RISC V Architecture

Mahesh Awati
Department of Electronics and Communication Engg.

RISC V ARCHITECTURE

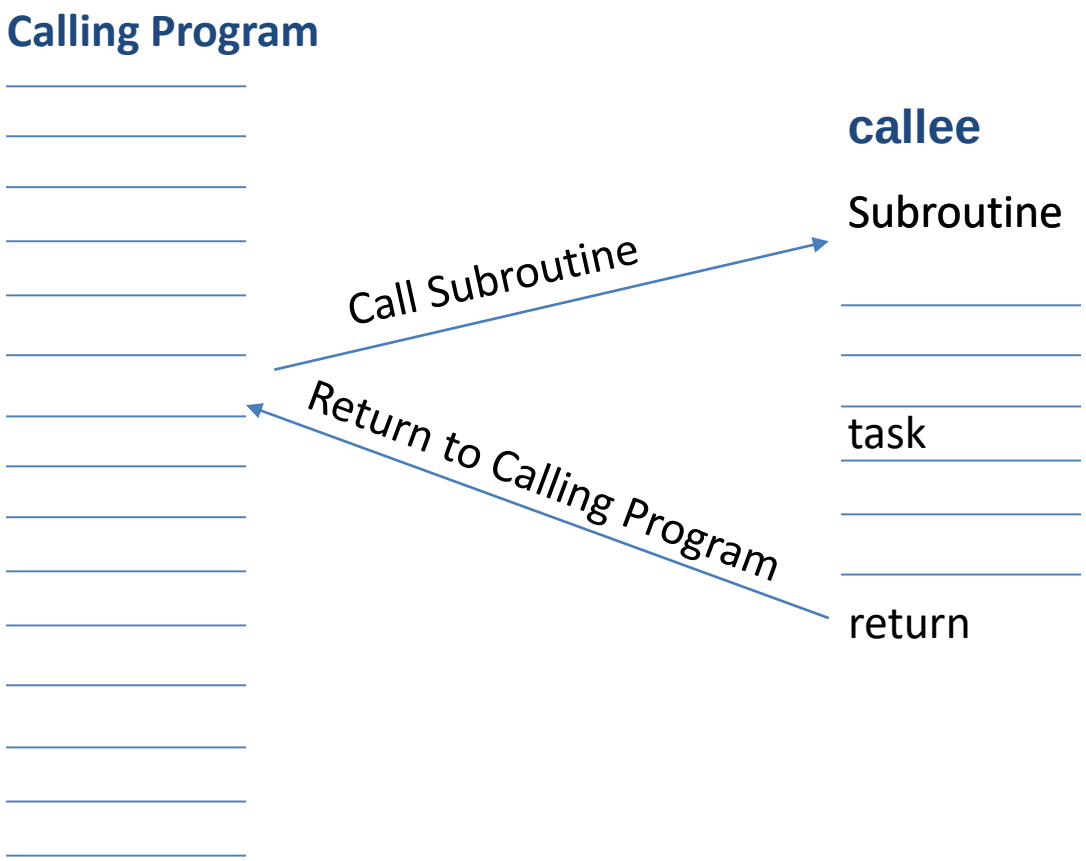
UNIT 2 – Instructions: The Language of Computer

Mahesh Awati
Department of Electronics and Communication Engineering

Instructions – Language of Computer

Supporting Procedures in Computer Hardware

- **Procedure:** A stored subroutine that performs a specific task based on the parameters with which it is provided.



- Parameters act as an interface between the procedure and the rest of the program and data, since they can pass values and return results.
- A procedure basically
  - ✓ Acquires resources,
  - ✓ performs the task,
  - ✓ covers his or her tracks, and
  - ✓ then returns to the point of origin with the desired result.

