GTU Department of Computer Engineering

CSE 331

Project 1 Bomberman Report

BEYZA ACAR

200104004065

13.11.2023

1- *System Requirements (RULES)*

- We will use MARS ISS.
- Pseudo instructions are allowed.
- You MUST comment each line of your assembly, explaining why you wrote that line.
- You will at least implement 3 assembly subroutines. Implement them wisely.
- It is better to have a working lower version than to have a not working high version. So be sure to submit a
- working code. One step at a time and keep every version. Because, not simulating submissions or totally
- wrong results can get 30pts at most. Therefore, do not target Version 4 in the beginning. Start with the
- first version and update it to the next versions.
- Obey the contract.
- This is not a rule but strong advice: First, implement the solution in C, then test that implementation and then write your assembly with the same algorithm. Because rewriting in C is much easier.
- Each late day after the due date results in 20% grade loss.
- If you use compiler you get 0, if you cheat you get 0. If you cannot fully explain your own code you get 0.
- Moreover, those who attempt any of these will cause a highly negative impression over the instructors.

2- *Aims of this project*

- Learn very well to use assembly for generic programming.
- Understand how such simple instructions can solve generic problems.
- Think similar to the MIPS CPU.
- Understand better how compilers work.

3- *Problem*

https://www.hackerrank.com/challenges/bomber-man/problem
the problem can be reached from the link above.

4- *General Approach*

   a. *Data*

      i. *The variables in the data field of assembly program*

         1. *Data Types:*

            a. *The "grid" array is implemented as a char array, where each element occupies 1 byte of memory. However, all other variables outside the "grid" array are of type "word," consuming 4 bytes of memory each.*

         2. *Access data:*

            a. *To access elements in the one-dimensional array, set and get functions have been created. The access formula grid[i * column + j] efficiently navigates through the array based on the provided column value.*

      ii. *Rules for register usage*

         1- *A function always takes its arguments from a registers ($a0, $a1, $a2, $a3)*

         2- *A function can use $t0, $t1, $t2, $t3 registers without saving their values to the stack*

         3- *A function must save its argument values in the stack for later use*

         4- *A function must save the local variables in the stack for later use*

         5- *A function must return its result in $v0 register*

         6- *A function must save the value of $ra register in the stack for later use if it calls another function*

   b. *Label Rule (Unique Label Identification for Collision Prevention)*

      ▪ *To address the potential collision of labels within functions, a unique identification numbering system has been implemented. Each function is assigned a distinct ID number, which is indicated at the top of the function with comments. This practice is in accordance with the GENERAL RULES OF REGISTER USAGE outlined at the beginning of the file, particularly RULE 6.*

      ▪ *The use of unique ID numbers for functions enhances code readability and mitigates the risk of label collisions. Developers can easily reference these IDs, as they are documented at the beginning of each function, providing clarity and structure to the codebase.*

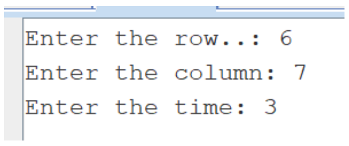
      ▪ *Example: outerLoop1, outerLoop2, else1, else2 …*

   c. *Method Descriptions with Corresponding C Equivalents*

      i. *To mitigate label collisions and enhance code readability, each assembly method is accompanied by a comment line indicating its equivalent function in C. These comments, placed at the top of each assembly method, provide a quick reference to the corresponding C representation.*

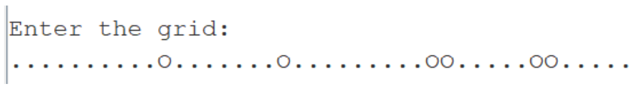
5- *Input and Output Type Examples*

- *For r, c and n values:*



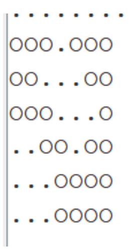
```
Enter the row..: 6
Enter the column: 7
Enter the time: 3
```

- *For grid:*



```
Enter the grid:
..........O.......O.........OO.....OO.....
```

- *Output :*



```
........
OOO.OOO
OO...OO
OOO...O
..OO.OO
...OOOO
...OOOO
```