Instruction Selection: Peephole Matching
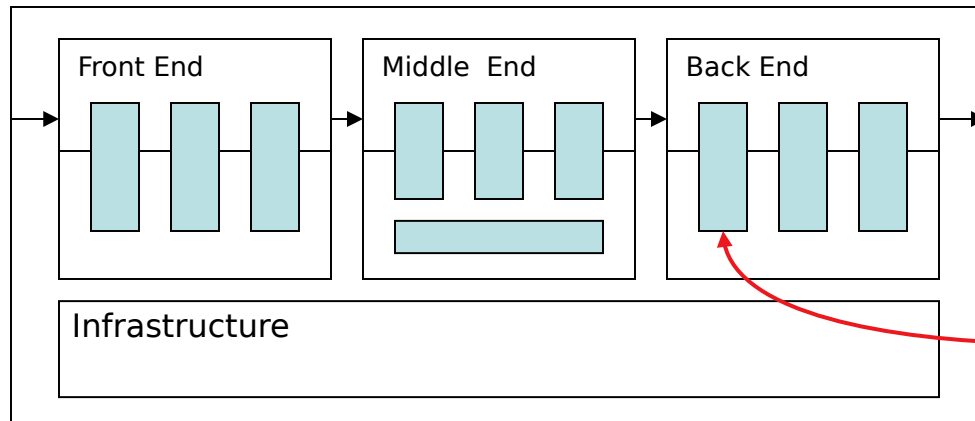
The Problem

Writing a compiler is a lot of work

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

Front End

Middle End

Back End

Infrastructure

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 in I**LOC**
- Obvious operation is `i2i rᵢ ⇒ rⱼ`
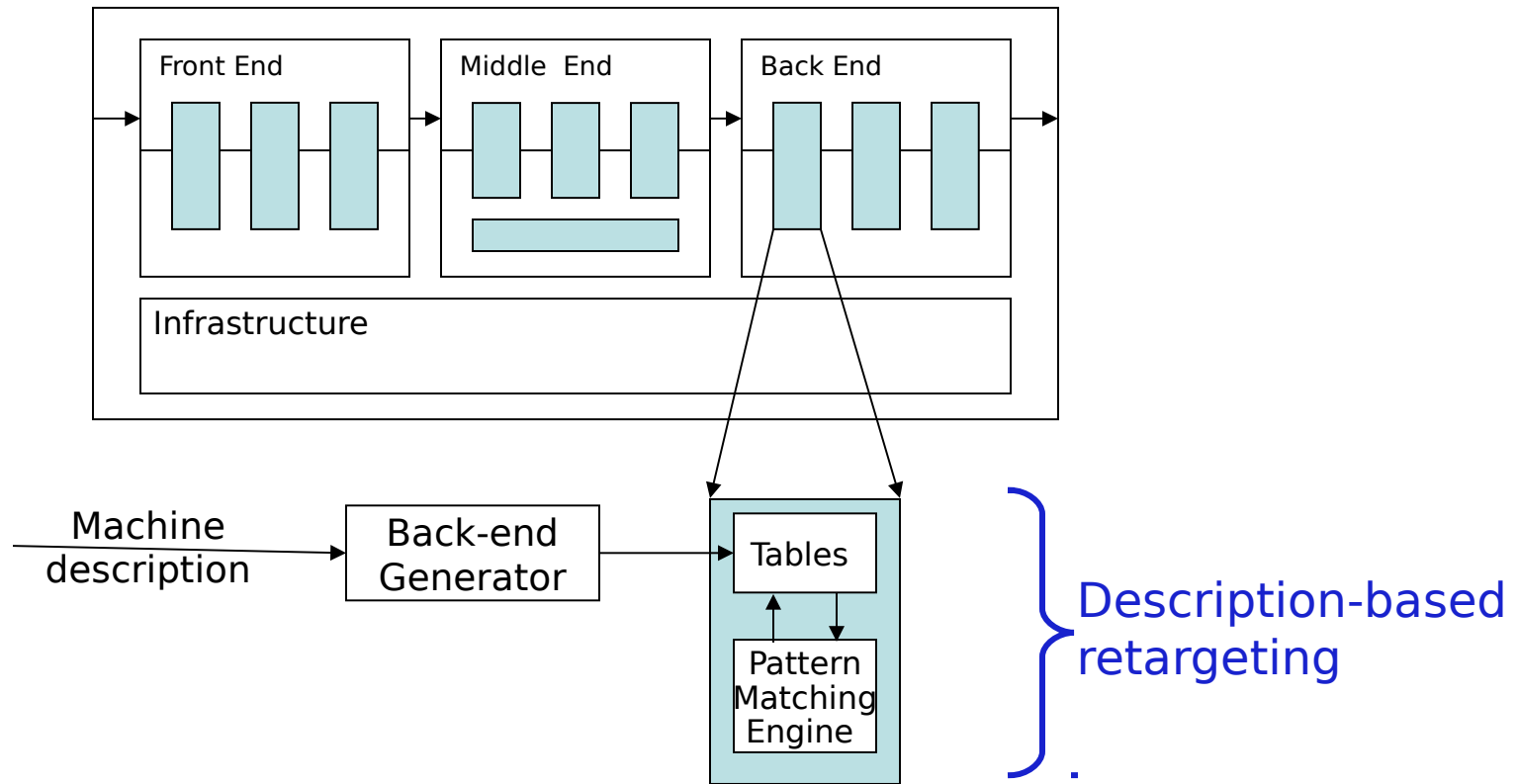- Many others exist