Instructions – Language of Computer

Supporting Procedures in Computer Hardware

In the execution of a procedure, the program must follow these six steps:

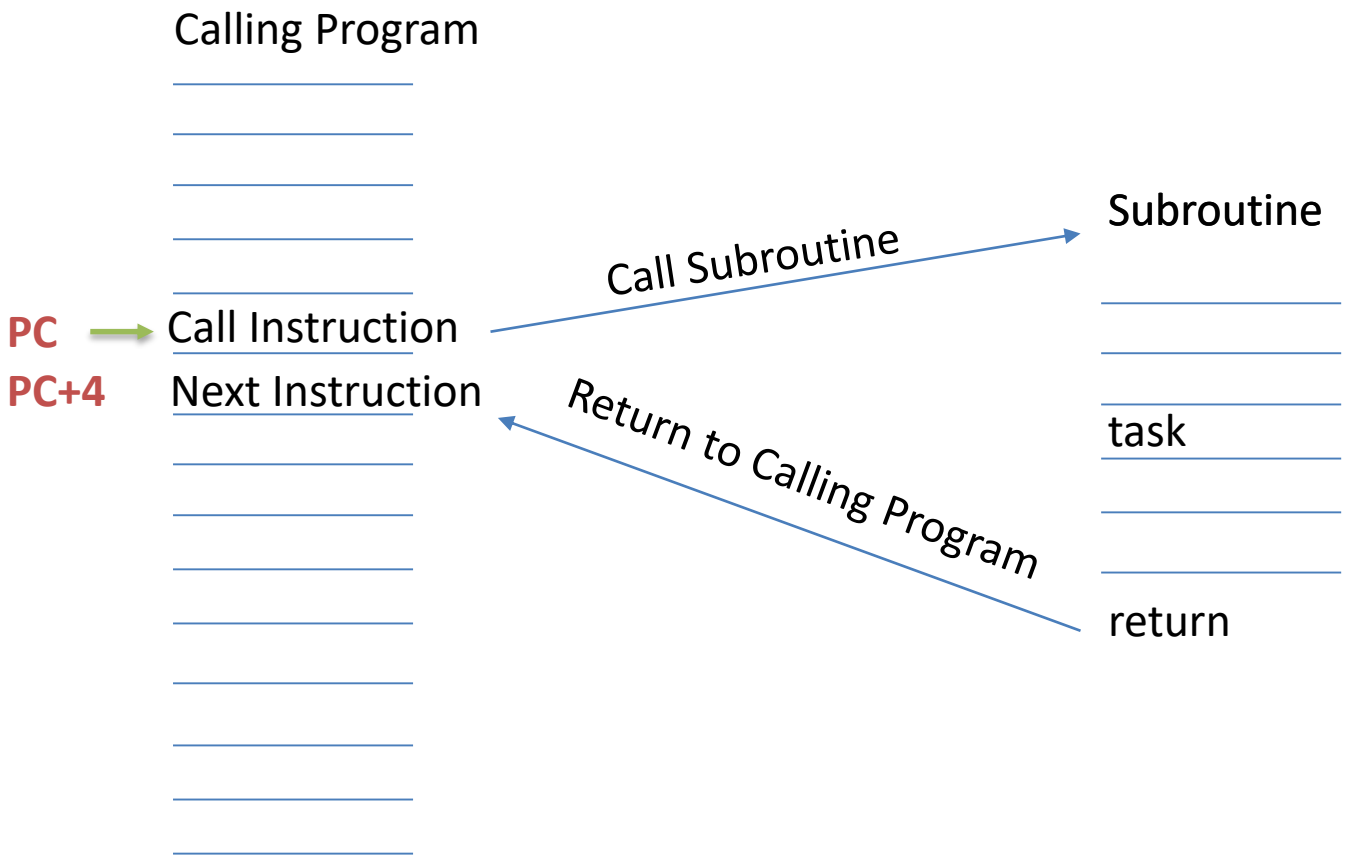
1. Put parameters in a place where the procedure can access them.
2. Transfer control to the procedure.
3. Acquire the storage resources needed for the procedure.
4. Perform the desired task.
5. Put the result value in a place where the calling program can access it.
6. Return control to the point of origin, since a procedure can be called from several points in a program.

