**Target Code Generation**

Input: intermediate language (IL)

Output: target language program Target languages:

– absolute binary (machine) code

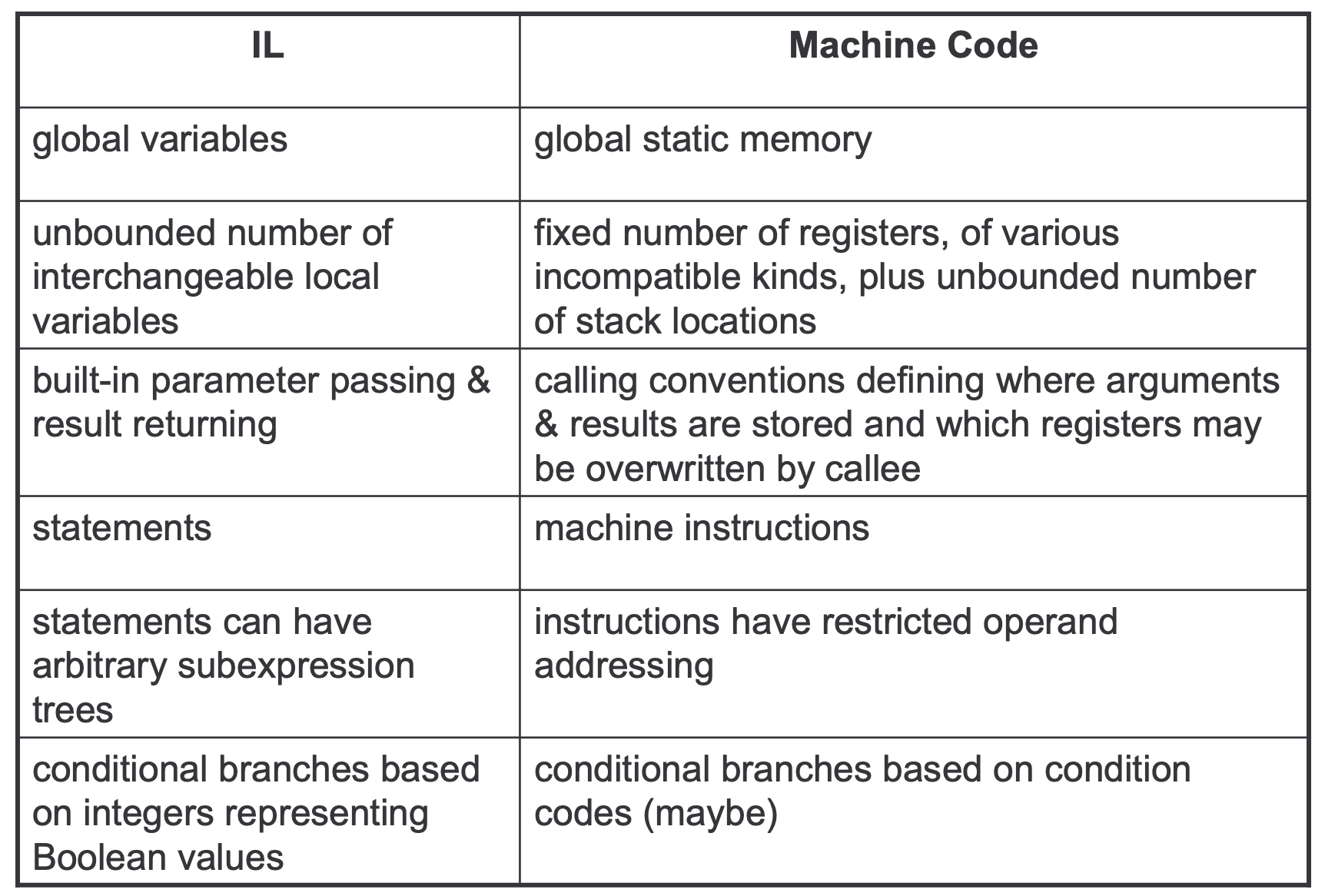
– relocatable binary code

– assembly code

– C

Target code generation must bridge the gap.

The gap, if target is machine code.



**Tasks of Code Generator**

**Register allocation**

– for each IL variable, select register/stack location/global memory location(s) to hold it.

• can depend on type of data, which operations manipulate it

• **Stack frame layout**

– compute layout of each function’s stack frame

• **Instruction selection**

– for each IL instruction (sequence), select target language instruction (sequence).

• includes operand addressing mode selection

• **Can have complex interactions**

– instruction selection depends on where operands are allocated

– some IL variables may not need a register, depending on the instructions & addressing modes that are selected.

**Register Allocation**

Intermediate language uses unlimited temporary variables

• makes ICG easy

Target machine has fixed resources for representing “locals” plus other internal things such as stack pointer

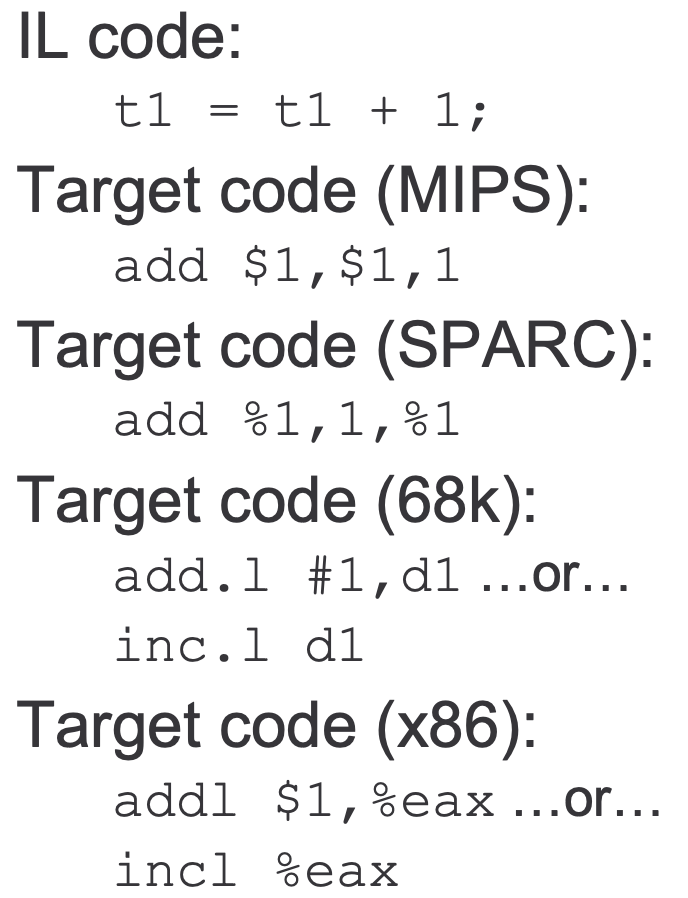
• MIPS, SPARC: 31 registers + 1 always-zero register

• 68k: 16 registers, divided into data and address regs

• x86: 8 word-sized integer registers (with a number of instruction specific restrictions on use) plus a stack of floating-point data manipulated only indirectly.

Registers are much faster than memory.

**Example:**



**References:**

* 1. <https://www.collegenote.net/pastpapers/4254/question/#gsc.tab=0>
  2. <https://www.geeksforgeeks.org/intermediate-code-generation-in-compiler-design/>
  3. https://courses.cs.washington.edu/courses/cse401/06sp/codegen.pdf
  4. Compilers: Principles, Techniques, and Tools: Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman