CIT 593 - Module 07 Assignment

Operating System, IO Assembly Instructions

Contents

Assignment Overview

In this assignment, you will continue programming in LC4 Assembly. We will be working in the Operating System portion of memory, so you will learn about TRAPs and memory-mapped devices.

Learning Ob

This assignment will [illegible] pics:

- Work with TR[illegible]ng System
- Implement an
- Work with the [illegible]nd Video devices
- (optional) Work with the Timer device

Advice

- Start early
- Ask for help early
- Do not try to do it all in one day

Getting Started

Codio Setup

Be sure to open Codio from the Codio Assignment page in Canvas. Refer to the Module 06 Instructions for detail orks.

Run `user_echo`

1. From the File m. This will open a new Codio tab named `os.asm` containing the
   a. Revie file. This is the basic framework for the operating system, which you will be writing for this assignment.
   b. Look in the `TRAP` vector table for the line
      `JMP TRAP_PUTC`

   Notice how this is the second instructions is after the `.ADDR x8000` directive.
   - When loaded into PennSim, this instruction will be placed at address `x8001`.
   - We can call `TRAP_PUTC` by using the `TRAP` instruction with offset `x01`. This will be done later in `user_echo.asm`.

   c. Scroll down (approximately 115 lines or so) until you reach the lines that read:
      ```
      .CODE
      TRAP_PUTC
      ```
      This label marks the start of the `PUTC` TRAP (an OS subroutine)
   d. When Penn Sim executes `JMP TRAP_PUTC` instruction from the vector table, the program counter will be advanced to the address labeled by `TRAP_PUTC`
   e. Further examine the code that follows; this is the operating system subroutine (a `TRAP`) called `TRAP_PUTC`.
   f. Notice that this is the program we created in lecture to write one character to the ASCII display device on the LC4.

2. From the File Tree, click on `user_echo.asm`.
   a. This file is a program to test some of the operating systems `TRAP`s in `os.asm`
   b. Scroll down to about the 24th line, look for the lines that read:
      ```
      CONST R0, x54
      TRAP x01
      ```
   c. The `CONST` instruction places the number `0x54` (which represents the letter `'T'` in ASCII code) into `R0`. This serves as the argument to the `TRAP_PUTC` trap. You can look up the ASCII code mappings in the Resources section.
   d. The `TRAP x01` instruction sets PC to `x8001`, and also sets `PSR[15]=1` (OS mode). `TRAP x01` is `TRAP_PUTC`, so this `TRAP` call will have the effect of outputting a `'T'` to the LC4's ASCII display.
   e. Examine the rest of this program, you'll see that its purpose is to output `Type Here>` on the LC4 ASCII display.

3. From the File Tree, click on `user_echo_script.txt`. This opens a new Codio tab displaying the script contents.
    a. This file is the PennSim script that assembles and loads `os.asm` and `user_`
    b. Look c ts and compare how it differs from your last HW. Notice d loads both os.asm and user_echo.asm
4. Open a Penn ch PennSim from the Codio command line. Refer to the Module 6 ed a refresher on how to do this.
5. Go to the com trols section and enter

```
script user_echo_script.txt
```

6. Press the `Step` button and carefully go line by line until you see the letter `'T'` output to the screen.
    a. Carefully watch how you start in `user_echo.asm`'s code (in user program memory) and then with the call to `TRAP`, you enter into OS program memory.
    b. Understanding this process is crucial to understanding, writing, and debugging this assignment.
7. Finally, press the `Continue` button to see PennSim run the program until it encounters the `END` label.
8. Make certain you understand how these files work together before beginning the assignment.

Starter Code

We have provided some starter files. You will need to modify some files and generate completely new files to succeed in this assignment.

`os.asm`	- contains the operating system framework, and the TRAP_PUTC m lecture. Your will write the remaining TRAPs.
`user_echo.asm`	ram from lecture
`user_echo_script`	t file for user_echo example
`PennSim.jar`	rograms, debugging, etc.

Requirements

General Requirements

- Your script files **MUST** contain all the necessary commands to assemble and load your programs, as the `multiply` example from Module 6.
- Your program e requirements of the problem when loaded into PennSim and e button.
- Your program ny Exception unless otherwise noted.
- Your program LC4 Assembly.
- You **MUST** co ns your code so we can grade it. This will also help with partial credit.
- You **MUST** use `END` as the label that indicates the end of the program.
- You **MUST** submit to Codio and Gradescope as outlined in the Submission section.
- You **SHOULD** do all the work in Codio. Do not attempt to run these programs locally. TAs will not assist you if you are trying to do the projects outside of Codio.
    - Codio provides a standard environment to ensure consistent functionality.
    - Codio backs up your code. You can restore deleted/modified files by going to Tools->Code Playback.
    - TAs can login and view your code for asynchronous debugging.
    - Different operating systems handle endianness differently. Your submission **MUST** work in the Codio environment, which is where we will be performing all tests.

