

# CS230: Digital Logic Design and Computer Architecture

## Lecture-7: MIPS Instructions-II

<https://www.cse.iitb.ac.in/~biswa/courses/CS230/main.html>

# Logistics

---

Exam on 27<sup>th</sup> January at 11 AM

Content: Digital Logic only

---

Look at the seating

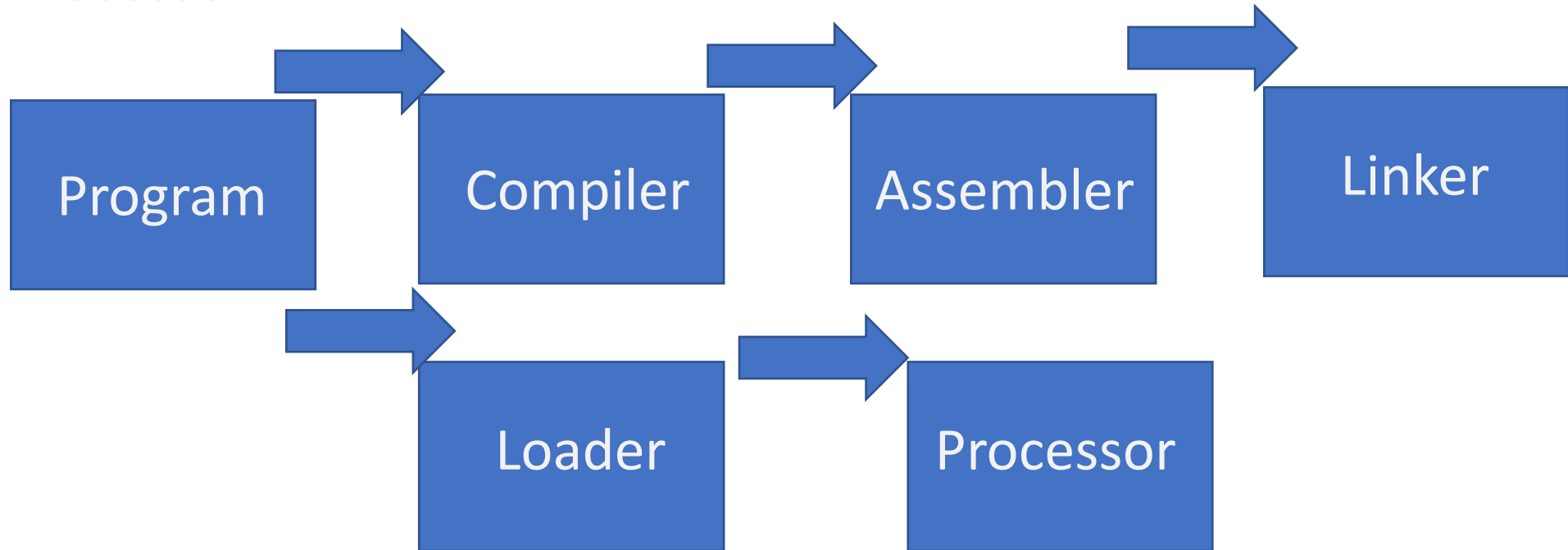
plan@Piazza, LA-001, CC-105

---

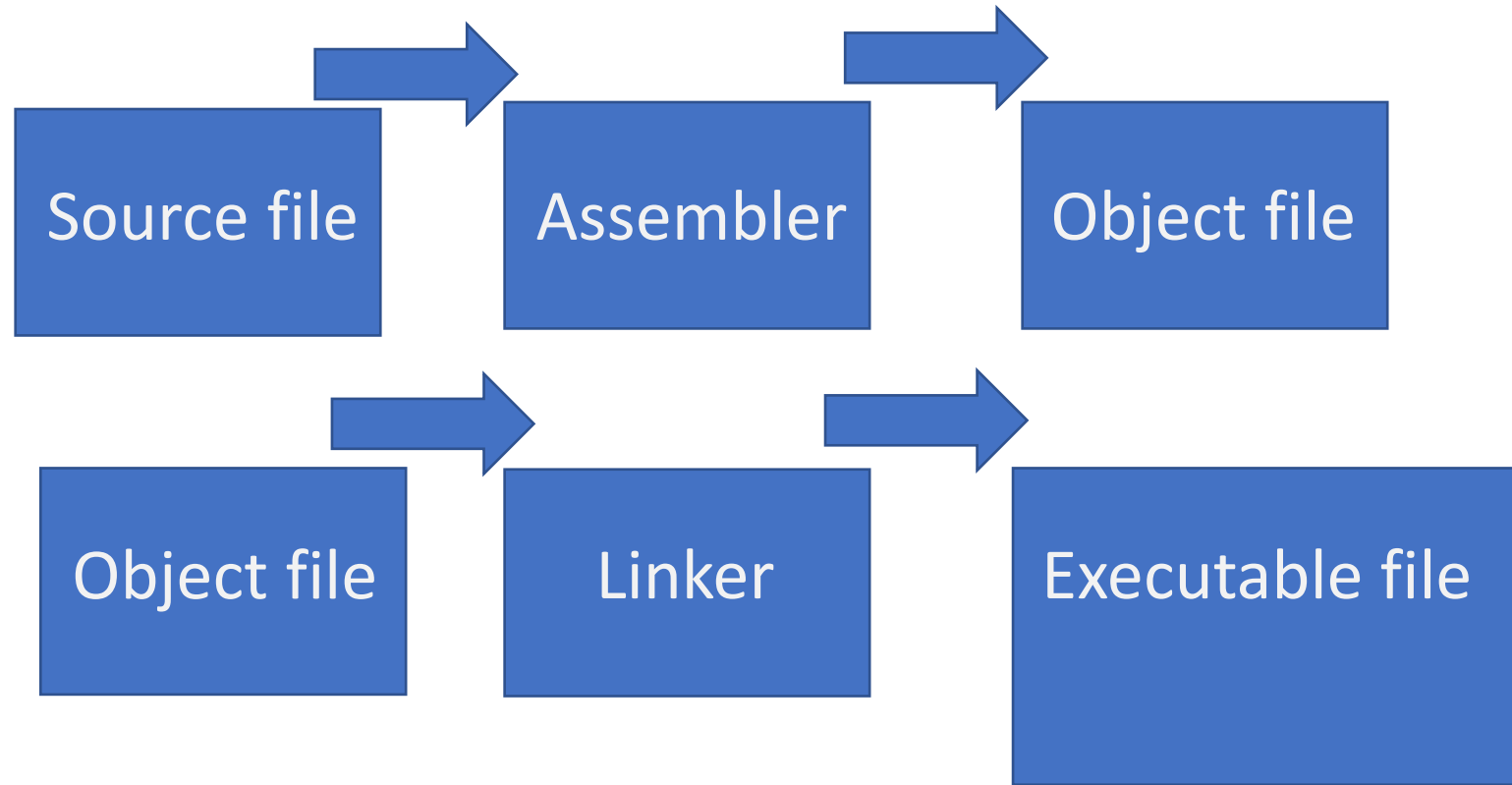
Lab-2, Please start doing it, last-minute plans may not work.

