ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

KIẾN TRÚC MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN

FIVE IN A ROW

LỚP TN01 – HK242

Giảng viên hướng dẫn: Nguyễn Thành Lộc
Sinh viên thực hiện: Nguyễn Hữu Thịnh 2313292
Nguyễn Chí Thanh 2313078
Nguyễn Thái Sơn 2312968

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 5/2025

Mục lục

1 Giới thiệu

Báo cáo này trình bày chi tiết việc triển khai trò chơi Gomoku bằng ngôn ngữ lập trình MIPS assembly, theo yêu cầu của bài tập. Gomoku là một trò chơi chiến thuật dành cho hai người với lịch sử lâu đời, trong đó mục tiêu là tạo thành một đường thẳng liền mạch gồm năm quân cờ - theo chiều ngang, dọc hoặc chéo - trên một bàn cờ kích thước 15x15. Trò chơi nổi tiếng với luật chơi đơn giản nhưng chiến lược sâu sắc, khiến nó trở thành một ứng cử viên lý tưởng để khám phá logic tính toán và quản lý trạng thái trò chơi.

Yêu cầu của bài tập bao gồm:

- **Khởi tạo và hiển thị bàn cờ:** Thiết lập và vẽ bàn cờ kích thước 15x15.
- **Luân phiên lượt chơi:** Xen kẽ lượt chơi giữa hai người chơi, với Người chơi 1 được đại diện bởi "X" và Người chơi 2 bởi "O".
- **Nhập nước đi:** Yêu cầu người chơi nhập nước đi theo định dạng "x,y", đồng thời kiểm tra tính hợp lệ và khả dụng của nước đi trên bàn cờ.
- **Phát hiện chiến thắng:** Kiểm tra chiến thắng bằng cách xác định năm ký hiệu liên tiếp theo hàng, cột hoặc đường chéo.
- **Phát hiện hòa:** Nhận diện trường hợp hòa khi bàn cờ được lấp đầy mà không có người chiến thắng.
- **Xuất kết quả:** Ghi lại trạng thái cuối cùng của bàn cờ và kết quả trò chơi vào tệp "result.txt".

Báo cáo này tập trung vào việc giải thích các thuật toán và logic đằng sau quá trình triển khai, cung cấp cái nhìn sâu sắc về cách thiết kế từng thành phần của trò chơi và cách chúng tích hợp để tạo nên một ứng dụng hoạt động thống nhất. Các phần sau sẽ trình bày cấu trúc chương trình, các thuật toán được sử dụng cho các chức năng quan trọng, cấu trúc dữ liệu được tận dụng, cũng như các cơ chế xử lý lỗi nhằm đảm bảo trải nghiệm người dùng mượt mà.

2 Cấu trúc chương trình

Trò chơi Gomoku được xây dựng dựa trên một Máy trạng thái hữu hạn (FSM) nhằm điều khiển luồng của trò chơi, đảm bảo rằng mỗi giai đoạn—khởi tạo, lượt chơi của người chơi, và kết thúc—đều được xử lý một cách hệ thống. FSM mang lại một cách tiếp cận rõ ràng và có tổ chức trong việc quản lý tiến trình trò chơi, giúp cho chương trình trở nên mô-đun và dễ bảo trì.