Instructions – Language of Computer

Supporting Procedures in Computer Hardware

RISC-V features for handling Procedures

Case a) Leaf Procedure: Procedures that do not call others.



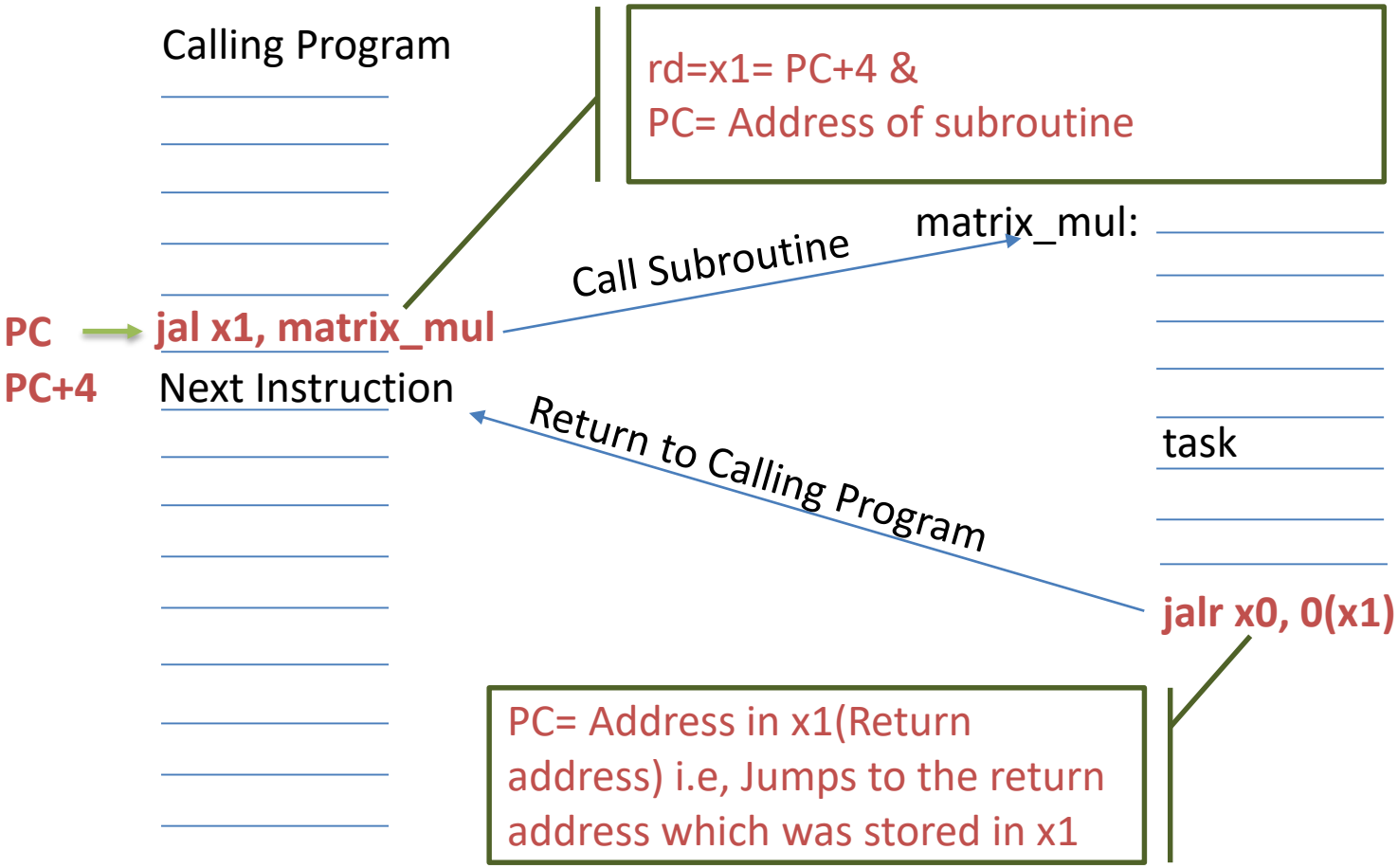
Requirement to handle the procedure

1. Passing & returning parameters:
   x10–x17: eight parameter registers
2. Instruction to call procedure:
   jal x1, ProcedureAddress
3. Making Note of the return address:
   x1:one return address register to return to the point of origin.
4. Instruction to return from subroutine - which should make sure that calling program should start from the point where it was stopped:
   jalr x0, 0(x1)

Instructions – Language of Computer

Supporting Procedures in Computer Hardware

RISC-V features for handling Procedures



Instructions – Language of Computer

Supporting Procedures in Computer Hardware

Using More Registers

Since we must cover our tracks after our mission(procedure is completed) is complete.

What is that we need to do with content of these the register needed by the Caller ????

Any registers needed by the caller must be restored to the values that they contained *before* the procedure was invoked. This situation is an example in which we need to spill registers to memory

In RISC-V , X10-X17 are used as argument registers.

What if a compiler needs more registers for a procedure than the eight argument Registers ?????

Need to use other registers in addition to X10-X17. Again, any registers needed by the caller must be restored to the values that they contained *before* the procedure was invoked. This situation is an example in which we need to spill registers to memory

