4,1
        addi $t5,$t5,1
        j fill
    exit:
        jr $ra
```

- In strcpy subroutine, $s0 used as pointer for grid1, $s1 is used as pointer for grid2, $t6 is used as loop counter. If loop counter is equal to $t2 that holds rows X columns, then branch to exit. A byte at first index is loaded into $t5 from grid2 and it is stored in the first index of grid1. Then counter is incremented by 1 and pointers points to next element.

(Note: I removed the comments to fit the image)

```
354 strcpy:
355     la $s0, grid1
356     la $s1, grid2
357     li $t6, 0
358     strcopy_loop:
359         beq $t6, $t2, end
360         lb $t5, 0($s1)
361         sb $t5, 0($s0)
362         addi $s0, $s0, 1
363         addi $s1, $s1, 1
364         add $t6,$t6,1
365         j strcopy_loop
```

- In printSteps subroutine, firstly msg5 is getting printed that is "Step ". Then the number at $a3 gets printed that holds the number of steps. In line 347, step counter is incremented by 1 and a new line gets printed.

(Note: I removed the comments to fit the image)

```
338 printSteps:
339     la $a0,msg5
340     li $v0, 4
341     syscall
342
343     li $v0,1
344     add $a0,$zero,$a3
345     syscall
346
347     add $a3,$a3,1
348
349     la $a0,newline
350     li $v0, 4
351     syscall
352
353     jr $ra
```

- printGrid and printRes are almost the same. printGrid prints the content of grid1, printRes prints the content of grid2.

$s0 and $s1 is used as a pointer to strings. If the loop counter is equal to rows X columns then branch to exit. $t4 is used to be loaded with first byte of grid1 and grid2 and it gets printed after loading. From line 261 to 263 and 286 to 288 is written to compute this in C ((i+1)%c) and in beqz, $t6 is checked if it is equal to zero to print a new line. Because grid1 and grid2 are strings in order to print them as a string we need to consider the number of columns to print new lines.

(Note: I removed the comments to fit the image)

```
252 printGrid:
253     la $s0, grid1
254     li $t3, 0
255     print_loop:
256         beq $t3, $t2, exit_print
257         lb $t4, 0($s0)
258         li $v0, 11
259         move $a0, $t4
260         syscall
261         add $t5, $t3, 1
262         div $t5, $t1
263         mfhi $t6
264         addi $s0, $s0, 1
265         addi $t3, $t3, 1
266         beqz $t6, newLine_Loop
267         j print_loop
268     newLine_Loop:
269         li $v0, 4
270         la $a0, newline
271         syscall
272         j print_loop
273
274     exit_print:
275         jr $ra
276
```

```
277 printRes:
278     la $s1, grid2
279     li $t3, 0
280     loop:
281             beq $t3, $t2, last_exit
282             lb $t4, 0($s1)
283             li $v0, 11
284             move $a0, $t4
285             syscall
286             add $t5, $t3, 1
287             div $t5, $t1
288             mfhi $t6
289             addi $s1, $s1, 1
290             addi $t3, $t3, 1
291             beqz $t6, nl
292             j loop
293             nl:
294                 li $v0, 4
295                 la $a0, newline
296                 syscall
297                 j loop
298             last_exit:
299             jr $ra
```

Sample Input and Outputs:

```
Enter the number of rows: 3
Enter the number of columns: 3
Number of Seconds: 4
Enter elements of the 2D array as if it's a 1D array: ....O....
Step 1
...
.O.
...
Step 2
OOO
OOO
OOO
Step 3
O.O
...
O.O
Step 4
OOO
OOO
OOO

-- program is finished running --
```

```
Enter the number of rows: 3
Enter the number of columns: 3
Number of Seconds: 5
Enter elements of the 2D array as if it's a 1D array: ....O....
Step 1
...
.O.
...
Step 2
OOO
OOO
OOO
Step 3
O.O
...
O.O
Step 4
OOO
OOO
OOO
Step 5
...
.O.
...

-- program is finished running --
```

```
Enter the number of rows: 6
Enter the number of columns: 7
Number of Seconds: 4
Enter elements of the 2D array as if it's a 1D array: ..........O.......O.........OO.....OO.....
Step 1
.......
...O...
....O..
.......
OO.....
OO.....
Step 2
OOOOOOO
OOOOOOO
OOOOOOO
OOOOOOO
OOOOOOO
OOOOOOO
Step 3
OOO.OOO
OO...OO
OOO...O
..OO.OO
...OOOO
...OOOO
Step 4
OOOOOOO
OOOOOOO
OOOOOOO
OOOOOOO
OOOOOOO
OOOOOOO

-- program is finished running --
```

```
Enter the number of rows: 6
Enter the number of columns: 7
Number of Seconds: 5
Enter elements of the 2D array as if it's a 1D array: ..........O.......O.........OO.....OO.....
Step 1
.......
...O...
....O..
.......
OO.....
OO.....
Step 2
OOOOOOO
OOOOOOO
OOOOOOO
OOOOOOO
OOOOOOO
OOOOOOO
Step 3
OOO.OOO
OO...OO
OOO...O
..OO.OO
...OOOO
...OOOO
Step 4
OOOOOOO
OOOOOOO
OOOOOOO
OOOOOOO
OOOOOOO
OOOOOOO
Step 5
.......
...O...
....O..
.......
OO.....
OO.....

-- program is finished running --
```