Ho Chi Minh City University of Technology

School year 2022-2023 – Semester 2

COMPUTER ARCHITECTURE - CO2007

ASSIGNMENT:
FOUR IN A ROW - A NEW VERSION

Lecturer: Prof. Pham Quoc Cuong

Instructor: Dr. Bang Ngoc Bao Tam

Student: Phan Huu Thanh - 2112305

1. Introduction **2**
a. Four in a Row 2
b. Applying into MIPS 2
2. General Outline for the program **4**
a. Board Setup 4
b. Asking for each player names and randomly assign them their pieces 4
c. Initializing all value and first move for both players 5
d. Starting the Game loop 6
3. The Game Loop 6
i. The Dropping 8
ii. The Undoing 9
iii. The Removing 10
iv. The Blocking 11
v. Checking for Violations 12
vi. Printing 13
4. Conclusion 13

1.Introduction

a. Four in a Row

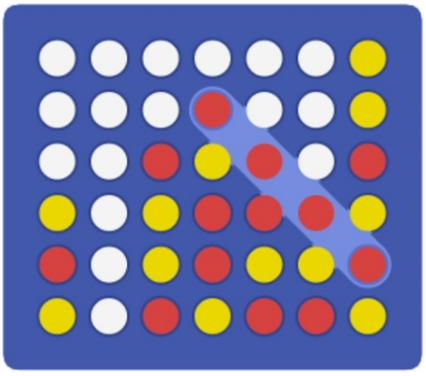
Four in a row is a classic strategy board game that has been enjoyed by people of all ages for generations. The objective of the game is to be the first player to connect four of your own colored discs in a row, either vertically, horizontally, or diagonally.

The game is simple to learn but difficult to master, requiring players to think strategically and anticipate their opponent's moves. In this particular version of the game, the game is built using MIPS (Microprocessor without Interlocked Pipeline Stages) technology, which is a popular instruction set architecture used in many embedded systems and microcontrollers. This allows for a streamlined and efficient game experience that is sure to provide hours of entertainment for players of all skill levels.

b. Applying into MIPS

Four in a row (also known as Connect 4, Four Up, Plot Four, Find Four, Captain's Mistress, Four in a Row, Drop Four, and Gravitrips in the Soviet Union):

- Connect Four is a two-player board game in which each player chooses a color and takes turns dropping colored tokens into a vertical grid consisting of seven columns and six rows.
- The tokens are placed in the lowest available spot in the column and drop straight down.
- The objective of the game is to be the first player to align four of their colored tokens in a horizontal, vertical, or diagonal line.



Picture 1: An example of the Four in a Row Game

The detailed rules and requirements for the game are shown as below:
First, the players must choose their names. After that, randomly assign the piece for them. Then, let the game begin.

The output of the game is:

If we have a winner, the program will show the name of the winner according to the number of the piece of this player on the board.

On the other hand (TIE GAME), the program just show the result information with the final board.

Moreover, there are some addition requirements of the game, such as:

1. In the first move of each player, they must drop the piece in the center column of the board.
2. In the middle of the game (after their first move), each player has 3 times to undo their move (before the opponent's turn).
3. Each player also has one time to block the next opponent's move. However, if the opponent has a chance to win (already had three pieces of the same color vertically, horizontally or diagonally), the player can not use this function.
4. And, instead of dropping a piece, each player has one time to remove one arbitrary piece of the opponent. It means that if a player chooses to drop a piece, the player can not remove the opponent's piece and vice versa. In case of removing a piece, if there are any pieces above this piece, they will fall down.
5. In addition, students have to handle the exception of placing a piece at an inappropriate column (outboard or column that has full pieces) by restarting the move. And it also counts as a violation.
6. If any players try to violate all of the above conditions over 3 times. This player will lose the game.
7. In each turn, the program has to show: the number of remaining violations, undo, and the player's name according to this turn.

These are the rules and requirements that we need to implement to the program.

2.General Outline for the Program

a. Board Setup

For the table of the game, I will be using an array with the length of 7 columns x 6 rows = 42 to store the player token that will be continuously dropped into the table.

