CS/CoE 0447 Computer Organization
Spring 2015

Programming Project 1
Assigned: February 12, 2015. Due: March 5, 2015 by 11:59 PM.

1 Description

Put on your seat belts! It's time to step into your DeLorean time machine, set the date to 1978, and accelerate to 88 MPH. Bam! It's 1978! The Bee Gees rule, the Incredible Hulk is number one, the Vax is king and satin disco jackets are *de rigueur.* And, best of all, it's the beginning of the arcade video game revolution. The pinball machine will never be the same!

As you practice your best dance moves under the spinning glitter ball to "Stayin' Alive," you have a brilliant idea for an arcade computer game, called Concentrate, that will most certainly land you a job at Atari (well, maybe...). You realized that card games are hugely popular, and what better way to play a card game than to use a computer! You decide to implement the classic card game of concentration with a couple fun twists.

Initially, the game shows 40 cards, arranged as 5 rows of 8 cards on the computer display (the golden age of the CRT). The cards are numbered from 0 to 9 (that is, their face values are 0 to 9). Each card is turned over so its back is showing. The player tries to select pairs of cards with the same value until all cards are selected. When the player selects a card, it is turned over to show the face (value). The player then tries to select another card that has the same value. When a match is made, the matching card faces are left showing on the game board. If a match was not made, the selected cards are turned over to show their backs after a few seconds. For example, if the player selects a pair of 2s, a successful match was made. If the player selects two cards with different values, a match was not made and the cards are turned back over to show their backs. The objective is to match all cards as quickly as possible.

Now, for the fun twists. First, the game has a count-down timer; the player has to match all cards before the timer reaches 0 minutes, 00 seconds. The initial value of the timer is 3:00 (3 minutes and 00 seconds). The timer is shown on the computer display and animated to show the seconds and minutes ticking away (Oh My!! what pressure!). Second, the game has a "smiley card" that randomly moves around the board. The smiley card is a special card back; when it is displayed at a card position on the board, the player receives extra time by matching the value of the smiley card. The smiley card is randomly moved every 10 seconds to another position. If the player doesn't match it before it moves, then a new match has to be made to receive the time bonus. The time bonus is 30 seconds.

Your job is simple: Implement Concentrate in MIPS with the Mars LED display simulator[1] used in lab 5 (see CourseWeb). The game should follow the rules below.

2 Game Rules

The game board is drawn on a display of LEDs. An LED is a light emitting diode that can be turned on/off. The display uses tri-color LEDs; each LED in the display can be off, red, yellow/orange or green. The LEDs are arranged as a grid of 64 columns by 64 rows, as shown in Figure 1. A position in the grid is denoted by a coordinate (x, y). The x coordinate is the column position, such that $0 \leqslant x \leqslant 63$. The y coordinate is the row position, such that $0 \leqslant y \leqslant 63$. The game board includes

[1] MIPS was actually invented in 1981, but let's suspend disbelief and pretend it was invented in 1977.

an arrow keypad, with up, down, left, right and center keys. The cards and timer are drawn on the board by turning on/off the LEDs.

