Sequence 4.1 – Intermediate Representation

P. de Oliveira Castro S. Tardieu

Intermediate Representation

- An Intermediate Representation (IR) should be,
  - Simple and terse (remove high-level *syntactic sugar*)
  - Optimizable (must preserve enough *information* to enable optimizations)
- **Tradeoffs** in the level of abstraction are necessary
- Eg. Should the IR have an array type ?
  - **Yes**: eases dependencies analysis
    - easy to know that two references map to the same array
  - **No**: complicates scalar optimizations such as constant propagation or force reduction
    - require a special case for array elements

Design Choices

- **Register vs. Stack**: keep local variables in named registers or a stack ?
  - Stack: simple code generation and interpretation
  - Register: eases dependency analysis and optimizations
- **Flat vs. Hierarchical**: is the IR a list of instructions or a tree ?
  - Hierarchical: preserves scopes and structure of the code
  - Flat: closer to target assembly; eases moving around instructions

Lowering and Three Address Code

Lowering: transforming a high-level representation (AST) into IR

- Classical lowering of expressions into **three-address code**, where each operation has at most three operands,

```
y := 4*x*x - 2*x + 1
```

becomes

```
x1 := x * x;
x2 := 4 * x1;
x3 := 2 * x;
x4 := x2 - x3;
y := x4 + 1;
```

Multiple IR ?

In some compilers multiple IR levels are used.

GCC

- Generic (common AST format for all frontends)
- Gimple (register + hybrid: hierarchical / flat)
  - lowers all control structures to a canonical form
  - lowers expressions to 3-address code
- Gimple SSA (a Single Static Assignment form of GIMPLE)
  - we'll see later what SSA is
- RTL, Register Transfer Language (hybrid: register / stack + hierarchical)
  - lisp like
  - very close to machine assembly

Original Program

```
int f(int x) {
  int y = 0;
  if (x>0)
    y = 4*x*x + 1;
  return y;
}
```

GIMPLE

```
[...]

  y = 0;
  if (x > 0) goto <D.4170>; else goto <D.4171>;
  <D.4170>:
  D.4172 = x * 4;
  D.4173 = D.4172 * x;
  y = D.4173 + 1;
  <D.4171>:
  D.4174 = y;
  goto <D.4175>;
  <D.4175>:
  return D.4174;
}
```

RTL

```
[...]
(insn 10 9 11 4 (set (reg:SI 115 [ D.4180 ])
  (ashift:SI (reg/v:SI 114 [ x ])
   (const_int 2 [0x2]))) ex1.c:4 -1
  (nil))
(insn 11 10 12 4 (set (reg:SI 116 [ D.4180 ])
  (mult:SI (reg/v:SI 114 [ x ])
   (reg:SI 115 [ D.4180 ]))) ex1.c:4 -1
  (nil))
(insn 12 11 22 4 (set (reg/v:SI 110 [ y ])
  (plus:SI (reg:SI 116 [ D.4180 ])
   (const_int 1 [0x1]))) ex1.c:4 -1
  (nil))
(jump_insn 22 12 23 4 (set (pc)
  (label_ref 13)) -1
  (nil)
 -> 13)
(barrier 23 22 25)
```