

# Instruction Selection: Peephole Matching

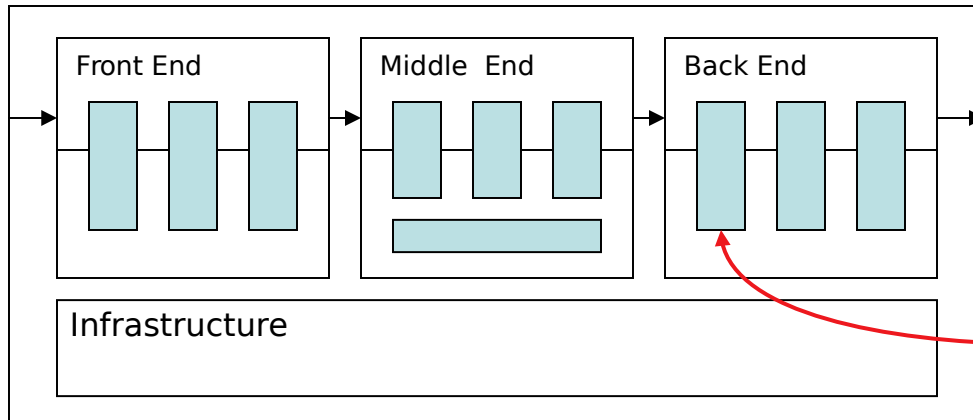
Copyright 2003, Keith D. Cooper, Ken Kennedy & Linda Torczon, all rights reserved.

# The Problem

---

Writing a compiler is a lot of work

- Would like to reuse components whenever possible
- Would like to automate construction of components



Today's lecture:  
Automating  
Instruction  
Selection

- Front end construction is largely automated
- Middle is largely hand crafted
- (Parts of) back end can be automated

# Definitions

---

## Instruction selection

- Mapping *IR* into assembly code
- Assumes a fixed storage mapping & code shape
- Combining operations, using address modes

## Instruction scheduling

- Reordering operations to hide latencies
- Assumes a fixed program (*set of operations*)
- Changes demand for registers

## Register allocation

- Deciding which values will reside in registers
- Changes the storage mapping, may add false sharing
- Concerns about placement of data & memory operations

# The Problem

---

Modern computers (still) have many ways to do anything

Consider register-to-register copy

- Obvious operation is `i2i ri ⇒ rj`
- Many others exist

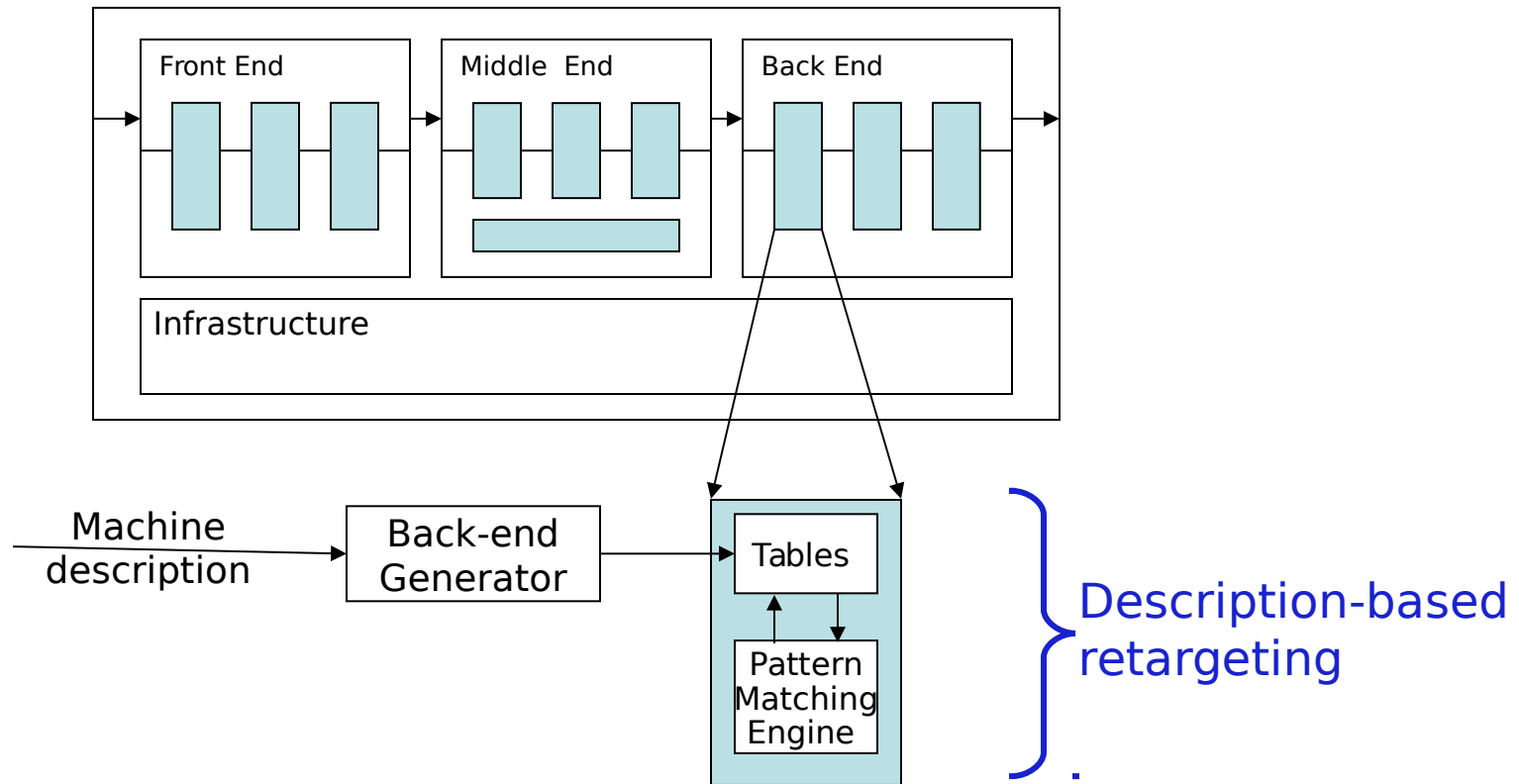
|                                                    |                                                   |                                                      |
|----------------------------------------------------|---------------------------------------------------|------------------------------------------------------|
| <code>addI r<sub>i</sub>,0 ⇒ r<sub>j</sub></code>  | <code>subI r<sub>i</sub>,0 ⇒ r<sub>j</sub></code> | <code>lshiftI r<sub>i</sub>,0 ⇒ r<sub>j</sub></code> |
| <code>multI r<sub>i</sub>,1 ⇒ r<sub>j</sub></code> | <code>divI r<sub>i</sub>,1 ⇒ r<sub>j</sub></code> | <code>rshiftI r<sub>i</sub>,0 ⇒ r<sub>j</sub></code> |
| <code>orI r<sub>i</sub>,0 ⇒ r<sub>j</sub></code>   | <code>xorI r<sub>i</sub>,0 ⇒ r<sub>j</sub></code> | ... and others ...                                   |

- Human would ignore all of these
- Algorithm must look at all of them & find low-cost encoding
  - Take context into account *(busy functional unit?)*

And this is an overly-simplified example

# The Goal

Want to automate generation of instruction selectors



Machine description should also help with scheduling & allocation

# The Big Picture

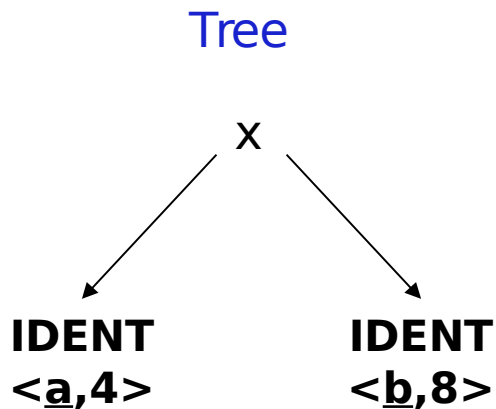
---

Need pattern matching techniques

- Must produce good code *(some metric for good)*
- Must run quickly

Our treewalk (visitor) code generator ran quickly

How good was the code?



Treewalk Code

```
loadl 4 ⇒ r5
loadA r5 ⇒ r6
loadl 8 ⇒ r7
loadA r7 ⇒ r8
mult r6, r8 ⇒ r9
```

Desired Code

```
loadAl 4 ⇒ r5
loadAl 8 ⇒ r6
mult r5, r6 ⇒ r7
```

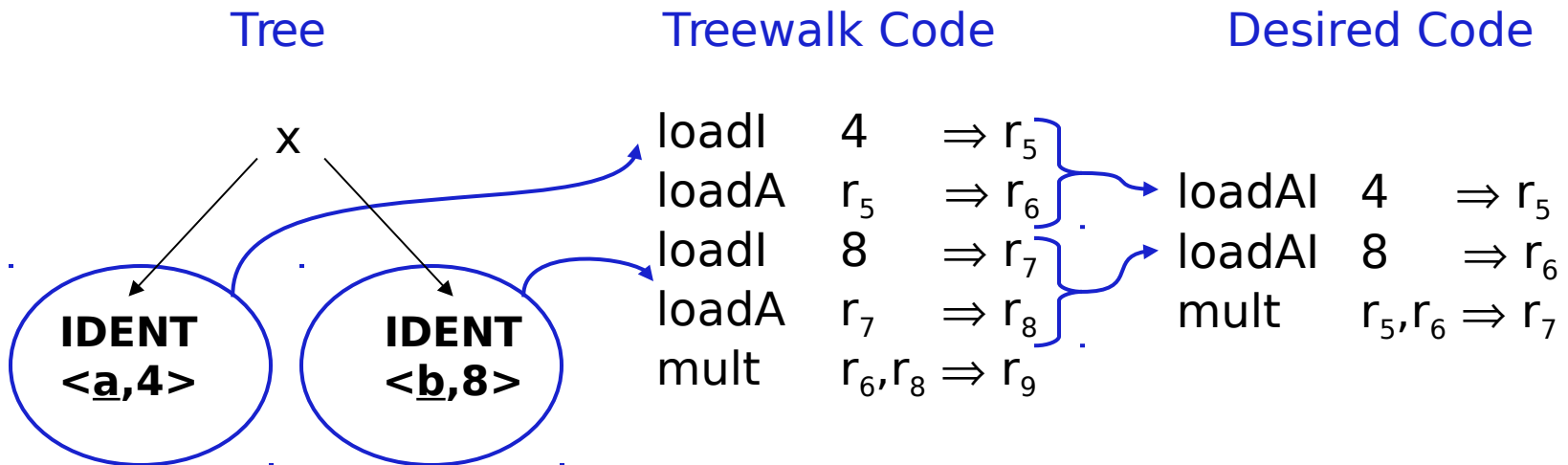
# The Big Picture

Need pattern matching techniques

- Must produce good code *(some metric for good)*
- Must run quickly

Our treewalk code generator ran quickly

How good was the code?



# The Big Picture

---

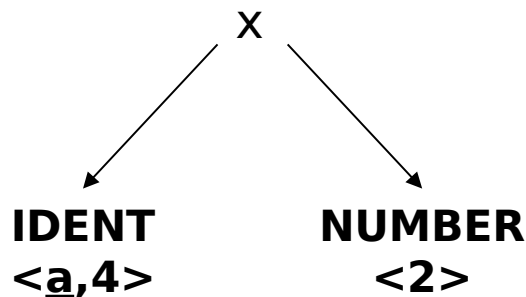
Need pattern matching techniques

- Must produce good code *(some metric for good )*
- Must run quickly

Our treewalk code generator ran quickly

How good was the code?

Tree



Treewalk Code

```
loadl 4 ⇒ r5
loadA r5 ⇒ r6
loadl 2 ⇒ r7
mult r6, r7 ⇒ r8
```

Desired Code

```
loadAl 4 ⇒ r5
multl r5, 2 ⇒ r7
```

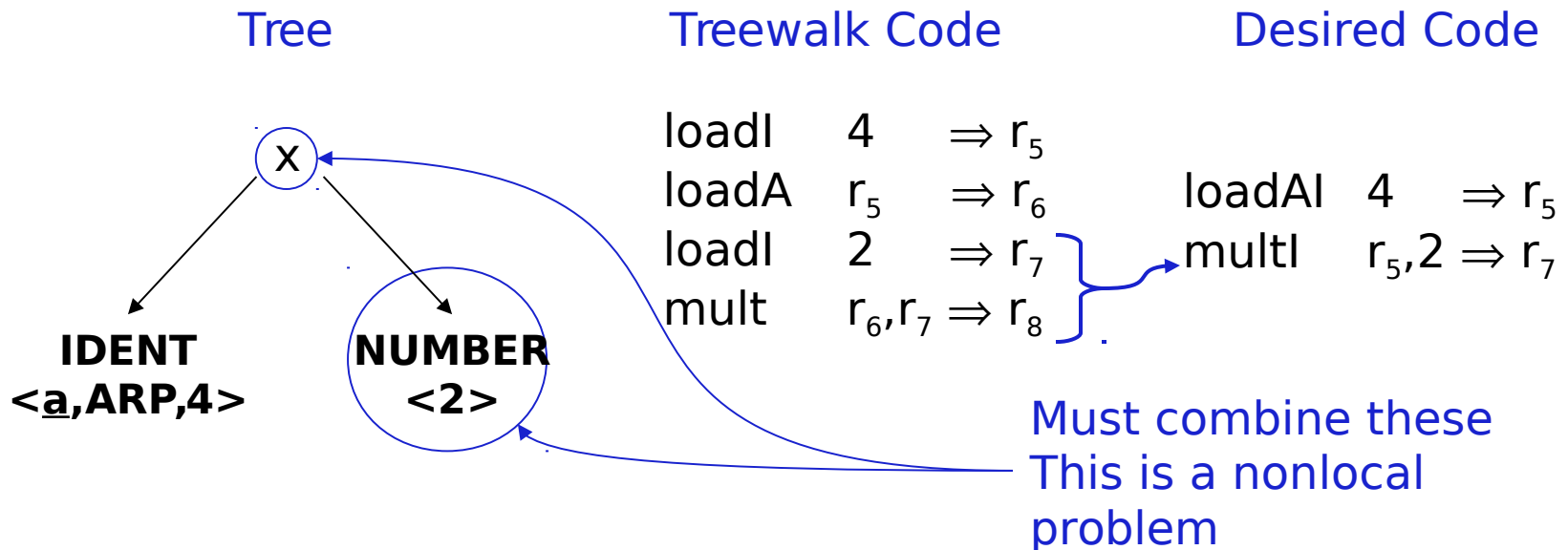
# The Big Picture

Need pattern matching techniques

- Must produce good code *(some metric for good)*
- Must run quickly

Our treewalk code generator ran quickly

How good was the code?



