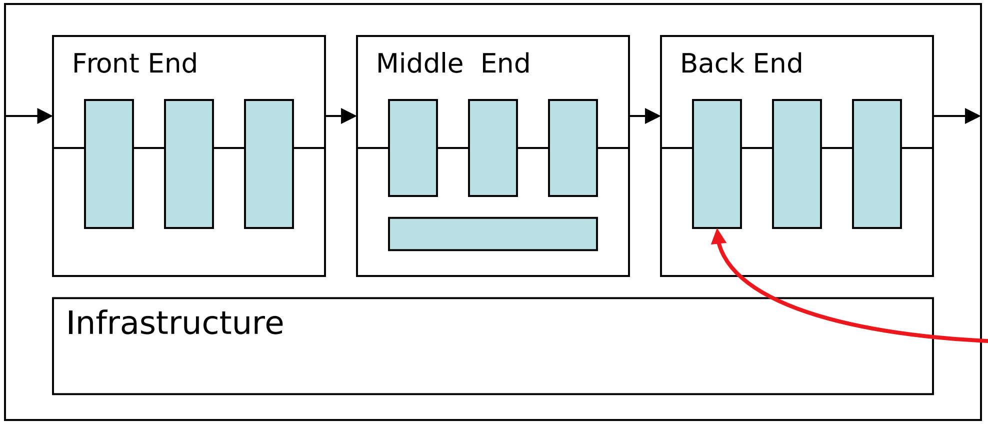
Instruction Selection: Peephole Matching

The Problem

Writing a compiler is a lot of work

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



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 $r_i \Rightarrow r_j$
- Many others exist

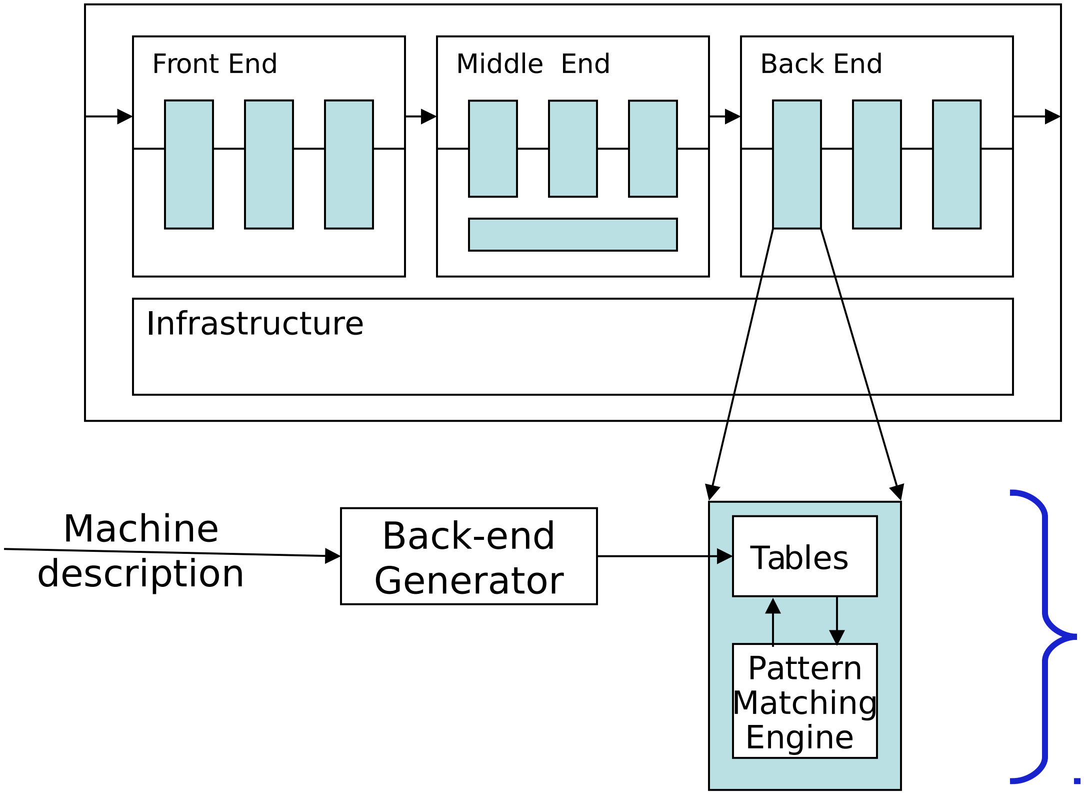
addI $r_i, 0 \Rightarrow r_j$	subI $r_i, 0 \Rightarrow r_j$	lshiftI $r_i, 0 \Rightarrow r_j$
multI $r_i, 1 \Rightarrow r_j$	divI $r_i, 1 \Rightarrow r_j$	rshiftI $r_i, 0 \Rightarrow r_j$
orI $r_i, 0 \Rightarrow r_j$	xorI $r_i, 0 \Rightarrow r_j$	**… and others …**

