ECE 190: Introduction to Computing Systems, Spring 2010
Machine Problem 2

Checkpoint 1 Due: 5 p.m., Wednesday 3 March, 2010
Checkpoint 1 Due: 5 p.m., Wednesday 10 March, 2010

Sudoku

Introduction: For this machine problem you will be writing a program in LC-3 assembly to implement the game of Sudoku. Sudoku is a numbers game in which there is a 9x9 grid of numbers, and the player must compete the grid in such a way that each row, column, and 3x3 sub-block of the grid contain every number from 1 to 9. Your program will use ASCII characters to display the 9x9 game boards, and a player will be able to fill out the grid and solve the puzzle one number at a time. If a spot on the game board does not have a number held within, a space will be displayed. Your implementation should be able to take row and column coordinates as user input, detect if the number placement is "legal," and determine when the game is finished. A skeleton mp2.asm file and test cases will be provided (read the MP2 README in the Machine Problems section).

Checkpoint 1: For the first checkpoint you will need to correctly display the 9x9 gameboard and detect if a board is completely filled out. You do not, at this time, need to be concerned with the board being correctly filled out. You will be given a database that contains 81 numbers stored sequentially starting at memory address x4000 as shown below.

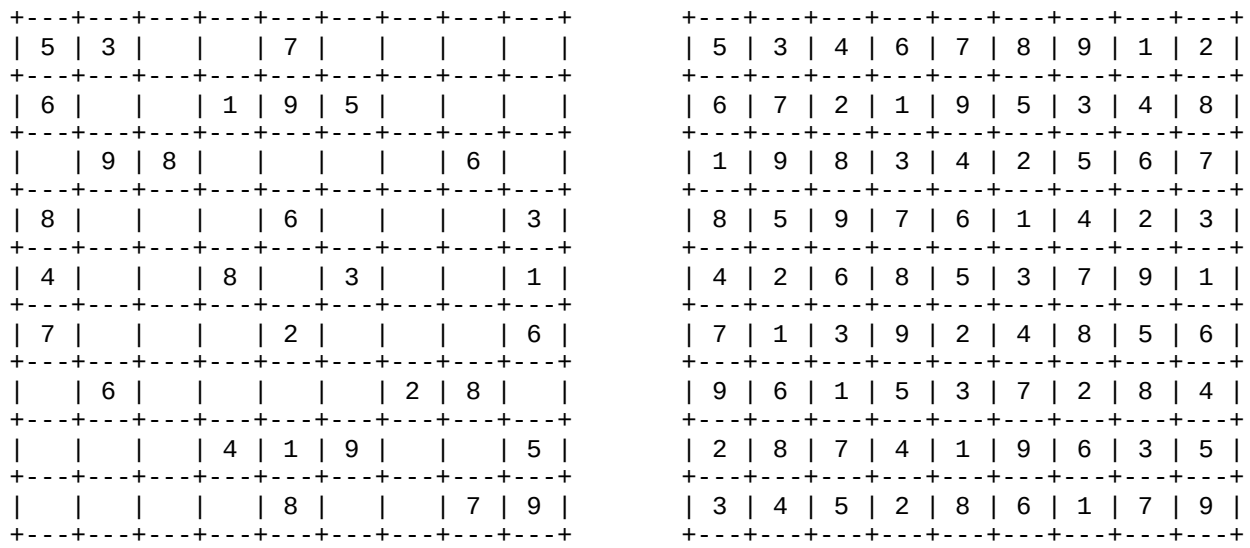
Address	Memory Value (row, column)
x4000	Memory Value (0,0)
x4001	Memory Value (1,0)
x4002	Memory Value (2,0)
...	...
x4050	Memory Value (8,8)

Each memory location will have data in the following format:

15	14	13	12	11	10	9	8	7	6	5	4	3 2 1 0
0	0	0	0	0	0	0	0	0	0	0	0	Binary Number

If the number is a zero (x0000) that shall mean that there is no number in that space on the game board. Any non-zero number will always be between the values of 1 and 9.

The board shall be a 9x9 grid rendered using ASCII '+', '-', '|' and ' ' (space) characters. An empty spot of the game board would be printed as three spaces in the location, whereas a spot with a number would have a single space followed by the ASCII number followed by another space. The board is shown below.



The board on the left is what a player would see at the beginning of the game, while the board on the right is the final solution for the puzzle.