Part 1: Echo with TRAP_GETC/TRAP_PUTC

- You **MUST** use the traps `TRAP_GETC` and `TRAP_PUTC`.
- You **MUST** implement the traps in `os.asm`, write the test program in `user_echo.asm`, and provide a working script `user_echo_script.txt`.
- `TRAP_GETC` **MUST** take no arguments as input and return the read-in character in `R0`.
- `TRAP_PUTC` **MUST** take a single argument in `R0` (the character to display) as input and not return any value.
- Your test program **MUST** call these two traps in a loop, to echo the user keystrokes to the display.
    - It **MUST** break out of the loop when the user presses the `enter` key.

Part 2: Print Strings with TRAP_PUTS

- You **MUST** implement and use the trap `TRAP_PUTS`.
- You **MUST** implement the traps in `os.asm`, write the test program in `user_string.asm`, and provide a working script `user_string_script.txt`.
- `TRAP_PUTS` **M**
  - take a `0` (the address of the start of the string to display) as input a ue.
  - check ument is a valid User Data Memory address.
    - valid, immediately return without attempting to print
  - print the entire string to the display; that is each character from the starting address through the entire array up to but not including the `NULL` terminator.
- Your test program **MUST**:
  - use `.FILL` to pre-load the `NULL`-terminated string

    ```
    I love CIT 593
    ```

    into User Data Memory, starting at address `x4000`.
  - print this string to the display by populating `R0` with the starting address and then calling `TRAP_PUTS` with the appropriate Trap Number from the Trap Vector Table

Part 3: Get Strings with TRAP_GETS

- You **MUST** implement the trap `TRAP_GETS` and use the traps `TRAP_GETS` and `TRAP_PUTC`.
- You **MUST** implement the traps in `os.asm`, write the test program in `user_string2.asm`, and provide a `string2_script.txt`.
- `TRAP_GETS` **M**
  - take a `0` (the address to start storing the string) as input and return `g` (not including the `NULL` terminator) in `R1`.
  - check `ument` is a valid User Data Memory address.
    - valid, immediately return without attempting to get a string.
  - continue to read characters entered by the user until the user presses the `enter` key.
  - store each character typed by the user up to but not including the `enter` key and add the `NULL` terminator at the end of the string.
- Your test program **MUST**:
  - call `TRAP_GETS` with address `x2020`.
  - read the arbitrary user string and store it into Data Memory.
  - Print the text:

    `Length = X`

    using `TRAP_PUTS` for the `Length = ` portion and `TRAP_PUTC` for `X`, where `X` is the length of the string.
    - You **MAY** assume that the strings will be less than `10`.
  - Call `TRAP_PUTS` with address `x2020` to print the string previously entered by the user.

Part 4: Draw Rectangles with TRAP_DRAW_RECT

- You **MUST** implement and use the trap `TRAP_DRAW_RECT`.
- You **MUST** implement the traps in `os.asm`, write the test program in `user_draw.asm`, and provide a working script `user_draw_script.txt`.
- `TRAP_DRAW_RECT` **M**
  - take fi
    - the upper-left corner of the rectangle, the horizontal
    - the upper-left corner of the rectangle, the vertical
    - `R2` - the horizontal length of the rectangle
    - `R3` - the vertical width of the rectangle
    - `R4` - the color of the rectangle
  - check the bounds of the rectangle compared to the video display
    - if the starting coordinates are outside the video display, immediately return without attempting to draw any part of the rectangle
    - if the starting coordinates are inside the video display, but the values for length or width would cause the rectangle to be drawn outside the video display, immediately return without attempting to draw any part of the rectangle.
  - draw the rectangle with the appropriate color, filling all interior pixels
- Your test program **MUST** call `TRAP_DRAW_RECT` for the following rectangles:
  - a `red` rectangle with starting coordinates `(50,5)`, length `10`, and width `5`
  - a `green` rectangle with starting coordinates `(10, 10)`, length `50`, and width `40`
  - a `yellow` rectangle with starting coordinates `(120, 100)`, length `27`, and width `10`

