DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Homework Assignment No. 01:

MIPS Assignment #1

submitted to:

Professor Son Nguyen
ECE 4612: Advanced Processor Systems
Temple University
College of Engineering
1947 North 12th Street
Philadelphia, Pennsylvania 19122

September 20, 2024

prepared by:

Leo Berman
Email: leo.berman@temple.edu

A. OBJECTIVES

The purpose of this assignment is to demonstrate basic knowledge of MIPS assembly by performing several tasks. These tasks are:

- Copy a string of characters
- Find factorial of a number
- Print a diamond shape to I/O window

Copying a string of characters requires manipulation of data. Finding the factorial of a number requires taking input from the command line, printing to the command line, and utilizing leaf procedures. Finally, printing a diamond shape to the I/O window, requires everything from task 1 and task 2 plus the ability to execute an algorithm in non-leaf procedures.

B. TOOLS/EQUIPMENT

The tool used in this assignment is the MARS MIPS Simulator.

C. PROCEDURE

Task 1:

- Declare string variable Y containing name
- Declare second variable X containing enough memory to contain name
- Load the beginning address of X and Y into a0 and a1 respectively.
- Loop through copying each byte of Y into X until a Null character is found
- Exit the program

Task 2:

- Load 2 variables in with the input and output prompts
- Print the input prompt
- Wait for the user to input a number
- Load that input into a register
- Jump to a routine that multiplies the number by the original number – the loop index and stores the output in the same register
- At the end of the loop print the output prompt and the output
- Exit the program

Task 3:

- Load 5 string variables into memory holding input and output responses and also “*”,” “, “\n”
- Print the input prompt
- Wait for the user to input a number
- Load that input into a register
- Check if the input is zero and if it is print the output prompt and exit the program
- Check if the input is one and if it is print a single start and exit the program
- Set 2 registers s0 and s1 to have the input number - 1 and input number – 2 respectively
- Set an iterator index to zero and jump to a leaf procedure that iterates printing spaces until the iterator equals input number -1
- Return to main and print an asterisk
- Initialize the stack pointer by moving it down 4 bytes
- Set an iterator index(i) to zero and jump to a non-leaf procedure that iterates until the iterator equal input number – 1. For each of those iterations do the instructions indented below
    - Save $ra onto stack
    - Set an iterator index(j) to zero and iterate until j = input number – i – 2 printing spaces
    - Return to the i level of the loop and print an asterisk
    - Set an iterator index(j) to zero and iterate until j = 2 * i + 1 printing spaces
    - Return to the i level of the loop and print an asterisk
    - Print an asterisk and a newline character
    - Pop from stack into $ra
    - Return to the main function
- Set an iterator index(i) to zero and jump to a non-leaf procedure that iterates until the iterator equal input number – 2. For each of those iterations do the instructions indented below

  - Save $ra onto stack
  - Set an iterator index(j) to zero and iterate until j = i + 1 printing spaces
  - Return to the i level of the loop and print an asterisk
  - Set an iterator index(j) to zero and iterate until j = 2 * (input size - i – 2) + 1 printing spaces
  - Return to the i level of the loop and print an asterisk
  - Print an asterisk and a newline character
  - Pop from stack into $ra
  - Return to the main function
- Exit the program

D. RESULTS

Task 1:

Initially we load a string into the Y variable which is stored in static memory. We also allocate space in static memory for the X variable to be the size of the ascii string. Next, we store the addresses of X and Y into a0 and a1 respectively. Finally, we increment through by 4 bytes checking for a null terminating character copying each set of 4 bytes from the Y register to the X register.

1. X & Y get placed in static memory
2. Then the beginning address of X & Y get placed into a0 and a1 respectively
3. Load those X & Y addresses into t3 and t0 respectively
4. Load the bytes stored at those address from Y into t2
5. Store those bytes into the address in t3
6. Repeat steps 3 -> 5 until a null pointer is found, then break and exit.