- 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



Front End

Middle End

Back End

Infrastructure

Machine description

Back-end Generator

Tables

Pattern Matching Engine

Description-based retargeting

Machine description should also help with scheduling & allocation

The Big Picture

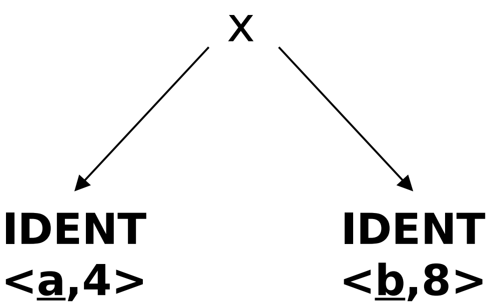
Need pattern matching techniques

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

Our treewalk (visitor) code generator ran quickly

How good was the code?

Tree



Treewalk Code

loadI 4 ⇒ r_5
loadA r_5 ⇒ r_6
loadI 8 ⇒ r_7
loadA r_7 ⇒ r_8
mult r_6, r_8 ⇒ r_9

Desired Code

loadAI 4 ⇒ r_5
loadAI 8 ⇒ r_6
mult r_5, r_6 ⇒ r_7

The Big Picture

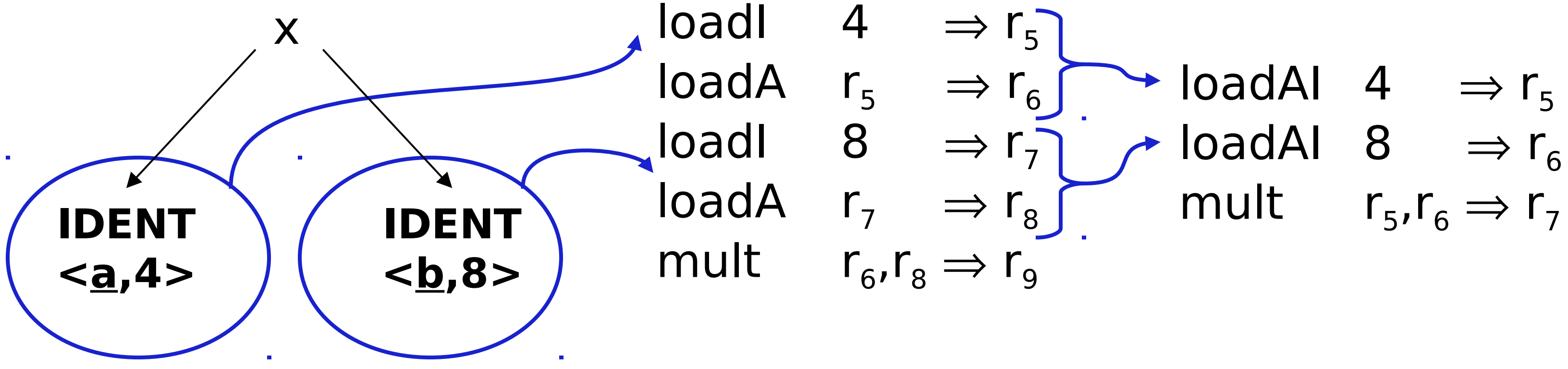
Need pattern matching techniques

- 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,4>

IDENT
<b,8>

loadI 4 ⇒ r_5
loadA r_5 ⇒ r_6
loadI 8 ⇒ r_7
loadA r_7 ⇒ r_8
mult r_6,r_8 ⇒ r_9

loadAI 4 ⇒ r_5
loadAI 8 ⇒ r_6
mult r_5,r_6 ⇒ r_7

The Big Picture

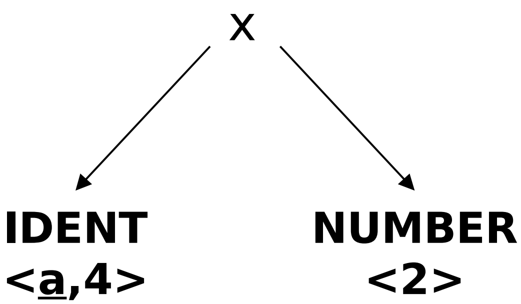
Need pattern matching techniques

- 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

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

Desired Code

loadA 4 $\Rightarrow r_5$