To display the whole table, I will be printing out all of the values in the array and also adding some borderlines so that it will look clean.

```
gameBoard:              .space 42
```

Picture 2: The gameBoard array

Next step is storing the whitespace character in a random temporary variable so that we can display our board by using it.

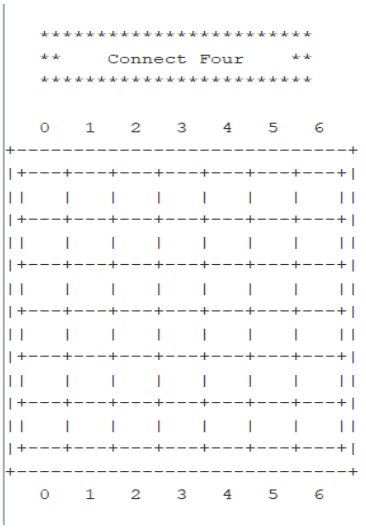
```
createBoard:
    li      $s1, 42          # store the size of the board array in $s1
    la      $s2, gameBoard   # store the address of the array in $s2
    move    $s3, $zero       # set $s3 to 0
    la      $t2, whitespace
    lb      $t1, 0($t2)      # store the whitespace character in $t1
    j       createLoop       # begin another iteration of the loop
```

We will begin by running the loop 42 times since we are working with a 7*6 table. Our objective is to print the contents of each row, starting from the first row and moving down to the last. To achieve this, we will use the following set of statements.

```
createLoop:
    addi    $t3, $s3, -42
    bgez    $t3, createRet  # if count >= 42 exit loop
    sb      $t1, 0($s2)     # store whitespace character
    addi    $s2, $s2, 1     # increment the array index
    addi    $s3, $s3, 1     # increment the counter
    j       createLoop
displayFirstBoard:
    j       displayTitle    # display the title to the users
displayBoard:
    move    $s4, $zero      # set the row value to 0
    move    $s5, $zero      # set the column value to 0
    j       displayRowLoop  # display the initial game board to the users
rowLoopRet:
    jr      $ra             # return from the subroutine
displayTitle:
    li      $v0, 4          # print_string
    la      $a0, titleString
    syscall                 # print the title string
    j       displayBoard    # return from the subroutine
displayRowLoop:
    bne     $s4, $zero, notFirstRow     # if row != 0 then branch to notFirstRow
    li      $v0, 4
    la      $a0, topRow                 # print the topRow string
    syscall

notFirstRow:
    move    $s5, $zero          # set the column value to 0
    j       displayColumnLoop
columnLoopRet:
    addi    $t4, $s4, -5
    bgez    $t4, doneRowLoop    # if row >= 5 then branch to doneRowLoop
    li      $v0, 4
    la      $a0, middleRow      # load the middleRow string
    syscall                     # print the middleRow string
    addi    $s4, $s4, 1         # increment the row value
    j       displayRowLoop
```

These are just a few of the functions that I have created. If you would like to see additional functions, you can refer to my source code. The program will generate a table that looks like this:



b. Asking for each player names and randomly assign them their pieces

To enable users to enter a name for each player and assign random tokens to them, I have implemented the following functions:

```
nameAndPieces:
        li      $v0, 4
        la      $a0, player1_setName
        syscall
        li      $v0, 8
        la      $a0, player1_name
        li      $a1, 100
        syscall
        li      $v0, 4
        la      $a0, player2_setName
        syscall
        li      $v0, 8
        la      $a0, player2_name
        li      $a1, 100
        syscall
        li      $v0, 42
        li      $a1, 2
        syscall
        beq     $a0, 0, player1X
        li      $t0, 'O'
        li      $t1, 'X'
        sw      $t0, -36($sp)
        sw      $t1, -40($sp)
        j       Print_pieces
player1X:
        li      $t0, 'X'
        li      $t1, 'O'
        sw      $t0, -36($sp)
        sw      $t1, -40($sp)
        j       Print_pieces
```