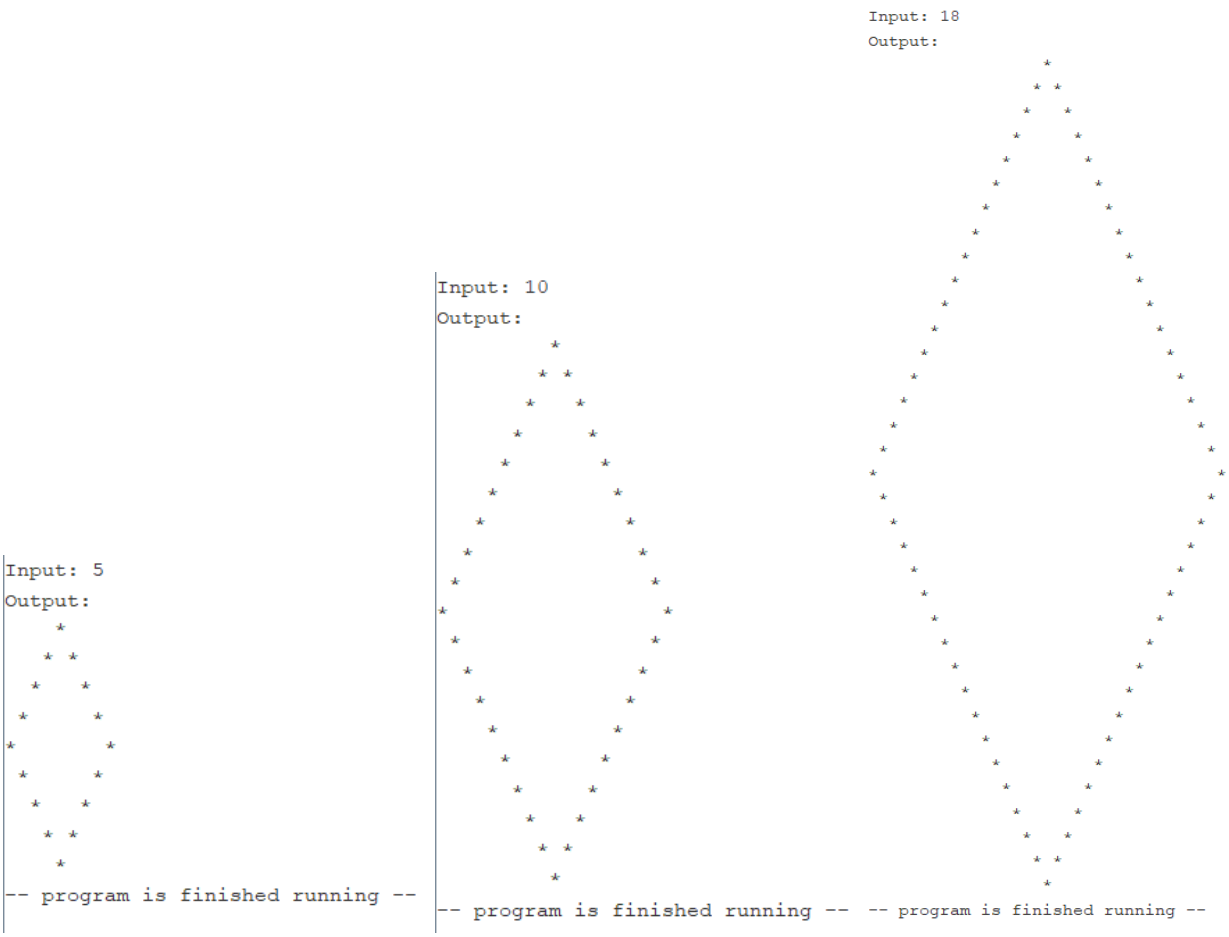
Task 2:

```
Enter a number to find factorial: 1
Here's your answer: 1
-- program is finished running --
Enter a number to find factorial: 2
Here's your answer: 2
-- program is finished running --
Enter a number to find factorial: 4
Here's your answer: 24
-- program is finished running --
Enter a number to find factorial: 8
Here's your answer: 40320
-- program is finished running --
Enter a number to find factorial: 12
Here's your answer: 479001600
-- program is finished running --
```

The program breaks at 13 due to overflow limits.

Task 3:

```
Input: 1
Output:
*
-- program is finished running --
```



18 was about where the diamond stopped being able to be shown in a single screenshot.

E. CONCLUSION

Overall, the most valuable the first and second problems didn't demand very much thought to get working due to the availability of the sample code. However, problem 3 forced me to go back and really understand what was going on in problems 1 & 2. After understanding some more of the basics, I adjusted the code for readability and just tried out other instructions. Problem 3 was a great learning experience in terms of learning to use non-leaf procedures. I spent a long time trying to figure out how to access the program counter so I could manually load the current address onto the stack. Later I learned that it doesn't work that way and although there is a way to access the program counter, it's inefficient due to having to use a command to jump to the next line while storing that same line in a register then loading that register onto the stack. Instead, I utilized nested labels for non-leaf procedures with nested loops. I think that because I prototyped problem 3 in Python first, I did kind of narrow my vision down to doing it a particular way, but it did make the process a little more painless.