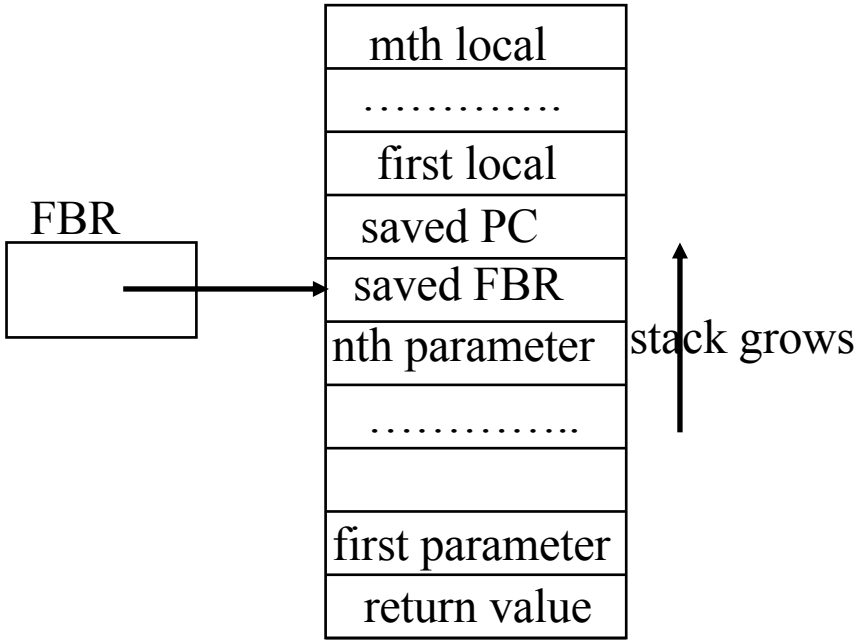
Method Invocation and Return

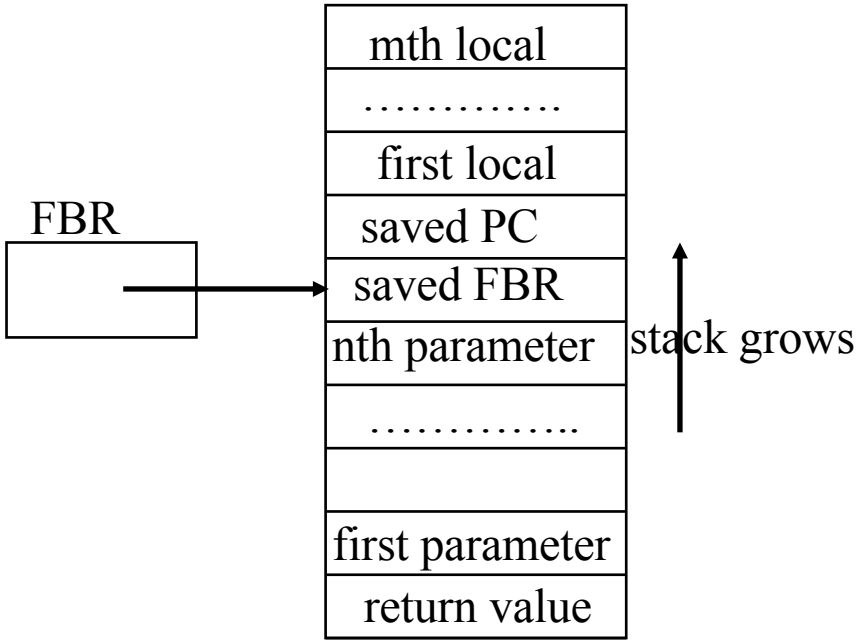
- Method invocation and return requires maintaining the values of the control registers (SP, FBR, PC) in a LIFO fashion.
  - The natural implementation is to make this data part of the SaM stack.
  - We do this by imposing structure on the SaM stack and viewing it as a collection of *stack frames*.

- It is also convenient to custom-craft some commands to make method invocation/return easier to implement.

