Is R5 – R6 a Constant Inside a Function?

One common question:
- **why use both R5 and R6?***
- (Aren't R5 and R6 always the same distance apart?)

One answer:
- code adds/removes values from the stack
- (so, no, the **difference is not constant**).

*Note that the x86 (IA-32) ISA calling convention also uses two registers.

Compiler Does Know R5 – R6 Most of the Time

What kinds of things are pushed?
- callee-saved registers
- arguments to subroutines
- spilled values (when compiler runs out of registers for performing calculations)
- certain types of temporary allocation (not covered in our class—see `alloca`).

But—except for the last case—**the compiler KNOWS when R6 moves**, so it could still generate the right code…

Compiler Often Does Not Provide Such Information

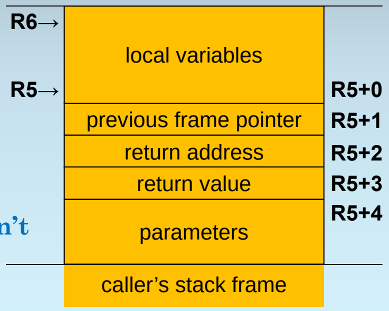
However, information about **R6's movement is often not passed to a debugger**.

So …
- you can turn on high levels of optimization
- and compilers (x86 compilers, for example) will reclaim the frame pointer,
- but good luck trying to debug (debugger will not be able to identify stack frames).

What is the Order of Local Variables?

What about the order of local variables?

Used only within the function, so choice doesn't matter.


R6→
local variables
R5→
R5+0
previous frame pointer
R5+1
return address
R5+2
return value
R5+3
R5+4
parameters
caller's stack frame