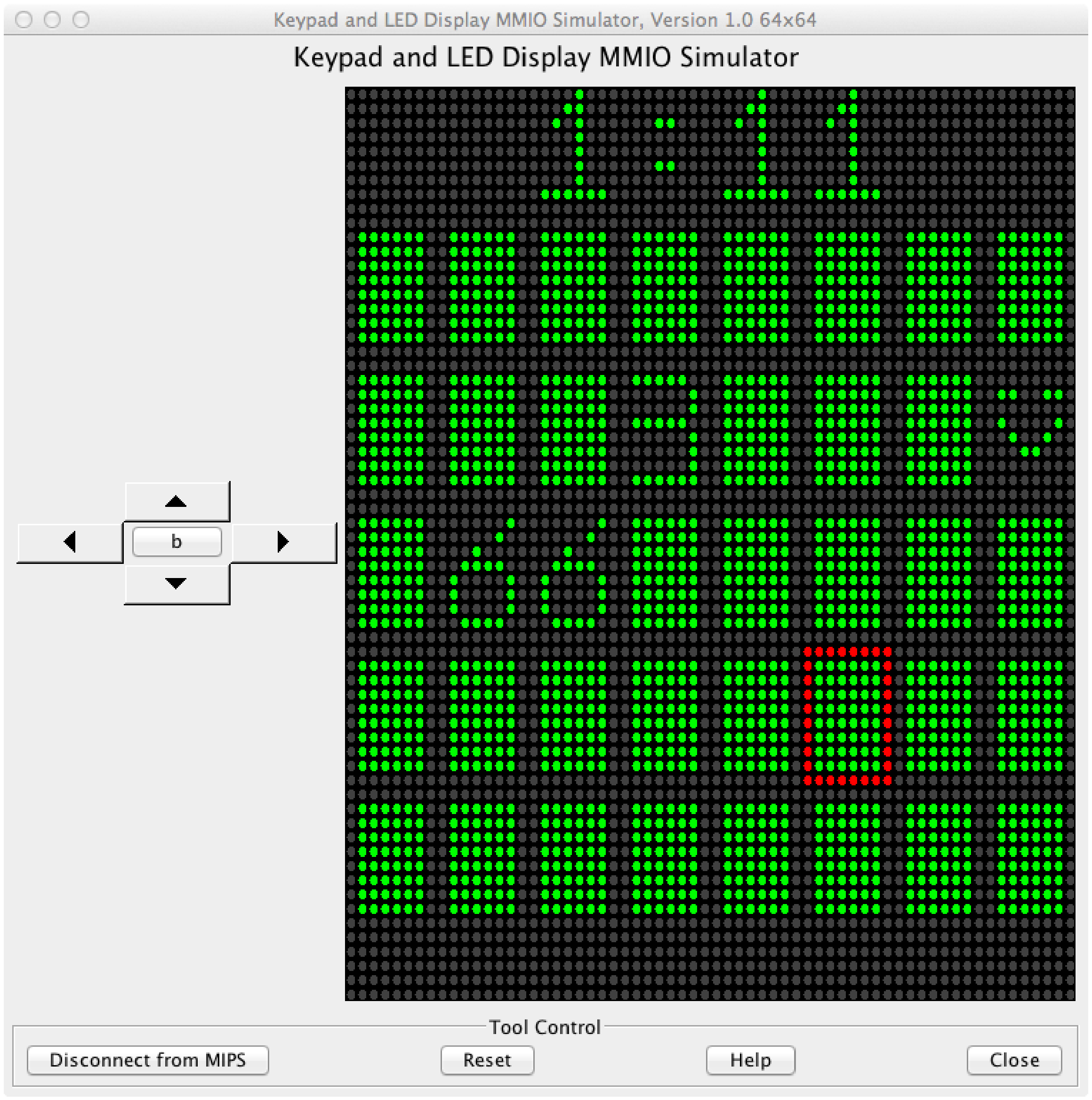
Keypad and LED Display MMIO Simulator, Version 1.0 64x64
Keypad and LED Display MMIO Simulator
b
Tool Control
Disconnect from MIPS
Reset
Help
Close

Figure 1: Example LED display with matched cards, smiley card, and one selected card.

2.1 Deck of Cards and Game Board

There are several requirements for the deck of cards and the game board:

- The game has one deck, which has 40 cards. Cards are numbered 0 to 9. There are 4 cards of each number in the deck (e.g., 2 pairs of 2s). Card suits are not marked.
- The cards are placed on the game board in 5 rows of 8 cards each.
- Each card is represented by a rectangle of 6x8 LEDs.
- A "font" for the cards is provided on CourseWeb. You must use this font.
- The game board shows a count-down timer at the top.

- The layout of the game board should be the same as Figure 1.

2.2 Selecting a Card

To select a card, the player moves a cursor and presses a select button:

- A cursor highlights the current card. The player can move the cursor up, down, left or right.
- The cursor is represented by a red border of LEDs around the current card. In Figure 1, the cursor is currently positioned at row 4, column 6.
- The current card is selected by pressing 'b' (center button).
- When the current card is selected, it is turned over to show its face. For example, the card at row 2, column 4 was previously selected, and the player is trying to find a matching 3.
- If the player selects a card that is already showing its face value, no action is taken.
- The cursor is moved with the arrow keys. The cursor is moved left, right, up or down in the direction of the arrow key pressed.
- If the player selects up/down at the top/bottom of the board, the cursor is not moved. Hence, the cursor does not wrap around to the other side at the board's top or bottom edges.
- Likewise, if the player selects left/right at the leftmost/rightmost side of the board, the cursor is not moved. Hence, the cursor does not wrap at the left or right edges.

