CS 0447 Computer Organization and Assembly Language

Midterm Project – Connect 4

Introduction

In this project, you will implement a 2 player game in MIPS assembly: Connect 4 aka Four-in-line. The game consists a board representing the play area. Two players face each other and drop tokens, one at a time, until one of them manages to place four in line!

Start early

The deadline will approach fast! Life happens, sickness happens, so if you start early, you can minimize the impact. Do a little bit every day! 1 hour every day! 30 minutes every day! SOMETHING!

Game mechanic

The game works like this:

1. Initially, the players have a blank board

```
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
```

2. Player 1 takes the first turn

```
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
```

```
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
Player 1, it's your turn.
Select a column to play. Must be between 0 and 6
-1
That play is invalid. Try again
Select a column to play. Must be between 0 and 6
7
That play is invalid. Try again
Select a column to play. Must be between 0 and 6
1
```


3. When a valid number is input, a token is placed in that column, at the first (lowest) free position.


```
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|*|_|_|_|_|_|
```

4. Next, it's player 2 turn.

```
Player 2, it's your turn.
```

```
Select a column to play. Must be between 0 and 6
```

5. The game ends when one of the players manages to place 4 tokens in a horizontal, vertical, or diagonal line.

```
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|*|_|_|_|_|_|_|
|_|*|_|_|_|_|_|
|_|_|*|_|_|_|_|
|_|_|_|*|_|_|_|
Congratulations player2 1. You won!
```



```
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|*|_|_|_|
|_|_|*|_|_|_|_|
|_|*|_|_|_|_|_|
|*|_|_|_|_|_|_|
Congratulations player 1. You won!
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|+|+|+|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
```

```
Congratulations player 2. You won!

 0 1 2 3 4 5 6

|_|_|_|_|_|_|_|

|_|+|_|_|_|_|_|

|_|+|_|_|_|_|_|

|_|+|_|_|_|_|_|

|_|+|_|_|_|_|_|

|_|_|_|_|_|_|_|

Congratulations player 2. You won!
```

Your assignment

Plan

Plan your implementation which includes data structures you are planning to use, user inputs that may be invalid and you need to account for, etc.

1. Think of which functions you will need to implement, and what they will do.
    1) Start from the main function and split your program into multiple steps.
    2) This plan is not going to be enforced, but it should be thought through.
2. Think of possible invalid user inputs, and how they will impact the program negatively.
    1) Board bounds.
    2) Filling a column to the top.

Implement

Implement the MIPS assembly code that executes the game described above. Your program will manage all interactions with the user and the board:

1. It begins by displaying a welcome message and an explanation of what the user should do. How is the game played?
2. Print the empty board.
3. Then, the game begins, and your program will:
    1) Ask player 1 to play:
        - Ask and validate user input (MARS will crash if the user gives no input or a letter, this is fine!)
        - Don't allow the user to select a non-existing tile.
        - Don't allow the user to select a full column.
        - "Drop" the token into the board at the requested column.

- Check for a winning condition.

2) Ask player 2 to play:
   - Ask and validate user input (MARS will crash if the user gives no input or a letter, this is fine!)
   - Don't allow the user to select a non-existing tile!
   - Don't allow the user to select a full column!
   - "Drop" the token into the board at the requested column.
   - Check for a winning condition.
3) Repeat until one of the players wins or the board is full.

4. In the end, print a message letting the winning player know the game has ended.

The welcome message

Bear in mind that you do your own thing, as long as it fits the project! So use the welcome message to explain to the user exactly how it should play the game. Explain the rules, and how the player can score points.

User input

Your program needs to ask the user in which column he/she wants to drop a token. If the user inputs an invalid value, you inform the user of that and ask again.
You must validate the user input! The exact way you implement this is up to you. You must ask the user to input something to select the column.

Representing the board

Feel free to implement all data structures that you need. However, it is suggested you'd better use matrices. You can implement your board as a matrix of **words** to keep the status of the game. Here is one suggestion:

```
board: .word

       0, 0, 0, 0, 0, 0, 0,

       0, 0, 0, 0, 0, 0, 0,

       0, 0, 0, 0, 0, 0, 0,

       0, 0, 0, 0, 0, 0, 0,

       0, 0, 0, 0, 0, 0, 0,

       0, 0, 0, 0, 0, 0, 0
```

Note: The board refers only to the contents of the board, not the frame around the tiles! The frame is always the same, it doesn't need to be stored anywhere! If you include the frame, it'll make your life harder!

For the status of each tile, it is suggested to create a matrix of 0s (empty) 1s (player 1 tokens), and 2s (player 2 tokens). When you want to print each tile, you simply need to check the status matrix to know if the tile was revealed.

```
if(board[i][j] == 0) { print('_') }

else if(board[i][j] == 1) { print('*') }

else { print('+') }
```

Check the example below:

The board - this is what you draw:

```
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|

|_|_|_|_|_|_|_|

|*|_|_|_|_|_|_|

|*|*|_|_|_|_|_|

|+|*|*|_|_|_|_|

|+|+|+|*|+|_|_|
```


The matrix representing the board - contains 1s for player 1, and 2 player 2:

```
0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0,

1, 0, 0, 0, 0, 0, 0,

1, 1, 0, 0, 0, 0, 0,

2, 1, 1, 0, 0, 0, 0,

2, 2, 2, 1, 2, 0, 0
```

Board size

You can do one of two things:

Easy route

The board should be a 7x6 matrix.

