COSC 211 Lab 4 – Procedures

(DUE Sunday, October 29th, 2023)

0.Introduction

In this lab, you will have the opportunity to explore using non-recursive and recursive procedures on the MIPS architecture. You will gain experience with stack management. Additionally, there are questions from the text book that will also cover ASCII characters. This lab will take considerable time so do not leave it to the end. You will have two lab periods to work on the materials. There are a total of 65 marks available for this assignment.

The assignment is broken into two parts. The written submission can be hand written and scanned or typed. Work must be clear with all work shown. The coding part of the assignment requires two submissions. Your code is to be submitted through Canvas before **by the end of the day on Sunday, October 29th** along with the written component. Submissions will be accepted earlier. **Late submissions will not be accepted**. Even if you do not complete all questions, please try them all and submit what you have for partial marks. This material will be covered on the midterm exam, so if you have questions, so please ensure that you review this material before the midterm exam.

1.Programming Assignment (/44)

MIPS Procedures and the Call Stack

Procedures

Unlike in higher-level languages such as Java and C, procedures (aka functions) require a bit of care-and-feeding in MIPS assembly. *Your textbook has a solid overview in **section 2.8, pg. 96** if you require a more in-depth explanation.*

Before continuing, it is important to keep in mind that the purpose of a procedure is to provide *self-contained, reusable* blocks of code. The most common errors faced by programmers when working with procedures are usually the result of side-effects related to the procedure not being as "self-contained" as was intended.

As a guideline for using procedures, you can follow these 6 steps (pg. 97 in the textbook)

1. Put the parameters in a place where the procedure can access them
   Registers `$a0 - $a3`
2. Transfer control to the procedure as `jal ProcedureLabel`.
3. Acquire the storage resources needed for the procedure.
   Save registers `$s0 - $s7` to memory IF you need to overwrite them in your procedure. Registers `$t0-$t9` do not need to be saved. Wise register usage and planning can save you work.
4. Perform the desired task.
5. Put the result value in a place where the calling program can access it.
   Registers `$v0, $v1` – if using saved registers `$s0 - $s7`, recover previous values from memory and load back in to the registers.
6. Return control to the point of origin, since a procedure can be called from several points in a program as `jr $ra`

Register Usage

As procedure calling is governed by caller-callee convention, registers are generally used for the following items:

- $a0 - $a3: Argument registers reserved for passing parameters to procedures.
- `$v0, $v1`: Result registers to store the values returned by procedures.
- `$ra`: Return address register which stores the address of the next instruction to execute when leaving the procedure.
- `$t0 – $t9`: Temporary registers that can safely be overwritten without worrying about their previous contents.
- `$s0 – $s7`: Saved registers that must have their contents saved to memory before being overwritten by your procedure. Before returning control back to the caller, these saved values must be loaded back in to their corresponding registers.

Assembly Commands for Procedures

As we saw in lecture, with reusable procedure code, we cannot just jump to a specific procedure and safely hope we will return to the correct place in the line of execution. Thus, we have two new instructions to help facilitate this action:

- `jal [Procedure Label]`**:** Jump and Link – stores the current instruction address + 4 in **$ra** (so that the PC can be set to the *next* instruction to execute when leaving the procedure) and sets the PC to the address of the called procedure.
- `Jr $ra`: Jump register – jumps to the address stored in **$ra**.

Call Stack

When working with procedures that require more than just the temporary registers, or itself calls another procedure, temporary data needs to be written out to memory in order to preserve the state of the procedure. There is a special location in memory called the *stack,* which is reserved specifically for this purpose. The stack is pointed at by the stack pointer (`$sp`), and grows/shrinks as data is **pushed** (added to) and **popped** (removed from). Historically the stack grows top-down, so **growing the stack is done with a *subtraction***, and **shrinking the stack is done with an *addition***.

Each procedure needs to know how much space it requires on the stack, and will manage pushing values on to the stack, and decrementing the stack pointer (at the start of the procedure), and then popping values off the stack and incrementing the stack pointer when done (before leaving the procedure). To manage the increasing complexity with the stack, you can use the prologue-body-epilogue model presented in lecture to help partition the different actions required for managing stack operations. ***Important Note****: It is the CALLEE's (procedure's) duty to push variables on to the stack when first entered, and to pop values off the stack before exiting.*

From textbook pg.102

Preserved	Not Preserved
Saved Registers: *$s0 - $s7*	Temporary Registers: *$t0 – $t9*
Stack Pointer: *$sp*	Argument Registers: *$a0 - $a3*
Return Address Register: *$ra*	Return Value Registers: *$v0 - $v1*
Stack above the stack pointer	Stack below the stack pointer

Programming Exercise

1. (/4) Download the program array.s and load it into QTSpim. In the panel ``Text Segments", there are 4 columns, representing respectively the memory address, machine encoding (in hexadecimal), assembled assembly code, and the original assembly code. Answer the following questions (**tracing the code if necessary**):

    1. What is the memory address of the array int_array?
    2. The branch instruction beq is encoded as 0x10200009. Explain how the digit 9 in the encoding is determined?
    3. Explain why the jump instruction j is encoded as 0x0810000f?

