

## Assumptions for code-generation problems

- You may show *cgen* code as we did in the examples in class, i.e., without `print` or `emit` statements. However, your *cgen* code must show any calls to the name generator (described below) and for any run-time constants in the generated code, include comments explaining how they would be computed at compile-time.
- We're assuming that we're generating code for a **stack machine** with the following characteristics and conventions:
  - There is a frame pointer `$fp` (points to the current stack frame)
  - There is a stack pointer (`$sp`) which points one word beyond the top element in the stack
  - The stack grows from large addresses to small addresses
  - memory is byte-addressable
  - temporaries are pushed on the stack, not put in the stack frame
  - the results of expressions are always left in `$a0`
  - after evaluating an expression the stack pointer, `$sp`, has the same value it had before evaluating the expression
- You may assume a global variable `lgen` of type `NameGenerator` is available for generating unique **labels**. The call `lgen.gen()` generates the next unused label (returns type `string`).
- You may use the following **pseudo-MIPS** syntax for the assembly language to generate, or if you know it, you may use real MIPS instructions instead:

```
x := y           For loading, storing, and moving between
                  registers.

                  for this operation, "x" or "y" (but not both) may be a
                  memory location using offset(reg). Either
                  one may be a register. "y" may be a
                  constant or a label representing an address
                  in memory.

x := y op z      Where op is one of +, -, *, /, %, "and", "or"
x := op y        Where op is one of -, "not"
goto L           jump to label L
if y relop z goto L      Conditional jump to label L, where
                        relop is one of <, >, =, !=, >=, <=
push y           shorthand for: 0($sp) := y; $sp := $sp-4;
pop              shorthand for: $sp := $sp + 4;
top              shorthand for: 4($sp)

                  in the above statements (besides the load/store listed
                  first): "x" must be a register and each of "y"
                  and "z" may be a register or a constant value
```