In Connect Four, two players alternately insert a colored diskinto a grid of 6 rows and 7 columns. The first player who manages to line up 4 of his own colored disks consecutively or horizontally, or vertically, or diagonally wins. The game grid is stored via a 6x7 connect4 matrix, where each element occupies 1 byte and can take the following values:

Duration: 90 minutes.

It is possible to consult:

- any paper material

- the slides downloaded from the course page on the didattica website

- the code of the laboratories, if uploaded to the didattica website in the “elaborati” section

Students caught communicating with each other will be immediately asked to leave the laboratory.

* 0: No colored disk has yet been placed
* 1: Player 1 put his own colored disk (in the picture, player 1 has yellow disks)
* 2: Player 2 put his own colored disk (in the picture, player 2 has red disks)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | |  | |  | |  |  | |  | |  | |  | |  | |  | |  | |  | |  |  |  | |  | |  | |  | |  | | |  | | | | | | | | | |
|  | |  | |  | |  | | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  |  | |  | Column index | | | | | | |
|  | |  | |  | |  | | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  |  | |  | *0* | *1* | *2* | *3* | *4* | *5* | *6* |
|  | |  | |  | |  | | |  | |  | |  | |  | |  | | 0 | | 0 | | 0 | | 0 | | 0 | | 0 | | 0 | |  | |  | Row index | | *0* | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|  | |  | |  | |  | | |  | |  | |  | |  | |  | | 0 | | 0 | | 2 | | 0 | | 0 | | 0 | | 0 | |  | |  | *1* | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|  | |  | |  | |  | | |  | |  | |  | |  | |  | | 0 | | 0 | | 2 | | 1 | | 1 | | 1 | | 1 | |  | |  | *2* | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|  | |  | |  | |  | | |  | |  | |  | |  | |  | | 0 | | 1 | | 2 | | 2 | | 1 | | 1 | | 2 | |  | |  | *3* | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
|  | |  | |  | |  | | |  | |  | |  | |  | |  | | 2 | | 2 | | 2 | | 2 | | 1 | | 2 | | 1 | |  | |  | *4* | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
|  | |  | |  | |  | | |  | |  | |  | |  | |  | | 1 | | 2 | | 1 | | 1 | | 1 | | 2 | | 2 | |  | |  | *5* | 35 | 36 | 37 | 38 | 39 | 40 | 41 |

1. Game grid b) matrix connect Four c) row, column and cell index

for matrix connect Four

It is required to write a program to check if a player has won.

You are asked to create a project with Keil, replace the contents of the startup\_LPC17xx.s file with the one in the **template** directory andadd the other files in the directory. Finally, write **debugged** and **working** subroutines that meet the following 3 specifications.

*Note 1*: You should not change the code calling the subroutines. It is only required to implement subroutines. Connect4 matrix is already declared.

*Note 2*: Specifications must be run in order. You can only switch to Specification 2 after verifying that the solution to Specification 1 is working correctly. Same for Specification 3.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Column index | | | | | | |
|  |  | *0* | *1* | *2* | *3* | *4* | *5* | *6* |
| Row index | *0* | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| *1* | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| *2* | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| *3* | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| *4* | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
| *5* | 35 | 36 | 37 | 38 | 39 | 40 | 41 |

**Specification** **1** (8 points). Write a checkRow subroutine that checks if there is a horizontal sequence of 4 discs of player X in the Connect4 matrix. The subroutine receives the following parameters in the order indicated:

* ID of player X (possible values are 1 or 2)
* address of Connect4 matrix

The checkRow subroutine returns the value 1 if the horizontal sequence is present, the value 0 otherwise.

The subroutine shall comply with the ARM Architecture Procedure Call Standard (AAPCS) standard, in particular with regard to switching the parameter to input/output and saving the registers.

*Example*: in Fig. a), there are 4 colored disks horizontally placed in cells 17, 18, 19, 20. If X = 1, the return value is 1.

*Suggestion*: Note that the first element of a sequence of 4 horizontal disks can only be found in one of the cells highlighted in blue in the figure at the bottom right of the previous page.

A possible implementation of checkRow is:

* with a double cycle all cells are marked in blue.
* For each iteration of the inner loop, the value of the current cell and the 3 cells on the right are checked. If all cells have value 1, the subroutine returns 1. You do not need to use a loop to perform the 4 controls: you can use 4 CMP instructions.

**Specification 2** (6 points). Write a checkDiagonal subroutine that controls if there is a sequence of 4 diagonal disks of player 1 in the matrix Connect4 . It is required to check only the diagonals starting at the bottom left and ending at the top right. The checkDiagonal subroutine receives the following parameters in the order indicated:

* ID of player X (possible values are 1 or 2)
* address of Connect4 matrix

The checkDiagonal subroutine returns the value 1 if the diagonal sequence is present, the value 0 otherwise. The subroutine shall comply with the AAPCS standard.

*Example*: in Fig. a), there are 4 colored disks diagonally in cells 38, 32, 26, 20. If X = 1, the return value is 1.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Column index | | | | | | |
|  |  | *0* | *1* | *2* | *3* | *4* | *5* | *6* |
| Row index | *0* | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| *1* | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| *2* | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| *3* | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| *4* | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
| *5* | 35 | 36 | 37 | 38 | 39 | 40 | 41 |
|  |  |  |  |  |  |  |  |  |

*Suggestion:* Note that the first element of a sequence of 4 diagonal colored disks can only be found in one of the cells highlighted in blue in the figure on the right.

A possible implementation of checkDiagonal is:

* with a double cycle all cells are marked in blue.
* For each iteration of the inner loop, the value of the current cell and the 3 cells are checked diagonally. If all cells have value 1, the subroutine returns 1. You do not need to use a loop to perform the 4 controls: you can use 4 CMP instructions.

**Specification** **3 (**5 points). Write appropriate instructions / C functions to implement the following features when pressing the buttons of the LPC 1768 board:

* INT0 key: switch off all LEDs.
* KEY1 key: Check if player 1 has won in the Connect4 grid represented with the grid variable (defined in the sample.c file). For the check, subroutines in ARM must be invoked as per specification 1 and 2. Possible outputs are:
  + Player 1 did not win: LED 11 must be turned on
  + Player 1 has won with a horizontal sequence: LED 10 must be turned on
  + Player 1 won with a diagonal sequence: LED 9 must be turned on.
* KEY2 key: Check if player 2 has won in the Connect4 grid represented with the grid variable (defined in the sample.c file). For the check, subroutines in ARM must be invoked as per specification 1 and 2. Possible outputs are:
  + Player 2 did not win: LED 7 must be turned on
  + Player 2 won with a horizontal sequence: LED 6 must be turned on
  + Player 2 won with a diagonal sequence: LED 5 must be turned on.

If a player has won at the same time with a horizontal sequence and a diagonal sequence, you can turn on any of the two corresponding LEDs (for example, LED 10 or 9 in case of player 1 victory).