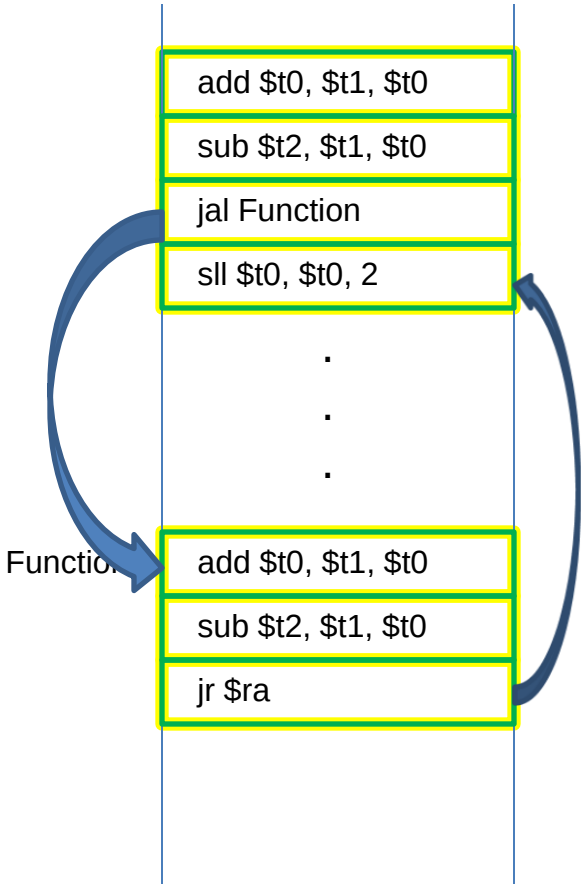
MIPS Function Continued

Review

- Function
  - A consecutive piece of code doing a specific thing
  - To go to the function, use `jal Function`, which does two things:
    - goes to the code starting at the address associated with label `Function`,
    - stores the address of the instruction immediately following the jal instruction into `$ra`.
  - To return from the function, use `jr $ra`, which takes the code back to the instruction following the jal instruction.



Review

- Stack
  - A piece of memory
  - Last in first out
  - Use `$sp` to keep track of the first used element on the stack

Character and String Operations

- Characters are encoded as 0's and 1's using ASCII most commonly
  - American Standard Code for Information Interchange
  - Each character is represented using 8 bits (or a byte)
  - If stored within an integer, the 8 bit character portion will be located within the lowermost ordered bits; this allows you to optionally store characters within integers
- MIPS provides instructions to move bytes
  - Load byte (`lb`) loads a byte to the rightmost 8 bits of a register
  - Store byte (`sb`) write the rightmost 8 bits of a register to memory

SPIM syscalls

- Syscalls are operations defined within the assembler (not the processor)
- The syscall operation is dependent on the value in $v0
- The parameters to the operation are in $a0-$a3 (except for floating point numbers and then it is $f12)
- Read operations store the value back in $v0 (again except for floating point numbers and then it is $f0)

SPIM syscalls
Integers

```
li $v0,1     # print an integer in $a0
li $a0,100
syscall

li $v0,5     # read an integer into $v0
syscall
```

SPIM syscalls
Characters

```
li $v0,11   # print a character in $a0
li $a0,'a'
syscall

li $v0,12   # read a character into $v0
syscall
```

SPIM syscalls
Strings

```
li $v0,4    # print an ASCIIZ string at $a0
la $a0,msg_hello
syscall
```

Don’t worry about reading in strings

SPIM syscalls
Floating Point

```
li $v0,2     # print a single precision
li $f12,5.5 # floating point number in $f12
syscall

li $v0,3     # print a double precision
li $f12,5.5 # floating point number in $f12
syscall

li $v0,6     # read a single precision
syscall      # floating point number into $f0

li $v0,7     # read a double precision
syscall      # floating point number into $f0
```

SPIM syscalls
Others

```
li $v0,10    #exit
syscall
```

String Copy Procedure

```
void strcpy (char x[], char y[])
{
   int i;
   i = 0;
   while ((x[i] = y[i]) != 0)  /* copy and test byte */
      i = i + 1;
}
```

```
                .data
msg_hello:
                .asciiz "Hello\n"

msg_empty:
                .space 400

                .text
                .globl main
main:

done:
                li $v0,4
                la $a0,msg_hello
                syscall

                li $v0,4
                la $a0,msg_empty
                syscall

                la $a0,msg_empty #dst
                la $a1,msg_hello #src
                jal strcpy

                li $v0,4
                la $a0,msg_empty
                syscall

                li $v0,10 #exit
                syscall


strcpy:
                lb $t0, 0($a1)
                sb $t0, 0($a0)
                addi $a0, $a0, 1
                addi $a1, $a1, 1
                bne $t0, $0, strcpy
                jr $ra
```

Stack

- Key things to keep in mind:
  - Stack is a software concept – last in first out, that's it.
  - In MIPS, you implement the stack by yourself by keeping `$sp` always pointing to the top element on the stack
  - Stack can be used in functions to save register values, and is the standard approach to save register values. But
    - You can also use stack for other purposes
    - This is not the only way to save register values.

```
        .data
msg:    .asciiz "hello
world"
endl:   .asciiz "\n"

        .text
        .globl main
main:
        addi $sp,$sp,-1
        sb $0,0($sp)
        la $t1, msg
L0:
        lb $t0,0($t1)
        beq $t0,$0, L1
        addi $sp,$sp,-1
        sb $t0,0($sp)
        addi $t1,$t1,1
        j L0
L1:
        la $t1,msg
L2:
        lb $t0,0($sp)
        addi $sp,$sp,1
        sb $t0,0($t1)
        beq $t0, $0, L3

        addi $t1,$t1,1
        j L2

L3:
        la $a0,msg
        li $v0,4
        syscall

        la $a0,endl
        li $v0,4
        syscall

        li $v0,10 #exit
        syscall
```

Inclass excercise