Compiler Design

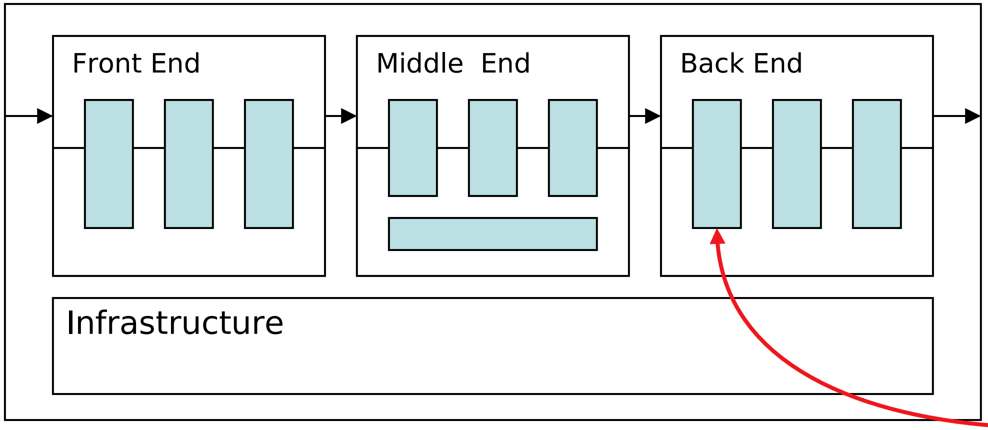
Lecture 18:
Instruction Selection via Peephole Matching

Christophe Dubach
Winter 2024

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

- Process of mapping IR into assembly code
  - → Assumes a fixed storage mapping & code shape
  - → Combining operations, using address modes

Instruction scheduling

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

Register allocation

- Process of 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 have many ways to do anything

Consider register-to-register copy

- Obvious operation is MOVE $r_j \leftarrow r_i$
- Many others exist

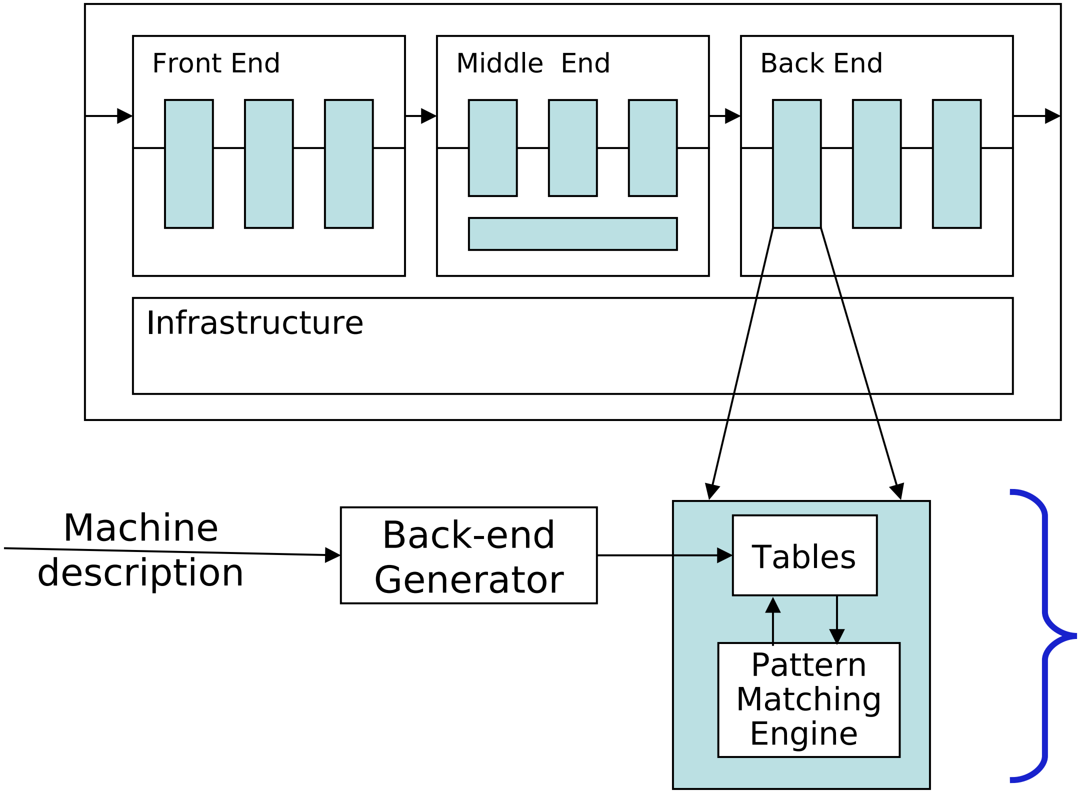
ADDI $r_j \leftarrow r_i, 0$	OR $r_j \leftarrow r_i, r_0$	SLL $r_j \leftarrow r_i, 0$
ADD $r_j \leftarrow r_i, r_0$	MULI $r_j \leftarrow r_i, 1$	RSHIFTI $r_j \leftarrow r_i, 0$
ORI $r_j \leftarrow r_i, 0$	XORI $r_j \leftarrow r_i, 0$	**… and others …**