Picture 3: Requesting for names

.

```
 Player 1 please enter name: me

 Player 2 please enter name: you
Coin tossing... Your randomly assigned pieces are:

me
    You are X
you
    You are O
```

Picture 4: Results

c. Initializing all value and first move for both players

I suggested that each player make their first move to the third column, as it is mandatory to start by moving to the middle square.

```
requestPlayer1Move:
    sw $ra,0($sp)
    jal print1
    lw $ra,0($sp)

    lw $t0,-60($sp)
    beq $t0,1,con1
    li $v0,4
    la  $a0,player1_first_move
    syscall

    li $v0,5
    syscall

    bne $v0, 3, con1_1
    move $a1, $v0
    addi $t0,$t0,1
    sw $t0,-60($sp)
    j processPlayer1Move
```

Picture 5: Player1_first_move

d. Starting the Game loop

To address the issue of alternating turns between players, I have implemented a loop that starts with player 1 and sequentially switches to player 2 until either a winner is determined or the game ends in a draw.

3. The Game Loop

i. The Dropping pieces

In each turn, a player is permitted to drop a token into the game by entering the corresponding column number. If the selected column is already full or does not exist, it will be considered an error for that player's turn. However, if the column is valid, the player can continue their turn.

```
processPlayer1Move:
    sw      $ra, 0($sp)
    jal     validIndex
    lw      $ra, 0($sp)
    move    $t3, $v0
    addi    $a2, $zero, 1
    bnez    $t3, invalidPlayerIndex
    sw      $ra, 0($sp)
    jal     availableSpace
    lw      $ra, 0($sp)
    addi    $t3, $zero, -1
    move    $a0, $v0
    addi    $a2, $zero, 1
    beq     $a0, $t3, invalidSpaceChoice
    sw      $ra, 0($sp)
    jal     getArrayIndex
    lw      $ra, 0($sp)
    la      $t2, -36($sp)
    lb      $t1, 0($t2)
    move    $s6, $v0
    move    $t8, $v0
    la      $s7, gameBoard
    add     $s7, $s7, $s6
```

Picture 6: The dropping piece

I understand that there will be a function to check if there is a winner after each move, and the details are included in your source code. However, based on what you have described, the function will start by checking the position of the latest token dropped in relation to its surrounding vertical, horizontal, and diagonal positions. The function will then loop through the adjacent tokens, starting from the latest one, until it detects a token that is different from the current one or until there are less than four tokens in a row. At that point, the function will start a new loop and repeat the same process for the next adjacent tokens until it detects a winner or concludes that there is no winner.

```
checkWinner:
    addi    $s1, $a0,0
    addi    $s2, $a1 ,0
    sw      $ra, 0($sp)
    sw      $a0, -4($sp)
    sw      $a1, -8($sp)
    j       check_Horizontal_Win
hLoopRet:
    j       check_Vertical_Win
vLoopRet:
    j       check_Diagonal_WIn
diagonalRet:
    j       checkTieGame

check_Horizontal_Win:
    lw      $s1, -4($sp)
    li    $s2, 0
    li    $s3, 0
    j       horizontalLoop

horizontalLoop:
    add    $t6, $s2,$zero
    subi    $t6, $t6, 6
    bgtz    $t6, hLoopRet
    add    $a0, $s1 ,$zero
    add    $a1, $s2 ,$zero
    sw      $ra, -12($sp)
    jal     getArrayIndex
    lw      $ra, -12($sp)
    add    $s6, $v0 ,$zero
    la      $s7, gameBoard
    add     $s7, $s7, $s6
    lb      $s4, 0($s7)
    bne     $a3, $s4, hLoopReset
    subi    $s3, $s3, -1

    subi    $t4, $s3, 3
    bgtz    $t4, winner
```

Picture 7: The checkWinner function(continued)

If there was no winner in the previous turn, the game will continue with the next player taking their turn. The game will proceed in the same manner as before, with each player taking turns to drop their token into the board and the function checking for a winner after each move. This process will continue until either a player wins or the game ends in a draw.

```
switchPlayers:
    sw      $ra, 0($sp)
    sw      $a2, -4($sp)
    jal     displayBoard
    lw      $ra, 0($sp)
    lw      $a2, -4($sp)
    addi    $t3, $zero, 1
    bne     $a2, $t3, requestPlayer1Move
    j       requestPlayer2Move
```