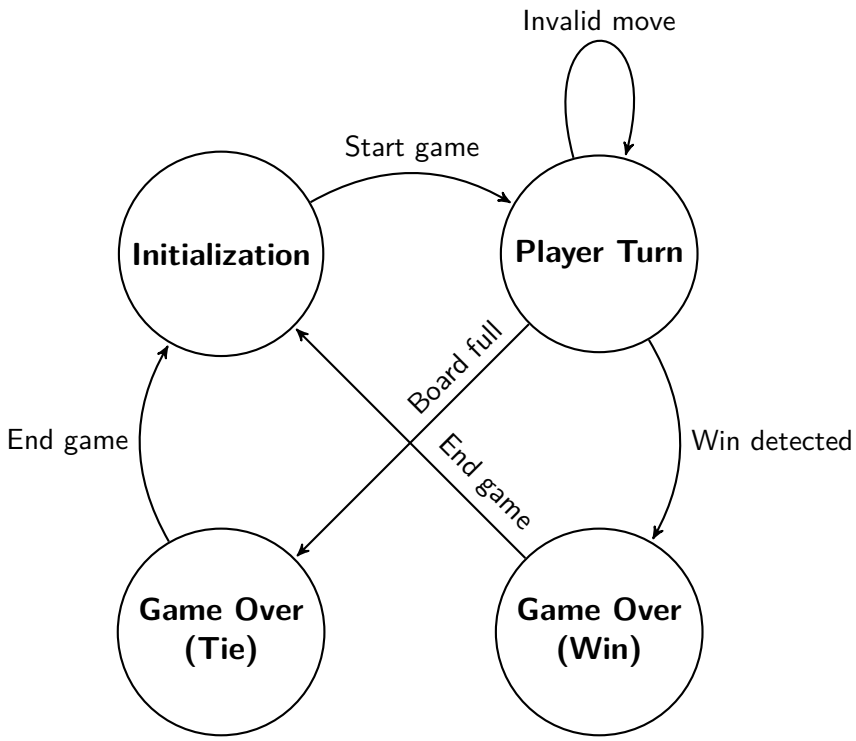
Chương trình được chia thành một số thủ tục chính, mỗi thủ tục đảm nhận một vai trò riêng biệt, như khởi tạo bàn cờ, xử lý đầu vào từ người chơi, cập nhật trạng thái bàn cờ, kiểm tra điều kiện thắng hoặc hòa, và quản lý đầu ra. Các thủ tục này được gọi tương ứng trong các trạng thái của FSM, tạo ra sự chuyển đổi liền mạch giữa các phần của trò chơi.

2.1 Máy trạng thái hữu hạn (FSM)

FSM bao gồm bốn trạng thái chính:

- **Initialization**: Thiết lập bàn cờ và các biến khởi đầu của trò chơi.
- **Player Turn**: Quản lý việc nhận và xác thực nước đi từ người chơi hiện hành.
- **Game Over (Win)**: Xử lý trường hợp khi một người chơi chiến thắng.
- **Game Over (Tie)**: Xử lý trường hợp trò chơi kết thúc hòa khi bàn cờ đầy mà không có người thắng.

Sự chuyển đổi giữa các trạng thái được điều khiển bởi các điều kiện cụ thể, như việc phát hiện nước thắng hoặc khi bàn cờ không còn vị trí trống. Hình **??** minh họa FSM với các trạng thái và mối quan hệ chuyển tiếp giữa chúng.



Hình 2.1: Máy trạng thái hữu hạn của trò chơi Gomoku

Trong FSM:

- Trò chơi bắt đầu ở trạng thái **Khởi tạo**, nơi bàn cờ được thiết lập và người chơi đầu tiên được chọn.
- Tiếp theo, trò chơi chuyển sang trạng thái **Lượt chơi**, nơi người chơi thực hiện lần lượt các nước đi. Nếu nước đi không hợp lệ, trạng thái sẽ yêu cầu người chơi thử lại. Sau mỗi nước đi hợp lệ, chương trình sẽ kiểm tra xem người chơi có chiến thắng hay bàn cờ đã đầy.

- Nếu phát hiện có chiến thắng, FSM sẽ chuyển sang trạng thái **Kết thúc trò chơi (Thắng)** để thông báo người chiến thắng và kết thúc trò chơi.
- Nếu bàn cờ được lấp đầy mà không có chiến thắng, chương trình chuyển sang trạng thái **Kết thúc trò chơi (Hòa)** để thông báo trận đấu hòa.