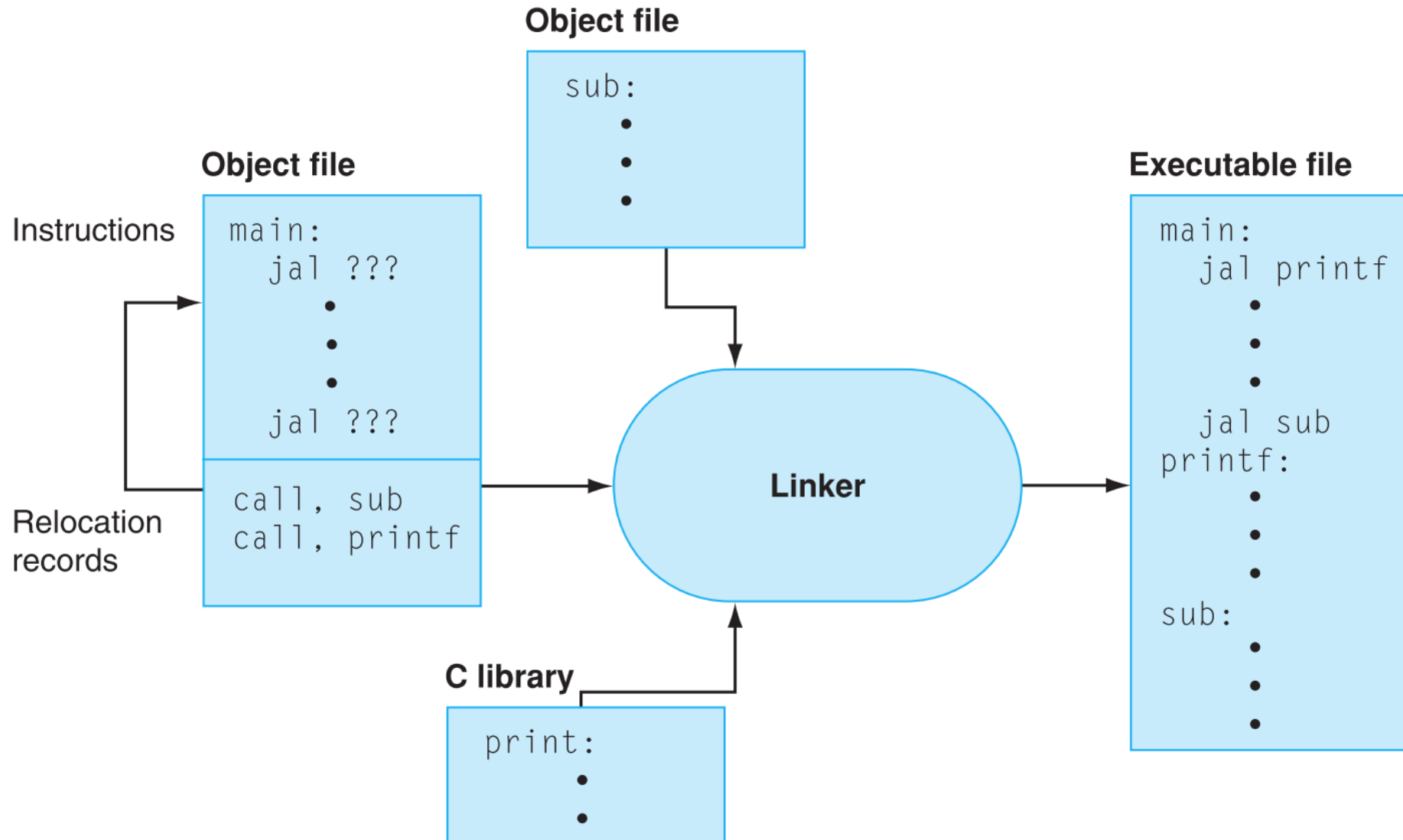
# The complete picture

Program -> Compiler -> Assembler -> Linker ->  
Loader -> Processor



# Source file, Object File and Executable File





# Sequential execution and jumps

PC, PC+4, PC+8, .....

PC, PC+4, {if condition here, TRUE} PC+32, .....

j instruction loads an immediate into the PC. It can be either specified as an offset or the label (assembler will convert this label into an offset). Next: jr, jal, ..

# Functions (Procedures)

```
int sum(int a, int b)
{
    int c=a+b;
    return c;
}
void main (void)
{
    int i=1;
    int j=2;
    int k = sum(i,j);
}
```

# Simple 😊

```
int sum(int a, int b)
{
    int c=a+b;
    return c;
}
void main (void)
{
    int i=1;
    int j=2;
    int k = sum (i,j);
}
```

**//jump to function**



# Simple 😊

```
int sum(int a, int b)
{
    int c=a+b;
    return c;
}
void main (void)
{
    int i=1;
    int j=2;
    int k = sum(i,j);
}
```