multI $r_5, 2 \Rightarrow r_7$

The Big Picture

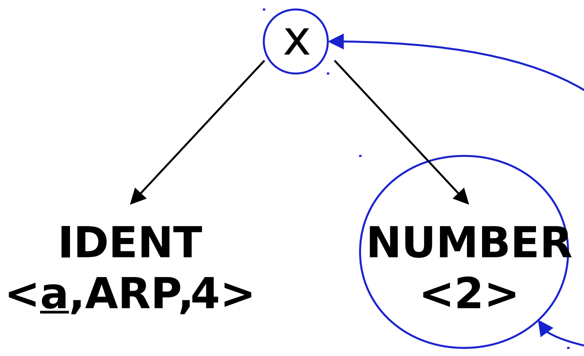
Need pattern matching techniques

- 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

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

Desired Code

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

Must combine these
This is a nonlocal problem

The Big Picture

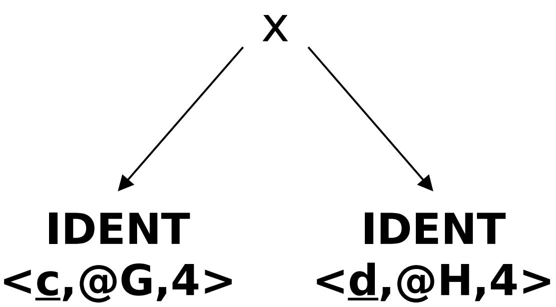
Need pattern matching techniques

- 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

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

Desired Code

loadI 4 ⇒ r_5
loadAI r_5,**@G** ⇒ r_6
loadAI r_5,**@H** ⇒ r_7
mult r_6,r_7 ⇒ r_8

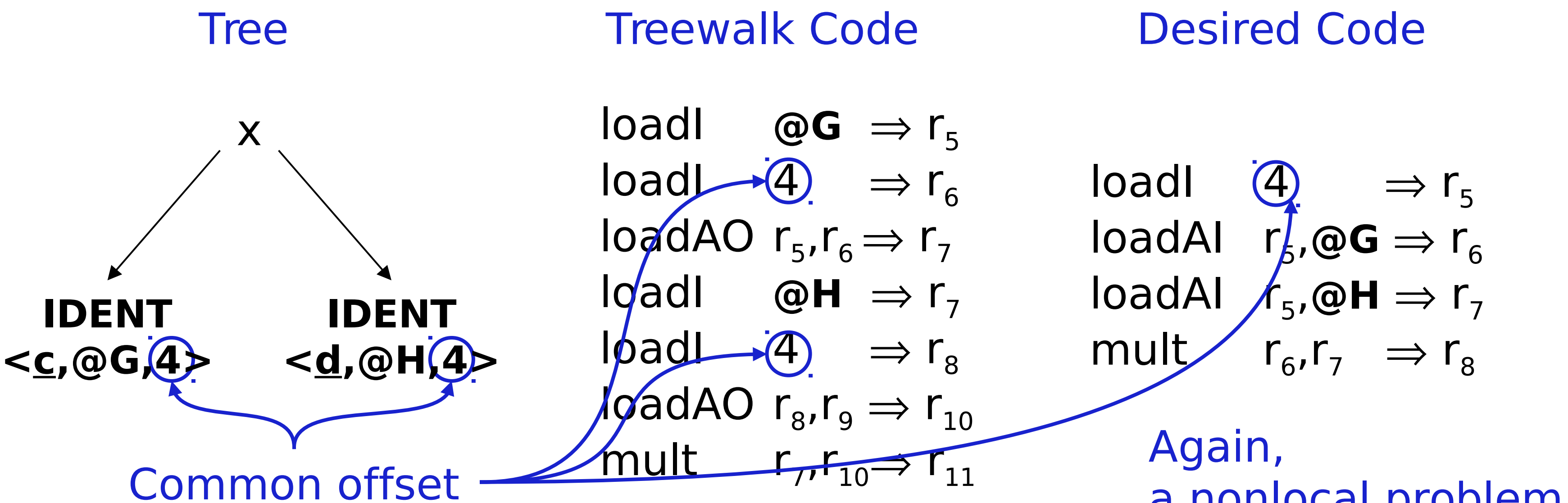
The Big Picture

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

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
  - → 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	Improved code
storeAI $r_1 \Rightarrow 8$	storeAI $r_1 \Rightarrow 8$
loadAI $8 \Rightarrow r_{15}$	i2i $r_1 \Rightarrow r_{15}$

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

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	Improved code
jumpI → L_{10} L_{10}: jumpI → L_{11}	L_{10}: jumpI → L_{11}

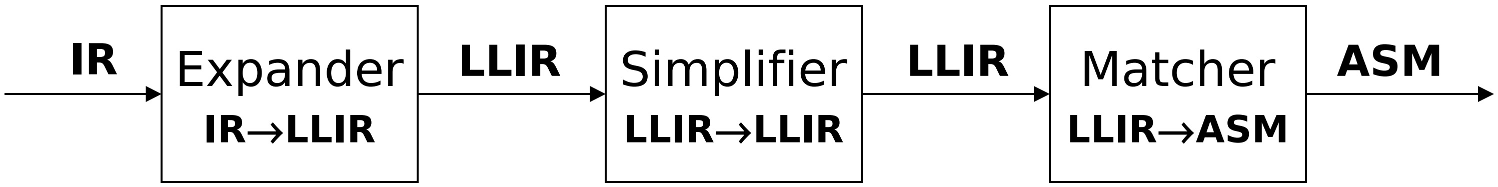
Peephole Matching

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



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
- 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

Example

Original IR Code

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

Expand

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

$r_{13} \leftarrow \textsc{Mem}(r_{12})$

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

$r_{15} \leftarrow @x$

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

$r_{17} \leftarrow \textsc{Mem}(r_{16})$

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

$r_{19} \leftarrow @w$

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

$\textsc{Mem}(r_{20}) \leftarrow r_{18}$

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	Match	ILOC Code
$r_{13} \leftarrow \text{MEM}(r_0 + @y)$		loadAI $r_0, @y \Rightarrow r_{13}$
$r_{14} \leftarrow 2 \times r_{13}$		multI $2 \times r_{13} \Rightarrow r_{14}$
$r_{17} \leftarrow \text{MEM}(r_0 + @x)$		loadAI $r_0, @x \Rightarrow r_{17}$
$r_{18} \leftarrow r_{17} - r_{14}$		sub $r_{17} - r_{14} \Rightarrow r_{18}$
$\text{MEM}(r_0 + @w) \leftarrow r_{18}$		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