2.2 Các thủ tục chính

Chức năng của chương trình được xây dựng qua một số thủ tục chính, mỗi thủ tục đảm nhận một nhiệm vụ cụ thể trong trò chơi:

- `INIT`: Khởi tạo bàn cờ 15x15 bằng cách thiết lập tất cả các vị trí với ký hiệu '.' (trống). Gán 1 cho `player` để xác định người chơi đầu tiên. Gán 0 cho biến `move_count` để theo dõi số nước đi đã thực hiện.
- `PRINT_BOARD`: Hiển thị trạng thái hiện tại của bàn cờ lên màn hình cùng với các nhãn hàng và cột.
- `GET_MOVE`: Yêu cầu người chơi nhập nước đi, đọc đầu vào, và kiểm tra tính hợp lệ thông qua `parse_input`.
- `parse_input`: Phân tích chuỗi đầu vào để trích xuất tọa độ, đồng thời kiểm tra định dạng, phạm vi hợp lệ và tính khả dụng của vị trí trên bàn cờ.
- `UPDATE_BOARD`: Cập nhật bàn cờ bằng cách ghi ký hiệu của người chơi ('X' hoặc 'O') vào vị trí đã chọn.
- `check_win`: Kiểm tra xem nước đi gần nhất có tạo thành chuỗi năm ký hiệu liên tiếp theo hàng, cột hoặc đường chéo hay không.
- `check_tie`: Xác định xem trò chơi có kết thúc hòa khi bàn cờ đã đầy mà không có chiến thắng.
- `SWITCH_PLAYER_TURN`: Chuyển lượt của người chơi giữa Người chơi 1 và Người chơi 2.
- `win_process, tie_process`: Ghi lại trạng thái cuối cùng của bàn cờ và kết quả trò chơi vào file "result.txt".

Các thủ tục này được gọi trong các trạng thái của FSM như sau:

- **Initialization**: Sử dụng `INIT` để thiết lập bàn cờ.
- **Player Turn**: Gọi `PRINT_BOARD`, `GET_MOVE`, `UPDATE_BOARD`, `check_win` và `check_tie`.
- **Game Over (Win)**: Gọi `PRINT_BOARD` và `win_process` để hiển thị kết quả.
- **Game Over (Tie)**: Tương tự, gọi `PRINT_BOARD` và `tie_process`.

Sau khi kết thúc trò chơi, người dùng có thể chọn chơi lại hoặc thoát chương trình.

3 Cấu trúc dữ liệu

Trong việc triển khai trò chơi Gomoku, bàn cờ được biểu diễn dưới dạng một mảng gồm 752 byte trong bộ nhớ, bao gồm cả hàng đầu hiển thị nhãn cột và 15 hàng ứng với lưới trò chơi 15x15. Mảng này, được gọi là `board`, được thiết kế nhằm mô phỏng giao diện hiển thị của terminal, bao gồm các ký tự xuống dòng, tạo điều kiện thuận lợi cho việc trực quan hóa trạng thái của trò chơi.

Mảng `board` được cấu trúc thành 16 dòng, mỗi dòng gồm 47 byte, trong đó có ký tự xuống dòng (\n) ở cuối dòng. Điều này tạo nên tổng kích thước là $16 \times 47 = 752$ byte, phù hợp với đặc tả đã cung cấp. Bố cục của mảng có thể được hình dung như một mảng hai chiều với kích thước 47×16, trong đó:

- **Dòng 0:** Là header, hiển thị nhãn cột từ 0 đến 14. Dòng này bắt đầu với ba khoảng trắng để căn lề, sau đó liệt kê các số cột, được định dạng sao cho phù hợp với các ô chơi bên dưới. Nội dung ban đầu là:

  ␣␣␣0␣␣1␣␣2␣␣3␣␣4␣␣5␣␣6␣␣7␣␣8␣␣9␣10␣11␣12␣13␣14\n

- **Các dòng 1 đến 15:** Đại diện cho các hàng từ 0 đến 14 của bàn cờ. Mỗi dòng bắt đầu với nhãn hàng (ví dụ, “0 ” cho hàng 0, “10 ” cho hàng 10), sau đó là 15 ô chơi và kết thúc bằng ký tự xuống dòng. Ví dụ, hàng 0 được khởi tạo như sau:

  0␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.\n

  Trong mỗi dòng, nhãn của hàng chiếm 3 byte đầu tiên, đảm bảo căn lề nhất quán:

  – Đối với các hàng 0 đến 9, nhãn là số của hàng theo sau là hai khoảng trắng (ví dụ “1␣␣”).
  – Đối với các hàng 10 đến 14, nhãn là số có hai chữ số theo sau là một khoảng trắng (ví dụ “10␣”).