    Submit your answers in a text file named exercise1.txt.

2. (/20) Write a program to compute and output the first 100 prime numbers. Your implementation will use a procedure **test_prime(n)** that accepts a parameter n and tests if n is a prime number. The procedure **test_prime(n)** returns 1 if n is prime, and 0 otherwise.

    Tips:

    - Based on the input to the procedure, you will need to determine the range over which a number needs to be tested.
    - A number is prime if and only if no number, except 1 and itself, divides it evenly. Thus, you will have to check each number to make sure that this holds true for each number. **You can use a for-loop to check this**.
    - To check if a number A can be divided evenly by a number B, use the division instruction div A, B where A is the dividend and B is the divisor held in registers(ie div $s0, $s1). This instruction puts the remainder of the division in the register HI and the quotient in the register LO.
    - To copy the value of HI to a general register, use the instruction mfhi. To copy the value of LO to a general register, use mflo. This usages follows as:

```
mfhi $s0    #this will move the remainder of the division
            #into the register $s0 which can then be checked

mflo  $s1   #this will move the quotient of the division
            #into the register $s0 which can then be checked
```

    - When developing your program, it would be best to write and test your procedure first to make sure that it works correctly.

    Include a program comment header that has your name, student number and program name. **Name your file excercise2.s.**

3. (/20) Write a program named hanoi.s to solve the Towers of Hanoi puzzle. For a detailed description of the puzzle, see https://en.wikipedia.org/wiki/Tower_of_Hanoi. You have to use recursions to solve the puzzle. Your program should first prompt the user to input the number of disks, and then solve the puzzle recursively, as illustrated in this C program hanoi.c and will use a procedure called hanoi Your program should also print, in each recursion, the current number of disks on the pegs in the following format :

```
Enter the number of disks: 3
[3, 0, 0]
[2, 0, 1]
[1, 1, 1]
[1, 2, 0]
[0, 2, 1]
[1, 1, 1]
[1, 0, 2]
[0, 0, 3]
```

Since your implementation uses recursive procedures, it is important that you follow the procedure call convention, including the parameter passing and the register usage. Otherwise, your program may not work correctly. **Hint: look at the c code implementation and think about writing this in assembly.**

This code presents the opportunity to use both recursive and non-recursive procedures. While the Hanoi procedure is recursive, you can build up 'helpers' to make coding easier and more compact. Once such helper you may wish to use is to print the state of each tower.

The global variable number_of_disks in hanoi.c is used to store the current number of disks on the pegs 0, 1, and 2. You can implement it as a global variable in your assembly program as well. Include a program comment header that has your name, student number and program name. **Name your file excercise3.s.**

Programming Submission

For questions 1, 2 and 3, submit both your files through Canvas. All files should be in a single .zip file. Submit your files through Canvas before the assignment due date.

2. Written Assignment (21 marks)

1. Instructions: Language of the Computer (Show all work. Part marks will be assigned for all questions)

Consider the following section of C code:

```
int my_global = 100;

main()
{
    int z;
    my_global  += 1;
    int x;
    z = leaf_function(x=0);
}

int leaf_function(int x)
{
    return x + 1;
}
```

Assume the calling convention discussed in class and that function inputs are passed using registers $a0-$a3 and the results are returned in $r0. **Assume that the leaf function can only use saved registers.**

a. (10 marks) Write the MIPS assembly code (complete program) for this code. (10 marks)
b. (4 marks) For this, show the contents on the stack before and after the procedure call. Assume that the stack pointer starts at `0x7ffffffc`. You may wish to draw the changes in the stack frame (Hint: please draw).
c. (5 marks) If the leaf function could use temporary registers ($t0, $t1, etc), write the MIPS assembly for this code.

2. Translate the following strings into hexadecimal ASCII byte values:
   a. (1 mark) hello word
   b. (1 mark) 0123456789

Written Submission

Submit your written (typed or scanned) assignment in via Canvas on the above indicated due date. Please take care and effort to make your assignment answers as clear as possible.