Picture 8: The switch player function

ii. The Undoing

During each of their turns, a player is allowed to make the same move up to three times. To facilitate this, the program saves the move made by the player in a variable and removes the corresponding token from the cell by replacing it with a whitespace character. This allows the player to make their move again without having to select a different column.

```
undo:
    beq $a2,2,undo_con
    lw $t0,-52($sp)
    addi $t0,$t0,-1
    sw $t0,-52($sp)
    undo_con_con:
    la      $t2, whitespace
    lb      $t1, 0($t2)
    la      $s7, gameBoard
    add     $s7, $s7, $t8
    sb      $t1, 0($s7)
    bne     $a2,1,requestPlayer2Move
    j requestPlayer1Move
    undo_con:
    lw $t0,-56($sp)
    addi $t0,$t0,-1
    sw $t0,-56($sp)
    j undo_con_con
undo_check1:
    lw $t0,-52($sp)
    sw $t0,-52($sp)
    beq $t0,$zero,undo_check_exit1
    jr $ra
    undo_check_exit1:
    lw $t0,-44($sp)
    addi $t0,$t0,-1
    sw $t0,-44($sp)
    move $t1,$ra
    jal violation1_check
    move $ra,$t1
    lw   $ra, 0($sp)
    j p_con1
```

Picture 9: The Undo function

iii. The Removing

In addition to dropping their own token, each player is also given the option to remove one of their opponent's tokens once during the game. However, the player can either choose to drop their own token or remove their opponent's token, but not both in the same turn. If the player chooses to remove an opponent's token, the selected token will be removed from the board, and any tokens above it will fall down accordingly to fill the empty space.

To facilitate the removal of an opponent's token, the program needs to keep track of four values: the row and column of the cell to be deleted, the type of token currently occupying the cell, and the player who issued the "remove" command. This information is necessary to ensure that the "remove" command is executed correctly and to update the game state accordingly. Additionally, it is important to note that the removal of an opponent's token can potentially lead to a winner if the removed token was part of a winning sequence.

After performing the "remove" operation, it is necessary to initiate a loop that begins from the deleted position and continues until it reaches the top of the table where "whitespace" is present. Within this loop, the "checkWinner" function should be utilized to determine if there is a winner or not.

If the "remove" function fails, instead of considering it as a bug, the player should be allowed to take their turn and the "remove" feature should not be counted towards their usage limit.

```
remove1:
    lw      $t0,-28($sp)
    beq     $t0,1,requestPlayer1Move
    addi    $t0,$t0,1
    sw      $t0,-28($sp)

    la      $t2, whitespace
    lb      $t3, 0($t2)

    la      $t2, -40($sp)
    lb      $t4, 0($t2)

    li      $v0, 4
    la      $a0, col
    syscall
    li      $v0, 5
    syscall
    move    $t0,$v0

    li      $v0, 4
    la      $a0, row
    syscall
    li      $v0, 5
    syscall
    move    $t1,$v0

    #t0 cot, t1 hang

    add $t2,$zero,$t1
    add $t2,$t2,$t1
    add $t2,$t2,$t1
    add $t2,$t2,$t1
    add $t2,$t2,$t1
    add $t2,$t2,$t1
    add $t2,$t2,$t1
    add $t2,$t2,$t0
```

```
    add     $s1, $s1, $t2
    lb      $t5, 0($s1)                     #gia tri cua

    bne     $t5, $t4, remove_exit1          #remove khong
    li      $v0,0    #v0=0
    li      $a0,7  #a0=7


    removeloop1:
    slt $v0, $t2,$a0      #v0=1 khi t2<7
    beq $v0, 1, remove_exitt1           #chay loop cho d
    la      $s7, gameBoard               #lay gia tri o
    add     $s7, $s7, $t2
    lb      $t6,-7($s7)
    sb      $t6,0($s7)
    subi    $t2,$t2,7
    j removeloop1

remove_exitt1:
    la      $s7, gameBoard
    add     $s7, $s7, $                 #o tren cung cua
    sb      $t3, 0($s7)

    j check1
check1:
    move $a0,$t1
    move $a1,$t0
    li $a2,1
    while1:
    li $v0,0
    lw $a3,-36($sp)
    li $a2,1
    slti    $v0,$a0,1
    beq     $v0,1,requestPlayer2Move

    sw      $ra, 0($sp)
    jal     checkWinner
    lw      $ra, 0($sp)
```