Extra Credit: Get a Character Within a Time Limit with TRAP_GETC_TIMER

- You **MUST** implement and use the traps `TRAP_GETC_TIMER` and `TRAP_PUTC`.
- You **MUST** im s.asm, write the test program in `user_string` a working script `user_string_ec_script.txt`.
- `TRAP_GETC_T`
  - take a `0` (the desired time to wait for a character) as input.
  - if a ke e time limit, return the entered character in `R1`.
  - if a ke n the time limit, return `NULL` in `R1`.
- Your test program **MUST**:
  - use a time limit of two seconds.
  - print the key that was entered within the time limit using `TRAP_PUTC`, or print nothing if a key was not entered.

Extra Credit: Reset Starting Coordinates When Out of Bounds

- You **SHOULD** make a copy of your `TRAP_DRAW_RECT` trap before working on this extra credit, in case you don't get it working before the submission deadline.
- You **MUST** modify `TRAP_DRAW_RECT`. It **MUST** follow the original requirements, except:
  - If the starting coordinates are outside the video display, it **MUST** reset the starting coordinates to `(0,0)` and draw the rectangle starting here instead, using the original length/width values.
  - It **MUST** follow the Wrap the Rectangle Horizontally extra credit exception if attempting both extra credit options.

Extra Credit: Wrap the Rectangle Horizontally

- You **SHOULD** make a copy of your `TRAP_DRAW_RECT` trap before working on this extra credit, in case you don't get it working before the submission deadline.
- You **MUST** modify `TRAP_DRAW_RECT`. It **MUST** follow the original requirements, except:
  - If the rectangle's horizontal length would take it outside the video display, it **MUST** wrap around the display and continue drawing the rectangle from the left side of the display at the same vertical width. Do not wrap rectangles if they go outside video memory vertically.
  - It **MUST** follow the Reset Starting Coordinates When Out of Bounds extra credit exception if attempting both extra credit options.

Suggested Approach

This is a *suggested* approach. You are not required to follow this approach as long as you follow all of the other requirements.

High Level Ov

Work on one problem

1. Review the st v the different files work together. This is critical for succeeding in RAP_PUTC is already written for you and you need to understand h
2. Implement us TRAP_GETC and TRAP_PUTC.
3. Complete the TRAP_PUTS implementation in `os.asm`.
4. Implement `user_string.asm` using `TRAP_PUTS`.
5. Complete the `TRAP_GETS` implementation in `os.asm`.
6. Implement `user_string2.asm` using `TRAP_GETS`, `TRAP_PUTS`, and `TRAP_PUTC`.
7. Review the implementation of `TRAP_DRAW_PIXEL`.
8. Complete the `TRAP_DRAW_RECT` implementation in `os.asm`.
9. Implement `user_draw.asm` using `TRAP_DRAW_RECT`.
10. (optional) Attempt the extra credits.

Great High Level Overview, but I really need a Slightly More Detailed Overview

Okay, I guess we can give some more details.

Part 1: Echo with TRAP_PUTC

File Setup

You only need to mo for this part. You do not need to create any additional files.

Your Task

Use the two provided TRAPs to "echo" the user's keystrokes to the display.

Extend the user_echo.asm code to read an ASCII character from the Keyboard (using `TRAP_GETC`) and print it to the ASCII Display (using `TRAP_PUTC`) in a loop. Repeat this process until the user presses the `Enter` key. Consider which type of loop is appropriate (for, while, do-while).

You may have noticed the hint to use `.STRINGZ`. This is optional and requires reading the textbook or do some outside research. This will likely be time consuming and more trouble than it is worth. Additionally, the TAs will not support you if you run into problems using `.STRINGZ` since this is 100% completely optional.

Do not continue to Part 2 until this is working.

Part 2: Print Strings with TRAP_PUTS

File Setup

For this problem, you will create a new trap `TRAP_PUTS` and add it to the operating system file `os.asm`. You'll also create a new file called `user_string.asm` to test your new trap, similar to how `user_echo.asm` [illegible] `C` and `TRAP_PUTC` traps in the last problem. This new trap will print (or [illegible] screen ASCII Display. Then create `user_string_scrip` [illegible] priate contents.

What is a String?

A string is simply a [illegible] ive sequence, of individual ASCII characters.

To be considered a string an ASCII character array must end with the `NULL` character. A `NULL` character is a non-printable character. Typically, the number `0` is used. It is never printed; it is just to mark the end of the string.

