

VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING



COMPUTER ARCHITECTURE LAB (CO2008)

Assignment Report

BATTLESHIP

Instructor: Prf. Nguyễn Thiên Ân, *CSE - HCMUT*

Student: Đặng Quốc Huy - 2053031, (*Group CCO1*)

HO CHI MINH CITY, DECEMBER 2023



Contents

1	Outcomes	2
2	Introduction	2
2.1	MIPS & MARS	2
2.1.1	What is MIPS assembly language?	2
2.1.2	What is MARS?	2
2.2	Battleship	3
3	Requirements	4
4	Idea	6
4.1	Whole picture idea	6
4.2	Set up idea	7
5	Implementation	7
5.1	Welcome players	7
5.2	Instruction	8
5.3	Set up phase	8
5.4	Combat phase	11
5.5	Game over	12
6	Conclusion	12



1 Outcomes

After finishing this assignment, students can proficiently use:

- MARS MIPS simulator.
- Arithmetic & data transfer instructions.
- Conditional branch and unconditional jump instructions.
- Procedures.

2 Introduction

2.1 MIPS & MARS

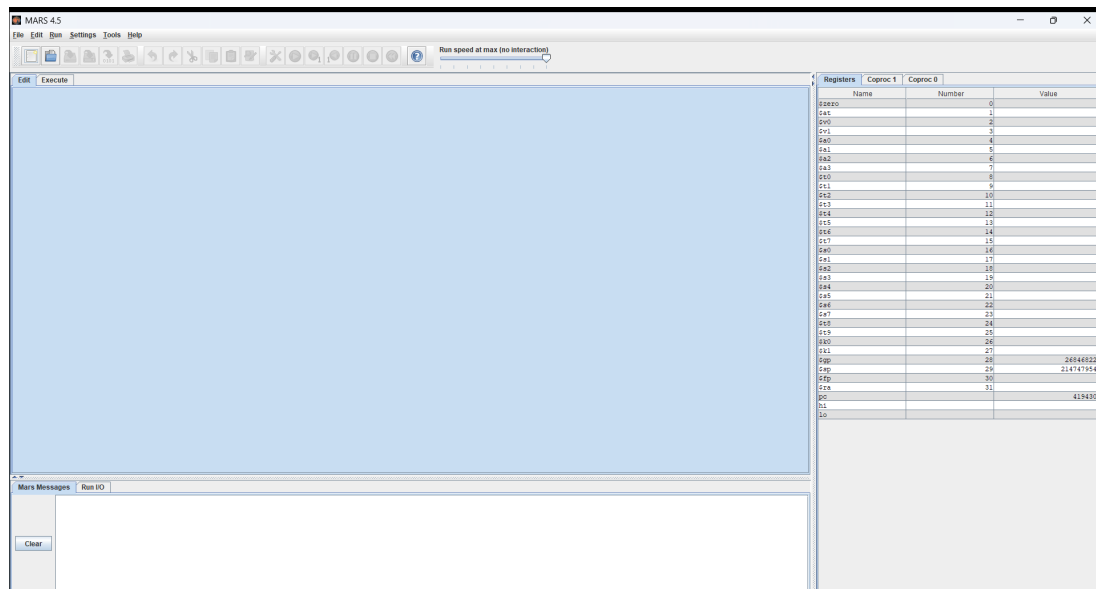
2.1.1 What is MIPS assembly language?

MIPS assembly language simply refers to the assembly language of the MIPS processor. The term MIPS is an acronym for Microprocessor without Interlocked Pipeline Stages. It is a reduced-instruction set architecture developed by an organization called MIPS Technologies.

The MIPS assembly language is a very useful language to learn because many embedded systems run on the MIPS processor. Knowing how to code in this language brings a deeper understanding of how these systems operate on a lower level.

2.1.2 What is MARS?

MARS is a lightweight interactive development environment (IDE) for programming in MIPS assembly language, intended for educational-level use with Patterson and Hennessy's Computer Organization and Design.



Screen of MARS version 4.5

2.2 Battleship

Battleship is a strategic board game between 2 players. It focuses on the process of planning deducting skill of both players. The game rules can be found in **Reference** , in summary:

- In a classic game, each player sets up a fleet of battleships on their map. A fleet must contain a predefined set of battleships with different sizes.
- After the ships have been positioned, the game proceeds in a series of rounds. In each round, each player takes a turn to announce a target square in the opponent's grid which is to be shot at. The opponent announces whether or not the square is occupied by a ship. If it is a "hit", the player who is hit marks this on their own or "ocean" grid (with a red peg in the pegboard version), and announces what ship was hit.
- If all of a player's ships have been sunk, the game is over and their opponent wins.