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

$r_{13} \leftarrow \text{M}\textbf{EM}(r_{12})$

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

$r_{15} \leftarrow @x$

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

$r_{17} \leftarrow \text{M}\textbf{EM}(r_{16})$

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

$r_{19} \leftarrow @w$

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

$\text{M}\textbf{EM}(r_{20}) \leftarrow r_{18}$

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

LLIR Code

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

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

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

LLIR Code

$$\left.\begin{array}{l} r_{10} \leftarrow 2 \\ r_{11} \leftarrow @y \\ r_{12} \leftarrow r_0 + r_{11} \end{array}\right\} \quad \boxed{\begin{array}{l} r_{10} \leftarrow 2 \\ r_{11} \leftarrow @y \\ r_{12} \leftarrow r_0 + r_{11} \end{array}} \Rightarrow \boxed{\begin{array}{l} r_{10} \leftarrow 2 \\ r_{12} \leftarrow r_0 + @y \\ r_{13} \leftarrow \text{M}\textbf{EM}(r_{12}) \end{array}}$$

$r_{13} \leftarrow \text{M}\textbf{EM}(r_{12})$

$r_{14} \leftarrow r_{10} \text{ x } r_{13}$

$r_{15} \leftarrow @x$

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

$r_{17} \leftarrow \text{M}\textbf{EM}(r_{16})$

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

$r_{19} \leftarrow @w$

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

$\text{M}\textbf{EM}(r_{20}) \leftarrow r_{18}$

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

LLIR Code

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

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

$\Rightarrow$

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

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

LLIR Code

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

1st op rolling out of window

$r_{10} \leftarrow 2$
$r_{13} \leftarrow \textsc{Mem}(r_0 + @y)$
$r_{14} \leftarrow r_{10} \text{ x } r_{13}$