Configurable route

You can make it configurable if you so wish, and then adjust for difficulty. AT THE TOP OF THE FILE! The simplest way is to name a number, like a **#define** in C. In MARS you can do that like this:

```
.eqv   BOARD_SIZE        42 # 7*6
.eqv   BOARD_WIDTH       7
.eqv   BOARD_HEIGHT      6
```

Then you can use the name instead of a number, i.e. in instructions that would normally use a number (the code is nonsense, don't use it):

```
lw    $t0, 3($t1)        ->  lw    $t0, N_PLAYS($t1)
li    $t0, 3             ->  li    $t0, N_PLAYS
beq   $t0, 3, _label     ->  beq   $t0, N_PLAYS, _label
x:   .word 3             ->  x:   .word N_PLAYS
arr: .word 0:100         -> arr: .word 0:BOARD_ELEMENTS
```


Or ask the user

Create variables, and ask what is the size of the board they want:

```
board_size:    .word    42 # 7*6
board_width:   .word     7
board_height:  .word     6
```

Printing the board

The board must be shown to the user. Check the example below if you are not sure how to proceed. The only requirement here is that Empty tiles must be empty, and players should have different tokens to represent the tokens. The following examples used an _ to represent empty tiles, * to represent "Player 1" tokens, and + to represent "Player 2" tokens.

Ending the game

The game should end when one of the players successfully drops 4 tokens in line. At that point, let the user know that the game has ended and who won.

Example run

This is only an example. Feel free to ignore everything except functionality.

```
Welcome to connect-4 the MIPS version

 This is a 2 player game, each player will take turns placing
a token.

 The objective is to create a line of 4 consecutive tokens.

 Good luck!

 0 1 2 3 4 5 6

|_|_|_|_|_|_|_|

|_|_|_|_|_|_|_|

|_|_|_|_|_|_|_|

|_|_|_|_|_|_|_|

|_|_|_|_|_|_|_|

|_|_|_|_|_|_|_|
Select a column to play. Must be between 0 and 6

3

 0 1 2 3 4 5 6

|_|_|_|_|_|_|_|

|_|_|_|_|_|_|_|

|_|_|_|_|_|_|_|

|_|_|_|_|_|_|_|

|_|_|_|_|_|_|_|

|_|_|_|*|_|_|_|

Select a column to play. Must be between 0 and 6

3

 0 1 2 3 4 5 6

|_|_|_|_|_|_|_|
```

```
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|+|_|_|_|
|_|_|_|*|_|_|_|
Select a column to play. Must be between 0 and 6
4
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|+|_|_|_|
|_|_|_|*|*|_|_|
Select a column to play. Must be between 0 and 6
5
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|+|_|_|_|
|_|_|_|*|*|+|_|
Select a column to play. Must be between 0 and 6
2
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
```

```
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|+|_|_|_|
|_|_|*|*|*|+|_|
Select a column to play. Must be between 0 and 6
1
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|+|_|_|_|
|_|+|*|*|*|+|_|
Select a column to play. Must be between 0 and 6
4
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|+|*|_|_|
|_|+|*|*|*|+|_|
Select a column to play. Must be between 0 and 6
3
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
```

```
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|+|_|_|_|
|_|_|_|+|*|_|_|
|_|+|*|*|*|+|_|
Select a column to play. Must be between 0 and 6
5
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|+|_|_|_|
|_|_|_|+|*|*|_|
|_|+|*|*|*|+|_|
Select a column to play. Must be between 0 and 6
2
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|+|_|_|_|
|_|_|+|+|*|*|_|
|_|+|*|*|*|+|_|
Select a column to play. Must be between 0 and 6
5
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
```

```
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|+|_|*|_|
|_|_|+|+|*|*|_|
|_|+|*|*|*|+|_|
Select a column to play. Must be between 0 and 6
4
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|+|+|*|_|
|_|_|+|+|*|*|_|
|_|+|*|*|*|+|_|
Select a column to play. Must be between 0 and 6
4
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|*|_|_|
|_|_|_|+|+|*|_|
|_|_|+|+|*|*|_|
|_|+|*|*|*|+|_|
Select a column to play. Must be between 0 and 6
2
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
```

```
|_|_|_|_|_|_|_|
|_|_|_|_|*|_|_|
|_|_|+|+|+|*|_|
|_|_|+|+|*|*|_|
|_|+|*|*|*|+|_|
Select a column to play. Must be between 0 and 6
0
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|*|_|_|
|_|_|+|+|+|*|_|
|_|_|+|+|*|*|_|
|*|+|*|*|*|+|_|
Select a column to play. Must be between 0 and 6
0
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|*|_|_|
|_|_|+|+|+|*|_|
|+|_|+|+|*|*|_|
|*|+|*|*|*|+|_|
Select a column to play. Must be between 0 and 6
1
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
```

```
|_|_|_|_|_|_|_|
|_|_|_|_|*|_|_|
|_|_|+|+|+|*|_|
|+|*|+|+|*|*|_|
|*|+|*|*|*|+|_|
Select a column to play. Must be between 0 and 6
1
 0 1 2 3 4 5 6
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|*|_|_|
|_|+|+|+|+|*|_|
|+|*|+|+|*|*|_|
|*|+|*|*|*|+|_|
Congratulations player 2. You won.
Thanks for playing!


-- program is finished running --
```


