Code Generation

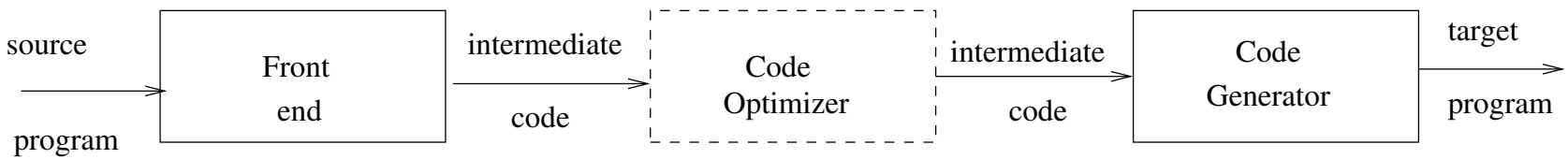
Sudakshina Dutta

IIT Goa

6^{th} May, 2022

Machine Code Generation

- The final phase
- Input : Intermediate representation (IR) produced by the front end and symbol table
- Output : Semantically equivalent target program



Machine Code Generation

- The target program has to be correct
- Mathematically, the problem of generating an optimal target program from a source program is undecidable
- Mostly based on heuristics
- With heuristics, generation of a fast code is possible

Machine Code Generation

Three tasks

- **Instruction selection**
- **Register allocation and assignment**
- **Instruction ordering**

Instruction selection

- We need to consider nature of instruction-set architecture and code quality
- Statement-by-statement code generator often produces poor code

$$
\begin{array}{ll}
a = b + c & LD\ R0, b \quad //R0 = b \\
d = a + e & ADD\ R0, R0, c \quad //R0 = R0 + c \\
 & ST\ a, R0 \quad //a = R0 \\
 & LD\ R0, a \quad //R0 = a \\
 & ADD\ R0, R0, e \quad //R0 = R0 + e \\
 & ST\ d, R0 \quad //d = R0
\end{array}
$$

- Here the fourth statement is redundant

Instruction selection

- The target machine may not support each instruction in uniform manner

  — For some machine, the floating point operations are performed using separate registers

- If the target machine supports "increment" instruction, then the three-address instruction $a = a + 1$ may be implemented by $INC\ a$ instead of
  $LD\ R0, a$
  $ADD\ R0, R0, \#1$
  $ST\ a, R0$
  for efficiency reason

Register allocation

- We select the set of variables that will reside in registers at each point in the program
- The reason is registers are the fastest computational unit and we do not have many registers
- The alternative of register is memory and it is much slower
- Mathematically, the problem of finding optimal assignment of registers is NP-complete

Instruction ordering

- The order in which computations are performed can affect the efficiency of the target code
- Some computation orders require fewer registers or fewer instructions
- Picking the best order is NP-complete problem

$c = a + b$
$p = a$

$LD\ R0, a$
$LD\ R1, b$
$ADD\ R0, R0, R1$
$ST\ R0, c$
$LD\ R0, a$
$ST\ R0, p$

$LD\ R0, a$
$ST\ R0, p$
$LD\ R1, b$
$ADD\ R0, R0, R1$
$ST\ R0, c$

The target program

- The instruction set architecture has significant impact on constructing a good code generator
- We only consider RISC (Reduced Instruction Set Computer) which has many registers, three-address instruction and a relatively simple architecture
- There are other architectures e.g., CISC and stack-based machines

The target program

- Producing absolute machine-language program has the advantage that the program can be compiled quickly
- Producing a relocatable machine-language program allows subprograms to be compiled separately and then link using linking loader
- Here we have considered the situation where assembly language programs are produced as output; it requires a further assembly step

A simple target machine model

- It is a three-address machine with load-store operations, computation operations, jump operations and conditional jumps
- Very limited set of instructions and assume the operands are integers

A simple target machine model

- **Load operations :** The instruction $LD\ dst, addr$ loads the value in location $addr$ into location dst
- **Store operation :** The instruction $ST\ x, r$ stores the value in register r into the location x
- **Computation operation :** The form is $OP\ dst, src_1, src_2$, where OP is a operator like ADD, SUB, and dst, src_1, src_2 are locations, not necessarily distinct

A simple target machine model

- **Unconditional jumps :** The instruction *BR L* causes control to branch to the machine instruction with label *L*
- **Conditional branch :** It is of the form *Bcond r*, *L*, where *r* is a register and *L* is a label and *cond* stands for any condition based on *r*

An example jump instruction

The machine code corresponding to

$$if\ x < y\ goto\ L$$

is

$$\begin{gathered} LD\ R1, x \\ LD\ R2, y \\ SUB\ R1, R1, R2 \\ BLTZ\ R1, M \end{gathered}$$

where M is the label representing the first instruction generated from the three-address instruction having label L