Stack Frames in SaM

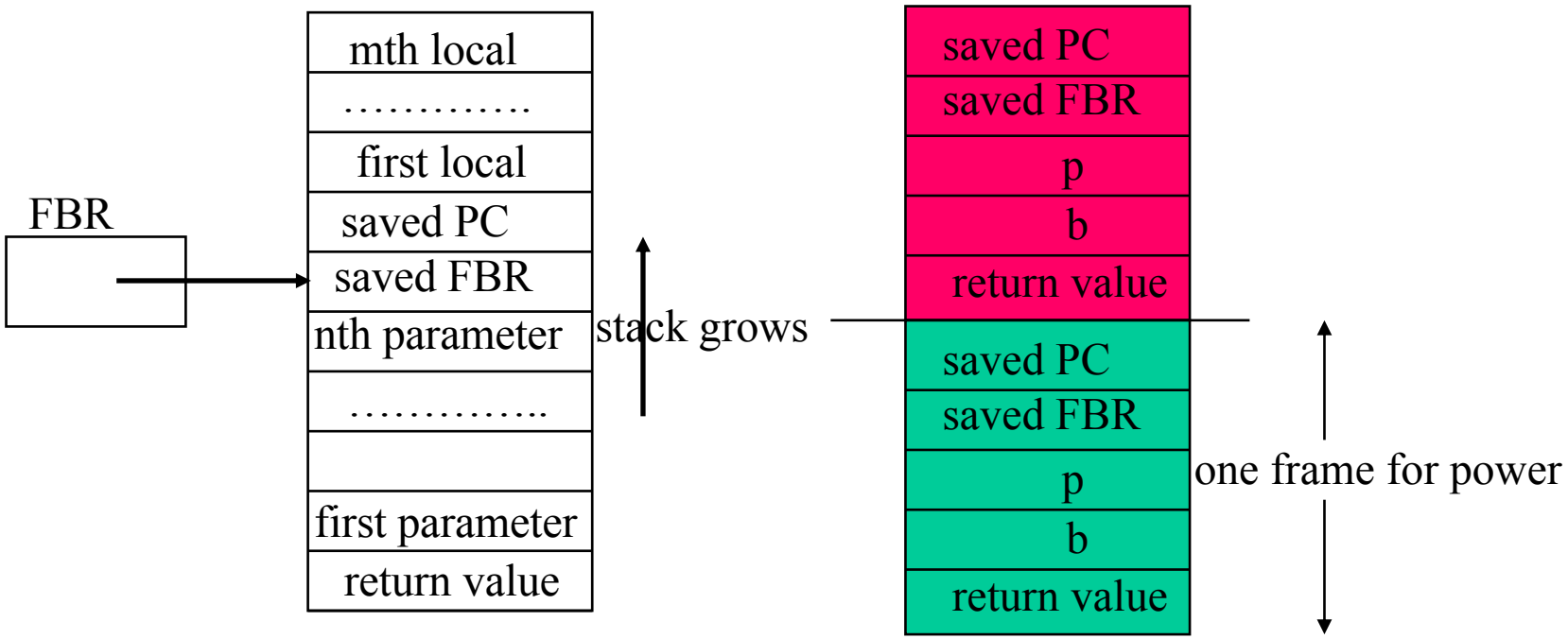


Stack Frames in SaM




```
int power(int b, unsigned p) {
    if (p == 0) return 1;
    else return b * power(b, p-1);
}
```

Stack Frames in SaM




```
int power(int b, unsigned p) {
    if (p == 0) return 1;
    else return b * power(b, p-1);
}
```

SP ↔ Stack Commands

- `PUSHSP`
  - Push value of SP to stack.
  - Stack[SP] ← SP; SP ← SP + 1.

- `POPSP`
  - Inverse of `PUSHSP`.
  - SP ← SP - 1; SP ← Stack[SP].

- `ADDSP` *n*
  - Convenient for method invocation.
  - SP ← SP + *n*.
  - Shorthand for the sequence [`PUSHSP`, `PUSHIMM` *n*, `ADD`, `POPSP`].

FBR ↔ Stack Commands

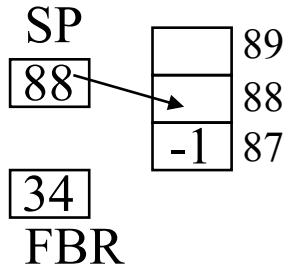
- `PUSHFBR`
  - Push value of FBR to stack.
  - Stack[SP] ← FBR; SP ← SP + 1.

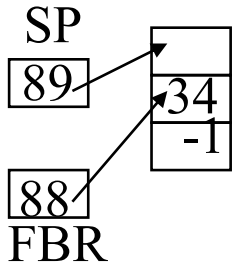
- `POPFBR`
  - Inverse of `PUSHFBR`.
  - SP ← SP - 1; FBR ← Stack[SP].

FBR ↔ Stack Commands

- `PUSHFBR`
  - Push value of FBR to stack.
  - Stack[SP] ← FBR; SP ← SP + 1.

- `POPFBR`
  - Inverse of `PUSHFBR`.
  - SP ← SP - 1; FBR ← Stack[SP].





- `LINK`
  - Convenient for method invocation.
  - Similar to PUSHFBR but also updates FBR so that it points to the location where FBR was saved.
  - Stack[SP] ← FBR; FBR ← SP; SP ← SP + 1.

PC ↔ Stack Commands

- A little different, since we don’t need exactly the PC value of the call site.

PC ↔ Stack Commands

- A little different, since we don't need exactly the PC value of the call site.
- `JSR` *`label`*
  - Save the value of PC + 1 on stack, and jump to *`label`*.
  - Stack[SP] ← PC + 1; SP ← SP + 1; PC ← *`label`*.

PC ↔ Stack Commands

- A little different, since we don't need exactly the PC value of the call site.
- `JSR` *`label`*
  - Save the value of PC + 1 on stack, and jump to *`label`*.
  - Stack[SP] ← PC + 1; SP ← SP + 1; PC ← *`label`*.

- `JUMPIND`
  - Use for return from method call.
  - SP ← SP - 1; PC ← Stack[SP].

PC ↔ Stack Commands

- A little different, since we don't need exactly the PC value of the call site.
- `JSR label`
  - Save the value of PC + 1 on stack, and jump to *`label`*.
  - Stack[SP] ← PC + 1; SP ← SP + 1; PC ← *`label`*.
- `JUMPIND`
  - Use for return from method call.
  - SP ← SP - 1; PC ← Stack[SP].
- `JSRIND`
  - Like JSR, but the address of the method is on the stack.
  - temp ← Stack[SP]; Stack[SP] ← PC + 1; PC ← temp.