

Stack machine characteristics and Pseudo-MIPS syntax

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-addressible
- 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

Syntax of the assembly language to generate (we'll call it **pseudo-MIPS**). It has operations like MIPS, but allows you to take operands directly from memory:

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

```
"x", "y", and "z" may be one of:
    a register
    offset(reg)
```

and "y" and/or "z" may also be constant values