**LAB6 - A Game: Dots and Boxes**

**Introduction**

Ever had a couple of minutes to spare? We all have. Chances are, during many such moments as a child, you decided to play a game. In this lab, you are going to write a command-line version of [this game](http://www.math.ucla.edu/~tom/Games/dots&boxes.html).

**Rules**

Dots and Boxes is a two-player game played on a variable-size grid of dots. For this assignment, the playing board is a 4x4 grid of dots, and the two players are denoted by two symbols: 1 and 2.

The rules are as follows:

* Player 1 always goes first.
* Each player takes a turn drawing a horizontal or vertical line between two adjacent dots.
* A player cannot draw a line between two dots that are already connected by a line.
* If a player completes a box (or boxes) by drawing the fourth side of it, the player writes his number (1 or 2) in the box and gets to draw another line.
* The game ends when all the adjacent dots have been connected with horizontal or vertical lines.
* The player with the most boxes at the end of the game wins.

**Game Board**

The game board is a 4x4 grid of dots, which are represented by the ASCII character '\*' (asterisk, ASCII code x002A). Vertical lines are represented by the ASCII character '|' (pipe, ASCII code x007C), and horizontal lines are represented by the ASCII character '-' (hyphen, ASCII code x002D). The rows of the grid are labeled with numbers 0 to 6, and the columns of the grid are labeled with letters A to G. When the game starts, the board looks as follows:

ABCDEFG

0 \* \* \* \*

1

2 \* \* \* \*

3

4 \* \* \* \*

5

6 \* \* \* \*

A move is specified by a two-character input pair designating the space between two adjacent dots. The first character should be a capital letter (A-G) specifying the desired column, and the second character should be a number (0-6) specifying the desired row. For example, an input move of A1 would result in the following board:

ABCDEFG

0 \* \* \* \*

1 |

2 \* \* \* \*

3

4 \* \* \* \*

5

6 \* \* \* \*

If the next move was D4, the board would now look as follows:

ABCDEFG

0 \* \* \* \*

1 |

2 \* \* \* \*

3

4 \* \*-\* \*

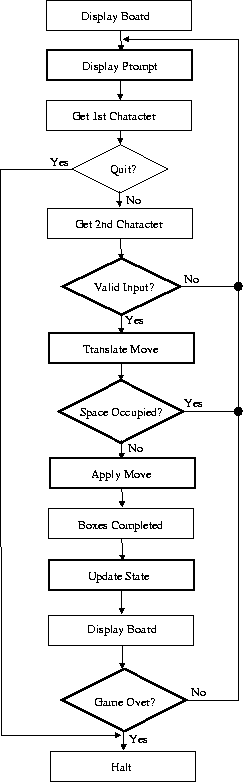
5

6 \* \* \* \*

The initial data structure of the game board is stored in memory using 7 .STRINGZ pseudo-ops, one for each row. During the execution of the program, we will modify the contents of these memory locations to reflect each player's move.

**Algorithm**

In order to design a Dots and Boxes game, or any other software project, we must break the problem into small pieces. To this effect, we have created the following flow chart for Dots and Boxes.



You will use this flowchart as the basis for designing the program. The blocks that you must implement are in bold.

**Your Job**

An example of the input/output for a game being played can be found [here](http://acsa.ustc.edu.cn/ics/downloads/lab5/example1.txt) and [here](http://acsa.ustc.edu.cn/ics/downloads/lab5/example2.txt). The following [starter code](http://acsa.ustc.edu.cn/ics/downloads/lab5/dots.asm) provides a general framework for the Dots and Boxes game, plus a few subroutines that we have provided for you. Your job is to complete the Dots and Boxes game by writing the following subroutines:

**DISPLAY\_PROMPT**

This subroutine prompts the current player for an input move. The current player, 1 or 2, is defined by the contents of the memory location labeled CURRENT\_PLAYER, which is initially set to 1.

Memory[CURRENT\_PLAYER] = 1 for player 1

2 for player 2

Based on the current player, this subroutine should print one of the following messages

Player 1, input a move (or 'Q' to quit):

Player 2, input a move (or 'Q' to quit):

**IS\_INPUT\_VALID**

This subroutine checks to see if the player's move is valid.

When this subroutine is called, R0 contains the ASCII code for the second character the player typed (the row), and R1 contains the ASCII code for the first character the user typed (the column). This subroutine

* checks if the character in R0 is between '0' and '6'. If it is not, the input is invalid.
* checks if the character in R1 is between 'A' and 'G'. If it is not, the input is invalid.
* checks if the (row, column) pair specified by (R0, R1) is the position corresponding to a '\*' (asterisk) in the game board. If it is, the input is invalid.
* some other invalid cases ...
* If the input is valid, the value 0 (zero) is returned in R3.
* If the input is invalid, the value -1 is returned in R3.

