FIT2085-S1-2021
Week 4 Workshop Malaysia

Learning Objectives

- Understanding local variables in MIPS.
- Understanding function calling conventions in MIPS
- Understanding function return conventions in MIPS

Stack and frame pointers

Tick all the correct answers.

Question 1

The stack pointer $sp:

A. Always points to the saved $ra of the main function.
B. Is always equal to $fp of the current function
C. Decreases when a function is called.
D. Always points to the next address in the stack that can be written into (but it may contain garbage).
E. Is automatically updated by the system.
F. Does not need to be saved on the stack between two function calls.

Question 2

The frame pointer $fp:

A. Always point to the saved $fp of the previous frame, if there is one.
B. Is always equal the frame number, stored as a 32 bits integer, which is equal to 0 if we are in the main function.
C. Always has the same value in the same function.
D. Can be used with a negative shift (with for instance -4($fp)) to access an argument of the current function.

Memory diagrams and calling convention

Consider the following Python code:

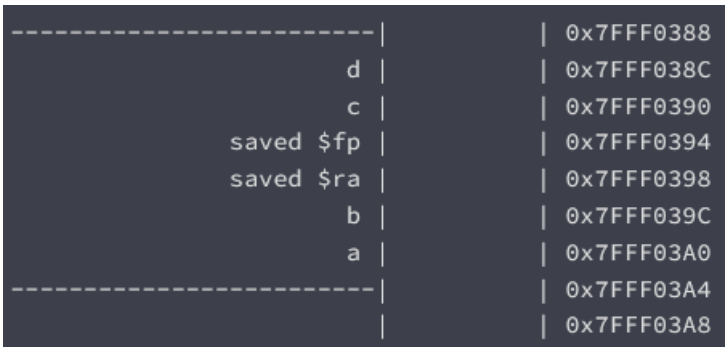
```python
import typing

def following(a:int, b:int) -> int:
    c = a-b
    d = a*b
    return d//c
```

Question 1

Which of the memory diagrams below correspond(s) to a correct faithful translation of the *following* function? (here we the values in the "middle" column do not matter - they are not represented).

Solution:



Question 2

Within the following function, what is the value of $fp?

Solution: d

Callees in MIPS

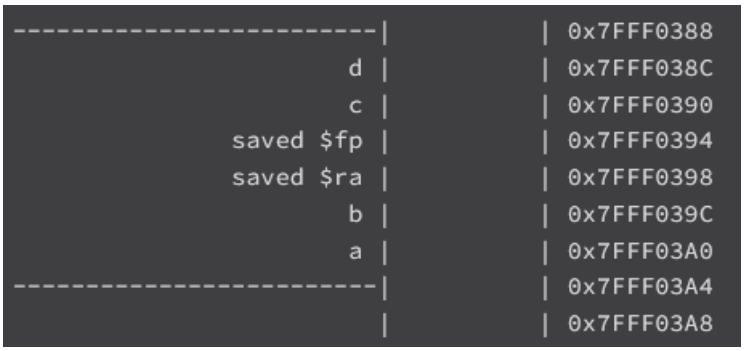
Consider the following Python code:

```python
import typing

def following(a:int, b:int) -> int:
    c = a-b
    d = a*b
    return d//c
```

Faithfully translate the following function into a properly commented MIPS program using the file provided by **replacing** the TODO lines with your code. You can "mark" this to check whether your function passes the tests.

We will use the diagram of the stack frame of the following function as shown below:



(Note that the addresses on the right are just an example: they will vary depending on the state of the stack at the point where the function is called.)

For convenience, we will write this as a comment in the code in this condensed and more useful format:

```
# d is at -8($fp)
# c is at -4($fp)
# saved fp is at ($fp)
# saved ra is at +4($fp)
# b is at +8($fp)
# a is at +12($fp)
```

Note that we restrict ourselves to a>b cases to avoid getting into the case where Python's // and MIPS's div disagree.

Callers in MIPS

Consider the following Python code:

```
import typing
from following import *

def main() -> None:
    x = int(input("Enter integer: "))
    y = int(input("Enter integer: "))
    print(following(x, y))

#in Python there is no default "main" function
#we need to indicate what to do if this file is run.
if __name__=="__main__":
    main()
```

Faithfully translate the main function into a properly commented MIPS program using the file provided by **replacing** the TODO lines with your code. You can "mark" this to check whether your function passes the tests.

For convenience, here is the stack frame diagram of the function main:

```
# y is at -8($fp)
# x is at -4($fp)
```

and the one of the following function:

```
# d is at -8($fp)
# c is at -4($fp)
# saved fp is at ($fp)
# saved ra is at +4($fp)
# b is at +8($fp)
# a is at +12($fp)
```

(where $fp refers to the frame pointer of following.)

Note (again) that we restrict ourselves to x>y cases to avoid getting into the case where Python's // and MIPS's div disagree.