**Project Report**

1. Description of the project:

The team project is the Connect 4 Game which is a two player game in which players take turn placing different colored coins into a 6 x 7 board with pockets with a goal to get 4 same colored coins in a row either vertically, horizontally or diagonally.

In our MIPS version of the game, the game board is represented by a 6 x 7 grid and instead of different colors, the players use Xs and Os. The user can choose to either play with the computer or with another player. The first player to get four of the same signs in a row either vertically, horizontally or diagonally wins the game.

4. A discussion of algorithms and techniques used in the program:

In this project, **we treated the game board as one single dimensional array with 72 elements** (that is 8 rows and 9 columns of which 2 end columns and 2 end rows on either side of the board represented the border of the board). The board was initialized using 0s and 3s, 3s representing the border and 0s representing the empty element. We used 1s to represent player 1’s coin and 2s to represent player 2’s coin. We stored the starting position (that is the address of the first element) of the board in a register using the **load address** instruction.

We used different subroutines for different parts of the game, mainly Placer, that places the user’s coin at the correct position in the board/grid, BoardDisplay, that displays the board, WinCheck, that checks if the player got 4 same symbols in a row either vertically, horizontally or diagonally, and main, that calls all the subroutines.

The Placer takes a valid user input of the column as an argument. The user enters a number between 1 and 7. So, the placer first **add**s 55 to the position to get to the end of the column (for example if the user enters 1, the last end of column one will be 1 + 55 = 56th position). Then it **word align**s the index (by using shift logical left instruction) and checks if the position is available to place the player’s coin (by using load word instruction). If the position is empty (that is if it holds a 0), it places the coin in the position. If not, it **sub**tracts 9 from the position (that is 56 - 9 = 47) to get to the second last position in the column, and repeats the whole process of word aligning and checking what is in the position an placing the coin. After the coins is successfully placed, the placer call the BoardDisplay subroutine using the **jump** instruction.

The BoardDisplay traverses the whole array and checks each position for either 0s, 1s, 2s or 3s. If the position has a 0, 1, 2, or 3 it displays an empty space ‘| |’, an ‘|X|’, an ‘|O|’, or just a ‘|’, respectively. We used the **branch if equal** instruction to do this branching. The end is reached when the end of the array is reach (that is when the incrementing variable reaches 292 (72\*4 = 288 (end of the array); 288 + 4 = 292 (outside the array)). The BoardDisplay then calls WinCheck subroutine using the **jump and link** instruction.

The WinCheck checks if the player playing the game won. It takes the position of the coin that the player just played as a argument. To check the **vertical win**, it checks for three other positions above or below the current position that hold the same coin as the player. If it finds a match (that is four in the same column), it returns 1 (that is the player won), else it returns 0. The same way for **horizontal win**, it checks for three other positions either on the right or left of the current position that have the same coin. If it finds a match (that is four in the same row), it returns 1 (that is the player won), else it returns 0. For the **diagonal win**. there are two conditions that it checks for, either left diagonal or right diagonal (that is / or \). If it finds a match (that is four in the same diagonal), it returns 1 (that is the player won), else it returns 0. After execution, the subroutine jumps back to the place it was called at, using the **jump register** instruction.

Other subroutines include validCol, that checks if the column number the user entered is valid or not, and changePlayer, that changes the player every turn. We used a bunch of jumping and branching conditions to jump though these subroutines.