FIT1008 – Intro to Computer Science
Tutorial 4

Semester 1, 2017

Objectives of this tutorial

- To understand the function calling and returning in MIPS.
- To be able to write simple MIPS functions.
- To understand memory maps.

Exercise 1

Consider each of the steps performed as part of the function call/return convention.

1. Caller: saves temporary registers by pushing their values on to the stack
2. Caller: prepares the arguments by pushing them on to the stack
3. Caller: calls the function using the `jal` instruction
4. Callee: saves `$ra` by pushing its value on the stack
5. Callee: saves `$fp` by pushing its value on the stack
6. Callee: copies `$sp` to `$fp`
7. Callee: allocates local variables by reserving enough space onto the stack
8. Callee: if there is a return value, stores it in `$v0`
9. Callee: deallocates local variables by popping the previously pushed space
10. Callee: restores `$fp` by popping its saved value off the stack
11. Callee: restores `$ra` by popping its saved value off the stack
12. Callee: returns using the `jr $ra` instruction
13. Caller: clears the function arguments by popping their allocated space off the stack
14. Caller: restores temporary registers by popping their values off the stack

15. Caller: uses the return value `$v0` if necessary

For each of the steps above:

- Explain the rationale behind the step i.e., explain what is the step trying to achieve in terms of functionality (e.g., needed to allow more than one functions to be called, needed to be able to pass a non-fixed number of parameters, etc).
- Disscus whether the step **must** be performed by the caller/callee or whether this is just a matter of convention (i.e., someone had to do it).

Exercise 2

Consider the following Python code:

```
def collatz(n):

    ## HERE

    if n % 2 == 0:
       return n/2

    return 3*n + 1

n = int(input("Enter integer: "))

while (n > 1):
   print(n)
   n = collatz(n)
```

(i) Draw a stack diagram at the time `## HERE` is found.

(ii) Translate the above program into MIPS. Try to make your translation as faithful as possible.

Exercise 3

Translate into MIPS the following function:

```
def even_product(a_list):
    product = 1
    for x in a_list:
        if x%2 ==0:
            product=product*x
    return product
```

Exercise 4

(i) The function calling convention given in lectures typically has functions accessing their first parameter at `8($fp)`, the second at `12($fp)`, the third at `16($fp)`, and so on.

Is this **order** necessary? In other words, would it be possible to have the last parameter at `8($fp)` instead, and the second-last at `12($fp)`, and so on (provided that all functions are changed to agree with this new convention)?

(ii) Why is the memory at address `4($fp)`seldom accessed by a called function?