- 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



Description-based retargeting

Machine description should also help with scheduling & allocation

The Big Picture

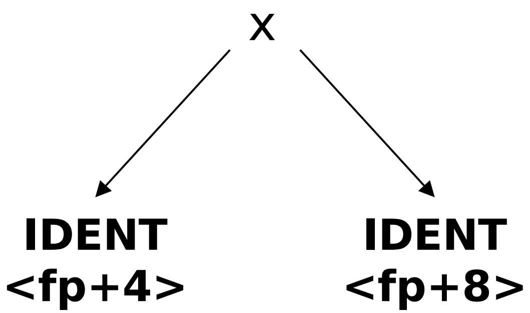
Need pattern matching techniques

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

Our treewalk (visitor) code generator ran quickly

How good was the code?

Tree



The Big Picture

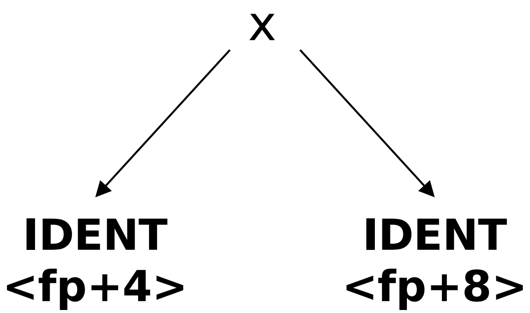
Need pattern matching techniques

- 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

```
MOVE r1 ← 4
ADD  r2 ← r1 , $fp
LW   r3 ← 0(r2)
MOVE r4 ← 8
ADD  r5 ← r4 , $fp
LW   r6 ← 0(r5)
MUL  r7 ← r3, r6
```

The Big Picture

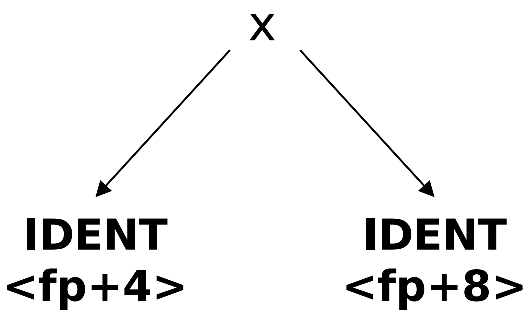
Need pattern matching techniques

- 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

```
MOVE r1 ← 4
ADD  r2  ← r1 , $fp
LW   r3 ← 0(r2)
MOVE r4 ← 8
ADD  r5  ← r4 , $fp
LW   r6 ← 0(r5)
MUL  r7 ← r3, r6
```

Desired Code

```
LW   r3 ← 4($fp)
LW   r6 ← 8($fp)
MUL  r7 ← r3, r6
```

The Big Picture

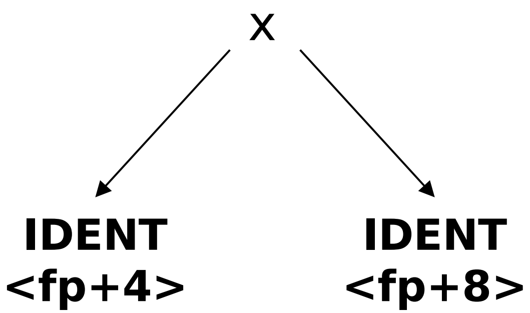
Need pattern matching techniques

- 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

```
MOVE r1 ← 4
ADD  r2  ← r1 , $fp
LW   r3 ← 0(r2)
MOVE r4 ← 8
ADD  r5  ← r4 , $fp
LW   r6 ← 0(r5)
MUL  r7 ← r3, r6
```