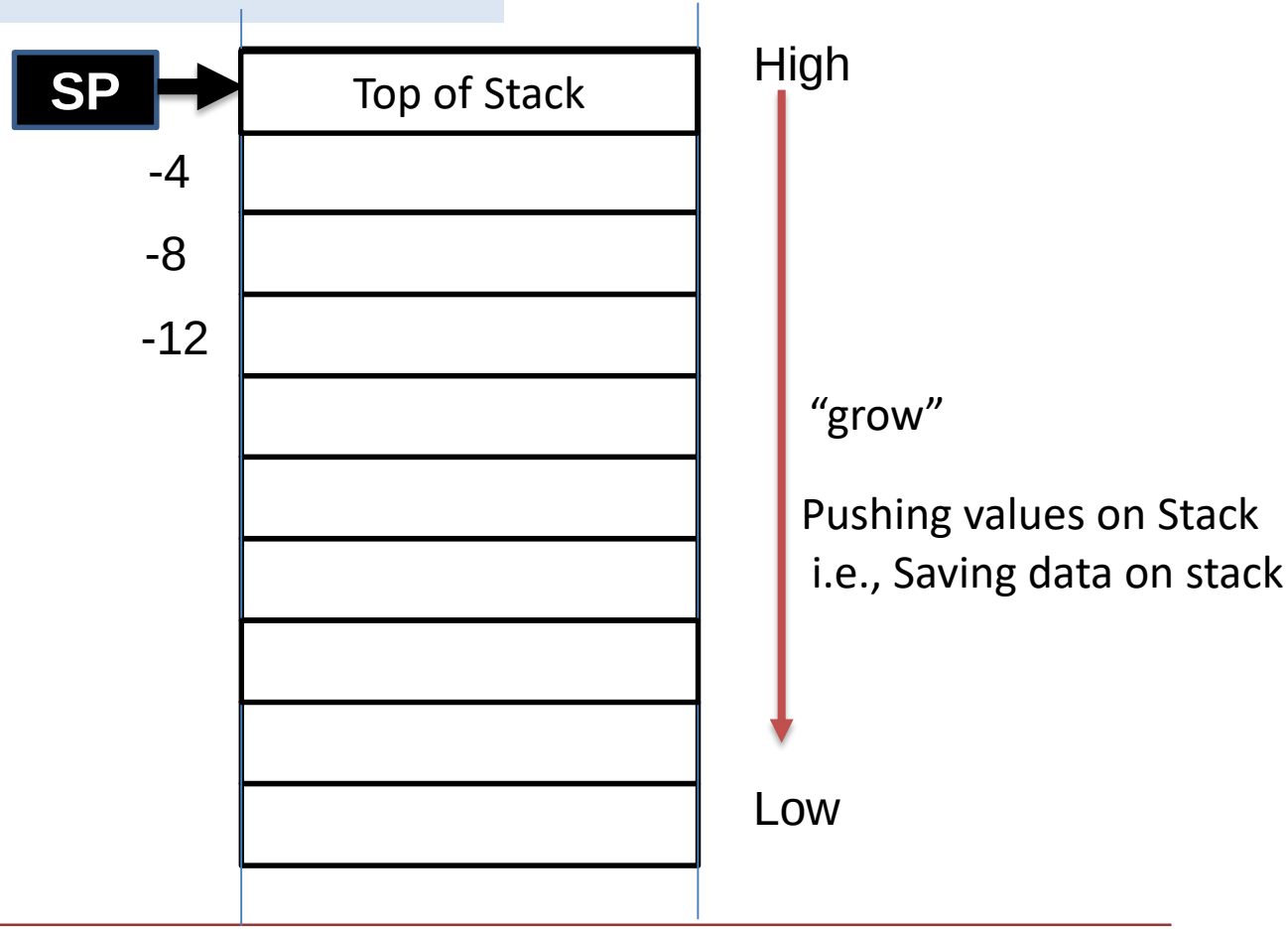
Instructions – Language of Computer

Supporting Procedures in Computer Hardware

Using More Registers

Stack—a last-in-first-out queue

- A data structure for spilling registers organized as a last-in-first-out queue.
- **Stack Pointer (SP) –** Holds the most recently allocated address in a stack. Basically SP shows where registers should be spilled or where old register values can be found.
- In RISC-V, it is register x2 which plays role of SP.
- The SP is adjusted by one word for each register that is saved (Push- Saving data on stack) or restored (Pop- Removing data on stack) .
- Stacks "grow" from higher addresses to lower addresses. This convention means that you push values onto the stack by decrementing SP by 4.
- Adding to the stack pointer shrinks the stack, thereby popping values off the stack.