**j** sum

How do you return? 😞

# Awesome Instructions

- **jal**: Jump and Link    and    **jr** \$ra

jal L1:

go to L1, the instruction that has to be **executed next** is in L1.

and

**save the address** of the next instruction in \$ra. ra is an awesome register that stores the return address.

# Awesome Instructions

- **jal**: Jump and Link      and      **jr** \$ra

jal L1:

Go to instruction whose address is stored in ra (**PC+4**)

go to L1, the instruction that has to be **executed next** is in L1.

and

**save the address** of the next instruction in \$ra. ra is an awesome register that stores the return address (ra).

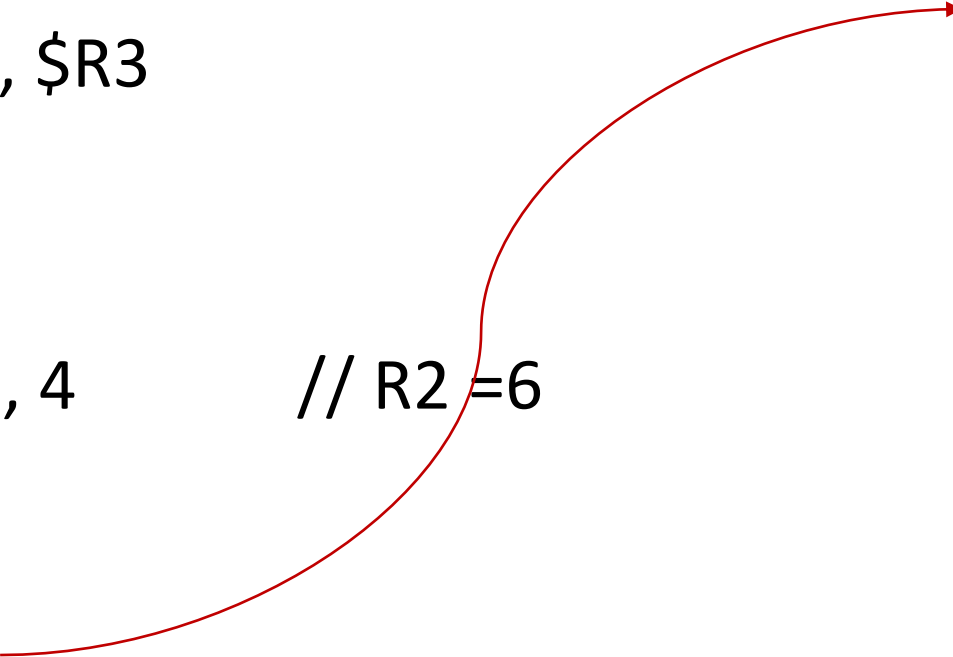
# Let's Have a Complete Picture

```
PC+4    addi $R1, $R0, 2        // R0 = 0, R1=2
PC+8    jal sum                // R31 (ra) = PC+12
PC+12   add $R0, $R3, $R3

sum:
PC+100  addi $R2, $R1, 4
PC+104  jr
```

# Let's Have a Complete Picture

PC+4	addi \$R1, \$R0, 2	// R0 = 0, R1=2
PC+8	jal sum	// R31 = PC+12 (ra)
PC+12	add \$R0, \$R3, \$R3	
sum:		
PC+100	addi \$R2, \$R1, 4	// R2 = 6
PC+104	jr \$R31	