# The Big Picture

---

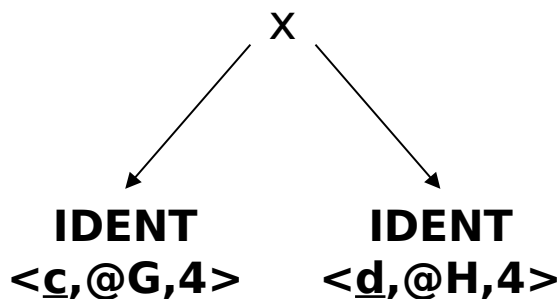
Need pattern matching techniques

- Must produce good code *(some metric for good)*
- Must run quickly

Our treewalk code generator ran quickly

How good was the code?

## Tree



## Treewalk Code

```
loadl  @G ⇒ r5
loadl  4  ⇒ r6
loadAO r5,r6 ⇒ r7
loadl  @H ⇒ r7
loadl  4  ⇒ r8
loadAO r8,r9 ⇒ r10
mult   r7,r10 ⇒ r11
```

## Desired Code

```
loadl  4      ⇒ r5
loadAl r5,@G ⇒ r6
loadAl r5,@H ⇒ r7
mult   r6,r7 ⇒ r8
```

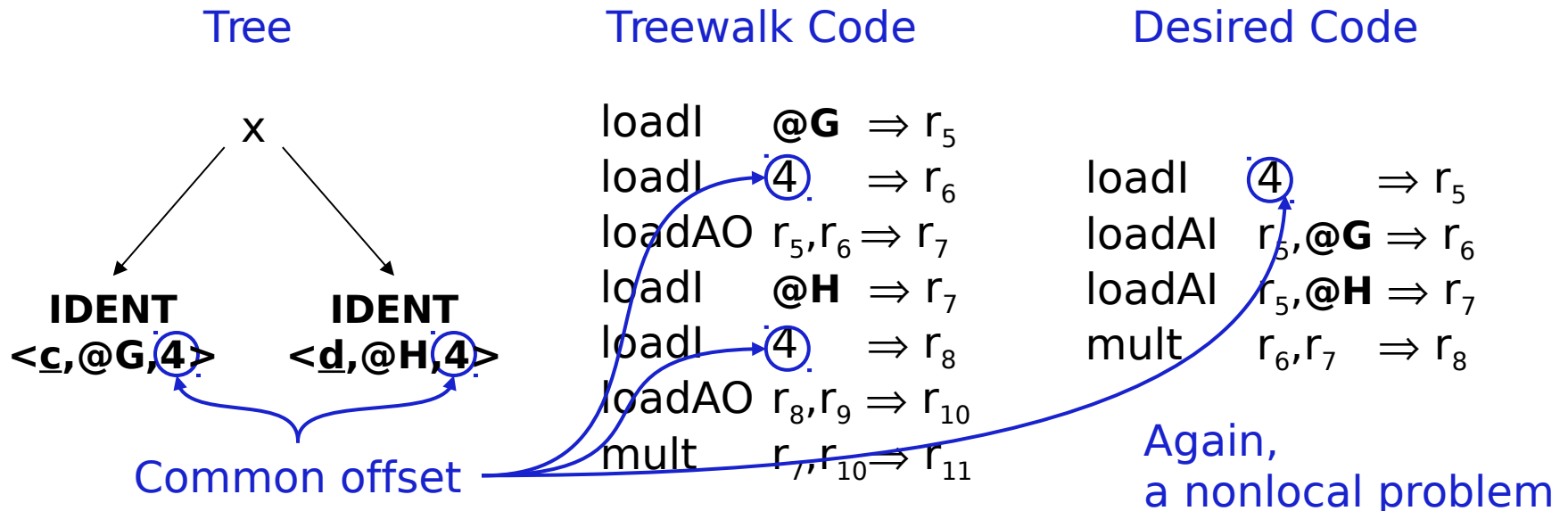
# The Big Picture

Need pattern matching techniques

- Must produce good code *(some metric for good)*
- Must run quickly

Our treewalk code generator met the second criteria

How did it do on the first ?



