MIPS Functions

Questions

A MIPS function is called by the jal instruction, which does two things: 1) going to the address of the first instruction in the function, 2) passing the arguments in $a0 to $a3.

Questions

A MIPS function must be ended by the jr $ra instruction.

Questions

MIPS functions are stored in a different part in the memory and not in the same part as the main function.

Questions

A MIPS function has explicitly declare a name along with the list of arguments to be passed to it, including the names and the types.

Questions

A function in MIPS cannot have loops.

Questions

In MIPS, the name of a function is a special data type.

Questions

The jal L1 instruction jumps to L1, and saves the address of L1 into $ra.

Questions

Suppose $s0, $v0, and $a0 are holding 60, 0, and 35, respectively. After the program executes till p9L2, what will be the value in $v0?

```
        jal p9L1
        j p9L2
p9L1:   add $v0, $v0, $a0
        blt $v0, $s0, p9L1
        jr $ra
p9L2:
```

(a) 100

(b) 70

(c) The program will never run to p9L2.

(d) None of the above.

In MIPS, "`la $t0, L1`" is a pseudo instruction which loads the address of the instruction associated with `L1` into `$t0`. If "`jal`" is not supported by hardware and is a pseudo instruction, which of the following correctly implements instruction "`jal f1`?"

```
(a)    la $ra, L1
       j f1
L1:    nop          # or any instruction after calling f1

(b)    la $t0, f1
       jr f1
L1:    nop          # or any instruction after calling f1

(c)    la $ra, f1
       j f1
L1:    nop          # or any instruction after calling f1
```

(d) None of the above.

MIPS Calling Conventions

- MIPS assembly follows the following convention in using registers
  - $a0 - $a3: four argument registers in which to pass parameters
  - $v0 - $v1: two value registers in which to return values
  - $ra: one return address register to return to the point of origin

MIPS Conventions

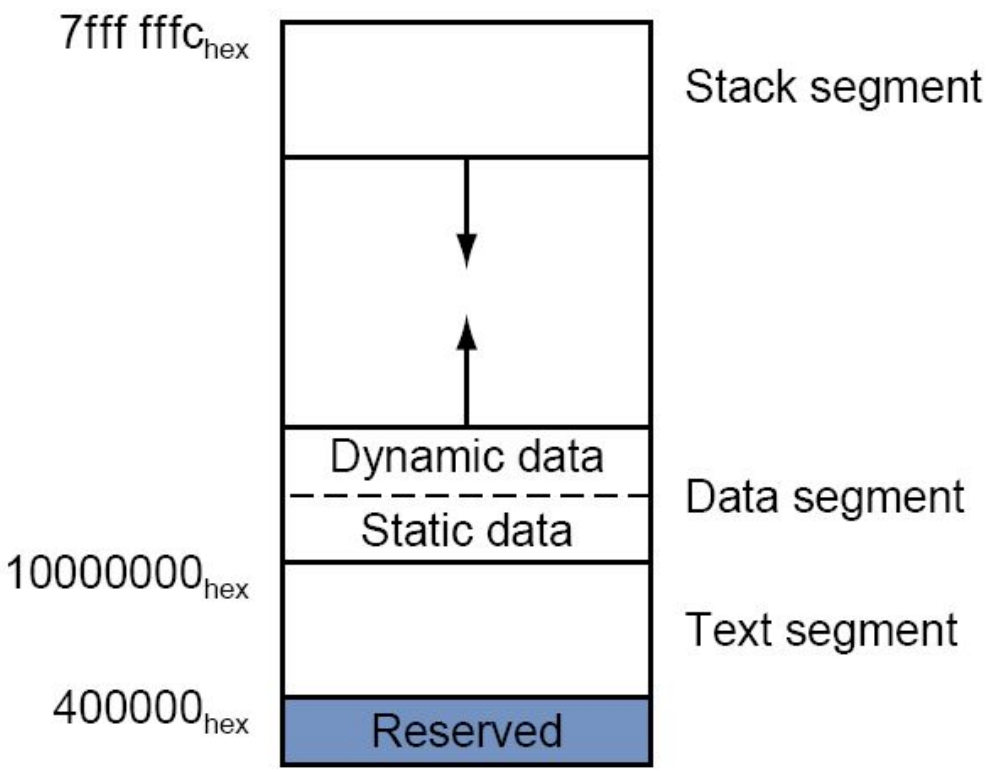
- Quite often, our function needs to use some registers to do dome calculation. So we will modify the values of them.
- We can use $t0-$t9 freely inside a function, because the caller does not expect the values inside $t0-$t9 to stay the same after the function call.
- But, the caller do expect the values in $s0 to $s7 to be the same after a function call.

MIPS Conventions

- So, just try to avoid using $s0 and $s7 inside a function whenever possible.
- But what if do need it? Such occasions will arise…

Stack

- So, if we do have to use $s0 - $s7, we MUST save it somewhere before entering the main part of the function, and restore it before we return (before we execute “jr $ra”).
- In MIPS, we save them in the **stack**.
- Stack is a part in the memory allocated for functions. It starts at 0x7ffffffc and grows **down** as we add more stuffs to it.
- Stack is “first in last out.”



`$sp`

- The top address of the stack, the address of the first word that is storing value, is (should be) always stored in `$sp`.
- So, adding a word into the stack (pushing a word onto the stack) is a two-step thing, because **you** have to maintain the correctness of $sp:
  - `addi $sp, $sp, -4`
  - `sw $s0, 0($sp)`

Suppose we want to

```
int weirdfun(int a, int b)
{
    int res;
    res = a + a + b - a / 2;
    return res;
}
```

```
int t = 0;
for (int i=0;i<10;i+=2)
{
    t += weirdfun(A[i],A[i+1]);
}
```

Stack and $sp

- Suppose we want to store a/2 in $s0.
  - How do we get a/2?
- At the beginning, we do
  - `addi $sp, $sp, -4`
  - `sw $s0, 0($sp)`
- At the end, we do
  - `lw $s0, 0($sp)`
  - `addi $sp, $sp, 4`

```
        .data
A:      .word 12, 34, 67, 1, 45, 90, 11, 33, 67, 19

        .text
        .globl main
main:
        la $s7, A
        li $s0, 0 #i
        li $s1, 0 #res
        li $s6, 9

loop:   sll $t0, $s0, 2
        add $t0, $t0, $s7
        lw $a0, 0($t0)
        lw $a1, 4($t0)
        jal weirdfun
        add $s1, $s1, $v0
        addi $s0, $s0, 2
        blt $s0, $s6, loop


done:   li $v0,10
        syscall


weirdfun:addi $sp, $sp, -4
        sw $s0, 0($sp)

        srl $s0, $a0, 1
        add $t0, $a0, $a0
        add $t0, $t0, $a1
        sub $t0, $t0, $s0

        ori $v0, $t0, 0

        lw $s0, 0($sp)
        addi $sp, $sp, 4

        jr $ra
```