Desired Code

```
LW   r3 ← 4($fp)
LW   r6 ← 8($fp)
MUL  r7 ← r3, r6
```

Okay wasn't too hard, could do it in the code generator

The Big Picture

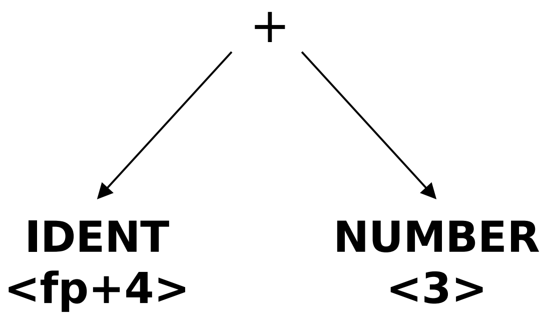
Need pattern matching techniques

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

Our treewalk (visitor) code generator ran quickly

How good was the code?

Tree



The Big Picture

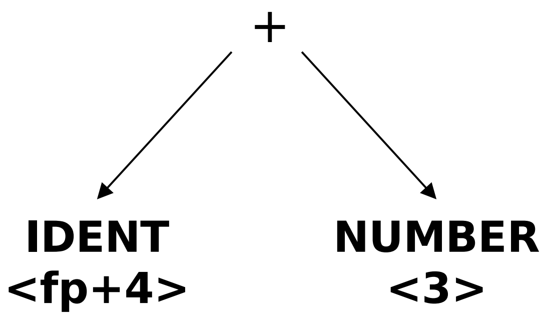
Need pattern matching techniques

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

Our treewalk (visitor) code generator ran quickly

How good was the code?

Tree



+

IDENT
<fp+4>

NUMBER
<3>

Treewalk Code

MOVE r_1 ← 4
ADD r_2 ← r_1 , \$fp
LW r_3 ← 0(r_2)
MOVE r_4 ← 3
ADD r_5 ← r_3 , r_4

The Big Picture

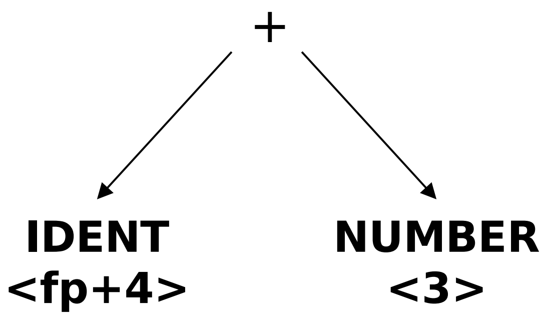
Need pattern matching techniques

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

Our treewalk (visitor) code generator ran quickly

How good was the code?

Tree



+

IDENT <fp+4>

NUMBER <3>

Treewalk Code

```
MOVE r1 ← 4
ADD  r2 ← r1 , $fp
LW   r3 ← 0(r2)
MOVE r4 ← 3
ADD  r5 ← r3 , r4
```

Desired Code

```
LW   r3 ← 4($fp)
ADDI r5 ← r3 , 3
```

The Big Picture

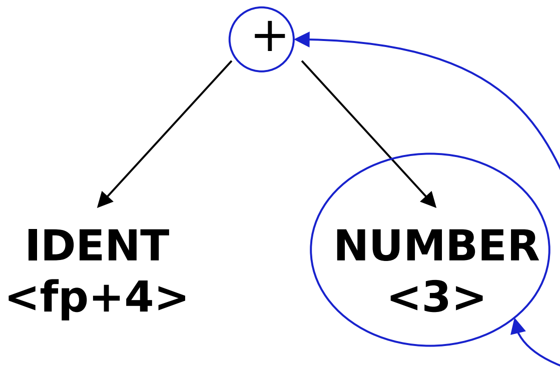
Need pattern matching techniques

- 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

```
MOVE r1 ← 4
ADD  r2  ← r1 , $fp
LW   r3 ← 0(r2)
MOVE r4 ← 3
ADD  r5  ← r3 , r4
```