2.3 Matching Cards

A pair of cards is matched according to the requirements:

- A match is made by selecting two cards with the same face value.
- When a pair of cards is successfully matched, their faces remain showing.
- When a pair of cards is not successfully matched, the face value of the cards is shown for **3 seconds** after the second card of the pair was selected. After 3 seconds, the card faces are turned over so the card backs are shown.

2.4 Smiley

The smiley is a special card that can be matched for extra time:

- One unselected card (back showing) is randomly chosen every **10 seconds** to be the smiley.
- The smiley card is depicted by showing the card back as a smiley face. Only one card chosen as the smiley should display the smiley back. For example, in the figure, the card currently chosen as the smiley is in the 2^{nd} row and 8^{th} column[2].
- The smiley card behaves similar to a normal card for matches; the only difference is the smiley face must be shown whenever the back should be shown.
- The smiley is independent of other game actions. Thus, the smiley can be selected, and then move during an attempted match! There are other conditions to be aware of as well.
- When the smiley is part of a successful match, 30 seconds are added to the timer. The timer display is updated with the new time.

2.5 Timer

The count-down timer determines how much time is left to play the game:

- The timer is shown at the top of the board, as depicted in Figure 1.

[2]Yea, I know it barely looks like a smiley face. But you try drawing a smiley in a 6x8 grid!

- The timer is centered in the top eight rows of LEDs
- The timer shows the minutes (one digit), and the seconds (two digits). There is a colon separator between the minutes and seconds.
- The counter is animated. It ticks down one second at a time.
- The timer is decremented by 1 second, every second. The timer display is updated every second with the new time remaining.
- When the timer reaches 0:00, the game is over.

2.6 Game Start

When the game starts, several initial conditions are set:

- The cards are randomly shuffled before being placed on the board.
- The cards are initially placed with their backs facing up.
- The cursor is on the card in the upper left corner of the board.
- The timer has an initial value of 3 minutes, 00 seconds.

2.7 Game End

The game ends when certain conditions are satisfied, and it prints the number of matched and unmatched card pairs:

- If the timer reaches 0 minutes and 00 seconds, the game is over.
- If all cards are matched, the game is over.
- When the game is over, the number of matched card pairs (call this value x) and unmatched card pairs (call this value y) are displayed in the Run I/O window.
- The output for game over must match:

  ```
  Game over.
  Number of card pairs matched:   x
  Number of card pairs not matched:   y
  ```

3 Animation

To animate the game, you must update the timer, cursor, smiley card and a pair of unmatched cards. The animation is straightforward: Just update the displayed values, according to the times noted in the requirements above. For easy reference, these are the animations that need to be handled:

- Update the timer every 1 second. Hint: Update only the digits that change.
- Update the cursor whenever the arrow keys are pressed.
- Update the current card display (back to face) when a card is selected.
- After 3 seconds, update the card display (face to back) for an unmatched pair of selected cards.
- After 10 seconds, update the smiley card (show back of current smiley, select another smiley and show it).

The "efficiency" of your assembly language program will influence the animation, so it's worth spending time to make your code simple (i.e., use as few instructions as you can). Try to find ways to shorten instruction sequences involved in timing.

4 Mars LED Display Simulator

The project needs a special version of Mars available from CourseWeb. This version of Mars has a Keypad and LED Display Simulator developed and extended by former graduate students in the CS Department.

4.1 Display

The LED Display Simulator has a grid of LEDs. An LED is a light. Each LED has a position (x, y), where $0 \leqslant x \leqslant 63$ and $0 \leqslant y \leqslant 63$. The upper left corner is $(0, 0)$ and the lower right corner is $(63, 63)$. An LED can be turned on/off by writing a 2-bit value to a memory location that corresponds to the LED. An LED can be set to one of three colors: green (11_2), yellow (10_2) or red (01_2). The LED can also be turned off (00_2). Details about the LED Display are available from the LED Display Simulator manual (see your TA's Courseweb site).

4.2 Keypad

The game must use the keypad provided by the LED display. The keypad is the set of arrow keys on the left side of the LED Display Simulator. The keypad has up (▴), down (▾), left (◂) and right (▸) buttons. The keypad also has a center button ('`b`').

Each arrow and center button in the keypad has a keyboard character equivalent. When you click a keypad button or press the equivalent keyboard character, a value is stored to memory location `0xFFFF0004`. To indicate a button click, the value `1` is written to memory location `0xFFFF0000`. When your program loads this address, memory location `0xFFFF0000` is set to `0`.

