Ho Chi Minh City – University of Technology

COMPUTER ARCHITECTURE

Report of Assignment

Tic – Tac – Toe

May 2022

I. Introduction

Tic-tac-toe is a paper-and-pencil game for two players who take turns marking the spaces in a n-by-n grid (n can be changed but at least equal to 3) with X or O.

The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row is the winner. It is a solved game, with a forced draw assuming best play from both players.

In this report, I am going to illustrate the general idea as well as how tic-tac-toe can be performed and implemented using MIPS Assembly Language Programming (Mars 4.5 version).

The idea of this game is very simple. We need 3 steps to decide who is the winner between two players:

1. Create a friendly interface for two participants.
2. Each person takes turn to mark a point in a grid. A 5x5 board is chosen to be the space so there are 25 spots in total. Each player can only mark one spot for each turn. There will be a request whether players want to undo a move. Players can not mark again the spot which is already marked. During the first turn, both players are not allowed to choose a central point (number 13, I will clarify this in the **Implementation** section).
3. If the number of marked spots of each person is greater than or equal to 3, we will check for the winner. 3 points in a row, a column or a diagonal will be the winner. If the checking process is done without any winner, players will continue to mark spots and the checking process will continue until we have a winner. We also have a special case that the result is draw. This is called the checking process.

II. General idea

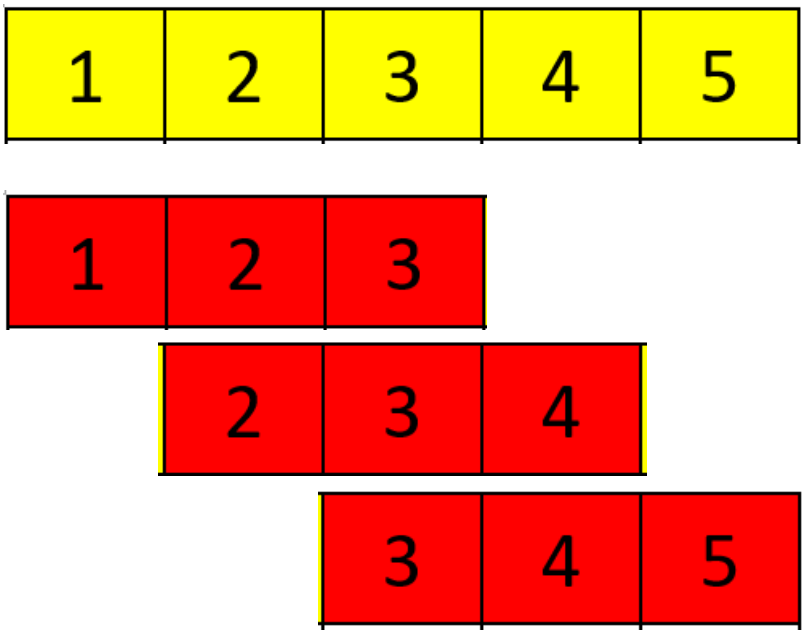
Supposedly, we have a 5x5 board and it is numbered.

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

The most important part of this game is the checking winner process. Here's my idea:

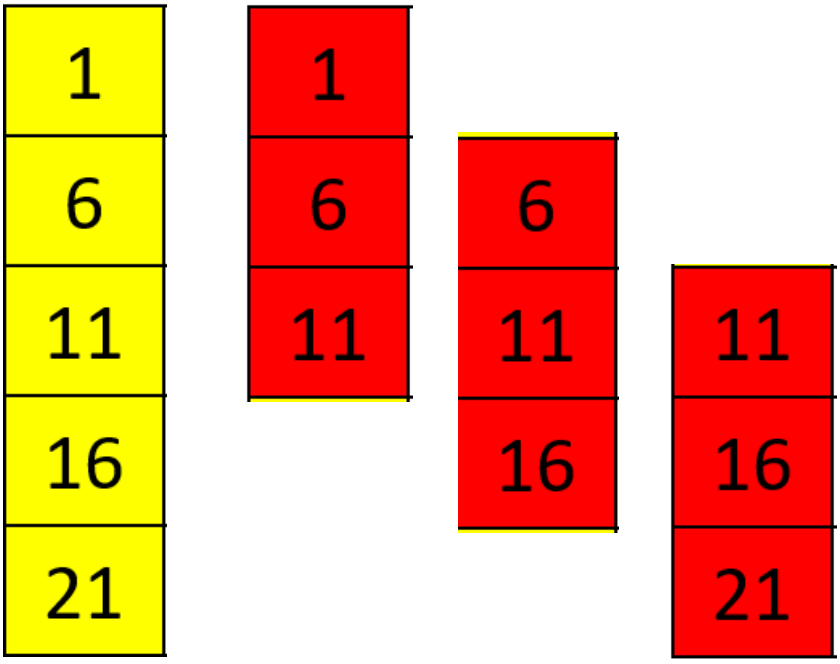
1. **Rows:**

   We have 5 rows in total. Each person only needs 3 points in a row to win the game. Hence, there will be only 3 cases in a single row. These pictures below show an example of the first row.



There are 5 rows having same properties. Therefore, the number of winning cases in this part is 15.

2. **Columns:**
   We have 5 columns in total. Each person only needs 3 points in a column to win the game. Hence, there will be only 3 cases in a single column. These pictures below show an example of the first column.



There are 5 columns having same properties. Therefore, the number of winning cases in this part is 15.

3. **Diagonals:**
   My idea is that I am going to divide the board into smaller pieces. Each piece is a 3x3 square so there are 9 squares of size 3 in a 5x5 board in total. As can be seen, a 3x3 square can contains no more than 2 diagonals. Hence, the total cases of diagonals are 18 cases. Here's an example of a square, note that those boxes in red make up a diagonal:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

III. Implementation

Using MIPS Assembly Language Programming (Mars 4.5 version), I used the knowledge of stack, jump, loop and branch to solve this game.

Firstly, in .data, where all strings and commands of the game are stored, I included the title, the introduction, rules to win the game or the game draws as well as the command requesting players to choose their movements.

In the .text part, I mainly used standard MIPS instructions to write the code. Particularly, I used the knowledge of stack to create a 25-blank memory representing 25 choices for players. In my program, there's a line which is "add $sp, $sp, -100" and this creates a stack. Each box contains 0 as the default value.

For the movements of players, the first turn has a regulation of choosing a central point of a board (in my description, it is the position whose number is 13). If they choose 13 during their first turn, the program pops out a violation message and asks them to choose a move again. If they enter a number which is greater than 25 (25 is the limitation of the movements), the program asks them to choose again. If the number is suitable, the program jumps to the undo request. I ask the player to press 1 for undo a move and press any other number to continue the game. There is no chance that any player can undo 2 movements during a turn. '1' will be marked as player 1's movements and '-1' will be the value of player 2's movements in the stack.

Winner checking process:

The process starts with rows. Each row contains no more than 3 cases of winning the game. These pictures below show all possibilities of the first row:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

The idea was that I used a temporary register to do an addition of 3 consecutive boxes in the stack. I also used the loop to make sure it only run 3 times each row. If the temporary register returns a value of 3, player 1 will take the victory. If the temporary register returns a value of -3, player 2 will win the game. After each row is finished checking, I designed it to jump to the next row if there is no winner.

When there’s no winner by rows, I considered all possibilities in columns. There are also 5 columns and the method to find the winner is familiar when I did with rows. These pictures below show all cases of the first column:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Starting from 1, I also use a temporary register to store the result of the addition of 3 boxes making up a column. By creating an iterator, it helped me to traverse the stack so conveniently, by simply adding 20 to the iterator. Because the iterator starts at 1 and each box is a word with 4 bytes, 20 is the suitable number to be added. After the first column is done checking, then the second one and the rest are respectively checked. Column checking process is only occurred when there is no winner by rows. Hence, if there is still no winner by columns, I continue considering the diagonals.

