"Procedure Calling"

* Let's assume procedure like a spy.

Execution process of spy

(i) leaves with a secret plan.

(ii) acquires resources.

(iii) performs the task.

(iv) covers traces.

(v) returns to the point of origin with desired answers.

Execution of a procedure

Step 1: Put parameters in a place where the procedure can access them. → ($X_{10} - X_{17}$)

Step 2: Transfer control to the procedure.

Step 3: Acquire storage resources needed for the proc.

Step 4: Perform the desired task.

Step 5: Put the result value in a place where the calling program can access it.

Step 6: Return control to the point of origin. → X_1

[A proc. can be called from several points in a program]

JAL => Jump and Link instruction

Syntax: JAL X_1, procedureLabel

(X_1: Fixed; procedureLabel: Jump Destination)

Explination: Jump to procedureLabel and write return Address to X1.

JALR => Jump and Link Register

Syntax: JALR X_0, $0[X_1]$

($0[X_1]$: Jump Destination)

* The calling program (caller) puts the parameter values in $X_{10} - X_{17}$.
* Uses Jal X1, abc to branch to procedure label abc (callee)
* Callee performs the calculations, places the results in the same parameter registers
* Returns control to the caller using jalr X_0, $0(X_1)$

PC (Program Counter) ⇒ is the register that holds the address of the current instruction being executed.

Jal X_1, procedureLabel
↑
(PC+4) is stored.

Jal X_0, Label } unconditional branch within a procedure.
↑
0 is hardwired;
Discard the return Address

What if compiler needs more registers for a procedure than the 8 arg. registers?

⇒ If the callee require any register that is already in use by the caller, callee must restore the values that were contained before the procedure was invoked.

↳ First store the values in stack

You want to store 3 register datas.

⇒ each register size = 64 bits
= 8 locations

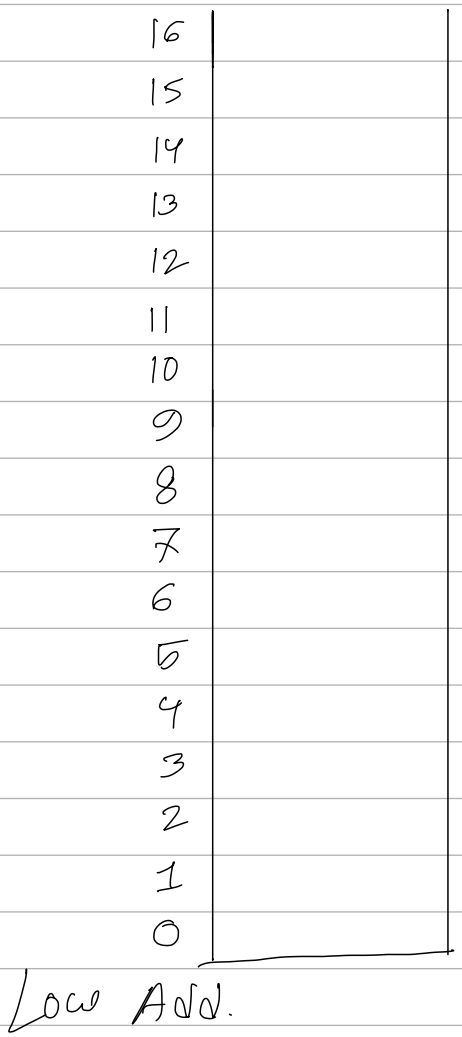
High Add.

SP ⇒ 24
23
22
21
20
19
18
17

Stack

⇒ to store data of 3 registers
= (3 * 8) = 24 locations are needed

(LIFO)



(i) Decrease the SP by the #locations you need

(ii) Then store the values in stack

```
def leaf_example (g, h, i, j):
    f = (g+h) - (i+j)
    return f
```

	$g = x_{10}$
leaf_procedure:	$h = x_{11}$
addi sp, sp, -24	$i = x_{12}$
sd x5, 16(sp) } Stored into	$j = x_{13}$
sd x6, 8(sp) } stack	f = x20 } Save these
sd x20, 0(sp) }	temp = x5, x6 } 3 in the stack first.
Add x5, x10, x11	
Add x6, x12, x13	
Sub x20, x5, x6	
Addi x10, x20, 0 } Store the return value	

```
ld   x20,  0(sp)   ┐
ld   x6,   8(sp)   ├ Stored into stack
ld   x5,   16(sp)  ┘
addi sp, sp, 24
jalr x0, 0[x1]
```

To avoid saving and restoring a register whose value is never used,

$x_5 - x_7$ and $x_{28} - x_{31}$ => temp. register values are not preserved by the callee.

$x_8 - x_9$ and $x_{18} - x_{27}$ => saved register values must be preserved by the callee.

Hence, from the above example we do not need to restore the values of x_5, x_6