  Sau nhãn, mỗi ô trong 15 ô được biểu diễn bởi chuỗi 3 byte: hai khoảng trắng ở đầu, ký tự nội dung của ô (’.’, ’X’, hoặc ’O’). Ví dụ, ô trống được biểu diễn là “.”, trong khi ô chứa quân của Người chơi 1 được biểu diễn là “X”. Định dạng này đảm bảo rằng ký tự trong ô được căn chỉnh đồng nhất, hỗ trợ việc hiển thị và thao tác. Hình **??** minh họa cách mà mảng `board` được tổ chức trong bộ nhớ, với các ký tự đại diện cho các ô chơi và nhãn hàng/cột được căn chỉnh để dễ đọc.

Các thủ tục `put` và `get` được thiết kế để tương tác với bàn cờ này bằng cách đặt và lấy quân cờ của người chơi (với ‘`X`’ dành cho Người chơi 1 và ‘`O`’ dành cho Người chơi 2) tại các tọa độ cụ thể trong `board`.

Thủ tục `put` chịu trách nhiệm đặt quân cờ của người chơi vào vị trí xác định trên bàn cờ. Nó nhận ba đối số:

- `$a0`: Tọa độ x (ngang, cột), có giá trị từ 0 đến 14.
- `$a1`: Tọa độ y (dọc, hàng), có giá trị từ 0 đến 14.

```
␣␣␣0␣␣1␣␣2␣␣3␣␣4␣␣5␣␣6␣␣7␣␣8␣␣9␣10␣11␣12␣13␣14\n
0␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.\n
1␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.\n
2␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.\n
3␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣X␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.\n
4␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣O␣␣X␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.\n
5␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣O␣␣O␣␣O␣␣X␣␣.␣␣.␣␣.␣␣.␣␣.\n
6␣␣.␣␣.␣␣.␣␣.␣␣X␣␣.␣␣X␣␣O␣␣O␣␣X␣␣.␣␣.␣␣.␣␣.␣␣.\n
7␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣X␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.\n
8␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣O␣␣.␣␣.␣␣.\n
9␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.\n
10␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.\n
11␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.\n
12␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.\n
13␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.\n
14␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.␣␣.\n
```

Hình 3.1: Mảng dữ liệu của bàn cờ

- `$a2`: Ký tự cần đặt, có thể là ‘X’ hoặc ‘O’.

Thủ tục `put` tính toán một độ lệch bộ nhớ trong mảng `board` và lưu ký tự vào vị trí tương ứng. Cụ thể như sau:

```
put:
    li $t0, 50
    mul $a0, $a0, 47
    mul $a1, $a1, 3
    add $t0, $a0, $t0
    add $t0, $a1, $t0
    sb $a2, board($t0)

    jr $ra
```

Offset được tính theo công thức: $50 + x * 47 + y * 3$. Hằng số 50 đại diện cho offset cơ sở, tức là vị trí bắt đầu của ô chơi đầu tiên (hàng 0, cột 0) sau khi đã loại trừ hàng header và định dạng ban đầu ở hàng chơi đầu tiên. Nhân tọa độ `x` với 47 tính đến sự đóng góp của vị trí cột trong bố cục theo hàng, khi mỗi hàng chiếm 47 byte. Nhân tọa độ `y` với 3 điều chỉnh vị trí trong hàng, vì mỗi ô chơi có sự dịch chuyển vị trí cố định. Offset cuối cùng, được lưu trong `$t0`, chỉ định chính xác byte trong mảng `board` nơi quân cờ sẽ được đặt. Lệnh `sb` (store byte) ghi ký tự từ `$a2` vào địa chỉ bộ nhớ `board($t0)`. Ví dụ: Tại tọa độ (0, 0), offset là 50, tương ứng với vị trí của dấu chấm đầu tiên trong hàng “`0␣␣.␣␣(...)`”.

Thủ tục `get` truy xuất ký tự tại vị trí cụ thể của bàn cờ, cho phép logic trò chơi kiểm tra trạng thái của ô (ví dụ: để xác định điều kiện chiến thắng). Thủ tục này nhận hai đối số:

- `$a0`: Tọa độ x (cột).