**TRANSLATE\_MOVE**

When this subroutine is called, R0 contains the ASCII code for a row number ('0'-'6'), and R1 contains the ASCII code for a column letter ('A'-'G'). This subroutine should

* convert the ASCII code for the row number to the binary representation of the number (0-6) and return the result in R0.
* convert the ASCII code for the column letter to a binary number between 0 and 6 (where 'A' corresponds to 0, 'B' corresponds to 1, ... , 'G' corresponds to 6) and return the result in R1.

For example, if R0 contains the ASCII code for '5' (x0035) and R1 contains the ASCII code for 'C' (x0043) when the subroutine is called, then R0 should contain the value 5 and R1 should contain the value 2 when the subroutine returns.

**IS\_OCCUPIED**

When this subroutine is called, R0 contains a row number (0-6, which you have previously provided from TRANSLATE\_MOVE), and R1 contains a column number (0-6, which also came from TRANSLATE\_MOVE). You should check the board position corresponding to the (row, column) pair and see if this position is already occupied.

* If the position is occupied, you should return the value -1 in R3.
* If the position is not occupied, you should return the value 0 (zero) in R3.

**NOTE:** If it is not occupied the ASCII code for space (x0020) will be stored there.The values in R0 and R1 when the subroutine returns MUST be EXACTLY the same as they were when the subroutine was called.

**GET\_ADDRESS**

This subroutine determines the address in memory of the position specified by a (row, column) pair. When this subroutine is called, R0 contains a row number (0-6) and R1 contains a column number (0-6). When this subroutine returns, R3 should contain the address in the game board data structure corresponding to this (row, column).

**NOTE:** The string for each row takes up 8 consecutive locations in memory (7 for the characters in the string + 1 for the NULL termination). Since in our assembly file we have placed the 7 strings for the board sequentially, this means that each row is 8 memory locations apart. Suppose that R5 contains the starting address of the the string for row 0. If you add 24 (i.e., 8\*3) to R5, what address do you now have in R5? If you now add 4 to R5, what is the significance of the value in R5? **You will probably need to call this subroutine from many of the other subroutines.**

**APPLY\_MOVE**

This subroutine applies a player's move to the game board by writing either a '-' (hyphen, ASCII code x002D) or a '|' (pipe, ASCII code x007C) in the correct position.

When this subroutine is called, R0 contains a row number (0-6) and R1 contains a column number (0-6) that have been provided by TRANSLATE\_MOVE. The position specified by the (row, column) pair is guaranteed to be unoccupied (if you have written IS\_OCCUPIED correctly). This subroutine should

* based on the (row, column) pair, determine if the character to be written should be a '-' or a '|'.
* write the character into the correct location in the game board data structure.

**NOTE:** You do not need to do error checking on the (row, column) pair; this has already been done by IS\_INPUT\_VALID. The values in R0 and R1 when the subroutine returns MUST be EXACTLY the same as they were when the subroutine was called.

**FILL\_BOX**

This subroutine writes the current player's number in a completed box and will be called from the BOXES\_COMPLETED subroutine (which we have provided).

When this subroutine is called, R0 contains a row number (0-6) and R1 contains a column number (0-6). The (row, column) pair specifies the position of the center of a box that has just been completed. This subroutine should

* determine which player completed the box.
* write the ASCII code for the player's number in the center of the completed box.

**NOTE:** The values in R0 and R1 when the subroutine returns MUST be EXACTLY the same as they were when the subroutine was called.

**UPDATE\_STATE**

This subroutine updates the score and player, if necessary. When this subroutine is called, R0 contains the number of boxes that the current player has just completed. This value is provided by the BOXES\_COMPLETED subroutine (which we have provided), and will be either 0, 1, or 2. This subroutine should

* update the memory location containing the current player's score to reflect the number of boxes he has just completed.
  + The score for player 1 is stored in the memory location labeled SCORE\_PLAYER\_ONE.
  + The score for player 2 is stored in the memory location labeled SCORE\_PLAYER\_TWO.
* update the contents of the memory location CURRENT\_PLAYER to reflect which player's turn is next. Remember that if a player has just completed a box, he gets to go again.

**IS\_GAME\_OVER**

This subroutine determines if the game is over or not. The game ends when all of the boxes on the game board have been completed.

* If the game is over, you should
  + determine the winner.
  + print one of the following messages
* Game over. Player 1 is the winner!

Game over. Player 2 is the winner!

* + return the value 0 (zero) in R3.
* If the game is not over yet, you should return the value -1 in R3.

**HINT:** What will the combined score of the two players be when the game is over?

We have provided the following subroutines for you:

**DISPLAY\_BOARD**

This subroutine prints the game board and the current score for both players to the screen.

**BOXES\_COMPLETED**

This subroutine determines if a box (or two) was completed in this move.

When this subroutine is called, R0 contains a row number (0-6) and R1 contains a column number (0-6) that correspond to the move that was just applied to the game board. For each box that was completed, FILL\_BOX is called with the (row, column) pair that specifies the center of the completed box.

**NOTE:** You must correctly implement FILL\_BOX for BOXES\_COMPLETED to work correctly. The number of boxes completed (0, 1, or 2) is returned in R3.

**Submission**

Please hand in a LAB5\_ID.asm file and a report as well.