## How do we perform this kind of matching ?

---

Tree-oriented IR suggests pattern matching on trees

- Tree-patterns as input, matcher as output
- Each pattern maps to a target-machine instruction sequence
- Use dynamic programming or bottom-up rewrite systems

Linear IR suggests using some sort of string matching

- Strings as input, matcher as output
- Each string maps to a target-machine instruction sequence
- Use text matching (Aho-Corasick) or peephole matching

In practice, both work well; matchers are quite different

# Peephole Matching

---

- Basic idea
- Compiler can discover local improvements locally
  - Look at a small set of adjacent operations
  - Move a “peephole” over code & search for improvement
- Classic example was store followed by load

## Original code

```
storeAl  r1    ⇒ 8  
loadAl   8      ⇒ r15
```

## Improved code

```
storeAl r1    ⇒ 8  
i2i     r1    ⇒ r15
```

# Peephole Matching

---

- Basic idea
- Compiler can discover local improvements locally
  - Look at a small set of adjacent operations
  - Move a “peephole” over code & search for improvement
- Classic example was store followed by load
- Simple algebraic identities

## Original code

```
addl    r2,0  ⇒ r7
mult    r4,r7 ⇒ r10
```

## Improved code

```
mult    r4,r2 ⇒ r10
```

# Peephole Matching

---

- Basic idea
- Compiler can discover local improvements locally
  - Look at a small set of adjacent operations
  - Move a “peephole” over code & search for improvement
- Classic example was store followed by load
- Simple algebraic identities
- Jump to a jump

## Original code

```
    jumpl    → L10  
L10: jumpl    → L11
```

## Improved code

```
L10: jumpl    → L11
```

# Peephole Matching

---

Implementing it

- Early systems used limited set of hand-coded patterns
- Window size ensured quick processing

Modern peephole instruction selectors

- Break problem into three tasks



- Apply symbolic interpretation & simplification systematically

# Peephole Matching

---

## Expander

- Turns IR code into a low-level IR (LLIR) such as RTL\*
- Operation-by-operation, template-driven rewriting
- LLIR form includes all direct effects
- Significant, albeit constant, expansion of size



\*RTL = Register transfer language

# Peephole Matching

---

## Simplifier

- Looks at LLIR through window and rewrites it
- Uses forward substitution, algebraic simplification, local constant propagation, and dead-effect elimination
- Performs local optimization within window



- This is the heart of the peephole system
  - Benefit of peephole optimization shows up in this step

# Peephole Matching

---

## Matcher

- Compares simplified LLIR against a library of patterns
- Picks low-cost pattern that captures effects
- Must preserve LLIR effects, may add new ones (*e.g., set cc*)
- Generates the assembly code output



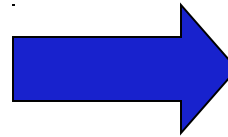
# Example

---

Original IR Code

| OP   | Arg <sub>1</sub> | Arg <sub>2</sub> | Result         |
|------|------------------|------------------|----------------|
| mult | 2                | Y                | t <sub>1</sub> |
| sub  | x                | t <sub>1</sub>   | w              |

Expand



LLIR Code

```
r10 ← 2
r11 ← @y
r12 ← r0 + r11
r13 ← MEM(r12)
r14 ← r10 × r13
r15 ← @x
r16 ← r0 + r15
r17 ← MEM(r16)
r18 ← r17 - r14
r19 ← @w
r20 ← r0 + r19
MEM(r20) ← r18
```

Each register is  
single use

# Example

---

## LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

$r_{12} \leftarrow r_0 + r_{11}$

$r_{13} \leftarrow \text{MEM}(r_{12})$

$r_{14} \leftarrow r_{10} \times r_{13}$

$r_{15} \leftarrow @x$

$r_{16} \leftarrow r_0 + r_{15}$

$r_{17} \leftarrow \text{MEM}(r_{16})$

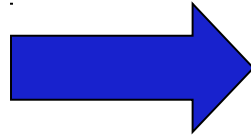
$r_{18} \leftarrow r_{17} - r_{14}$

$r_{19} \leftarrow @w$

$r_{20} \leftarrow r_0 + r_{19}$

$\text{MEM}(r_{20}) \leftarrow r_{18}$

Simplify



## LLIR Code

$r_{13} \leftarrow \text{MEM}(r_0 + @y)$

$r_{14} \leftarrow 2 \times r_{13}$

$r_{17} \leftarrow \text{MEM}(r_0 + @x)$

$r_{18} \leftarrow r_{17} - r_{14}$

$\text{MEM}(r_0 + @w) \leftarrow r_{18}$

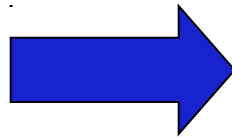
# Example

---

## LLIR Code

$r_{13} \leftarrow \text{MEM}(r_0 + @y)$   
 $r_{14} \leftarrow 2 \times r_{13}$   
 $r_{17} \leftarrow \text{MEM}(r_0 + @x)$   
 $r_{18} \leftarrow r_{17} - r_{14}$   
 $\text{MEM}(r_0 + @w) \leftarrow r_{18}$