# Let's Have a Complete Picture

PC+4      `addi $R1, $R0, 2`      // R0 = R3 = 0, R1=2

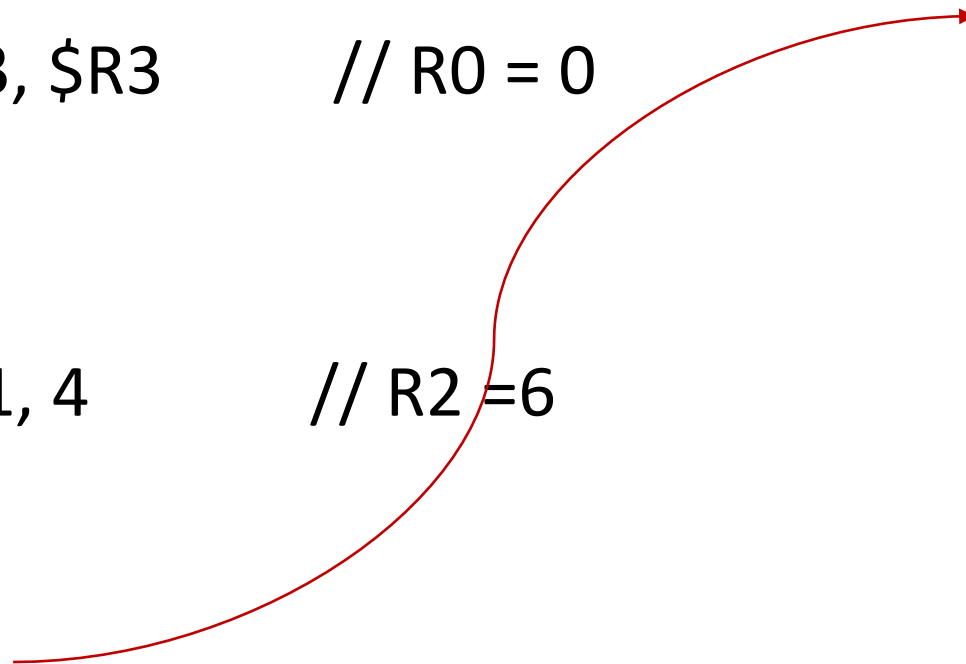
PC+8      `jal sum`      // R31 = PC+12 (ra)

PC+12     `add $R0, $R3, $R3`      // R0 = 0

sum:

PC+100   `addi $R2, $R1, 4`      // R2 = 6

PC+104   `jr $R31`



# Let's Have a Complete Picture

PC+4      `addi $R1, $R0, 2`      // R0 = R3 = 0, R1=2

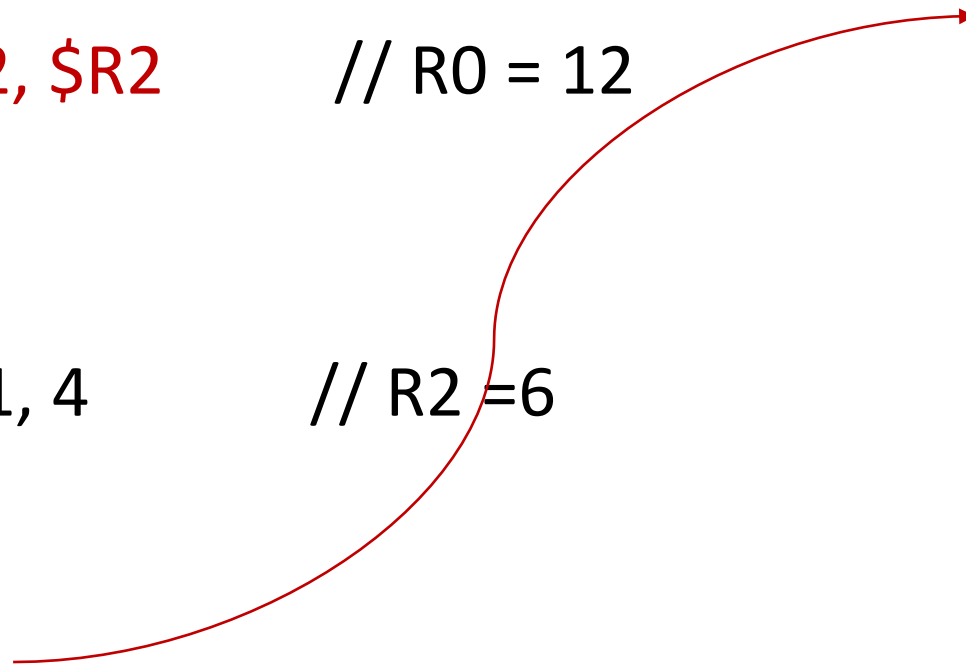
PC+8      `jal sum`      // R31 = PC+12 (ra)

