Local Optimization

- Given a basic block the student will be able to apply local value numbering to that block.

Why Local Optimization?

- Consider the following

```
iLDI    5       ⇒ r1
iADD    r1 r2   ⇒ r3
i2i     r1      ⇒ r4
iADD    r4 r2   ⇒ r5
iSUB    r4 r1   ⇒ r6
```

What operations can be removed?

Assumptions

- If one instruction in the basic block is executed, they all are.
- What happens outside of the basic block is unknown
- Compiler temporaries may be live outside the basic block
- All lexically identical expressions store into the same result register

Value Numbering

- Each "value" within a basic block is assigned a unique number
- A variable may have different value numbers at different points in a basic block
- Variables with the same value numbers are equivalent
- Optimizations
  1. constant propagation and folding
  2. scalar propagation
  3. common subexpression elimination
  4. expose dead code

Information

- What information is needed?
  1. Is a variable or value number constant?
     - constant table
     - symbol table
  2. Has this expression already been computed?
     - available expression table

Constant Table

Value #	Value
...	...

Symbol Table

Name	Value #	Subsumed-by	Subsumes
...	...	...	...

Information

Available Expression Table

r_1-value #	operator	r_2-value #	l-value name
…	…	…	…

Notes

- valnum and setvalnum must add entries to Constant Table and Symbol Table when there is no entry
- Moves between the same register are not added to the table
- This algorithm won't catch redundancies due to instructions changed to an iLDI
- Assume dead code elimination will be done afterwards

Local Value Numbering

```
while ∃ an instruction do
    I = the next instruction
    applySubsume(I)
    l = l-value(I)
    if I is an iLDI {
      r = valnum(r-value(I))
      setvalnum(l,r)
    }
    else if I is a move {
      r = valnum(r-value(I))
      removeSubsume(l)
      setvalnum(l,r)
      if isConst(r)
         change I to iLDI
      else
          subsume(l,r-value(I))
     }
    else {
      r1 = valnum(r1-value(I))
      r2 = valnum(r2-value(I))
      op = operator(I)
      if isConst(r1) && isConst(r2) {
        v = r1 op r2
        change I to load immed.
        removeSubsume(l)
        setvalnum(l,valnum(v))
      }
      else {
        if <r1,op,r2> in expression
          table {
            lt = l-value(<r1,op,r2>)
            v = valnum(lt)

// continued on next slide
```

Local Value Numbering

```
            change I to move
            removeSubsume(I)
            setvalnum(I,v)
            subsume(I,I_t)
        }
        else {
            propagate constants
            insert(<r_1,op,r_2>,I)
            setvalnum(I,newval())
        }
    }
enddo
```

Example

```
iLDI   5       ⇒ r1
iADD   r1 r2   ⇒ r11
i2i    r11     ⇒ r3
iLDI   3       ⇒ r4
iADD   r4 r1   ⇒ r12
i2i    r12     ⇒ r6
i2i    r2      ⇒ r7
iLDI   5       ⇒ r1
iADD   r1 r7   ⇒ r13
```