COMPSCI 2GA3 Tutorial 4 Note

Note:

This note does NOT cover all the materials in Chapter 2 -- Only the ones rated to sample questions of this tutorial are included.

For any questions about the tutorials and courses, feel free to contact me. (Email: wangm235@mcmaster.ca)

GLHF :)
Mingzhe Wang

NO ADDITIONAL MATERIAL IN THIS WEEK'S NOTE --- HAPPY READING WEEK :)

What is the “Caller”? What is the “Callee”?

```java
public class Main {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

In this simple Java code:

`main` is the caller,

`System.out.println` is the callee.

Quite easy, right? :)

Leaf Procedure Example

- C code:

```
long long int leaf_example (
 long long int g, long long int h,
 long long int i, long long int j) {
   long long int f;
   f = (g + h) - (i + j);
   return f;
}
```

By convention, we only need to store x20 (callee).
Saving x5, x6 is just for the purpose of example.

- Arguments *g, h, i, j* in x10, …, x13, *f* in x20
- Temporaries x5, x6
- Need to save x5, x6, x20 on stack

Leaf Procedure Example

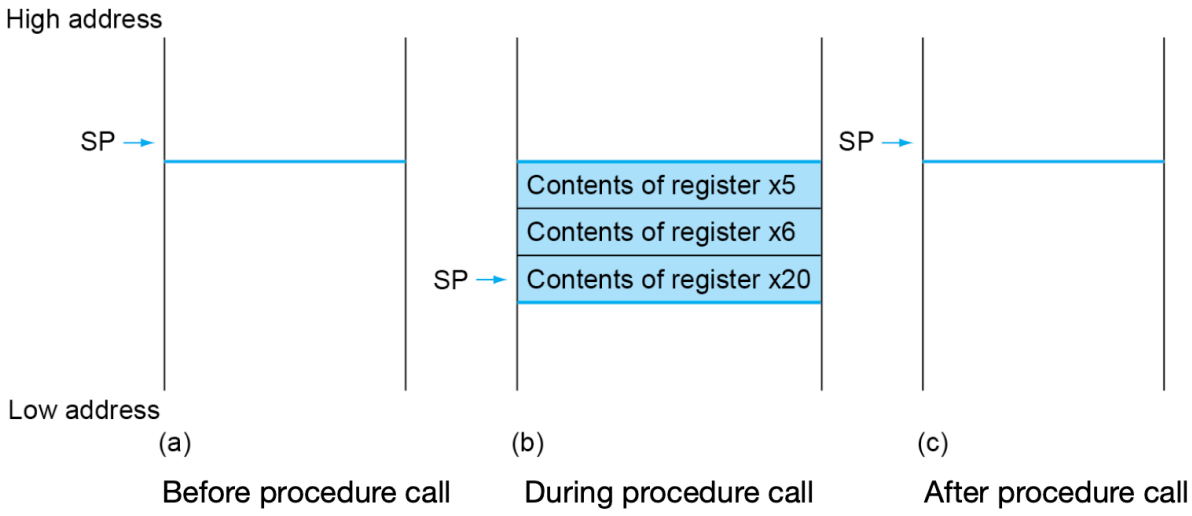
Assume 64-bit (8 bytes) memory.

- RISC-V code:

```
leaf_example:            // label of the procedure
   addi sp,sp,-24        // adjust stack to make room for 3 items
   sd   x5,16(sp)        // save register x5
   sd   x6,8(sp)         // save register x6
   sd   x20,0(sp)        // save register x20
   add  x5,x10,x11       // register x5 contains g + h
   add  x6,x12,x13       // register x6 contains i + j
   sub  x20,x5,x6        // f = x5 - x6
   addi x10,x20,0        // returns f (x10 = x20 + 0)
   ld   x20,0(sp)        // restore register x20 for caller
   ld   x6,8(sp)         // restore register x6 for caller
   ld   x5,16(sp)        // restore register x5 for caller
   addi sp,sp,24         // adjust stack to delete 3 items
   jalr x0,0(x1)         // branch back to calling routine
```

(Note: Leaf procedure --- procedure that doesn't call another precedure in its body.
Non-leaf procedure -- procedure that calls another precedure in its body.)

Local Data on the Stack



The values of the stack pointer and the stack

(More example: Textbook P98 – P102)

Difference between jal and jalr

Procedure Call Instructions

- Procedure call: jump and link (another procedure)

  `jal x1, ProcedureLabel`
  - Address of following instruction put in x1
  - Jumps to target address

- Procedure return: jump and link register (to your caller)

  `jalr x0, 0(x1)`
  - Like `jal`, but jumps to 0 + address in x1
  - Use x0 as rd (x0 cannot be changed)
  - Can also be used for computed jumps e.g., for case/ switch statements

One word : when you call another procedure, use jal; when you want to return to your caller (the procedure which calls you), use jalr.

Calling Convention

RISC-V Calling Convention			
Register	ABI Name	Saver	Description
x0	zero	---	Hard-wired zero
x1	ra	Caller	Return address
x2	sp	Callee	Stack pointer
x3	gp	---	Global pointer
x4	tp	---	Thread pointer
x5-7	t0-2	Caller	Temporaries
x8	s0/fp	Callee	Saved register/frame pointer
x9	s1	Callee	Saved register
x10-11	a0-1	Caller	Function arguments/return values
x12-17	a2-7	Caller	Function arguments
x18-27	s2-11	Callee	Saved registers
x28-31	t3-t6	Caller	Temporaries

In another word, when you are writing RISC-V code:

If that register is callee saved by convention, we should make sure that register's value not changed after your procedure. (Solution: use a stack to keep its original value at the beginning of our procedure and restore the original value at the end of our code.)

If that register is caller saved by convention, NO need to do anything. Because it's NOT our (callee) responsibility to keep that's register's value not changed. However, that also means when you try to call another procedure and your data is stored on a caller saved register (this time, you are the caller), your original data could be lost after calling, because it's NOT callee's responsibility to maintain this register's value. (Solution: store your data in callee saved registers / use a stack to store your data before calling another procedure).
e.g.
addi x10, x6, 1 // x10 = x6 + 1
jal x1, g // call function g
what's in x10? God knows!!!