COMP2611: Computer Organization
Spring 2020
Programming Project: 2048 Game
(Deadline 11:59PM, May 22, 2020)

1 Introduction

Have you ever played the popular 2048 game? It is a single-player sliding block puzzle game designed by the Italian web developer Gabriele Cirulli. The game's objective is to slide numbered tiles on a grid to combine them to create a tile with the number 2048. Fig. 1 shows a sample screen shot of the game. You can try to play the game online. You can read more about the game at 2048 wiki.

2048 was originally written in JavaScript and CSS during a weekend. Your task is to implement the 2048 game with MIPS assembly within a month. Hope you enjoy the fun!

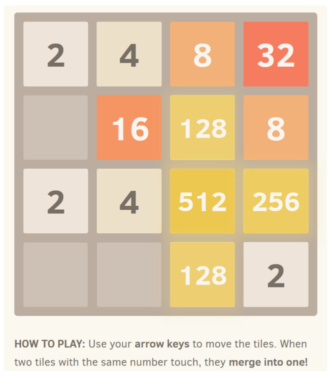
2
4
8
32
16
128
8
2
4
512
256
128
2
HOW TO PLAY: Use your arrow keys to move the tiles. When two tiles with the same number touch, they merge into one!

Fig. 1: The 2048 game on a 4x4 grid.

2 Game Rules

2.1 The Sliding Operation

2048 is played on a 4×4 grid, with numbered tiles that slide smoothly when a player moves them to either up, down, left or right direction.

Tiles slide as far as possible in the chosen direction until they are stopped by either another tile or the edge of the grid. For example, [2, 0, 4, 0] → [0, 0, 2, 4], where → denotes a sliding right operation and 0 is an empty tile.

If two tiles of the same number collide while moving, they will merge into a tile with the total value of the two tiles that collided. The resulting tile cannot merge with another tile again in the same move, e.g., [0, 2, 2, 4] → [0, 0, 4, 4].

If a move causes three consecutive tiles of the same value to slide together, only the two tiles farthest along the direction of motion will combine, e.g., [0, 2, 2, 2] → [0, 0, 2, 4].

If all four spaces in a row or column are filled with tiles of the same value, a move parallel to that row/column will combine the first two and last two, e.g. [2, 2, 2, 2] → [0, 0, 4, 4].

2.2 New Tile

A new tile will randomly appear in an empty spot on the 4x4 grid with a value of either 2 or 4, if the sliding operation has changed the values of the grid. Take the game grid in Fig. 1 as an example. If the player slides up, down, or left, a new tile with 2 or 4 will generated randomly after the sliding operation. But if the player slides right, no new tiles will be generated as the corresponding sliding operation leads to no change on the grid.

2.3 Wining or Losing the Game

A player wins the game when a tile with a value of 2048 appears on the board, hence the name of the game.

A player loses the game if the 4×4 grid is full AND there are no more neighboring tiles that can be further combined (i.e., with the same value). Fig. 2 shows examples of game win, lose, and a game grid which is full but not lost yet.

3 Game Implementation

2048 in MIPS assembly is challenging, but don't worry, you won't start everything from scratch. The fancy user interface is handled by modified Mars (copyright: COMP2611 teaching team). The MIPS code mainly works on the logic of the game. Download skeleton file `COMP2611_Spring20_2048_skeleton.s` to start your coding.

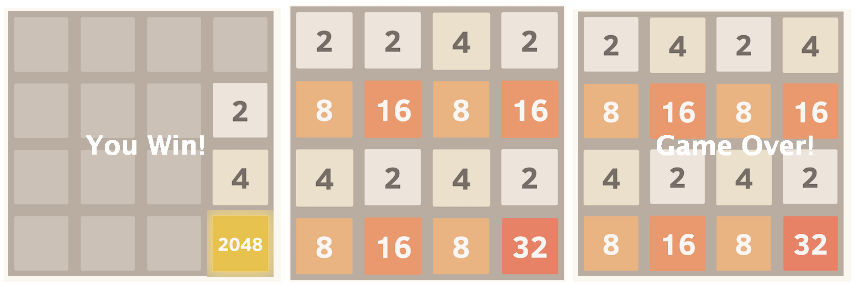

Fig. 2: **Left:** An example of winning a game. **Middle:** An example of an ongoing game with a full grid (not lose yet). **Right:** An example of losing a game.

'Divide-n-conquer' strategy is used in the skeleton. The skeleton code is loooong, but you will find it is well organized with sub-tasks handled by different MIPS procedures. Although you're recommended to read through the skeleton and understand the details of the code, you should still be able to survive if you get a big picture of the code structure, and then only focus on the individual tasks (implemented by different procedures with well-defined interface).

3.1 Data Structure

All data structures used are static. They're defined at the very beginning of the skeleton code.

The 4×4 game board is a 4×4 matrix, which is internally saved as a 1D array `puzzle_map` with 16 elements (row major). Two additional 1D arrays of 16 elements, `puzzle_map_prev` and `puzzle_map_temp`, are used for Step 3 check whether the sliding operation has changed the grid, and Step 5 check game lose respectively.