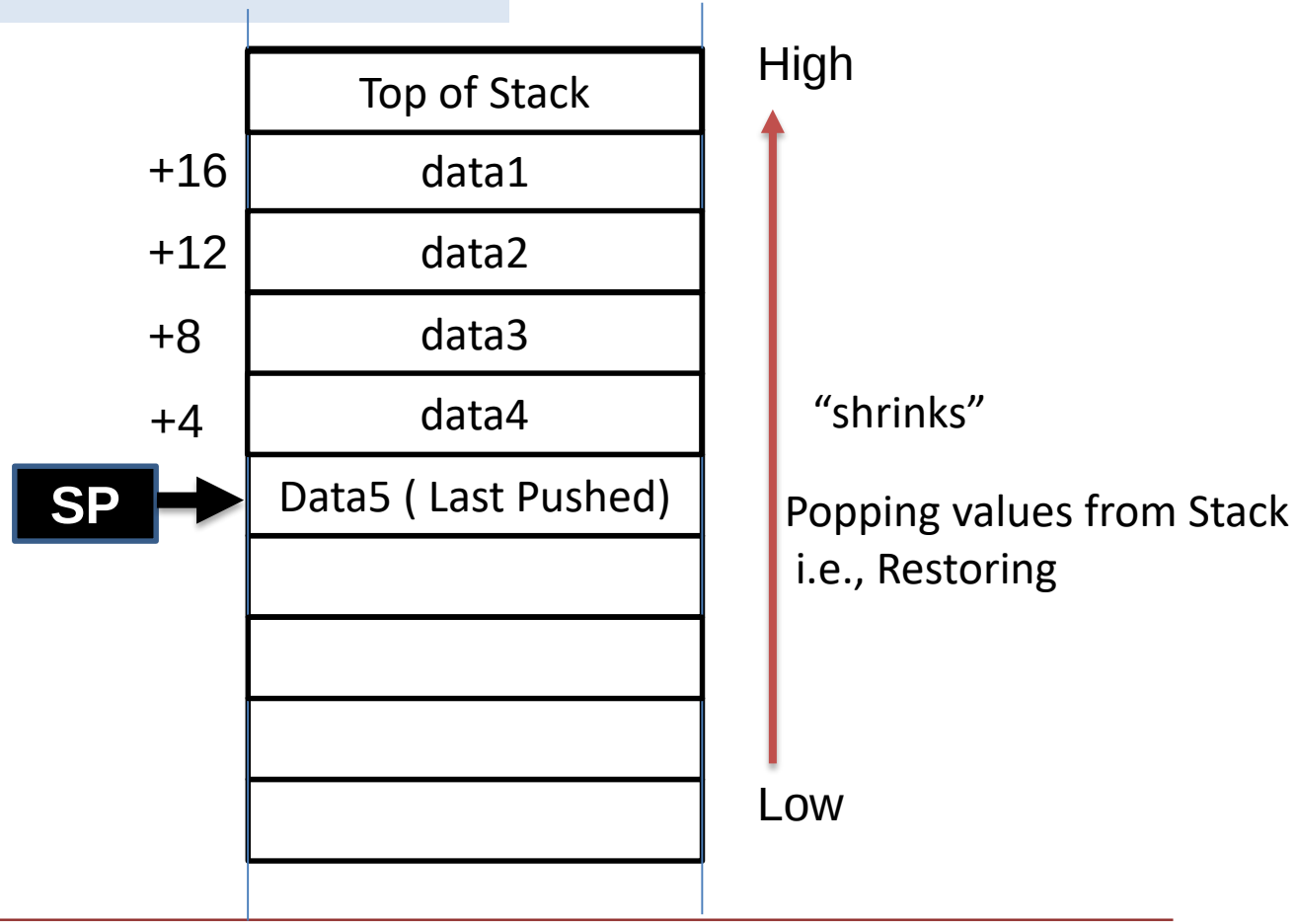
Instructions – Language of Computer

Supporting Procedures in Computer Hardware

Using More Registers

Stack—a last-in-first-out queue

- Stacks "grow" from higher addresses to lower addresses. This convention means that you push values onto the stack by decrementing SP by 4.
- Adding to the stack pointer shrinks the stack, thereby popping values off the stack.



Instructions – Language of Computer

Supporting Procedures in Computer Hardware

Compiling a C Procedure that Doesn't Call Another Procedure

The C procedure:

```
int leaf_example (int g, int h, int i, int j)
{
int f;
f = (g + h) − (i + j);
return f;
}
```

What is the compiled RISC-V assembly code?

The parameter variables g, h, i, and j correspond to the argument registers x10, x11, x12, and x13, respectively, and f corresponds to x20.

Compiled RISC-V assembly

Assembly procedure should involve

1. Save the registers used by the procedure.
2. Body of the procedure to perform the task.
3. The value of f, should be in parameter register
4. Restore the old values of the registers we saved by "popping" them from the stack.
5. Return from procedure

Note: In RISC-V don't have exclusive Instructions for PUSH data and POP data from stack

leaf_example:

```
addi sp, sp, -12
sw x5, 8(sp)
sw x6, 4(sp)
sw x20, 0(sp)

add x5, x10, x11 //x5 =g + h
add x6, x12, x13 // x6= i + j
sub x20, x5, x6

addi x10, x20, 0

lw x20, 0(sp)
lw x6, 4(sp)
lw x5, 8(sp)
addi sp, sp, 12
jalr x0, 0(x1)  // Return
```