The top left corner of the grid is considered to be the coordinate (0,0) and the bottom right is (8,8). The top right is (8,0) and the bottom left is (0,8).

CHECK_FOR_WIN is the last subroutine you will need to create. It is responsible for determining if all 81 spaces on the board have numbers in them, and exits the game after printing the WIN_MSG if the board is full. If the board is not full, then return to the main program.

Required Subroutines (Checkpoint 1):

- **Display Subroutine:** To display the game board you should write a display subroutine. This subroutine should first clear the screen by writing 50 linefeeds (\n or x000A) to the screen using the OUT trap. Then display the current game board using OUT. Entries with 0's in all bits should be displayed as a space and entries with numbers should be displayed as their ASCII equivalents.
- **Check For Win:** This subroutine displays the WIN_MSG using PUTS and then HALTS if all 81 entries of the Sudoku table are filled, otherwise it returns to the main routine.

Checkpoint 2: Any user playing the game will need to be able to input information to the program, so your program will also need to take in user input using a GET_VALID_INPUT subroutine that will be given to you in the template code. For any input you are given, you will need to store it in any labeled memory location and also print it back out to the screen. You will need to write a GET_INPUTS subroutine that asks the user for all the needed information and uses GET_VALID_INPUT to help with this task. Finally, the new data is to be stored in the database of numbers for the puzzle.

For the second checkpoint, you will need to create a CHECK_INPUT subroutine that will enforce the rules of Sudoku onto the player. That is to say that CHECK_INPUT will make sure the new entry does not break these rules:

- Each row must contain no more than one of the same number.
- Each column must contain no more than one of the same number.
- Each 3x3 sub-block must contain no more than one of the same number.

If any of those three conditions are not met, the new board entry should be removed from the database and INVALID_MSG2 should be displayed using PUTS.

You will be given several look up tables in the template code that may or may not help you fulfill the objectives of this checkpoint.

After all of this is done, you need to display the WAIT_MSG using PUTS and wait for the user to input any character prior to exiting.

Required Subroutines (Checkpoint 2):

- **Get Inputs Subroutine:** First display the ASK_ROW_MSG using PUTS. Then get user input using the GET_VALID_INPUT subroutine provided. If a 'q' is passed into the program GET_VALID_INPUT will end the program. Store the return value into a memory location labeled INPUT_ROW, then display the ASK_COL_MSG using PUTS. Store the return into INPUT_COL and finally ask the user for a number by displaying ASK_NUM_MSG and storing the return value from GET_VALID_INPUT into INPUT_NUM.
- **Check Input:** Check the board to make sure it follows the rules of Sudoku. If it does not, remove the most recently placed entry into the board and display INVALID_MSG2 using PUTS. Try to optimize this code as much as possible. You do not want to check every row, column, and sub-block to see if it only has one 1, 2, 3, 4, 5, 6, 7, 8, and 9 every time this subroutine is called.

You are allowed to create and use additional subroutines if needed in either checkpoint.

The game procedure should look like this:

1. Display the game board (Display Subroutine)
2. Display FIRST_MSG and get user's row selection (Get Input Subroutine)
3. Display SECOND_MSG and get user's column selection (Get Input Subroutine)
4. Display THIRD_MSG and get user's entry for the game board (Get Input Subroutine)
5. If the game board is finished (use CHECK_FOR_WIN), display WIN_MSG.
6. If the game board is now illegal, undo change and display INVALID_MSG2.
7. Pause the game and wait for the user to press a 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.

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, through 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 information. This file will be updated if and when needed.

Grading:

Functionality (55%)

- 10% - Display Subroutine (Checkpoint 1)
- 10% - Check for Win Subroutine (Checkpoint 1)
- 10% - Input Subroutines (Checkpoint 2)
- 20% - Check for Valid Input Subroutine (Checkpoint 2)
- 5% - Full game functionality, including proper error handling (Checkpoint 2)

Style (15%)

- 5% - Lines are kept to 80 characters or fewer in length
- 10% - Assembly code and comments are formatted properly

Comments (30%)

- 10% - Code is clear, easy to understand and follow
- 10% - Introductory paragraph explaining your overall approach to the machine problem, and brief comments at the beginning of every subroutine detailing how each register is used (input data, temporary loop counter, output data, no used, etc)
- 10% - Code is well commented (including subroutine explanations)