Match



## LLOC Code

loadAI  $r_0, @y \Rightarrow r_{13}$   
multl  $2 \times r_{13} \Rightarrow r_{14}$   
loadAI  $r_0, @x \Rightarrow r_{17}$   
sub  $r_{17} - r_{14} \Rightarrow r_{18}$   
storeAI  $r_{18} \Rightarrow r_0, @w$

- Introduced all memory operations & temporary names
- Turned out pretty good code

# Steps of the Simplifier

(3-operation window)

## LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

$r_{12} \leftarrow r_0 + r_{11}$

$r_{13} \leftarrow \mathbf{MEM}(r_{12})$

$r_{14} \leftarrow r_{10} \times r_{13}$

$r_{15} \leftarrow @x$

$r_{16} \leftarrow r_0 + r_{15}$

$r_{17} \leftarrow \mathbf{MEM}(r_{16})$

$r_{18} \leftarrow r_{17} - r_{14}$

$r_{19} \leftarrow @w$

$r_{20} \leftarrow r_0 + r_{19}$

$\mathbf{MEM}(r_{20}) \leftarrow r_{18}$

# Steps of the Simplifier

(3-operation window)

## LLIR Code

$r_{10} \leftarrow 2$   
 $r_{11} \leftarrow @y$   
 $r_{12} \leftarrow r_0 + r_{11}$   
 $r_{13} \leftarrow \mathbf{MEM}(r_{12})$

$r_{10} \leftarrow 2$   
 $r_{11} \leftarrow @y$   
 $r_{12} \leftarrow r_0 + r_{11}$

$r_{14} \leftarrow r_{10} \times r_{13}$

$r_{15} \leftarrow @x$

$r_{16} \leftarrow r_0 + r_{15}$

$r_{17} \leftarrow \mathbf{MEM}(r_{16})$

$r_{18} \leftarrow r_{17} - r_{14}$

$r_{19} \leftarrow @w$

$r_{20} \leftarrow r_0 + r_{19}$

$\mathbf{MEM}(r_{20}) \leftarrow r_{18}$

# Steps of the Simplifier (3-operation window)

## LLIR Code

$r_{10} \leftarrow 2$   
 $r_{11} \leftarrow @y$   
 $r_{12} \leftarrow r_0 + r_{11}$   
 $r_{13} \leftarrow \mathbf{MEM}(r_{12})$

$r_{14} \leftarrow r_{10} \times r_{13}$

$r_{15} \leftarrow @x$

$r_{16} \leftarrow r_0 + r_{15}$

$r_{17} \leftarrow \mathbf{MEM}(r_{16})$

$r_{18} \leftarrow r_{17} - r_{14}$

$r_{19} \leftarrow @w$

$r_{20} \leftarrow r_0 + r_{19}$

$\mathbf{MEM}(r_{20}) \leftarrow r_{18}$

$r_{10} \leftarrow 2$   
 $r_{11} \leftarrow @y$   
 $r_{12} \leftarrow r_0 + r_{11}$



$r_{10} \leftarrow 2$   
 $r_{12} \leftarrow r_0 + @y$   
 $r_{13} \leftarrow \mathbf{MEM}(r_{12})$

# Steps of the Simplifier (3-operation window)

## LLIR Code

$r_{10} \leftarrow 2$   
 $r_{11} \leftarrow @y$   
 $r_{12} \leftarrow r_0 + r_{11}$   
 $r_{13} \leftarrow \mathbf{MEM}(r_{12})$   
 $r_{14} \leftarrow r_{10} \times r_{13}$   
 $r_{15} \leftarrow @x$   
 $r_{16} \leftarrow r_0 + r_{15}$   
 $r_{17} \leftarrow \mathbf{MEM}(r_{16})$   
 $r_{18} \leftarrow r_{17} - r_{14}$   
 $r_{19} \leftarrow @w$   
 $r_{20} \leftarrow r_0 + r_{19}$   
 $\mathbf{MEM}(r_{20}) \leftarrow r_{18}$

$r_{10} \leftarrow 2$   
 $r_{12} \leftarrow r_0 + @y$   
 $r_{13} \leftarrow \mathbf{MEM}(r_{12})$



$r_{10} \leftarrow 2$   
 $r_{13} \leftarrow \mathbf{MEM}(r_0 + @y)$   
 $r_{14} \leftarrow r_{10} \times r_{13}$

# Steps of the Simplifier

(3-operation window)

## LLIR Code

$r_{10} \leftarrow 2$   
 $r_{11} \leftarrow @y$   
 $r_{12} \leftarrow r_0 + r_{11}$   
 $r_{13} \leftarrow \text{MEM}(r_{12})$   
 $r_{14} \leftarrow r_{10} \times r_{13}$   
 $r_{15} \leftarrow @x$   
 $r_{16} \leftarrow r_0 + r_{15}$   
 $r_{17} \leftarrow \text{MEM}(r_{16})$   
 $r_{18} \leftarrow r_{17} - r_{14}$   
 $r_{19} \leftarrow @w$   
 $r_{20} \leftarrow r_0 + r_{19}$   
 $\text{MEM}(r_{20}) \leftarrow r_{18}$

this instruction disappear  
(r10 not needed)