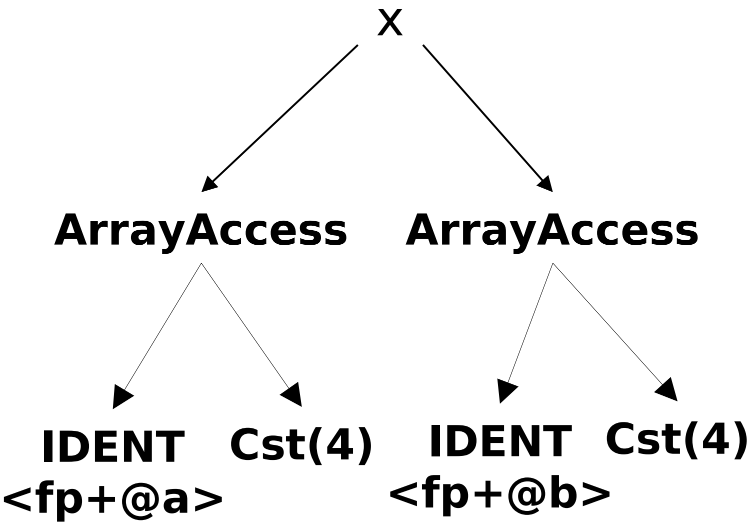
Desired Code

```
LW   r3 ← 4($fp)
ADDI r5  ← r3 , 3
```

Must combine these.
This is a nonlocal problem

The Big Picture

Tree



x

ArrayAccess

ArrayAccess

IDENT
<fp+@a>

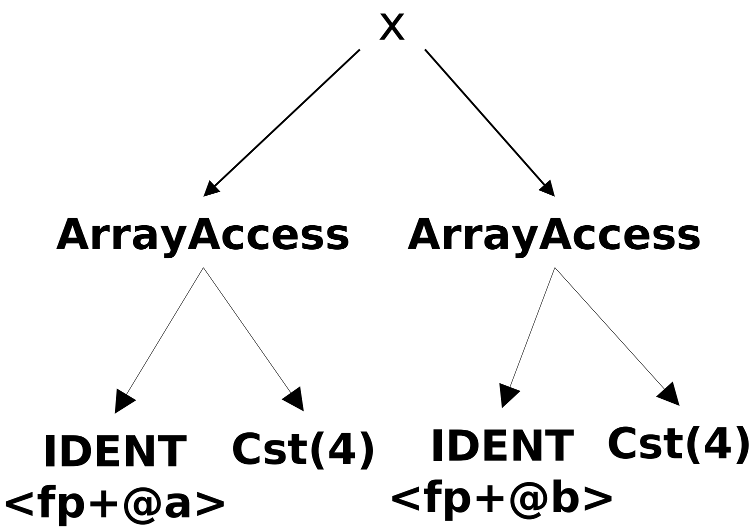
Cst(4)

IDENT
<fp+@b>

Cst(4)

The Big Picture

Tree



x

ArrayAccess

ArrayAccess

IDENT <fp+@a>

Cst(4)

IDENT <fp+@b>

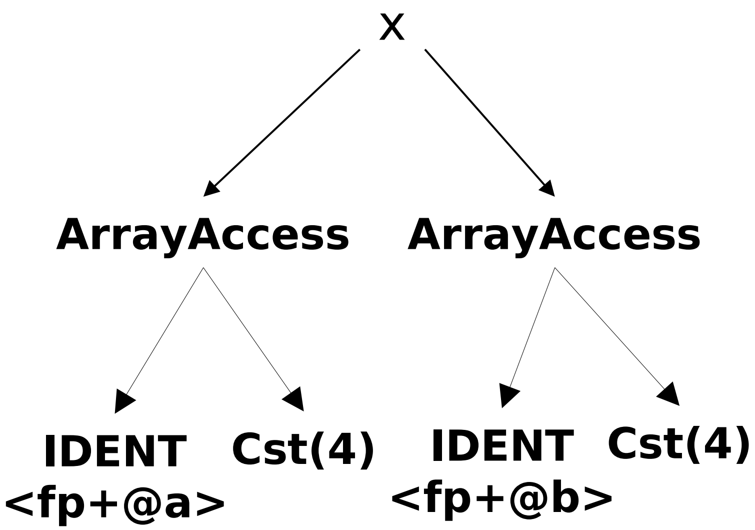
Cst(4)

