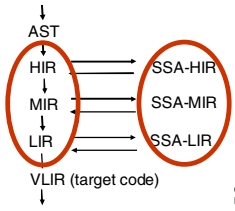
Introduction to Static Single Assignment (SSA) Form

Christoph Kessler, IDA, Linköpings universitet, 2009.

Recall: Multi-Level IR, Standard vs. SSA Form



Christoph Kessler, IDA, Linköpings universitet, 2009.

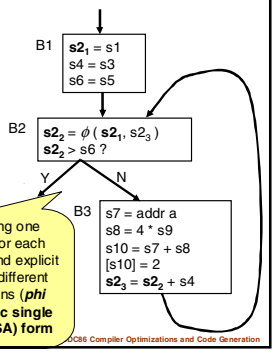
Example with SSA-LIR

(adapted from Muchnick'97)

s2 is assigned (written, defined) multiple times in the program text (i.e., multiple static assignments)

- LIR:

```
    s2 = s1
    s4 = s3
    s6 = s5
L1: if s2 > s6 goto L2
    s7 = addr a
    s8 = 4 * s9
    s10 = s7 + s8
    [s10] = 2
    s2 = s2 + s4
    goto L1
L2:
```



After introducing one ***version*** of s2 for each static definition and explicit merger ops for different reaching versions (***phi nodes***, ϕ): **Static single assignment (SSA) form**

Static Single Assignment (SSA) Form

Goal:

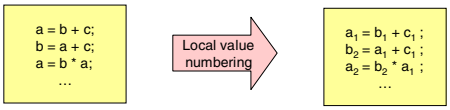
- increase efficiency of inter/intra-procedural analyses and optimizations
- speed up dataflow analysis
- represent def-use relations explicitly

Idea:

- Represent program as a directed graph of operations op
  - Represent statements / quadruples / instructions as assignments v = v' op v'' with *v*, *v'*, *v''* a variable / label / symbolic register / temporary (edge) connecting operations
- **SSA-Property:**
  There is only *one position* (statement, quadruple, instruction) in a program/procedure defining a variable version *v* → *static value*
  - Does not mean that *v* is computed only once at runtime:
    Due to iteration / recursion, the program point may be executed more than once with different *dynamic* values.

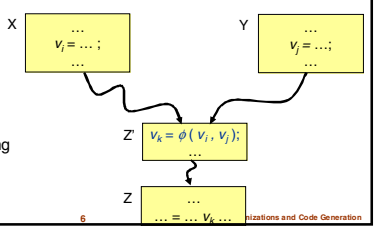
SSA Construction (1): Value Numbering in a Single Basic Block

- Assign a distinct name (e.g. variable name + index) to each static value computed in the block
- Can be done on-the-fly when constructing DAGs (see Lecture 1)
- Makes local Def-Use chains explicit
- For several basic blocks: use (procedure-wide) unique indices



SSA Construction (2) – Insert Phi nodes to stitch DU-chains between blocks together

- For different basic blocks X and Y both defining a variable *v*, say v_i in X and v_j in Y, if non-empty paths X→⁺ Z and Y→⁺ Z exist in the control flow graph to a common successor block Z containing a use v_k of *v* that is reached by both definitions, with Z' being the first common node on the two paths, then a Phi node $v_k = \phi(v_i, v_j)$ must be inserted in Z'.
- A Phi node in a block *B* has | Pred(*B*) | operands



→ Global value numbering

Algorithms for SSA Construction

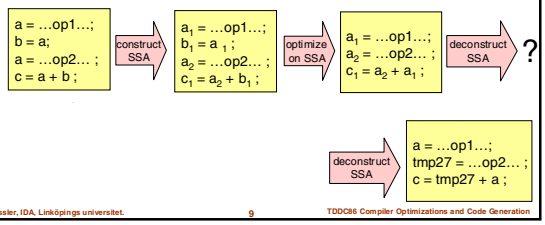
- Standard algorithm by Cytron et al. 1989 (iterated dominance frontiers)
  - See Muchnick, Section 8.11
- Other algorithms for SSA construction exist
  - Optimize number of Phi nodes
- Standard transformations like constant folding, arithmetic simplification, common subexpression elimination can also reduce the number of Phi nodes.

Some details…

- Array variables??
  - Phi-node to copy entire array if only one element is written
  - Use special array Phi operators
- Dynamically allocated objects??
  - Example:
    while (…) {
    ptr = **new** Listitem();
    ptr->next = list;
    list = ptr;
    }
  - Different created Listitem fields can no longer be identified and named statically
- **Memory-SSA**
  - For target-level SSA form: Dependences between Load and Store instructions through memory should be made explicit with special memory-Phi nodes

Converting from SSA back to standard IR

- Simply throwing away the indices and Phi nodes????
  - **No!**
    Optimizations on SSA representation may have created overlaps of different static values of the same variable…
  - Example:



References

- R. Cytron, J. Ferrante, B. Rosen, et al.:
  Efficiently computing static single assignment form.
  Proc. POPL-1989, pp. 25-35, ACM.
- Muchnick, Section 8.11