PC+12      `add $R0, $R2, $R2`      // R0 = 12

sum:

PC+100      `addi $R2, $R1, 4`      // R2 = 6

PC+104      `jr $R31`



# JAL: Jump and Link, What's wrong?

PC+4      `addi $R1, $R0, 2`

PC+8      `jal sum`                      *// R31 = PC+12 (ra)*

PC+12     `add $R0, $R2, $R2`



# Well

As per MIPS specification, Check P&H MIPS sheet 😞

PC: jal label

ra = PC + 8

PC+4:

PC+8:

Jump	j	J	PC=JumpAddr
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr
Jump Register	jr	R	PC=R[rs]



PC+4 or PC+8? Why this

PC+4 at the  
moment

PC+8 after a  
month or so



# Quick recap

Usage of `j`, `jr`, `jal`, and `$ra`

# MIPS provides

Upto **four** arguments can be passed from the caller to the callee while using **jal**. It uses registers \$a0 to \$a3

A callee can return upto two values to the caller. It uses registers \$v0 and \$v1

# What if?

```
main(){  
  a = a + f1(a);  
}
```

```
f1(a) {  
    a = a - f2(a); return a;}  
f2(a) {  
    a = a + f3(a); return a;}  
f3(a) {  
    a = a + 1;    return a;}  
}
```

f1:

f2's argument in \$a0 to \$a3  
jal f2

# What if?

f1:

f2's argument in \$a0 to \$a3

jal f2

...

f2:

f3's argument in \$a0 to \$a3

jal f3

...

# What is the big deal?

f1:

f2's argument in \$a0 to \$a3

jal f2

...

f2:

f3's argument in \$a0 to \$a3

jal f3

...

# What is the big deal? Oh no!

f1:

PC: f2's argument in \$a0 to \$a3

PC+4: jal f2                   // \$ra = PC+8

...

f2:

PC+100: f3's argument in \$a0 to \$a3

PC+104: jal f3               // \$ra = PC+108

...

f3: ...

jr \$ra



# What is the big deal? Oh no!

f1:

PC: f2's argument in \$a0 to \$a3

PC+4: jal f2 // \$ra = PC+8

...

f2:

PC+100: f3's argument in \$a0 to \$a3

PC+104: jal f3 // \$ra = PC+108

jr \$ra 😞 Oh no!!

...

f3: ...

jr \$ra

# Saving and Restoring Registers (limited)

**caller** registers

**callee** registers

Why?

**Callee** does not know, registers used by callers, can be many callers too

**Caller** does not know the callee's plan 😊

# MIPS 32 registers

Registers	Total Regs
• \$Zero	1
• (Return) Value registers (\$v0,\$v1)	3
• Argument registers (\$a0-\$a3)	7
• Return Address (\$ra)	8
• Saved registers (\$s0-\$s7)	16
• Temporary registers (\$t0-\$t9)	26
• Global Pointer (\$gp)	27
• Stack Pointer (\$sp)	28
• Frame Pointer (\$fp), or \$t10	29
• 2 for OS (\$k0, \$k1), 1 for assembler (\$at)	

# Before that: Who does what?

In MIPS,

\$t0 to \$t9 (R8 to R15, R24, and R25) are temporary and *caller* saved registers. Register values not preserved across function calls (call-clobbered).

\$s0 to \$s7 (R16 to R23) are *callee* saved registers. Register values are preserved across function calls (call-preserved).

**\$ra is caller or callee saved register ?**

# Before that: Who does what?

In MIPS,

\$t0 to \$t9 (R8 to R15, R24, and R25) are temporary and *caller* saved registers. Register values not preserved across function calls (call-clobbered).

\$s0 to \$s7 (R16 to R23) are *callee* saved registers. Register values are preserved across function calls (call-preserved).

*\$ra* is *callee* saved register.



# Coffee Credits

---

Yashwant, 210050171, +2

A white ceramic mug with a red heart design and a red rim, containing a light brown beverage, sits on a white surface. The scene is lit from the left, casting a soft shadow. The background is a textured, light-colored fabric.

buona giornata