sas 2.0

Feel free to change "sas.y"

The machine has a stack (the size of which is up to you), 26 registers: a .. z, and there can be up to 1000 labels (L000 .. L999) in a program.

A very simple stack machine and its assembly instructions

Implemented by sas.l + sas.y (sas = simple assembler)

N: an integer variable (one of a .. z) or an integer literal
R: a register name (one of a .. z)

top: the stack pointer, pointing at the topmost element of the stack

push N	stack[++top] = N
pop R	R = stack[top--]
compLT	if stack[top-1] < stack[top] then stack[--top] = 1 else 0
compGT	if stack[top-1] > stack[top] then stack[--top] = 1 else 0
compGE	if stack[top-1] >= stack[top] then stack[--top] = 1 else 0
compLE	if stack[top-1] <= stack[top] then stack[--top] = 1 else 0
compNE	if stack[top-1] != stack[top] then stack[--top] = 1 else 0
compEQ	if stack[top-1] == stack[top] then stack[--top] = 1 else 0
print	print stack[top]
read	stack[++top] = input an integer
add	X = stack[top-1] + stack[top]; stack[--top] = X
sub	X = stack[top-1] - stack[top]; stack[--top] = X
mul	X = stack[top-1] * stack[top]; stack[--top] = X
div	X = stack[top-1] / stack[top]; stack[--top] = X
neg	stack[top] = -stack[top]
and	X = stack[top-1] && stack[top]; stack[--top] = X
or	X = stack[top-1] || stack[top]; stack[--top] = X
jz Lxxx	if (stack[top--] == 0) then jump to label **Lxxx**
jmp Lxxx	jump to label **Lxxx**

Note: these instructions are "destructive" – the two operands are replaced by the result; similarly, for the add, sub, ...

"and" "or" are implemented in the stack machine in the same fashion as the arithmetic operators, which makes complex logical expressions very easy to compile into assembly code; otherwise, it would be quite difficult (we'll see in the near future)

This one is destructive too

```
flex sas2.l
bison -d sas2.y
gcc -o sas2 lex.yy.c sas2.tab.c
./sas2 num.sas2
```

Instructions added:

- **pushi R** -> stack[top] = R[stack[top]]
- **popi R** -> R[stack[top--]] = stack[top--]
- **//** comments