Consider this hypothetical example of a string containing: `"Tom"` in data memory. It starts at `x4000` and uses four rows of Data Memory to hold each character, including the `NULL` terminator.

Example Address	Contents in hexadecimal	ASCII translation
x4000	x0054	'T'
x4001	x004F	'O'
x4002	x004D	'M'
x4003	x0000	NULL

Your Task

First, create `TRAP_PUTS`. Open up the file os.asm and find where `TRAP_PUTS` is located in the vector table. Then scroll down to the label `TRAP_PUTS` to see the trap's implementation. You'll notice it's empty, but this is where you will be working.

This purpose of this function is to output a `NULL` terminated string to the ASCII display. We can't pass the entire "string" to a trap because we'd run out of registers quickly if we had strings with more than 8 characters. Instead, we will pass the address of the first character of a string to the trap using `R0`.

When `TRAP_PUTS` is called, register `R0` must contain the address of the first character of the string where the caller has stored the string in Data Memory. Therefore, `R0` is considered the argument to the trap. Using the example string above, `R0` would be set to `x4000` when calling the trap.

The last character in the string must be zero, which is the null terminator, to give us a null-terminated string.

This trap does not return anything to the caller

We have provided pseudocode for this TRAP:

```
TRAP_PUTS (R0) {
      check the value of R0, is it a valid address in User Data memory?
            if it                    if not, return to caller
      load the AS                    m the address held in R0 while (ASCII
      character !
            check                    for ASCII Display
                                     ntinue, if not, keep checking until its free
            write                    to ASCII Display's data register
            load                     character from data memory
       }
       return to caller
}
```


Once you have implemented the trap, it is time to test it. Copy `user_echo.asm`, and call the copy `user_string.asm`. Open up `user_string.asm` and perform the following tasks:

1. Using the `.FILL` directive, populate User Data Memory starting at address `x4000` with the hexadecimal code for the string:

   `I love CIT 593`

   Look at the back cover of your book to find the hexadecimal code for each character. Don't forget to also set the value of the last address after your string with `NULL`. You may label address `x4000` if you'd like, but its not required.
2. Populate `R0` with the address of the first character in your string.
3. Call `TRAP_PUTS` using the appropriate TRAP number from the TRAP Vector Table shown at the top of `os.asm`.
4. Create `user_string_script.txt` by copying `user_echo_script.txt` and modifying it.
5. Test our your work, if it's not working, debug by going `Step` by `Step`.
6. Do not move on to the next part until this one is working correctly.

Part 3: Get Strings with TRAP_GETS

File setup

For this problem, you will create a new trap `TRAP_GETS` and add it to the operating system file `os.asm`. You'll also create a new file called `user_string2.asm` to test your new trap. This new trap will get a str[illegible]reate `user_string2_script.txt` with the appropriate contents.

Your Task

Back in `os.asm`, scro[illegible]`RAP_GETS`. You'll notice under the label is where you will be working. The p[illegible]n is to continually read characters from the keyboard until the `enter` key is pressed, storing each character into user specified location in User Data Memory as a string, then return the length of the string to the caller.

When the program calls the trap, the caller must pass the desired starting address as an argument in `R0`. `R0` will be the address in User Data Memory where the string that will be read from the keyboard will be stored.

The trap needs to check to ensure `R0` contains a valid address in User Data Memory. If not, return immediately without attempting to read anything from the keyboard..

Otherwise, the trap must then read in characters one by one. As it reads in each character, it must store them in data memory consecutively starting from the address passed in by the caller. Once the `enter` key is pressed (which is hexadecimal `x0D` or `x0A` depending on your machine) the trap must "NULL terminate" the string in data memory and finally return the length of the string (without including the `NULL` or `enter`) to the caller in `R1`.

As an example, let's say a program called the TRAP as follows:

```
; program sets R0=x2000
; program calls TRAP_GETS(R0)
; R1 contain the length of the string after the TRAP returns
```

For a more concrete example, let's say that, when the program calls the TRAP, the user types in `Hello` on the console, followed by an `enter`. User Data Memory would contain the following after the TRAP returns:

Example Address	Contents in hexadecimal	ASCII translation
x2000	x0048	'H'
x2001	x0065	'e'
x2002	x006C	'l'
x2003	x006C	'l'
x2004	X006F	'o'
x2005	x0000	NULL

And R1 would contain the number 5 when the trap returns. The above is just an example; any valid address in data memory can be passed in by the caller and any string could be entered by the caller on the keyboard.

