Local Optimization

converting 3ac into assembly

- Simple approach: **macro expansion**
- Treat each 3AC instruction separately, generate code in isolation

```
ADD C, A, B
```

→

```
LA r1 <addr of A>
LW r2, 0(r1)
LA r3 <addr of B>
LW r4, 0(r3)
ADD r5, r2, r4
LA r6 <addr of C>
SW r5, 0(r6)
```

converting 3ac into assembly

- Simple approach: **macro expansion**
- Treat each 3AC instruction separately, generate code in isolation
- Problem: inefficient!
  - Too many registers
  - Redundant loads, adds

```
ADD C, A, B
ADD D, A, B
```

→

```
LA r1 <addr of A>
LW r2, 0(r1)
LA r3 <addr of B>
LW r4, 0(r3)
ADD r5, r2, r4
LA r6 <addr of C>
SW r5, 0(r6)
```

```
LA r7 <addr of A>
LW r8, 0(r7)
LA r9 <addr of B>
LW r10, 0(r9)
ADD r11, r8, r10
LA r12 <addr of D>
SW r11, 0(r12)
```

converting 3ac into assembly

- Simple approach: **macro expansion**
- Treat each 3AC instruction separately, generate code in isolation
- Problem: inefficient!
  - Too many registers
  - Redundant loads, adds

```
ADD C, A, B
ADD D, A, B
```

→

LA r1 <addr of A>
LW r2, 0(r1)
LA r3 <addr of B>
LW r4, 0(r3)
ADD r5, r2, r4
LA r6 <addr of C>
SW r5, 0(r6)

~~LA r7 <addr of A>~~
~~LW r8, 0(r7)~~
~~LA r9 <addr of B>~~
~~LW r10, 0(r9)~~
~~ADD r11, r8, r10~~
LA r12 <addr of D>
SW r5, 0(r12)

converting 3ac into assembly

- Simple approach: **macro expansion**
- Treat each 3AC instruction separately, generate code in isolation
- Problem: inefficient!
  - Too many registers
  - Redundant loads, adds

ADD C, A, B
~~ADD D, A, B~~
MOV D, C

→

LA r1 <addr of A>
LW r2, 0(r1)
LA r3 <addr of B>
LW r4, 0(r3)
ADD r5, r2, r4
LA r6 <addr of C>
SW r5, 0(r6)

~~LA r7 <addr of A>~~
~~LW r8, 0(r7)~~
~~LA r9 <addr of B>~~
~~LW r10, 0(r9)~~
~~ADD r11, r8, r10~~
LA r12 <addr of D>
SW r5, 0(r12)

one perspective on optimization

- Almost all compiler transformations fall into one of two categories:
  - **Optimizing** computation: simplifying computations, removing unnecessary or redundant computations
  - **Scheduling** computation: changing the order of when computations occur to improve code behavior

- These types of optimization can interact with each other: optimizing computations can change the impact of scheduling decisions, and scheduling decisions can enhance (or inhibit) opportunities for simplifying code

optimizating computations

- Optimize translation of 3AC to assembly to improve generated code
  - Eliminate redundant computation: **common subexpression elimination**
  - Eliminate unused code: **dead code elimination**
  - Optimize use of registers, eliminate unneeded loads/stores: **register allocation**