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

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

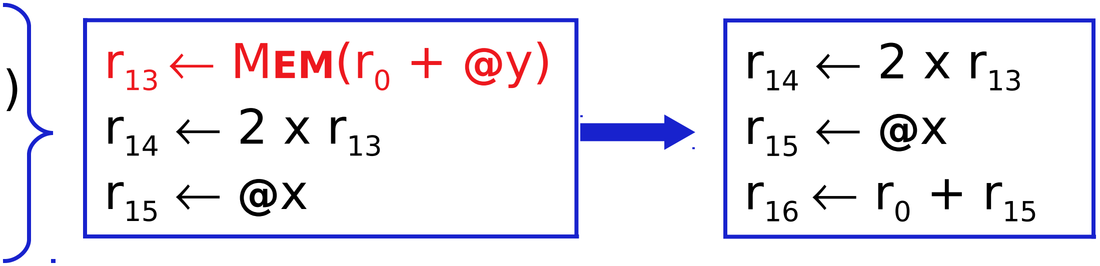
$r_{17} \leftarrow \text{MEM}(r_{16})$

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

$r_{19} \leftarrow @w$

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

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



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

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

$r_{15} \leftarrow @x$

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

$r_{15} \leftarrow @x$

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

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

$r_{13} \leftarrow \textsc{Mem}(r_{12})$

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

$r_{15} \leftarrow @x$

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

$r_{17} \leftarrow \textsc{Mem}(r_{16})$

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

$r_{19} \leftarrow @w$

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

$\textsc{Mem}(r_{20}) \leftarrow r_{18}$

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

$r_{15} \leftarrow @x$

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

$\Rightarrow$

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

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

$r_{17} \leftarrow \textsc{Mem}(r_{16})$

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

$r_{13} \leftarrow \textsc{Mem}(r_{12})$

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

$r_{15} \leftarrow @x$

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

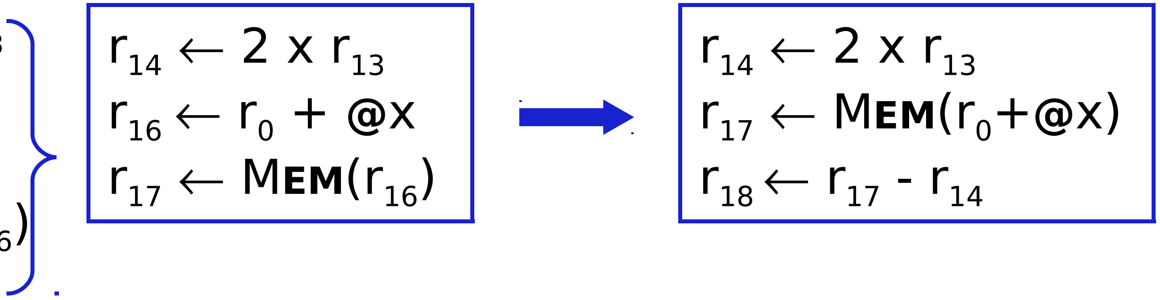
$r_{17} \leftarrow \textsc{Mem}(r_{16})$

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

$r_{19} \leftarrow @w$

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

$\textsc{Mem}(r_{20}) \leftarrow r_{18}$



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

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

$r_{17} \leftarrow \textsc{Mem}(r_{16})$

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

$r_{17} \leftarrow \textsc{Mem}(r_0+@x)$

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

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

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

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

$r_{19} \leftarrow @w$

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

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

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

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

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

$\Rightarrow$

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

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

$r_{19} \leftarrow @w$

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

$r_{13} \leftarrow \textsc{Mem}(r_{12})$

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

$r_{15} \leftarrow @x$

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

$r_{17} \leftarrow \textsc{Mem}(r_{16})$

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

$r_{19} \leftarrow @w$

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

$\textsc{Mem}(r_{20}) \leftarrow r_{18}$

$r_{17} \leftarrow \textsc{Mem}(r_0 + @x)$

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

$r_{19} \leftarrow @w$

$\Rightarrow$

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

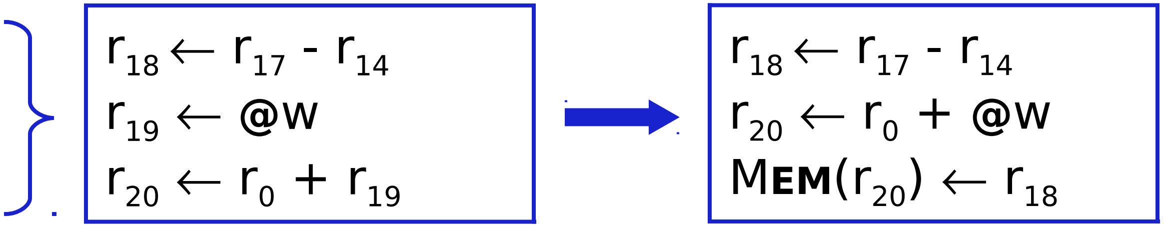
$r_{19} \leftarrow @w$

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

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

LLIR Code

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



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

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

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

$r_{14} \leftarrow r_{10} \text{ x } r_{13}$

$r_{15} \leftarrow @x$

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

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

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

$r_{19} \leftarrow @w$

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

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

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

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

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

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

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

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

$r_{13} \leftarrow \text{M}\textbf{EM}(r_{12})$

$r_{14} \leftarrow r_{10} \text{ x } r_{13}$

$r_{15} \leftarrow @x$

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

$r_{17} \leftarrow \text{M}\textbf{EM}(r_{16})$

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

$r_{19} \leftarrow @w$

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

$\text{M}\textbf{EM}(r_{20}) \leftarrow r_{18}$

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

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

Example

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

$r_{13} \leftarrow \textsc{Mem}(r_{12})$

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

$r_{15} \leftarrow @x$

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

$r_{17} \leftarrow \textsc{Mem}(r_{16})$

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

$r_{19} \leftarrow @w$

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

$\textsc{Mem}(r_{20}) \leftarrow r_{18}$

Simplify

LLIR Code

$r_{13} \leftarrow \textsc{Mem}(r_0 + @y)$

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

$r_{17} \leftarrow \textsc{Mem}(r_0 + @x)$

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

$\textsc{Mem}(r_0 + @w) \leftarrow r_{18}$

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 lecure

Instruction selection

- Tree-based pattern matching