After you complete te  following in `user_string2.asm`:

1. Call TRAP_GE`20` in `R0`. This will allow a string to be entered by the user and i`ress R0`
2. Using TRAP_P print

   `Length = X`

   where X is the length of the read-in string from step 1 (you can assume length will be less than 10)
3. Call TRAP_PUTS with address x2020 in R0. This will output the same string that was read in from step 1 to the Display.

Part 4: Draw Rectangles with TRAP_DRAW_RECT

File setup

For this problem, you create a new trap `TRAP_DRAW_RECT` and add it to the operating system file `os.asm`. You'll also create a new file called `user_draw.asm` to test your new trap. This new trap will get the i (starting location, size, color) from the user. Finally, create a scrip `ipt.txt` with the appropriate contents.

Your Task

Review the TRAP_DI ow to interact with the video display.

Implement TRAP_DF p will draw a rectangle whose location and dimensions will be set by the user. The color of the rectangle will also be an argument passed in by the caller.

When the TRAP is called the following registers must contain the following:

`R0` – "x coordinate" of upper-left corner of the rectangle.

`R1` – "y coordinate" of upper-left corner of the rectangle.

`R2` – horizontal length of the rectangle (in number of pixels across the display).

`R3` – vertical width of the side of the rectangle (in number of pixels down the display).

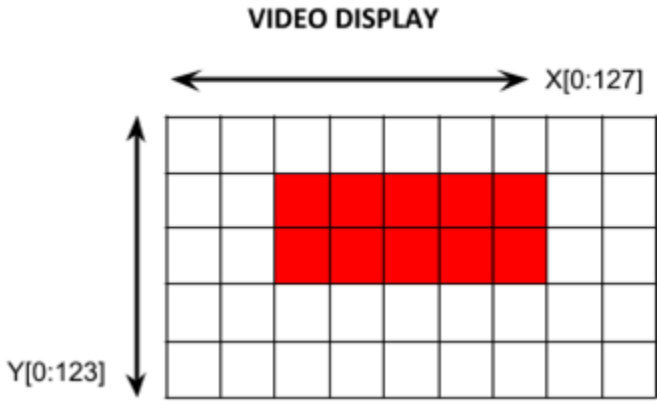
`R4` – the color of the rectangle. Read the PennSim manual for how to generate colors.

The TRAP needs to do basic boundary Checking. The TRAP checks to see if the length and width are valid from the starting location of the rectangle. If the rectangle would start outside or go outside the video display, return without attempting to draw the rectangle.

Make certain to comment the TRAP's inputs and outputs as well as key components of your code so we can understand your work.

Also make certain to comment the trap "test" program you write.

As an example of using the TRAP, suppose the user calls the TRAP with the values of: `R0=#2`, `R1=#1`, `R2=#5`, `R3=#2`, `R4=x7C00` (red). The TRAP would then draw a red box to the video display and like this:



Demonstrate your mastery of this TRAP in `user_draw.asm` which draws the following rectangles (coordinates are given in `(x, y)` order):

- A red rectangle, upper left coordinates: `(50, 5)`, length = `10` and width `5`
- A green ectangle, upper left coordinates: `(10, 10)`, length = `50`, width `40`
- A yellow recta dinates: `(120, 100)`, length = `27`, width `10`

These are some edg ask about all the time. Now they are in the instructions!

1. For `R0=127`, draw 1 pixel at `(127,123)` (the bottom right pixel).
2. If the starting vertical width `R3` would go out of bounds, then do not draw the recta
3. If the starting x- or y-coordinate is out of bounds, do not draw the rectangle, unless you are attempting the Reset Starting Coordinates extra credit.
4. If the starting x-coordinate `R0` plus length `R2` would go out of bounds, then do not draw the rectangle, unless attempting the Wrapped Rectangle Horizontally extra credit.

Extra Credit: Get a Character in a Time Limit with TRAP_GETC_TIMER

We recommend attempting this optional extra credit after you have Part 3 working successfully. You will need to read up on the Timer Device in the PennSim manual on your own. Since this is a challenge problem, the TAs will not be able to help much.