Project Stages

In order to help you be aware of your progress, it is recommended to use a series of mile markers to help you divide up the work. You can ignore these if you wish. However, if you find you need some direction, by all means follow along.

Stage 1 - Create the main loop logic and user interaction

The tedious part of this program will be to create all the strings, display them to the user, and get user input. It is also the simpler bit. During the first stage, it is suggested you focus on creating an application that prints the strings to the user, implements the main loop, and asks the user for input (don't forget to make sure the input column is valid!). At this stage you don't have to worry about saving the input, etc.

If you finish early, move on to stage 2 and try to display the board. You can edit the matrix manually to "simulate" some plays have occurred.

Stage 2 - The board

Now that you display all strings to the user, and get all information from the user, you can move on to implement the next step: displaying the board and dropping tokens as requested by the user. In this stage, create the data structure that represents the board - matrix. Implement functions that help you access the matrix. You can start by implementing the code that prints the board to the user, as it will help you debug. Then, use the user input to drop a token. When the user chooses a column, you may spiral down the column to find the first empty cell.

If you finish early, move on to stage 3 and try to find winning game conditions for horizontal and vertical 4-in-line.

Stage 3 - Winning the game

Since you know where the token was dropped, the easiest way, probably not the smartest, is to check the matrix entries around the dropped token. For example, this was the last dropped token:


```
 0 1 2 3 4 5 6

|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|*|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
```

You can check the following 4 regions of interest:

1

```
 0 1 2 3 4 5 6

|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
|_|?|?|?|*|?|?|
|_|_|_|_|_|_|_|
|_|_|_|_|_|_|_|
```

```
|_|_|_|_|_|_|_|
```

2

```
 0 1 2 3 4 5 6
|_|_|_|_|?|_|_|
|_|_|_|_|?|_|_|
|_|_|_|_|*|_|_|
|_|_|_|_|?|_|_|
|_|_|_|_|?|_|_|
|_|_|_|_|?|_|_|
```

3


```
 0 1 2 3 4 5 6
|_|_|?|_|_|_|_|
|_|_|_|?|_|_|_|
|_|_|_|_|*|_|_|
|_|_|_|_|_|?|_|
|_|_|_|_|_|_|?|
|_|_|_|_|_|_|_|
```

4

```
 0 1 2 3 4 5 6
|_|_|_|_|_|_|?|
|_|_|_|_|_|?|_|
|_|_|_|_|*|_|_|
|_|_|_|?|_|_|_|
|_|_|?|_|_|_|_|
|_|?|_|_|_|_|_|
```

Also, don't forget to stop the game if the board is full. (Just keep track of how many tokens were dropped.)

Helpful Tidbits

Starting the code

This is a very simple program in a higher-level language! But it is much more complex in assembly. As such, here is some advice for developing your program.

Plan and start by writing high-level comments on how you plan to approach the problem:

- If you are not sure what to write, start with the items in the "Your assignment" section of the project :)
- Then add detail to those comments.
- If you need, write the program in a high-level language, draw a diagram, write pseudo-code, and then translate that into MISP assembly.

Testing

DO NOT TRY TO WRITE THE WHOLE PROGRAM BEFORE TESTING IT!

- It's the easiest way to get overwhelmed and confused without knowing what to do!
- Implement small parts of the code and test them!

Split your code into functions

Use functions! They will help you manage the cognitive load. Here is a starting point!

```
main:
    jal print_welcome
    jal display_board

_main_loop:
    ...
_main_player1:
  <stuff>
    j   _main_loop
    <more stuff>
```

Submission

Submit a single ZIP file with your project named studentID_MidtermProj.zip (e.g., 2023141520000_ MidtermProj.zip). In the zip file, there should be NO folder, just the following files:

- Your connect.asm file. (Put your name and student ID at the top of the file in the comments!)
- A readme.txt file (DO NOT SUBMIT A README.DOCX/README.PDF. SUBMIT A PLAIN TEXT FILE. PLEASE.) which should contain: a) your name, b) your student ID, c) anything that does not work, d) anything else you think might help the grader grade your project more easily.

Submit into the Blackboard. Let me know immediately if there are any problems submitting your work.