Optimization I

CS 4447 / CS 9545 – Stephen M. Watt

***The* University *of* Western Ontario**

Overview

- Machine independent optimizations. Purple Dragon book, Chapter 9.
- Things you might have heard about:
  - “Peep hole” optimization
  - Common sub-expression elimination
  - Dead code elimination
- Things you might not have heard about:
  - Flow graphs
  - Data flow analysis
  - Live variable analysis
  - UD and DU chains
  - Static Single Assignment
  - Register allocation by graph coloring

The Main Sources of Optimization

- Removing unnecessary code/computation
  - Global common sub-expressions
  - Copy propagation (re-using assignments)
  - Constant folding
  - Dead code elimination
  - Moving code out of loops
  - Strength reduction (using simpler operations)

- Removing unused data
  - Dead variable elimination

Peephole Optimization

- Examine the sequence of instructions for local improvements.

- Sliding window
- Replace sequence with shorter, faster, sequence giving the same result
- Multiple passes

- Redundant instruction elimination
- Flow of control optimization (e.g. when going to a goto)
- Algebraic simplifications
- Use of machine idioms (e.g. doing arithmetic by effective address computation)

Eliminating Redundant Instructions

- E.g. replace

```
LD a, R0
ST R0, a
```

with

```
LD a, R0
```

Eliminating Unreachable Code

- E.g.

```
        If (debug == 1) goto L1
        goto L2
L1:     print debugging info
L2:     stuff
```

becomes

```
        if (debug != 1) goto L2
        print debugging info
L2:     stuff
```

Flow of Control Optimization

- E.g.

```
        goto L1
        …
L1:     goto L2
```

becomes

```
        goto L2
        …
L1:     goto L2
```

Algebraic Simplification

- Use identites to eliminate useless instructions

  x = x + 0

  x = x * 1

- Strength reduction

  x = 4 * y => x = y<<2

Machine Idioms

- Use addressing modes to do real work.
  E.g.
  - Post increment or pre-decrement addressing

    *r++ *--r

  - Effective address calculation

    R[B + I]

The Principal Optimizations in Aldor

- The usual, as outlined on the previous slide, PLUS:

- Procedure integration (inlining)
- Jump-flow optimization (optimizing generic iteration)
- Data structure elimination
- Environment merging
- Leaf function optimization

Flow Graphs

- Basic blocks
  - One point of entry, one point of exit
  - Exit may be multi-way,
    e.g. if with fall-through, computed goto

- Directed graph with a basic block at each vertex.

- Each block has a (potentially empty) set of

  predecessors and

  successors.