`addI rᵢ,0 ⇒ rⱼ`	`subI rᵢ,0 ⇒ rⱼ`	`lshiftI rᵢ,0 ⇒ rⱼ`
`multI rᵢ,1 ⇒ rⱼ`	`divI rᵢ,1 ⇒ rⱼ`	`rshiftI rᵢ,0 ⇒ rⱼ`
`orI rᵢ,0 ⇒ rⱼ`	`xorI rᵢ,0 ⇒ rⱼ`	**…** 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 I**LOC** is an overly-simplified case

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 code generator ran quickly
How good was the code?

Tree	Treewalk Code	Desired Code

x

IDENT
<**a**,ARP,4>   IDENT
<**b**,ARP,8>

Treewalk Code:

$loadI \quad 4 \quad \Rightarrow r_5$
$loadAO \quad r_0, r_5 \Rightarrow r_6$
$loadI \quad 8 \quad \Rightarrow r_7$
$loadAO \quad r_0, r_7 \Rightarrow r_8$
$mult \quad r_6, r_8 \Rightarrow r_9$

Desired Code:

$loadAI \quad r_0, 4 \Rightarrow r_5$
$loadAI \quad r_0, 8 \Rightarrow r_6$
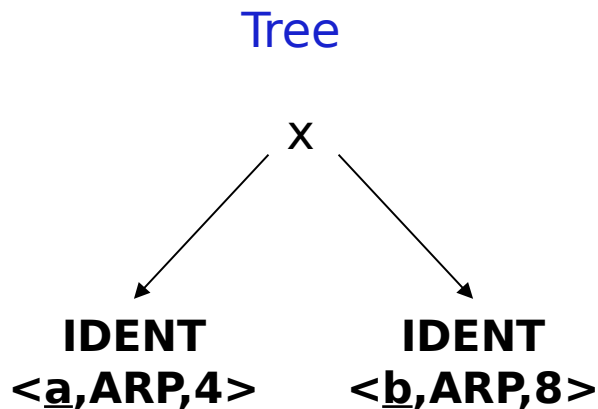$mult \quad r_5, r_6 \Rightarrow r_7$

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	Desired Code

x

IDENT
<a,ARP,4>

IDENT
<b,ARP,8>

loadI   4 $\Rightarrow r_5$
loadAO $r_0, r_5 \Rightarrow r_6$
loadI   8 $\Rightarrow r_7$
loadAO $r_0, r_7 \Rightarrow r_8$
mult $r_6, r_8 \Rightarrow r_9$

loadAI $r_0, 4 \Rightarrow r_5$
loadAI $r_0, 8 \Rightarrow r_6$
mult $r_5, r_6 \Rightarrow r_7$

