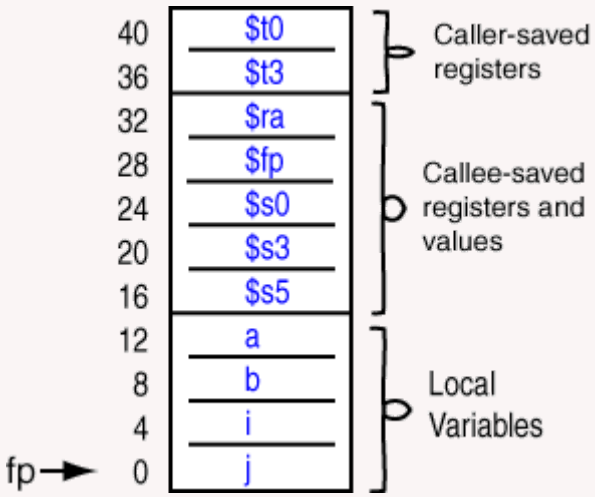
Answer:

Conceptually, the local variables cease to exist when the stack frame that implements them is popped.

Frame Pointer

Register `$30` is reserved, by software convention, for use as a **frame pointer**. In the extended assembler it has the mnemonic name `$fp`. When a subroutine starts running, the frame pointer and the stack pointer contain the same address.



While the subroutine is active, the frame pointer, points at the top of the stack. (Remember, our stacks grow downward, so in the picture `$fp` is correctly pointing at the last word that was pushed onto the stack, the top of the stack.)

But the stack (and the stack pointer) may be involved in arithmetic expression evaluation. This often involves pushing and popping values onto and off of the stack. If `$sp` keeps changing, it would be hard to access a variable at a fixed location on the stack.

To make things easy for compilers (and for human assembly language programmers) it is convenient to have a frame pointer that does not change

its value while a subroutine is active. The variables will always be the same distance from the unchanging frame pointer.

In the subroutine prolog, the caller's frame pointer is pushed onto the stack along with the stack pointer and any `$s` registers. Now the subroutine makes room on the stack for variables and points the frame pointer to the top of the stack frame.

QUESTION 4:

Write the instruction that loads register `$t5` with the value held in the variable **a**:

```
lw $t5,      (      )
```

(Hint: use **$fp** as a base register.)