Last but not least, I check the diagonals. I decide to divide the 5x5 board into smaller pieces. Each piece which is needed to check is a 3x3 square. In a square, there are at most 2 diagonals which can be found. In a 5x5 board, we can find no more than 9 squares size 3. To sum up, there are 18 diagonals must be checked to find a winner by diagonals. I use the same method which is already performed in rows and columns with labels and branches to find the winner in this case. I also found out the rules that diagonals can be classified into even smaller cases, with 9 cases of diagonals whose distance is 6 units and the rest is 4 units. These are all squares I could find:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

1	2	3	4	5
6	7	**8**	**9**	**10**
11	12	**13**	**14**	**15**
16	17	**18**	**19**	**20**
21	22	23	24	25

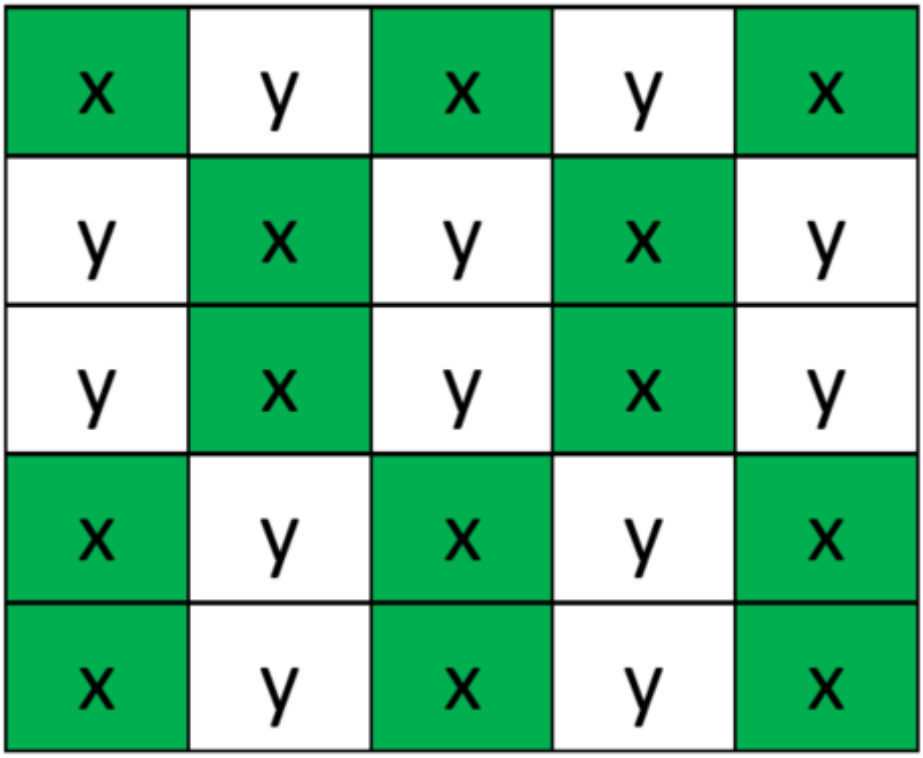
1	2	3	4	5
6	7	8	9	10
11	**12**	**13**	14	15
16	**17**	**18**	19	20
21	**22**	**23**	24	25

1	2	3	4	5
6	7	8	9	10
11	**12**	**13**	**14**	15
16	**17**	**18**	**19**	20
21	**22**	**23**	**24**	25

1	2	3	4	5
6	7	8	9	10
11	12	**13**	**14**	**15**
16	17	**18**	**19**	**20**
21	22	**23**	**24**	**25**

The iterator does its best job at theses cases. In the code, I only used 2 main labels for diagonals. Each label contains 3 loops and each loop repeats three times. With 6 units distance, we have 1-7-13, 2-8-14, 3-9-15, 6-12-18, 7-13-19, 8-14-20, 11-17-23, 12-18-24 and 13-19-25 are in diagonal positions. With 4 units distance, we have 3-7-11, 4-8-12, 5-9-13, 8-12-16, 9-13-17, 10-14-18, 13-17-21, 14-18-22, 15-19-23 are also in diagonal shape.

During my research and fixing time, I find out a draw result I used $t8 and $t9 to store each player's total movements. If the result of adding these two values is equal to 25, it means that the board is full. Therefore, this is the tie match. The case is:



In my program, in order to print the board and results after each turn, I also implemented some labels such as *print*, *print_x*, *print_y*, .etc. All lines of code are in the source file.