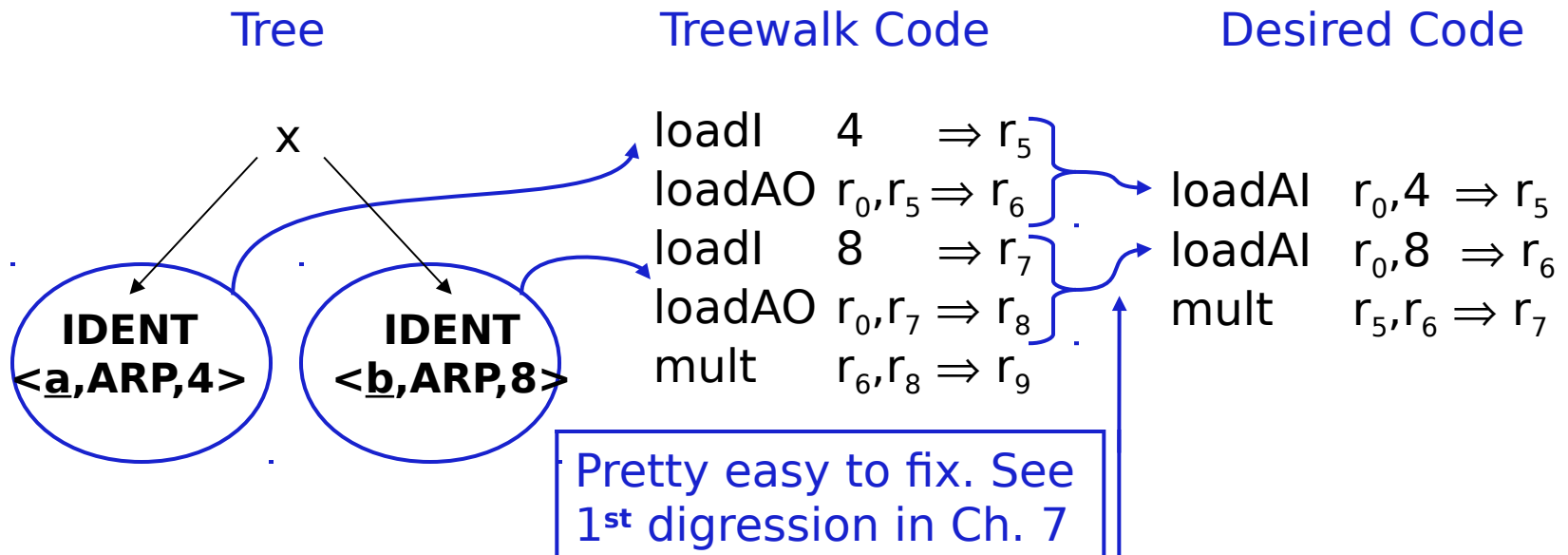Pretty easy to fix. See
1st digression in Ch. 7

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	Desired Code

$$x$$

IDENT
<u>a</u>,ARP,4>

NUMBER
<2>

loadI   4 $\Rightarrow r_5$
loadAO $r_0, r_5 \Rightarrow r_6$
loadI   2 $\Rightarrow r_7$
mult $r_6, r_7 \Rightarrow r_8$

loadAI $r_0, 4 \Rightarrow r_5$
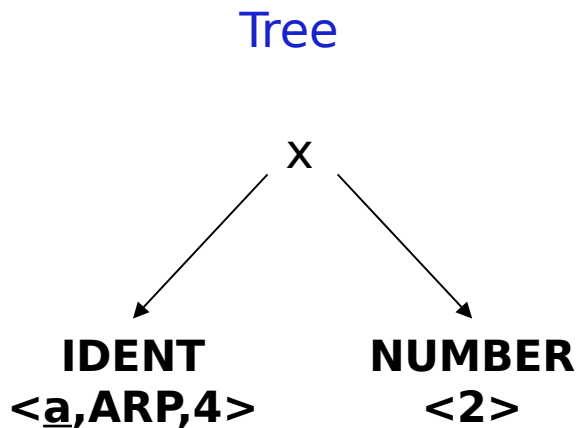multI $r_5, 2 \Rightarrow r_7$

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	Desired Code

loadI   4 $\Rightarrow r_5$
loadAO $r_0,r_5 \Rightarrow r_6$ loadAI $r_0,4 \Rightarrow r_5$
loadI   2 $\Rightarrow r_7$ multI $r_5,2 \Rightarrow r_7$
mult $r_6,r_7 \Rightarrow r_8$

