

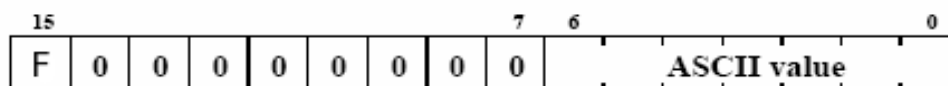
### Concentration

**Introduction:** For this machine problem you will be writing a program in LC-3 assembly to implement the game of Concentration also known as Memory. Concentration is a card game in which all of the cards are laid face down on a surface and two cards are flipped face up after each turn. The object of the game is to turn over pairs of matching cards. Instead of using playing cards, your program will use ASCII characters. Your program will display a NxN game board (where N is the number of ASCII characters in each column or row). Before a match is made, your program will hide the actual characters by displaying a 'X'. Your implementation should be able to take row and column coordinates as user input, detect if a match has been made, and keep track of which characters have been matched. Once all characters have been matched the game should end. A skeleton mp2.asm file and test cases will be provided (read the MP2-README in Machine Problems section).

**Checkpoint 1:** For the first checkpoint you will need to correctly display the NxN game board and detect if all characters have been paired. You will be given a database that contains N and the ASCII characters and it will be stored sequentially in memory starting at x4000 as shown below.

Address	Memory Value
x4000	N
x4001	Ascii Character
x4002	Ascii Character
...	...
x4000+N <sup>2</sup>	Ascii Character

Each “ASCII character” will be stored in memory in the following format:



The first entry at x4000 specifies N (the width/height of the game board). Your program will only work with square game boards of size NxN. For this MP, we will limit N to 2, 4, 6, or 8. The following entries will hold a ASCII value and a Flag bit. The Flag bit is stored in bit 15 and the ASCII value is stored in bits [6:0]. If the flag bit is set to 1 then the character has not been paired (and is hidden), and if the flag is set to 0 the character has been paired (revealed).

#### Required Subroutines (Checkpoint 1):

- **Display Subroutine:** To display the game board you should write a display subroutine. This subroutine should first clear the screen. To do this write 50 linefeeds ( $\backslash n$  or x000A) to the screen using the OUT trap. Then display the current game board using OUT. Entries with a 0 in bit 15 should be displayed and entries with a 1 in bit 15 should display an uppercase X (no entry will be an uppercase X). Print a single linefeed ( $\backslash n$  or x000A) character after every row.
- **Check for Win Subroutine:** This subroutine should check all entries in the game board. If all entries have been paired (Flag=0), the string at WIN\_MSG should be displayed with the PUTS trap and the game should HALT.

Once both subroutines have been written, the game should call Display and then Check for Win.

**Checkpoint 2:** For the second checkpoint, you will allow a user to choose two coordinates and check for a match. You will then create a fully working game of Concentration.

**Required Subroutines (Checkpoint 2):**

- **Get Input Subroutine:** First display the ASK\_ROW\_MSG using PUTS. Then get user input using IN trap. Check the user's input to see if it is the lowercase character 'q'. If it is, quit the game. If it is not, check that it is from 0 to N-1 (so that it is a valid selection). If the selection is not in range, use PUTS to display INVALID\_MSG and loop to beginning of subroutine. If row is valid, display ASK\_COL\_MSG and get the column as well. You will not need to check for 'q', but you must check that the column is in range. If it is not, display the INVALID\_MSG and loop to the beginning of the subroutine. If valid, convert the row and column into an address corresponding to the memory location of the specified piece. Lastly, check that the selected piece has not already been revealed. If it has, display the INVALID2\_MSG and loop to the beginning of the subroutine. Otherwise, return the memory address in a register.

After your subroutines have been written and thoroughly tested, you should start writing code for the actual game logic. The game procedure should look like this:

1. Display the game board (Display Subroutine)
2. Display FIRST\_MSG and get user's first selection. (Get Input Subroutine)
3. Change first piece's flag bit to shown and re-display the game board.
4. Display SECOND\_MSG and get user's second selection. (Get Input Subroutine)
5. Change second piece's flag bit to shown and re-display the game board.
6. If the two selections match, display MATCH\_MSG otherwise display NO\_MATCH\_MSG and change both pieces' flag bit to hidden.
7. Pause game and wait for the user to press any key. To do this display the WAIT\_MSG and use GETC trap to pause until the user presses a key. The input can be disregarded.
8. Use your check win subroutine to see if the game is over. If it is not, loop to the beginning.

**Testing:** We will provide you several game boards to use. To load these boards into the simulator, first load the gameboard .obj file and then load your mp2 .obj file. We will provide a series of test cases on the ECE 190 website that will be very useful for testing your code, though you should make sure to create your own tests to ensure correctness. **YOUR OUTPUT SHOULD MATCH THE GOLD OUTPUT EXACTLY TO GET CREDIT** (except for register values). Read the README in the Machine Problems section for more info.

**Grading:**

*Functionality (55%)*

- 15% - Display subroutine (Checkpoint 1)
- 10% - Check for win subroutine (Checkpoint 1)
- 15% - Get Input subroutine (Checkpoint 2)
- 10% - Main game functionality (Checkpoint 2)
- 5% - Proper error handling (Checkpoint 2)

*Style (15%)*

- 5% - Lines are kept to 80 characters or fewer
- 10% - Correctly displays messages

*Comments (30%)*

- 10% - Code is clear, easy to understand and follow
- 10% - Introductory paragraph explaining program's purpose, register contents, etc
- 10% - Code is well commented (including subroutine explanations)