$r_{10} \leftarrow 2$   
 $r_{13} \leftarrow \text{MEM}(r_0 + @y)$   
 $r_{14} \leftarrow r_{10} \times r_{13}$

$r_{13} \leftarrow \text{MEM}(r_0 + @y)$   
 $r_{14} \leftarrow 2 \times r_{13}$   
 $r_{15} \leftarrow @x$

# Steps of the Simplifier

(3-operation window)

## LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

$r_{12} \leftarrow r_0 + r_{11}$

$r_{13} \leftarrow \mathbf{MEM}(r_{12})$

$r_{14} \leftarrow r_{10} \times r_{13}$

$r_{15} \leftarrow @x$

$r_{16} \leftarrow r_0 + r_{15}$

$r_{17} \leftarrow \mathbf{MEM}(r_{16})$

$r_{18} \leftarrow r_{17} - r_{14}$

$r_{19} \leftarrow @w$

$r_{20} \leftarrow r_0 + r_{19}$

$\mathbf{MEM}(r_{20}) \leftarrow r_{18}$

1<sup>st</sup> op rolling out of window

$r_{13} \leftarrow \mathbf{MEM}(r_0 + @y)$

$r_{14} \leftarrow 2 \times r_{13}$

$r_{15} \leftarrow @x$

$r_{14} \leftarrow 2 \times r_{13}$

$r_{15} \leftarrow @x$

$r_{16} \leftarrow r_0 + r_{15}$

# Steps of the Simplifier

(3-operation window)

## LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

$r_{12} \leftarrow r_0 + r_{11}$

$r_{13} \leftarrow \text{MEM}(r_{12})$

$r_{14} \leftarrow r_{10} \times r_{13}$

$r_{15} \leftarrow @x$

$r_{16} \leftarrow r_0 + r_{15}$

$r_{17} \leftarrow \text{MEM}(r_{16})$

$r_{18} \leftarrow r_{17} - r_{14}$

$r_{19} \leftarrow @w$

$r_{20} \leftarrow r_0 + r_{19}$

$\text{MEM}(r_{20}) \leftarrow r_{18}$

$r_{14} \leftarrow 2 \times r_{13}$

$r_{15} \leftarrow @x$

$r_{16} \leftarrow r_0 + r_{15}$



$r_{14} \leftarrow 2 \times r_{13}$

$r_{16} \leftarrow r_0 + @x$

$r_{17} \leftarrow \text{MEM}(r_{16})$

# Steps of the Simplifier

(3-operation window)

## LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

$r_{12} \leftarrow r_0 + r_{11}$

$r_{13} \leftarrow \text{MEM}(r_{12})$

$r_{14} \leftarrow r_{10} \times r_{13}$

$r_{15} \leftarrow @x$

$r_{16} \leftarrow r_0 + r_{15}$

$r_{17} \leftarrow \text{MEM}(r_{16})$

$r_{18} \leftarrow r_{17} - r_{14}$

$r_{19} \leftarrow @w$

$r_{20} \leftarrow r_0 + r_{19}$

$\text{MEM}(r_{20}) \leftarrow r_{18}$

$r_{14} \leftarrow 2 \times r_{13}$   
 $r_{16} \leftarrow r_0 + @x$   
 $r_{17} \leftarrow \text{MEM}(r_{16})$



$r_{14} \leftarrow 2 \times r_{13}$   
 $r_{17} \leftarrow \text{MEM}(r_0 + @x)$   
 $r_{18} \leftarrow r_{17} - r_{14}$

# Steps of the Simplifier

(3-operation window)

## LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

$r_{12} \leftarrow r_0 + r_{11}$

$r_{13} \leftarrow \text{MEM}(r_{12})$

$r_{14} \leftarrow r_{10} \times r_{13}$

$r_{15} \leftarrow @x$

$r_{16} \leftarrow r_0 + r_{15}$

$r_{17} \leftarrow \text{MEM}(r_{16})$

$r_{18} \leftarrow r_{17} - r_{14}$

$r_{19} \leftarrow @w$

$r_{20} \leftarrow r_0 + r_{19}$

$\text{MEM}(r_{20}) \leftarrow r_{18}$

$r_{14} \leftarrow 2 \times r_{13}$   
 $r_{17} \leftarrow \text{MEM}(r_0 + @x)$   
 $r_{18} \leftarrow r_{17} - r_{14}$



$r_{17} \leftarrow \text{MEM}(r_0 + @x)$   
 $r_{18} \leftarrow r_{17} - r_{14}$   
 $r_{19} \leftarrow @w$

# Steps of the Simplifier

(3-operation window)

## LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

$r_{12} \leftarrow r_0 + r_{11}$

$r_{13} \leftarrow \text{MEM}(r_{12})$

$r_{14} \leftarrow r_{10} \times r_{13}$

$r_{15} \leftarrow @x$

$r_{16} \leftarrow r_0 + r_{15}$

$r_{17} \leftarrow \text{MEM}(r_{16})$

$r_{18} \leftarrow r_{17} - r_{14}$

$r_{19} \leftarrow @W$

$r_{20} \leftarrow r_0 + r_{19}$

$\text{MEM}(r_{20}) \leftarrow r_{18}$

$r_{17} \leftarrow \text{MEM}(r_0 + @x)$

$r_{18} \leftarrow r_{17} - r_{14}$

$r_{19} \leftarrow @W$

$r_{18} \leftarrow r_{17} - r_{14}$

$r_{19} \leftarrow @W$

$r_{20} \leftarrow r_0 + r_{19}$

# Steps of the Simplifier

(3-operation window)

## LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

$r_{12} \leftarrow r_0 + r_{11}$

$r_{13} \leftarrow \text{MEM}(r_{12})$

$r_{14} \leftarrow r_{10} \times r_{13}$

$r_{15} \leftarrow @x$

$r_{16} \leftarrow r_0 + r_{15}$

$r_{17} \leftarrow \text{MEM}(r_{16})$

$r_{18} \leftarrow r_{17} - r_{14}$

$r_{19} \leftarrow @w$

$r_{20} \leftarrow r_0 + r_{19}$

$\text{MEM}(r_{20}) \leftarrow r_{18}$

$r_{18} \leftarrow r_{17} - r_{14}$

$r_{19} \leftarrow @w$

$r_{20} \leftarrow r_0 + r_{19}$



$r_{18} \leftarrow r_{17} - r_{14}$

$r_{20} \leftarrow r_0 + @w$

$\text{MEM}(r_{20}) \leftarrow r_{18}$

# Steps of the Simplifier (3-operation window)

## LLIR Code

```
r10 ← 2
r11 ← @y
r12 ← r0 + r11
r13 ← MEM(r12)
r14 ← r10 × r13
r15 ← @x
r16 ← r0 + r15
r17 ← MEM(r16)
r18 ← r17 - r14
r19 ← @w
r20 ← r0 + r19
MEM(r20) ← r18
```

$r_{18} \leftarrow r_{17} - r_{14}$   
 $r_{20} \leftarrow r_0 + @w$   
 $\text{MEM}(r_{20}) \leftarrow r_{18}$



$r_{18} \leftarrow r_{17} - r_{14}$   
 $\text{MEM}(r_0 + @w) \leftarrow r_{18}$

# Steps of the Simplifier

(3-operation window)

## LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

$r_{12} \leftarrow r_0 + r_{11}$

$r_{13} \leftarrow \mathbf{MEM}(r_{12})$

$r_{14} \leftarrow r_{10} \times r_{13}$

$r_{15} \leftarrow @x$

$r_{16} \leftarrow r_0 + r_{15}$

$r_{17} \leftarrow \mathbf{MEM}(r_{16})$

$r_{18} \leftarrow r_{17} - r_{14}$

$r_{19} \leftarrow @w$

$r_{20} \leftarrow r_0 + r_{19}$

$\mathbf{MEM}(r_{20}) \leftarrow r_{18}$

$r_{18} \leftarrow r_{17} - r_{14}$

$\mathbf{MEM}(r_0 + @w) \leftarrow r_{18}$

# Example

---

## LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

$r_{12} \leftarrow r_0 + r_{11}$

$r_{13} \leftarrow \text{MEM}(r_{12})$

$r_{14} \leftarrow r_{10} \times r_{13}$

$r_{15} \leftarrow @x$

$r_{16} \leftarrow r_0 + r_{15}$

$r_{17} \leftarrow \text{MEM}(r_{16})$

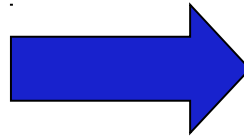
$r_{18} \leftarrow r_{17} - r_{14}$

$r_{19} \leftarrow @w$

$r_{20} \leftarrow r_0 + r_{19}$

$\text{MEM}(r_{20}) \leftarrow r_{18}$

Simplify



## LLIR Code

$r_{13} \leftarrow \text{MEM}(r_0 + @y)$

$r_{14} \leftarrow 2 \times r_{13}$

$r_{17} \leftarrow \text{MEM}(r_0 + @x)$

$r_{18} \leftarrow r_{17} - r_{14}$

$\text{MEM}(r_0 + @w) \leftarrow r_{18}$

# Making It All Work

---

## Details

- LLIR is largely machine independent (RTL)
- Target machine described as LLIR → ASM pattern
- Actual pattern matching
  - Use a hand-coded pattern matcher (gcc)
  - Turn patterns into grammar & use LR parser (VPO)
- Several important compilers use this technology
- It seems to produce good portable instruction selectors

Key strength appears to be late low-level optimization

# Next lecture

---

## Instruction selection

- Tree-based pattern matching