x

IDENT
<u>a</u>,ARP,4>

NUMBER
<2>

Must combine these
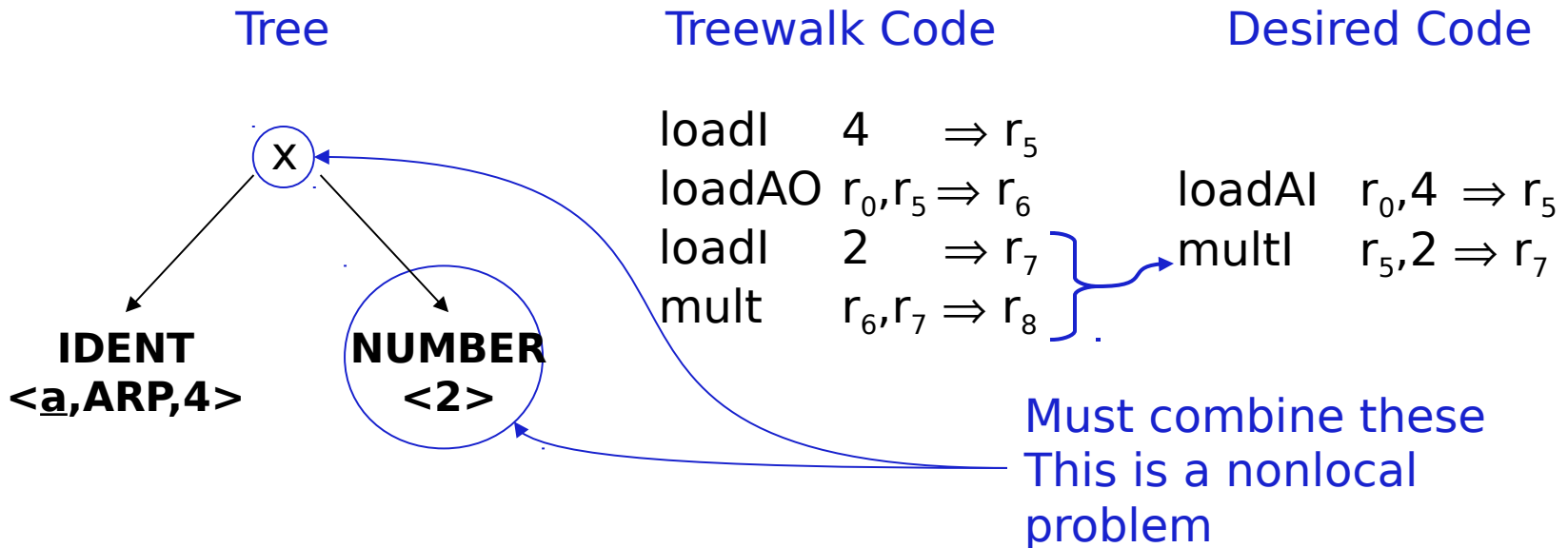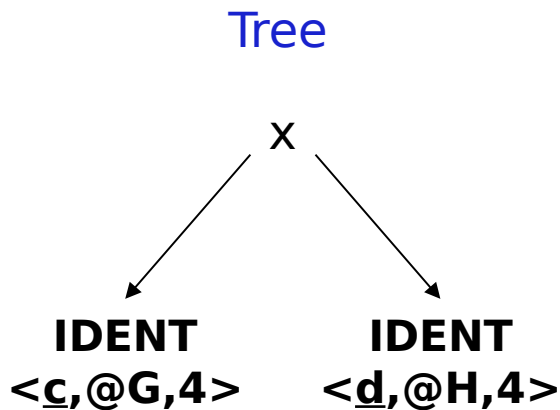This is a nonlocal
problem