- `$a1`: Tọa độ y (hàng).

Giá trị được trả về:

- `$v0`: Ký tự tại vị trí đó (có thể là 'X', 'O', hoặc '.' nếu ô trống).

Việc hiện thực của `get` tương tự thủ tục `put` về tính toán độ lệch. Công thức $50 + x \times 47 + y \times 3$ giống hệt như trong `put`, đảm bảo nhất quán trong truy cập các vị trí của bàn cờ. Lệnh `lb` (load byte) đọc ký tự từ `board($t0)` và lưu vào `$v0`, sau đó trả về cho caller.

```
get:
    li $t0, 50
    mul $a0, $a0, 47
    mul $a1, $a1, 3
    add $t0, $a0, $t0
    add $t0, $a1, $t0
    lb     $v0, board($t0)

    jr $ra
```

4 Các giải thuật và logic

4.1 Nhận tọa độ từ người chơi

1. **Nhận tọa độ di chuyển của người chơi trong trò chơi Gomoku**: Chương trình thực hiện một quy trình có cấu trúc để đảm bảo rằng đầu vào được định dạng đúng, nằm trong phạm vi hợp lệ và trùng với một vị trí trên bàn cờ chưa bị chiếm.

2. **Hiển thị thông báo yêu cầu người chơi**:

   - Đối với Người chơi 1: "Người chơi 1, vui lòng nhập tọa độ của bạn: "
   - Đối với Người chơi 2: "Người chơi 2, vui lòng nhập tọa độ của bạn: "

```
    # Display prompt based on current player
    li $v0, 4          # Syscall to print string
    beq $s0, 1, print_prompt1
    la $a0, prompt2    # "Player 2, please input your coordinates: "
    j print_prompt
print_prompt1:
    la $a0, prompt1    # "Player 1, please input your coordinates: "
print_prompt:
    syscall
```

3. **Đọc đầu vào**:

- Chương trình nhận đầu vào từ người dùng dưới dạng chuỗi thông qua một lệnh gọi hệ thống.
- Dữ liệu nhận được được lưu vào một bộ đệm để xử lý tiếp.

```
    # Read input string
    li $v0, 8          # Syscall to read string
    la $a0, input_buffer
    li $a1, 10         # Max length of input
    syscall
```

4. **Phân tích đầu vào**:

- Phân tách chuỗi để trích xuất tọa độ x và y theo định dạng "x,y", trong đó x và y là số nguyên từ 0 đến 14.
- **Trích xuất x**: Đọc các ký tự trước dấu phẩy và kiểm tra rằng chúng đều là chữ số, xây dựng giá trị x bằng cách nhân giá trị hiện tại với 10 và cộng thêm chữ số mới.
- **Bỏ qua dấu phẩy**: Di chuyển qua dấu phẩy để xử lý tiếp phần tọa độ y.
- **Trích xuất y**: Đọc các chữ số sau dấu phẩy cho đến khi gặp ký tự xuống dòng hoặc ký tự kết thúc, xây dựng giá trị y tương tự như x.
- Nếu bất kỳ ký tự nào không phải là số hoặc định dạng không đúng (ví dụ, thiếu dấu phẩy), đầu vào sẽ bị xem là không hợp lệ.

```
parse_x:
    lb $t0, 0($s0)    # Load current character
    beq $t0, 44, end_parse_x  # If comma (ASCII 44), end x parsing
    beq $t0, 0, invalid  # If null, invalid
    beq $t0, 10, invalid # If newline, invalid

    # Check if character is a digit (0-9)
    blt $t0, 48, invalid  # < '0'
    bgt $t0, 57, invalid  # > '9'

    # Update x = x * 10 + (char - '0')
    mul $s1, $s1, 10  # x *= 10
    sub $t0, $t0, 48  # Convert char to integer
    add $s1, $s1, $t0 # x += digit

    addi $s0, $s0, 1  # Move to next character
    j parse_x

end_parse_x:
    addi $s0, $s0, 1  # Skip comma
    # Step 2: Parse y (digits after comma)
parse_y:
    lb $t0, 0($s0)    # Load current character
    beq $t0, 10, end_parse_y  # If newline, end y parsing
    beq $t0, 0, end_parse_y   # If null, end y parsing