Important: The program *must* read the status memory location (`0xFFFF0000`) before trying to read the key value memory location (`0xFFFF0004`). The behavior is undefined if you read the key value memory location without first reading the status location. See the LED Display Simulator manual for a few more details.

The keypad button and keyboard character equivalents are:

'`w`' is ▴ button, which stores `0xE0` at memory address `0xFFFF0004`
'`s`' is ▾ button, which stores `0xE1` at memory address `0xFFFF0004`
'`a`' is ◂ button, which stores `0xE2` at memory address `0xFFFF0004`
'`d`' is ▸ button, which stores `0xE3` at memory address `0xFFFF0004`
'`b`' is `b` button, which stores `0x42` at memory address `0xFFFF0004`

Note: There have been reports that Mars locks up if the mapped keyboard keys are pressed too quickly.

Details about the keypad and how to use it are described in the LED Display Simulator manual. This document is available from CourseWeb. Additionally, an example program that uses the keypad is available from CourseWeb.

4.3 Java Versions

We built this version of Mars with JVM 1.6. It should run on any machine with JVM 1.6 installed. We have successfully tested it on recent versions of Windows, Mac and Linux. If you have an older version of Java, you may wish to upgrade to a more recent one. Please let your instructor or TA know if you encounter any problems.

5 Turning in the Project

5.1 What to Submit

You must submit a single compressed file (`.zip`) containing:

- `concentrate-<your pitt username>.asm` (the Concentrate game)
- `README-<your pitt username>.txt` (a help file – see next paragraph)

The filename of your submission (the compressed file) must have the format:

> `project1-<your pitt username>.zip`
> Here's an example: `project1-xyz30.zip`, which contains `concentrate-xyz30.asm` and `README-xyz30.txt`.

Put your name and e-mail address in both files at the top.

Use `README.txt` to explain the algorithm you implemented for your programming assignment. Your explanation should be detailed enough that the teaching assistant can understand your approach without reading your source code. If you have known problems/issues (bugs!) with your code (e.g., doesn't animate correctly, odd behavior, etc.), then you should clearly specify the problems in this file. The explanation and bug list are critical to grading. So, if you're uncertain whether to include something, then err on the side of writing too much and just include it.

Your assembly language program must be reasonably documented and formatted. Use enough comments to explain your algorithm, implementation decisions and anything else necessary to make your code easily understandable.

Deadline: Files submitted after March 5, 2015, 11:59 PM will not be graded. **It is strongly suggested that you submit your file well before the deadline.** That is, if you have a problem during submission at the last minute, it is very unlikely that anyone will respond to e-mail and help with the submission before the deadline.

5.2 Where to Submit

Follow the directions from your TA for submission via CourseWeb.

6 Collaboration

In accordance with the policy on individual work for CS/CoE 0447, this project is strictly an individual effort. Unlike labs, you must not collaborate with a partner. Please see the course web site (syllabus) for more information.

7 Programming Hints

Tackle the project in small steps, rather than trying to do it all at once. Test each step carefully before moving on to the next one.

7.1 Recommended Steps

Here are some recommended steps:

1. Think! Plan! Think some more! Write pseudo-code for your game strategy.
2. Perhaps start with the count-down timer. Identify a useful way to keep the state of the timer. You may find it easiest to treat each digit of the timer independently (e.g., use separate registers for the minutes, tens of seconds, and ones of seconds).