Picture 10: The Remove function

iv. The Blocking

To address the issue with the "block" function, one possible solution is to allow the player who failed to perform the function to forfeit their turn and remove the move they attempted. This would simplify the process of determining if the player has a chance to win or not. However, it is important to consider that this approach may impact the fairness of the game and the overall experience for the player. It may be worth exploring alternative solutions to address the issue of failed "block" functions.

.

```
block1:
    lw $t0,-20($sp)
    beq $t0,$zero,block1_1
    beq $t0,1,requestPlayer1Move

block2:
    lw $t0,-24($sp)
    beq $t0,$zero,block2_1
    beq $t0,1,requestPlayer2Move
block1_1:
    addi    $t0,$t0,1
    sw      $t0, -20($sp)
    la      $t2, whitespace
    lb      $t1, 0($t2)
    la      $s7, gameBoard
    add     $s7, $s7, $t8
    sb      $t1, 0($s7)
    j requestPlayer1Move
block2_1:
    addi $t0,$t0,1
    sw   $t0, -24($sp)
    la      $t2, whitespace
    lb      $t1, 0($t2)
    la      $s7, gameBoard
    add     $s7, $s7, $t8
    sb      $t1, 0($s7)
    j requestPlayer2Move
```

Picture 13: The Block Function

v. Checking for Violations

If a player violates any of the aforementioned conditions more than three times, they will lose the game. it would be appropriate to implement a check after every wrong move in the first_move and dropping_piece functions to see if the player has violated any of the conditions more than three times. If so, the player should be declared the loser of the game.

```
violation1_check:
    lw $t0,-44($sp)
    li $t9,0
    slt $t9,$t0,$zero
    li $a2,2
    sw $t0,-44($sp)
    beq $t9,1,winner
    li $a2,1
    jr $ra


violation2_check:
    lw $t0,-48($sp)
    li $t9,0
    slt $t9,$t0,$zero
    li $a2,1
    sw $t0,-48($sp)
    beq $t9,1,winner
    li $a2,2
    jr $ra
```

VI. Printing

In each turn, the program should display the following information:

- The number of remaining violations for the current player.
- The number of available undo actions for the current player.
- The name of the player whose turn it is.

```
print1:
    la $a0,socsocstring
    li $v0,4
    syscall
    la $a0,turnstring
    li $v0,4
    syscall
    la $a0,player1_name
    li $v0,4
    syscall
    la $a0,violationremain
    li $v0,4
    syscall
    lw $a0,-44($sp)
    li $v0,1
    syscall
    la $a0,undoremain
    li $v0,4
    syscall
    lw $a0,-52($sp)
    li $v0,1
    syscall
    la $a0,blockused
    li $v0,4
    syscall
    lw $a0,-20($sp)
    li $v0,1
    syscall
    la $a0,removeused
    li $v0,4
    syscall
    lw $a0,-28($sp)
    li $v0,1
    syscall
    la $a0,socsocstring
    li $v0,4
    syscall
```

4. Conclusion

In conclusion, Connect4 is a classic two-player game that has been enjoyed by people of all ages for decades. The objective of the game is simple, yet the gameplay is challenging and strategic, requiring players to think ahead and anticipate their opponent's moves. The game can be played both online and offline, and there are various versions of the game available on different platforms. Connect4 is a fun and engaging game that encourages social interaction and friendly competition. Whether playing with family, friends, or against computer opponents, Connect4 is a timeless game that never fails to entertain and provide hours of fun.