`arr4` can be used to assist the sliding operation. `zero_indices` records the empty spots in the game board, where the new tiles can be dropped.

3.2 Game Loop

During game initialization, `input_game_target_score` asks the player to input a target score. Feel free to input a small value (power of 2, e.g. 64) for fast debugging purpose. `generate_a_new_game_map_randomly` initializes the game board by randomly filling it with two tiles with values of 2 or 4 .

The game then starts. It works as a big loop. Each iteration follows the steps listed below:

1. Get the keyboard input: `get_keyboard_input`.
2. Perform the core sliding operation if the pressed key is 'w', 's', 'a' or 'd', corresponding to up, down, left and right respectively: `process_slide_operation`.

3. Update the game status after the slide operation

   – Update the current game score, which is defined to be the maximum value among all tiles: `update_game_score`.

   – Check whether the game win: `check_win`.

   – Check whether the sliding operation has changed the values of the grid: `check_map_changed`. If no tile moves, Go to Step 1 and restart the game loop.

   – Refresh the screen by syscall 201.

4. Generate a new tile randomly: `generate_a_random_tile`.
5. Check game lose: `jal check_lose`.
6. refresh the screen by syscall 201.
7. Go to step 1 and restart the game loop.

`actions_for_win` and `actions_for_lose` handle the game winning and losing, then the program finishes its execution.

4 Programming Tasks

Read the skeleton code carefully. But you don't need to understand every single line of MIPS code!

Top priorities:

1. Understand the data structures used in the skeleton;
2. Trace the game loop, and draw its flow chart;
3. Figure out the functionality of each procedure.

Once you build up a big picture of the project, zoom in to the programming tasks, and examine the comments and example codes carefully.

Note: You should not modify the skeleton code but only add your code to those procedures.

Table 1 lists all the programming tasks you need to implement. For some programming tasks in the skeleton code, you need to remove the code in the answer space and write your own code.

Try to follow good conventions, e.g. comment your code properly and use registers wisely.

You can discuss with your friends if you have difficulty in understanding the tasks. But every single line of your code should be your own work (not something copied from your friends).

Procedure	Input	Results	Description
`clear_map`		all elements in `puzzle_map` will bet set to 0.	clear up the 4x4 game grid.
`generate_a_random_tile`		a new tile will randomly appear in an empty spot on the `puzzle_map` with a value drawn randomly from 2,4 equally.	refer to Section 2.2
`slide`	\$a0, \$a1, \$a2, \$a3: the addresses of 4 elements in a column or row in the 4x4 game grid	the 4 numbers will be moved and merged in the direction of \$a1 → \$a3, according to the 2048 game rules.	refer to Section 2.1
`check_win`		\$v0=1 if win, \$v0=0 otherwise.	The game is won when a tile in `puzzle_map` has a value not less than the target game score (i.e., `input_target`).
`check_lose`		\$v0=1 if lose, \$v0=0 otherwise	The game is lost if the `puzzle_map` is full and there are no more neighboring tiles that are combinable.

Table 1: Programming tasks

5 Syscall Services

The MIPS 2048 game runs on a modified MARS (`Mars_2048game.jar`), which provides additional set of syscall services, e.g. update the game board, refresh the screen, and play sound effect, etc.

Table 2 lists these additional syscalls (starting from code 200). Note that not all the new syscalls are necessary in your code, some are described here for you to understand the skeleton. Syscall code should be passed to $v0 before usage.

If you have problem running the modified MARS, that is most likely due to the outdated JRE. Download the latest JDK v13 and install it to your system.

Service	Code	Parameters	Result
Create game screen	200		Create an empty game board with the size of 560×560 pixels
Update the game board and refresh the screen	201	$a0: the address of `puzzle_map`.	The 4x4 game board will be refreshed using the values in `puzzle_map`.
Play the game sound or stop it	202	$a0: sound_id; $a1 = 0: play once; 1: play repeatedly in loop; 2: stop playing. The sound IDs are described as follows: 0: the background music; 1: the sound effect of obtaining game scores; 2: the sound effect of losing game scores; 3: the sound effect of losing the game;	
Set the current game score	203	$a0: score	
Set the game target	204	$a0: target	Set the target score in the range of [64, 2048] for winning a game.
Get a random integer in the range [0, upper bound) uniformly	42	$a0: index of pseudo-random number generator, $a1: the upper bound	The random integer is returned in $a0

Table 2: syscall Services

6 Submission

You should ***ONLY*** submit a single file `COMP2611_Spring20_2048_yourStudentID.s` with your completed code for the project. Please write down your name, student ID, and email address (as code comments) at the beginning of the file.

Submission is via Canvas. The deadline is a hard deadline. Try to avoid uploading in the last minute. If you upload multiple times, we will grade the latest version by default.

7 Grading

Your project will be graded on the basis of the functionality listed in the project description and requirements. You should ensure that your completed program can run properly in our modified MARS.