3. Develop a function that can draw a “card” on the display. The function should be general enough for both drawing cards and the time. Indeed, the timer value is displayed simply as card faces of appropriate values.
4. Using your draw function, draw the timer on the display (without updating the timer).
5. Implement a loop to decrement the timer value. The loop should read the current time and subtract it from the previously read time. Do not use the sleep syscall because the game has to respond to keypresses and other “events” at any time. The sleep syscall will do what it says — your program will sleep and any keypresses during the sleep will be missed.
6. Now, whenever the time has elapsed by a second, update the timer display with the new time remaining. You may find it most efficient to decrement one digit at a time; e.g., decrement the ones of seconds position, and then check for roll over in this position (i.e., decrementing 0 to -1) to decide whether to decrement the tens of seconds position. This avoids redrawing all digits when only one or two changes, which makes the game more responsive.
7. Once the timer is working, you have the basic structure for the game! You’ll implement all game control functionality inside the timer loop. Note that the animation for drawing cards is similar to drawing the timer. Also, you’ll need to keep track of two elapsed time values: One for turning over an unsuccessfully matched pair of cards and one for moving the smiley around the board.
8. Identify a useful data structure for the game state and board. See below for a suggestion.
9. Implement the data structure and *thoroughly* test it.
10. Once you have the data structure, try writing code to respond to keypresses. In the timer loop, read the keypress status value. If a key was pressed, then read the keypress value. When you do this inside the timer loop, you are “polling” the keyboard to check whether the user pressed a key.
11. Next, try drawing cards on the board. This should be easy, if you correctly implemented the “card drawing” function.
12. You’ll need to initialize the game by randomly selecting the cards to display. You can do this with a random permutation function (see below).
13. Now, try implementing the functionality to highlight a card. Whenever a key is pressed, move the red box left/right, up/down. Did you handle the conditions for the edges of the board?
14. OK, the next part is to select a pair of cards. Try implementing this without worrying about turning cards back over again (on an unsuccessful match). Just draw the board with card backs shown, and then use the left/right, up/down arrow keys to move around the board. Select some cards, and show their face value whenever selected (i.e., center button is pressed).
15. Add the functionality to check for a card pair match. Update the count of card pairs matched.
16. Handle the exit game conditions.
17. Add functionality to turn unsuccessfully matched cards over. This is easy, once you have the timing loop. Simply record the time at which the second card was selected. Show that card’s face value. Continue the timing loop, checking whether 3 seconds has elapsed by subtracting the recorded time from the current time. If 3 seconds or more has elapsed, then turn the cards back over by drawing their backs.
18. Finally, add the smiley card. This is easy as well, once you have the game timing and know how to select random cards. Simply select a random card as the smiley. Check that this card hasn’t been matched. If it isn’t matched, then draw it as the smiley. Be sure to update the old smiley card! Keep track of how many seconds elapsed between when you draw the smiley, and after 10 seconds or more, select a new smiley.

7.2 Data Structure

You'll need a good data structure to represent the state of the board. My suggestion is to treat the board as an array of bytes. There is one byte per card position in the game board. Each array element records the value of a card at some position (x, y). To find the array index for the position (x, y), use the function $i = x \times 8 + y$. For example, in Figure 1, the card at position (1,3) has the value 3. Therefore, the array has the value 3 in element $1 \times 8 + 3 = 11$, e.g., `cardvalues[11]=3`. Note, in assembly, the array element for position (x, y) is at memory address `cardvalues`$+x \times 8 + y$.

You may want a second array that holds the "matched state" of the cards. In this array, an element value of 0 means the card is unmatched, while a value of 1 means it is matched. For example, in Figure 1, the 6 at position (2,1) was matched with the 6 at position (2,2). Array elements $2 \times 8 + 1 = 17$ and $2 \times 8 + 2 = 18$ would be 1, i.e., `cardstate[17]=1` and `cardstate[18]=1`.

7.3 Shuffling the Cards

Shuffling the cards requires a way to randomize order of the card value array. The random number syscalls in Mars can be helpful for this purpose. To use the random number syscall, you need to first initialize the random number generator with a "seed". You can use the time syscall, and pass the lower bits of the time as the seed for initializing the random number generator. Once initialized, you can call the random number generator to get a random number within some range $[0, max]$.

Using the random number generator, we can shuffle the deck. First, initialize the card values. There are 40 cards in the deck. Set 4 cards to each value of 0 to 9. You can do this "statically" as part of the data declaration (a list of bytes or words for the values). Name the array, `cardvalues`. Second, using the random number generator, write a function `permute` to randomly order an array. Call this function with `cardvalues` to shuffle (permute) the deck. Here is pseudocode for `permute`. The function inputs the array `cardvalues`, its length and outputs the array shuffled.

```
procedure permute(int cardvalues[], int length)
begin
  // permute the cardvalues with length array elements
  for i = length downto 2 do begin
     // pick random integer in range [0..i-1]
     int r = random(0, i-1);

     // values i and r in the array
     byte tmp = cardvalues[i-1];
     cardvalues[i-1] = cardvalues[r];
     cardvalues[r] = tmp;
  end
end
```

If you are interested in more details about this permutation algorithm, Wikipedia has a web page about it: `http://en.wikipedia.org/wiki/Fisher-Yates_shuffle`.