Treewalk Code

```
MOVE r2  ← @a
ADD  r3  ← $fp, r2
MOVE r4  ← 4
ADD  r5  ← r3, r4
LW   r6  ← 0(r5)
MOVE r8  ← @b
ADD  r9  ← $fp, r8
MOVE r10 ← 4
ADD  r11 ← r9, r10
LW   r12 ← 0(r11)
MUL  r13 ← r6, r12
```

The Big Picture

Tree



Treewalk Code

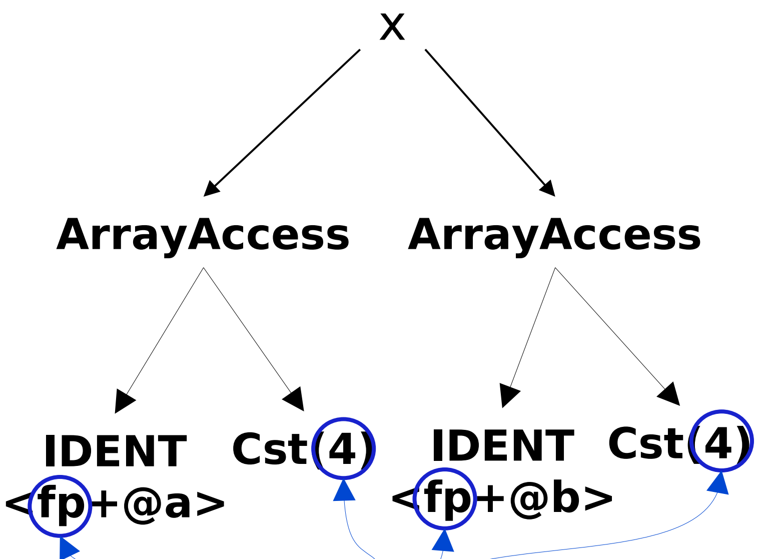
```
MOVE r2  ← @a
ADD  r3  ← $fp, r2
MOVE r4  ← 4
ADD  r5  ← r3, r4
LW   r6  ← 0(r5)
MOVE r8  ← @b
ADD  r9  ← $fp, r8
MOVE r10 ← 4
ADD  r11 ← r9, r10
LW   r12 ← 0(r11)
MUL  r13 ← r6, r12
```

Desired Code

```
ADDI r3  ← $fp, 4
LW   r6  ← @a(r3)
LW   r12 ← @b(r3)
MUL  r13 ← r6, r12
```

The Big Picture

Tree



Common offsets

Treewalk Code

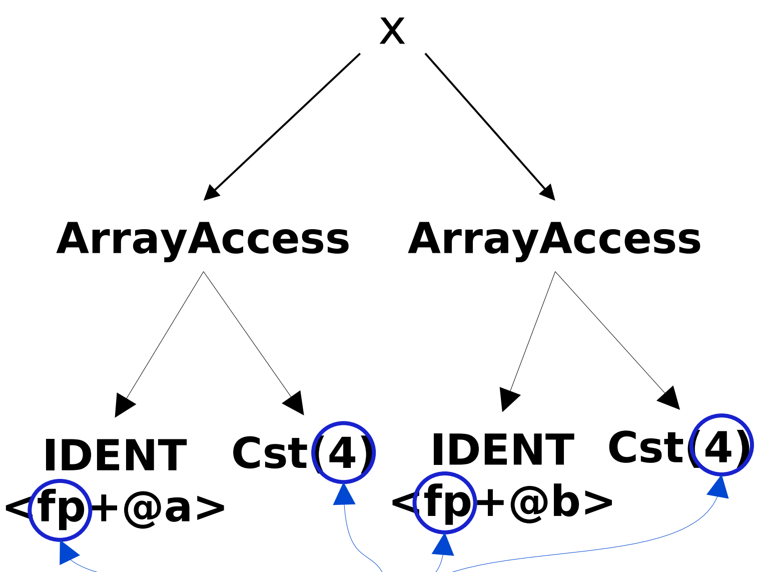
```
MOVE r2  ← @a
ADD  r3  ← $fp, r2
MOVE r4  ← 4
ADD  r5  ← r3, r4
LW   r6  ← 0(r5)
MOVE r8  ← @b
ADD  r9  ← $fp, r8
MOVE r10 ← 4
ADD  r11 ← r9, r10
LW   r12 ← 0(r11)
MUL  r13 ← r6, r12
```