```

```
        # Check if character is a digit
        blt $t0, 48, invalid  # < '0'
        bgt $t0, 57, invalid  # > '9'

        # Update y = y * 10 + (char - '0')
        mul $s2, $s2, 10  # y *= 10
        sub $t0, $t0, 48  # Convert char to integer
        add $s2, $s2, $t0 # y += digit

        addi $s0, $s0, 1  # Move to next character
        j parse_y

    end_parse_y:
```

5. **Kiểm tra tính hợp lệ của tọa độ**:
   - **Kiểm tra phạm vi**: Xác nhận rằng cả x và y đều nằm trong phạm vi từ 0 đến 14, tương ứng với bàn cờ 15x15.
   - **Kiểm tra vị trí**: Đảm bảo rằng vị trí tại tọa độ (x, y) trên bàn cờ đang trống.

```
        blt $s1, 0, invalid   # x < 0
        bgt $s1, 14, invalid  # x > 14
        blt $s2, 0, invalid   # y < 0
        bgt $s2, 14, invalid  # y > 14

        # Step 4: Check if board position is empty
        get($s1, $s2, $v0)
        bne $v0, empty, invalid  # If not empty, position is occupied

        # Step 5: Valid input, return x, y
        move $v0, $s1         # Return x
        move $v1, $s2         # Return y
        j parse_input_exit
```

6. **Xử lý đầu vào không hợp lệ**: Nếu đầu vào không hợp lệ do định dạng sai như “a,b” hoặc ‘4’, giá trị vượt phạm vi như ‘15,2’, hoặc vị trí đã bị chiếm, chương trình sẽ hiển thị thông báo lỗi ‘Đầu vào không hợp lệ, vui lòng nhập lại’ và quay lại bước 1 để yêu cầu người chơi nhập lại.

```
        # Check if input is invalid
        li $t0, -1
        beq $v0, $t0, invalid_input

        # Input is valid, return x,y
        # x already in $v0
        # y in $v1
        j get_move_exit

    invalid_input:
        # Print error message
        li $v0, 4
```

```
13      la $a0, error_msg # "Invalid input, try again"
14      syscall
15      j input_loop      # Re-prompt
```

7. **Trả về tọa độ hợp lệ**: Khi đầu vào vượt qua tất cả các kiểm tra, chương trình trả về tọa độ (x, y) để sử dụng trong logic trò chơi, ghi nhận nước đi của người chơi trên bàn cờ.

Quy trình trên đảm bảo rằng chỉ những nước đi hợp lệ được chấp nhận, đáp ứng các yêu cầu của bàn cờ 15x15 và định dạng đầu vào dự kiến, mang lại trải nghiệm người dùng thân thiện và chính xác.

4.2 Kiểm tra chiến thắng

1. **Tham số đầu vào và kết quả trả về:** Hàm nhận 2 tham số tọa độ x và y của lượt vừa đi của người chơi để kiểm tra điều kiện chiến thắng rồi trả về giá trị true hoặc false.
2. **Lưu thanh ghi vào stack trước khi xử lí:** Lưu trữ dữ liệu của các thanh ghi tham số `$a0`, `$a1`, thanh ghi `$ra` và `$s0`, . . . để sử dụng nhằm tránh mất mát dữ liệu khi gọi hàm.

```
1   addi $sp, $sp, -32
2   sw $ra, 0($sp)
3   sw $a0, 4($sp)
4   sw $a1, 8($sp)
5   sw $s0, 12($sp)
6   sw $s1, 16($sp)
7   sw $s2, 20($sp)
8   sw $s3, 24($sp)
9   sw $s4, 28($sp)
```

3. **Lấy giá trị tại tọa độ của bàn cờ:** gọi macro `get` để lấy giá trị của bàn cờ tại tọa độ x và y.

```
1       get($a0, $a1, $s0)
```