Example of Battleship game in real life

3 Requirements

In this project, we will replicate the battleship game in MIPS assembly language. Since the resource is limited and to keep things simple, we will work with a smaller grid size which is 7x7. Please create a program that does the following:

- A note before entering the program: A ship location is indicated by the coordinates of the bow and the stern of the ship (row_{bow} , $column_{bow}$, row_{stern} , $column_{stern}$). In the next page figure, the coordinates of the yellow 4x1 ship is "0 0 0 3"; the blue 2x1 ships are "1 4 2 4", "2 1 3 1", and "4 4 4 5"; the green 3x1 ships are "5 3 5 5" and "6 0 6 2".
- We begin with the preparation phase where the players set up their ship locations. The program should prompt to ask the players to insert the location of the ships. The values of the cells of the map will be either 1 or 0. 1 means that the box is occupied by a ship, otherwise it's 0. The players will need to setup the exact amount of ships with the same size in order to complete the setup phase. In detail, each player will have 3 2x1 ships, 2 3x1 ships, and 1 4x1 ship. Note that the ships can not overlap with each other. An example of an acceptable arrangement is shown in below image.

1	1	1	1	0	0	0
0	0	0	0	1	0	0
0	1	0	0	1	0	0
0	1	0	0	0	0	0
0	0	0	0	1	1	0
0	0	0	1	1	1	0
1	1	1	0	0	0	0

Example of the the ships that a player can place in this project

- After the setup phase, both players will take turn targeting a box on the other player's map blindly. If the attack hits a cell that is occupied by a ship, an announcement should pop up in the terminal and says "HIT!". This will allow the attacking player to know if they hit an opponent's ship or not. The players will need to remember the cells that have been hit themselves. If a cell got hit, its value should change into 0, as there it is no longer occupied by a ship. A ship is completely destroyed if all of the cells it's occupying are targeted. A player will lose if all of their ships got destroyed first. In other words, the game will end when a player's board contain only 0's.

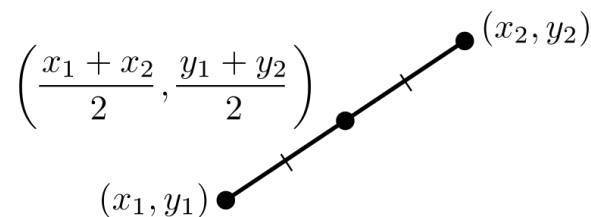
4 Idea

4.1 Whole picture idea

- In this game, we use 2 labels: **board1** and **board2** for storing each player's grid.
- There will be 2 phases, the setup phase and the combat phase. In the setup phase, the game will show the player's board for them to place ships.
- Each player will have 1 4x1 ship, 2 3x1 ships, 3 2x1 ships. Player 1 will input first, then Player 2. There will be 16 cells which contain 1s for each board of each player to represent ships' location. Therefore, the player loses if they are hit 16 times differently.
- Whenever one player successfully shoot the opponent, the cell value that was hit will down from 1 to 0.
- After the setup phase, we move to the combat phase, where each player will take turn to shoot other player's ships.
- If the shot was hit, system will print HIT!!!, otherwise system will move on to the next player.
- At the end of each turn, system will check if there were a winning player. If not, the game will continue; otherwise, the game will be over.
- When the game is over, system will ask if players want to play again.

4.2 Set up idea

- With 2x1 ship, we just need to insert coordinate of stern and bow of the ship into the grid.
- With 3x1 ship, we will have three locations of the ship: bow, middle-ship, stern. Middle-ship coordinate will be calculated as a midpoint of bow and stern. (Formula as picture below)


$$\left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right)$$

- With 4x1 ship, we have to identify whether the ship is horizontal or vertical. Then, we will add cells one by one depending on the direction of the ship from stern to bow (or vice-versa).

5 Implementation

5.1 Welcome players

Firstly, we welcome players to the game, and ask if they want to read instructions or play game immediately. If players choose 1, we will show the instruction, if players choose 2, we will move them to the game play.

```
***** Welcome to Battle Ship game >( ' v ' x)< *****
- Choose 1 to open the instruction.
- Choose 2 to play game.
Your choice: |
```

Welcome players!

5.2 Instruction

If the players choose 1, we simply show players the instructions to play the game and move to the game play.

The instruction includes: Set up, Input rule, Game play, Winning condition, and map to view.

```

**** Welcome to Battle Ship game >( ' v ' x)< ****
- Choose 1 to open the instruction.
- Choose 2 to play game.
Your choice: 1

INSTRUCTION
Set up:
1. Each player places their ships on their 7x7 grid. The ships vary in size: 1 ship of length 4, 2 ships of length 3, and 3 ships of length 2.
2. Ships cannot overlap or extend beyond the grid. Ships are only horizontal or vertical.
3. The game uses the following coordinate system: columns and rows are numbered 0-6 (e.g. 1 4, 2 5).

Input rule:
1. Players will input row-value first, then column-value. Each value is separated by an enter button.
2. Two enters are inputted consecutively or backspace/delete without any value will execute the game.

Gameplay:
1. Players place their ships by inputting the coordinate of bow and stern of the ship.
2. Players take turns to announce their shots by stating the coordinates of the target (e.g. 3 2).
3. If the shot hits the opponent's ship-box, the system will announce HIT!!!

Winning:
All the squares of all opponent ships have been hit, the first player did that win the game.

Map:
  0  1  2  3  4  5  6
  -----
0| 0  0  0  0  0  0  0
1| 0  0  0  0  0  0  0
2| 0  0  0  0  0  0  0
3| 0  0  0  0  0  0  0
4| 0  0  0  0  0  0  0
5| 0  0  0  0  0  0  0
6| 0  0  0  0  0  0  0

LET THE GAME BEGIN !

Player 1 input ship 2x1:

```

5.3 Set up phase

- We declare **board1** and **board2** to represent two players' grid respectively. Each board contains forty-nine 0s as the grid 7x7 is empty.

[illegible]

- To place the ship into the grid, we will read 4 integers (row-bow, column-bow, row-stern, column-stern) and then check their validity.

- To check validity, each value must stay between 0 and 6 is the first condition. Furthermore, row-bow should equal to row-stern if the ship is horizontal, column-bow should equal to column-stern if the ship is vertical.
- After we have row and column value, we need to calculate **offset** to find the position in the board. To calculate offset, we follow the below formula:

```
calculate_offset:  
li $t5, 7  
mul $s3, $s1, $t5  
add $s3, $s3, $s2  
mul $s3, $s3, 4
```

- Where t5 is size of the grid, s1 is row value, s2 is column value, and 4 is element size.
- After we have offset for each cell, plus with the **Set up idea** on 4.2, we can place ships into the grid.
- Player 1 will input their ships first. The syntax is **row** <enter> **column** <enter>. After inputting each ship, the map will be shown to help player check.

```
Player 1 input ship 2x1:  
1  
4  
2  
4  
-----  
0000000  
0000100  
0000100  
0000000  
0000000  
0000000  
0000000  
0000000  
-----  
Continuing input:  
Player 1 input ship 2x1:
```

- Overlap situation is counted when the cell's value reach to 2, which means the cell has been added by one twice.

- However, in my code, if the input is overlapped or out-ranged (below zero or greater than six) when setting up, the system will force player to re-input again from the beginning of that player's turn (also initializing the data in that grid). As I checked the conditions in general way instead of separating them for each kind of ships.

```
Player 1 input ship 2x1:
-2
1
3
1
Your ship coordinate may have been overlapped/out-ranged, please re-input from the beginning
Player 1 input ship 2x1:
```

- Meanwhile, **in other invalid situations**, the system will force player to re-input again in place (the previous data that has been input remains).
For example:

```
Player 1 input ship 2x1:
2
1
4
1
Your input may have been incorrect, please re-input the value
Player 1 input ship 2x1:
```

- After completing set up process, the player will have a chance to take a look whether all ships are located correctly as their wish. If not, the player can input 1 to set up again from the beginning.

```
Grid of player 1:
1111000
0010100
0110100
0100000
0000000
0001110
1110000

Please carefully check if your set up is correct, incorrect set up will affect the gameplay:
1. Choose 1 for resetting the ships.
2. Choose 2 for moving to the next process.
Your choice:
```

- After setting up, player 1 needs to choose 2 so it's player 2's turn to set up. During this process, a blank space will be printed so that player 2 cannot see player 1's grid.

- Finally, after player 2's setting up, they also need to choose 2 so the system can move to combat phase.

5.4 Combat phase

- In this phase, each player will take turn to shoot by inputting cell's coordinate. The system will only announce HIT!!!! if the shot is successful. If the shot coordinate is incorrect, the system will ask to re-input in place.
- A successful shot is the shot that its target cell's value is one.

```
It's combat time
Player 1 shoot:
3
3
Player 2 shoot:
2
2
HIT !!!!!!!
Player 1 shoot:
```

- Each player has 16 numbers 1 in their grid, each time the ship is hit, this number will decrease by one. A player loses if they are hit 16 times differently before the other player. Therefore, we will have two registers, both values are 16 for each players (also knows as health point). If the shot is success, the opponent's health point will be decreased by one.
- To identify the winner as soon as possible, we will check if we have someone losing (health point equals to zero) after every shot.
- The cell that has been hit will down value to zero. So, if the player hits again at that cell, the system will not count.

5.5 Game over

- When the game is over, the system will print out the winner. Additionally, players can input 0 to play a new battle ship game.

```
Player 2 shoot:  
3  
6  
HIT !!!!!!!  
Player 2 won!! -- Press 0 to play again --
```

6 Conclusion

It is very challenging to implement Battle ship game with MIPS. In my opinion, the hardest part of the project is the Set up phase, where we must check many conditions for the coordinates. By finishing this project, I have studied so much about MIPS programming.

References

- [1] Wikipedia, “Battleship (game).”
[https://en.wikipedia.org/wiki/Battleship_\(game\)](https://en.wikipedia.org/wiki/Battleship_(game))