Instructions – Language of Computer

Supporting Procedures in Computer Hardware

The values of Stack and Stack Pointer

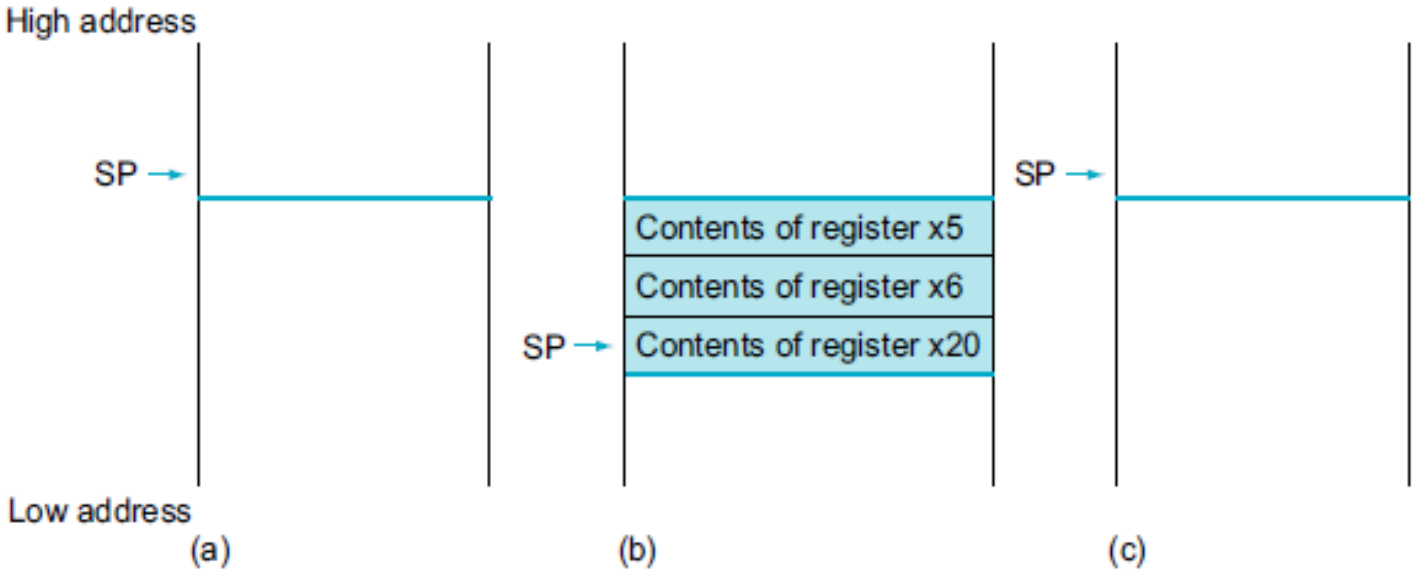


FIGURE 2.10 The values of the stack pointer and the stack (a) before, (b) during, and (c) after the procedure call. The stack pointer always points to the "top" of the stack, or the last word in the stack in this drawing.

Instructions – Language of Computer

Supporting Procedures in Computer Hardware

To avoid saving and restoring a register whose value is never used, which might happen with a temporary register, RISC-V software separates 19 of the registers into two groups:

1. x5–x7 and x28–x31: temporary registers that are *not* preserved by the callee (called procedure) on a procedure call
2. x8–x9 and x18–x27: saved registers that must be preserved on a procedure call (if used, the callee saves and restores them)

Name	Alias
x0	zero
x1	ra
x2	sp
x3	gp
x4	tp
x5	t0
x6	t1
x7	t2
x8	s0
x9	s1
x10	a0
x11	a1
x12	a2
x13	a3
x14	a4
x15	a5

Name	Alias
x16	a6
x17	a7
x18	s2
x19	s3
x20	s4
x21	s5
x22	s6
x23	s7
x24	s8
x25	s9
x26	s10
x27	s11
x28	t3
x29	t4
x30	t5
x31	t6

THANK YOU

Mahesh Awati
Department of Electronics and Communication

mahesha@pes.edu

+91 9741172822