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	Desired Code

x

IDENT   IDENT
<**c**,@**G**,4>   <**d**,@**H**,4>

loadI   @**G** $\Rightarrow r_5$
loadI   4 $\Rightarrow r_6$ loadI   4 $\Rightarrow r_5$
loadAO $r_5,r_6 \Rightarrow r_7$ loadAI $r_5,@\mathbf{G} \Rightarrow r_6$
loadI   @**H** $\Rightarrow r_7$ loadAI $r_5,@\mathbf{H} \Rightarrow r_7$
loadI   4 $\Rightarrow r_8$ mult r_6,r_7 $\Rightarrow r_8$
loadAO $r_8,r_9 \Rightarrow r_{10}$
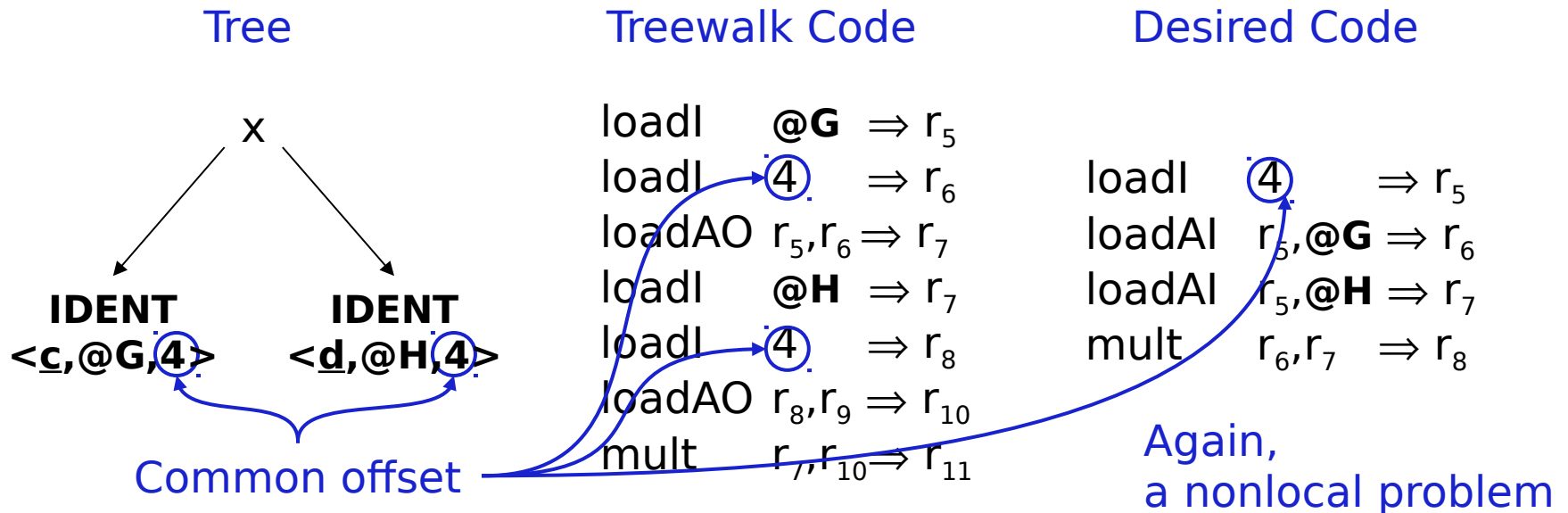mult $r_7,r_{10} \Rightarrow r_{11}$

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 ?

Tree	Treewalk Code	Desired Code

Tree

x

IDENT
<c,@G,4>

IDENT
<d,@H,4>

Common offset

Treewalk Code

loadI   @**G** $\Rightarrow r_5$
loadI   4 $\Rightarrow r_6$
loadAO $r_5,r_6 \Rightarrow r_7$
loadI   @**H** $\Rightarrow r_7$
loadI   4 $\Rightarrow r_8$
loadAO $r_8,r_9 \Rightarrow r_{10}$
mult $r_7,r_{10} \Rightarrow r_{11}$

Desired Code

loadI   4 $\Rightarrow r_5$
loadAI $r_5,@$**G** $\Rightarrow r_6$
loadAI $r_5,@$**H** $\Rightarrow r_7$
mult r_6,r_7 $\Rightarrow r_8$

Again,
a nonlocal problem

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
  - $\rightarrow$ Look at a small set of adjacent operations
  - $\rightarrow$ Move a "peephole" over code & search for improvement

