Translate Function to Assembly Code

1 Why do we need to save affected registers in the function?

Assume we have a caller and function func() and there are assembly code for them.

```
Caller:
L1:        addi  $t0, 0, 100
L2:        addi  $a0, 0, 50
L3:        jal  funct
L4:        addi  $t1, $t0, $v0

func:
L5:        addi   $t0, $a0, 2
L6:        addi   $v0, $t0, 0
L7:         jr      $ra
```

Let walk through the above code. In the caller, the value of $t0 is 10 before we call funct( ). But the value of $t0 is changed to be 52. What does make this happen? The modification of $t0 is done by func( ). In the func, the first instruction addi does change the value of t0, whose label is L5. This is NOT what we expect. In avoid this issue, we need to save the affected registers before entering the body of procedure/function.

Here is the correct code

```
func:
L51:    sw  $t0 , ($sp)
L52:    addi $sp, $sp, -4

L5:        addi   $t0, $a0, 2
L6:        addi   $v0, $t0, 0

L61:     lw $t0, 4($sp)
L62:     addi  $sp, $sp, 4
L7:        jr      $ra
```

2 How do we save affected registers?

We use the sw instructions to save registers to stack, which is a part of memory.
In the sw instruction, we need provide the offset and base register to specify the address which data are placed. Since storing registers on stack, we use the $sp as base register.

```
Sw   $t0, offset($sp)
```

How do we determine the value of offset? It depends on the number and order of registers you are going to store. For example, you have two registers, $t0 and $t1 to save.

```
Sw $t0, 0($sp)
Sw  $t1, -4($sp)
```

You may are confused by the offset in this example. Why are they negative numbers? This is the convention of MIPS. The stack growth direction is from high address to low address. If you want to allocate memory space on stack, these data are stored in the addresses lower than original $sp, which holds the stack pointer.

After that, you need to adjust the $sp to reflect the fact that you have allocated two words on the stack.

```
Addi  $sp, $0, -8
```

Alternatively, you can first change $sp and then store registers to stack. In this case, your code is little different.

```
Addi  $sp, $0, -8
Sw $t0, 8($sp)
Sw  $t0, 4($sp)
```

3 How do we restore affected registers?

We use the lw instructions to load saved in stack to registers. In the lw instruction, we need provide the offset and base register to specify the address which data are placed. Since loading data from stack, we use the $sp as base register.

```
lw    $t0, offset($sp)
```

The offset is similar to the saving registers to stack. But there is little difference.

4 Framework for function/procedure

Part 1 save Affected Registers

Part 2 Body of function

Part 3 Restore Affected Registers and return

Part 1:

1) Allocate stack frame (decrement stack pointer)
2) Save any registers (callee save registers)

Part 2:

3) Procedure body (remember some arguments may be on the stack!)

Part 3:

4) Restore registers (callee save registers)
5) Pop stack frame (increment stack pointer)
6) Return (jr $ra)

5 Tip to Write Procedure

1) Finish the body of procedure (Part 2)
2) Identify registers which are written in the body of the procedure
3) Write code to save these affected registers (Part 1)
4) Write code to restore these affected registers (Part 3)