4. **Kiểm tra điều kiện thắng của trò chơi:**
   - Từ tọa độ của người ta phải kiểm tra 4 hướng chính như sau: $\searrow \nearrow \uparrow \rightarrow$. Hàm sẽ sử dụng vòng lặp để chạy 4 hướng chính sau cũng như tạo sẵn 2 mảng dùng để dễ dàng tăng thêm theo hướng cho tọa độ x, y đang đứng.
   - Dùng vòng lặp để chạy cả 2 chiều theo chiều dương và chiều âm của hướng để kiểm tra lần lượt các điều kiện sau để dừng vòng lặp:
     - Ô đang đứng có phải là vị trí hợp lệ hay không, tức là kiểm tra $0 \leq x \leq 14$ và $0 \leq y \leq 14$

  - So sánh giá trị của bàn cờ tại tọa độ đang đứng có bằng với giá trị của người chơi hay không. *Ví dụ*: Hàm có đầu vào ban đầu là (x_0, y_0) có giá trị trên bàn cờ là **X**, sau khi chạy vòng lặp tới ô (x_1, y_1) nếu có giá trị bàn cờ là **X** thì tiếp tục còn là **O** thì dừng.
- Sau khi kiểm tra điều kiện thỏa mãn, ta tăng tổng ô liên tiếp 1 đơn vị và kiểm tra thêm nếu tổng lớn hơn bằng 5 thì ngừng lặp và trả về kết quả true. Nếu không thì tiếp tục vòng lặp. Khi hết mỗi hướng đi thì ta reset lại tổng đó về 1 để đếm lại.
- Sau khi hết vòng lặp mà chưa trả về kết quả thì hàm trả về fasle.

```
    #$s0 = Player
    get($a0, $a1, $s0)
    #$s1 = loop count
    li $s1, 4
check_win_for:
    beqz $s1, check_win_exit
    addi $s1, $s1, -1
    #$s2 = count
    li $s2, 1
#Direction (+)
    lw $s3, 4($sp)
    lw $s4, 8($sp)
check_win_loop1:
sll $t1, $s1, 2
la $t0, Dx
add $t0, $t0, $t1
lw $t0, 0($t0)
add $s3, $s3, $t0
la $t0, Dy
add $t0, $t0, $t1
lw $t0, 0($t0)
add $s4, $s4, $t0

li $t0, 14
blt $t0, $s3, check_win_exit1
blt $t0, $s4, check_win_exit1
bltz $s3, check_win_exit1
bltz $s4, check_win_exit1
get($s3, $s4, $t0)
bne $t0, $s0, check_win_exit1

addi $s2, $s2, 1
li $t0, 4
blt $t0, $s2, check_win_true
j check_win_loop1

check_win_exit1:
#Direction (-)
    lw $s3, 4($sp)
    lw $s4, 8($sp)
check_win_loop2:
sll $t1, $s1, 2
la $t0, Dx
```

```
44    add $t0, $t0, $t1
45    lw $t0, 0($t0)
46    sub $s3, $s3, $t0
47    la $t0, Dy
48    add $t0, $t0, $t1
49    lw $t0, 0($t0)
50    sub $s4, $s4, $t0
51
52    li $t0, 14
53    blt $t0, $s3, check_win_exit2
54    blt $t0, $s4, check_win_exit2
55    bltz $s3, check_win_exit2
56    bltz $s4, check_win_exit2
57    get($s3, $s4, $t0)
58    bne $t0, $s0, check_win_exit2
59
60    addi $s2, $s2, 1
61    li $t0, 4
62    blt $t0, $s2, check_win_true
63    j check_win_loop2
64
65    check_win_exit2:
66    j check_win_for
67
68    check_win_exit:
69    li $v0, 0
70    j check_win_return
71
72    check_win_true:
73    li $v0, 1
74    check_win_return:
```

5. **Load dữ liệu của thanh ghi trong stack sau khi xử lí:** Load dữ liệu của các thanh ghi `$ra` và `$s0`, . . . nhằm tránh mất mát dữ liệu khi gọi hàm.

```
1       lw $ra, 0($sp)
2       lw $s0, 12($sp)
3       lw $s1, 16($sp)
4       lw $s2, 20($sp)
5       lw $s3, 24($sp)
6       lw $s4, 28($sp)
7       addi $sp, $sp, 32
```

5 Kết luận