Create a new trap ca ER in `os.asm` and create two new files: `user_string_ec.as ec_script.txt`.
Your new `TRAP_GETC` ything that `TRAP_GETC` used to do, except now it must "time out" if a u ey in 2 seconds.
How to do this? Reca ecks the status register in a loop. Before entering that loop, set a timer you are inside the loop that checks the status register, you could also check the timer too. If the user doesn't press a key in 2 seconds, return back to the caller without checking the data register.
We've included a TRAP called `TRAP_TIMER` to give you an example of how to work with the timer I/O device.
Your `user_string_ec.asm` file must call `TRAP_GETC_TIMER` to get a character from the keyboard if it is entered within 2 seconds. If the user types a character within 2 seconds, print it to the ASCII display. Otherwise, your program must end gracefully without printing anything.

Extra Credit: Reset Starting Coordinates When Out of Bounds

To implement the extra credit, you will be modifying the behavior of `TRAP_DRAW_RECT`. Once you get the TRAP working, we encourage you to make a backup of your code in case your extra credit implementation is not functional in time for submission.

The original impleme v a rectangle if the rectangle would start outside the Video Display. In this entation, reset the starting coordinates to `(0,0)` and draw the rectangle st sing the original length/width values. If the reset rectangle would still g Memory bounds with the original length/width values, do not draw the recta

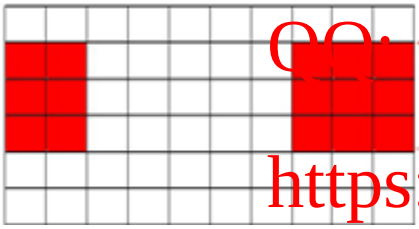
If you are attempting RECT extra credits, you should do both this extra credit and the followi same TRAP.

Extra Credit: Wrap the Rectangle Horizontally

To implement the extra credit, you will be modifying the behavior of `TRAP_DRAW_RECT`. Once you get the TRAP working, we encourage you to make a backup of your code in case your extra credit implementation is not functional in time for submission.

The original implementation does not draw a rectangle if the rectangle would go outside the horizontal limit of the video display. In this extra credit, if the box would go outside of Video Memory horizontally, correct the rectangle and make it "wrap around" the display (see diagram below). Do not wrap rectangles if they go outside video memory vertically.

If you are attempting both `TRAP_DRAW_RECT` extra credits, you should do both this extra credit and the previous extra credit in the same TRAP.



Submission

Where to put the files

Leave all the files in the working directory where the starter code started.

Pre-Submissio

There are no pre-sub

The Actual Su

There are two parts t ment: Codio and Gradescope.

Codio Submission

When you are ready (before the deadline), go to Education -> Mark Complete.

Gradescope Submission

You will need to create a single page PDF and upload it to Gradescope.

Match every question to this single page.

Do not submit a copy of your code in the PDF.

This PDF requires two things:

1. **Academic Integrity statement and signature.** You can use this as a template (an entire word document template is available on Canvas):

   Academic Integrity Agreement:

   By submitting this agreement I certify that I have completed this homework assignment **on my own** (without collaboration with another student or unauthorized outside source) and have not **plagiarized** on this assignment (in accordance with Penn's Code of Academic Integrity).

   _____ (your name, just type it in)

2. **A screenshot of your Completed Codio workspace.** It must show:
   a. your Codio username
   b. the Module number for this assignment (do not reuse the screenshot between assignments; you will get a 0)
   c. A screenshot of an indication that the workspace is complete:
      i. The Warning that pops up after Marking Complete and typing "yes", OR
      ii. The Education dropdown menu showing that Mark as Completed is inactive (greyed out)

Grading

Main Assignment

This assignment is worth 220 points, which will be normalized to 100% for gradebook purposes.

All problems have pa

10 points for script fil

10 points for commer

10 points for Part 1: E /TRAP_PUTC

50 points for Part 2: F P_PUTS

50 points for Part 3: G _GETS

90 points for Part 4: Draw Rectangles with `TRAP_DRAW_RECT`

Do note that we are only grading for correctness, not efficiency.

Extra Credit

The extra credits are worth a total of 10 percentage points. The maximum score on this assignment is 110%. As usual, the extra credit does not have partial credit.

5 percentage points for correctly implementing `TRAP_GETC_TIMER`.

1 percentage point for correctly implementing `TRAP_DRAW_RECT` to reset the coordinates.

4 percentage points for correctly implementing `TRAP_DRAW_RECT` to wrap around horizontally.

An Important Note of Plagiarism

- We will scan your assignment files for plagiarism using an automatic plagiarism detection tool.

- If you are unaware of the plagiarism policy, make certain to check the syllabus to see the possible repercussions of submitting plagiarized work (or letting someone submit yours).

FAQ

Quick Hints

These are some hints provided by TAs.

-

Resources

- ASCII table
  https://www.a
-