- Classic example was store followed by load

Original code

Improved code

```
storeAI r₁    ⇒ r₀,8
loadAI   r₀,8 ⇒ r₁₅
```

```
storeAI r₁  ⇒ r₀,8
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

$$addI \quad r_2,0 \Rightarrow r_7$$
$$mult \quad r_4,r_7 \Rightarrow r_{10}$$

Improved code

$$mult \quad r_4,r_2 \Rightarrow r_{10}$$

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

$$jumpI \rightarrow L_{10}$$
$$L_{10}: jumpI \rightarrow L_{11}$$

Improved code

$$L_{10}: jumpI \rightarrow L_{11}$$

Peephole Matching

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

Modern peephole instruction selectors   *(Davidson)*
- Break problem into three tasks

IR → | Expander
IR→LLIR | → **LLIR** → | Simplifier
LLIR→LLIR | → **LLIR** → | Matcher
LLIR→ASM | → **ASM** →

- 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   *(e.g., setting cc)*
- Significant, albeit constant, expansion of size

IR → | Expander **IR→LLIR** | **LLIR** → | Simplifier **LLIR→LLIR** | **LLIR** → | Matcher **LLIR→ASM** | **ASM** →

*RTL = Register transfer language

Peephole Matching

Simplifier

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

| IR → | Expander
IR→LLIR | **LLIR** → | Simplifier
LLIR→LLIR | **LLIR** → | Matcher
LLIR→ASM | **ASM** → |

- 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)*
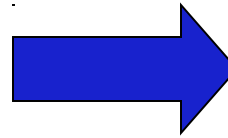- Generates the assembly code output

IR → | Expander
IR→LLIR | **LLIR** → | Simplifier
LLIR→LLIR | **LLIR** → | Matcher
LLIR→ASM | **ASM** →

Example

Original IR Code

OP	Arg_1	Arg_2	Result
mult	2	Y	t_1
sub	x	t_1	w

Expand →

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

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

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

$r_{19} \leftarrow @w$

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

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

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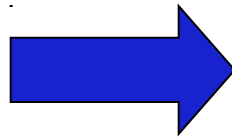$\text{MEM}(r_0 + @w) \leftarrow r_{18}$

Example

LLIR Code

$r_{13} \leftarrow \text{M\textbf{EM}}(r_0 + @y)$

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

$r_{17} \leftarrow \text{M\textbf{EM}}(r_0 + @x)$

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

$\text{M\textbf{EM}}(r_0 + @w) \leftarrow r_{18}$

Match

ILOC Code

loadAI $r_0, @y \Rightarrow r_{13}$

multI $2 \times r_{13} \Rightarrow r_{14}$

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

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

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

$r_{19} \leftarrow @w$

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

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

Steps of the Simplifier   (*3-operation window*)

LLIR Code

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

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

Steps of the Simplifier (*3-operation window*)

LLIR Code

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

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

$\longrightarrow$

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

Steps of the Simplifier   (*3-operation window*)

LLIR Code

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

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

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

Steps of the Simplifier　　(*3-operation window*)

LLIR Code

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

1st op rolling out of window

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

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

Steps of the Simplifier   (*3-operation window*)

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

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

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

$r_{19} \leftarrow @w$

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

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

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

$r_{14} \leftarrow 2 \times r_{13}$
$r_{15} \leftarrow @x$
$r_{16} \leftarrow r_0 + r_{15}$

Steps of the Simplifier   (*3-operation window*)

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

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

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

$r_{19} \leftarrow @w$

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

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

$r_{14} \leftarrow 2 \times r_{13}$
$r_{15} \leftarrow @x$
$r_{16} \leftarrow r_0 + r_{15}$

$\longrightarrow$

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

Steps of the Simplifier   (*3-operation window*)

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

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

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

$r_{19} \leftarrow @w$

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

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

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

$\longrightarrow$

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

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

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

$r_{19} \leftarrow @w$

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

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

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

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

$r_{13} \leftarrow M\textsc{em}(r_{12})$

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

$r_{15} \leftarrow @x$

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

$r_{17} \leftarrow M\textsc{em}(r_{16})$

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

$r_{19} \leftarrow @w$

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

$M\textsc{em}(r_{20}) \leftarrow r_{18}$

$r_{17} \leftarrow M\textsc{em}(r_0 + @x)$
$r_{18} \leftarrow r_{17} - r_{14}$
$r_{19} \leftarrow @w$

$r_{18} \leftarrow r_{17} - r_{14}$
$r_{19} \leftarrow @w$
$r_{20} \leftarrow r_0 + r_{19}$

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

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

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

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

$r_{19} \leftarrow @w$

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

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

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

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

Steps of the Simplifier   (*3-operation window*)

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

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

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

$r_{19} \leftarrow @w$

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

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

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

Example

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

$r_{16} \leftarrow r_0 + r_{15}$
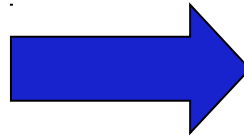
$r_{17} \leftarrow \text{MEM}(r_{16})$

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

$r_{19} \leftarrow @w$

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

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

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 $\rightarrow$ ASM pattern
- Actual pattern matching
    - $\rightarrow$ Use a hand-coded pattern matcher   (gcc)
    - $\rightarrow$ 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