Desired Code

```
ADDI r3  ← $fp, 4
LW   r6  ← @a(r3)
LW   r12 ← @b(r3)
MUL  r13 ← r6, r12
```

The Big Picture

Tree



Common offsets

Treewalk Code

```
MOVE  r2  ← @a
ADD   r3  ← $fp, r2
MOVE  r4  ← 4
ADD   r5  ← r3, r4
LW    r6  ← 0(r5)
MOVE  r8  ← @b
ADD   r9  ← $fp, r8
MOVE  r10 ← 4
ADD   r11 ← r9, r10
LW    r12 ← 0(r11)
MUL   r13 ← r6, r12
```

Desired Code

```
ADDI  r3  ← $fp, 4
LW    r6  ← @a(r3)
LW    r12 ← @b(r3)
MUL   r13 ← r6, r12
```

($fp+@a)+4=($fp+4)+@a

($fp+@b)+4=($fp+4)+@b

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. Today we will look at matchers on Linear IR.

Peephole Matching

- Basic idea:
  Compiler can discover local-*ish* improvements
  - → Look at a small set of adjacent operations
  - → Move a “peephole” over code & search for improvement

Marco Verch , CC BY 2.0 Source: https://foto.wuestenigel.com/a-mans-hand-opens-the-peephole-on-the-door/

Peephole Matching

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

Original code

SW $r_1 \rightarrow 0(r_2)$
LW $r_{15} \leftarrow 0(r_2)$

Improved code

SW $r_1 \rightarrow 0(r_2)$
MOVE $r_{15} \leftarrow r_1$

Peephole Matching

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

Original code	Improved code
ADDI $r_7 \leftarrow r_2, 0$ MUL $r_{10} \leftarrow r_4, r_7$	MUL $r_{10} \leftarrow r_4, r_2$

Peephole Matching

- Basic idea: Compiler can discover local improvements
  - → Look at a small set of adjacent operations
  - → Move a “peephole” over code & search for improvement
- Classic examples:
  - → store followed by load
  - → simple algebraic identities
  - → jump to a jump

Original code	Improved code
Jump L_{10} L_{10}: Jump L_{11}	L_{10}: Jump L_{11}

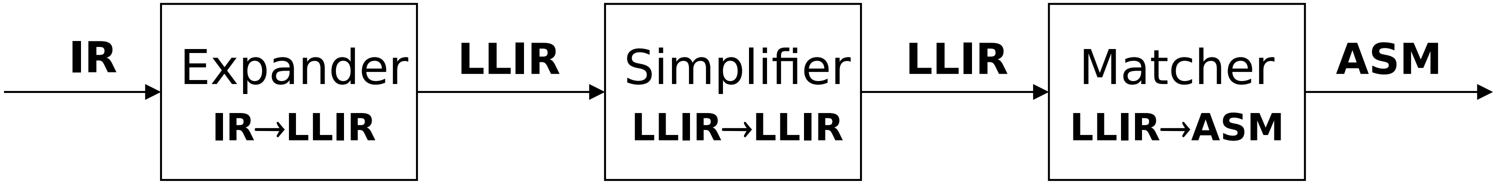
Peephole Matching

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

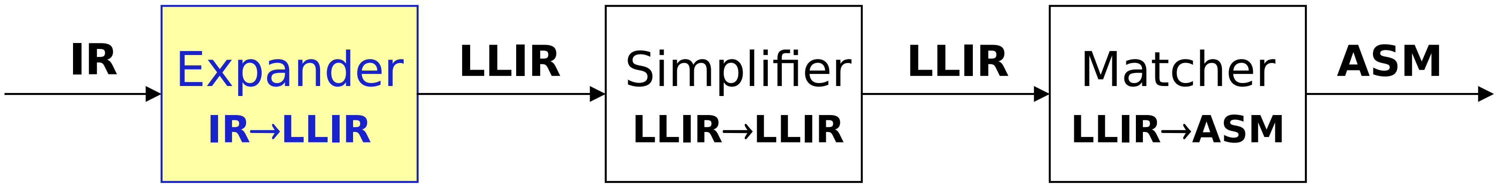


- Apply symbolic interpretation & simplification systematically

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

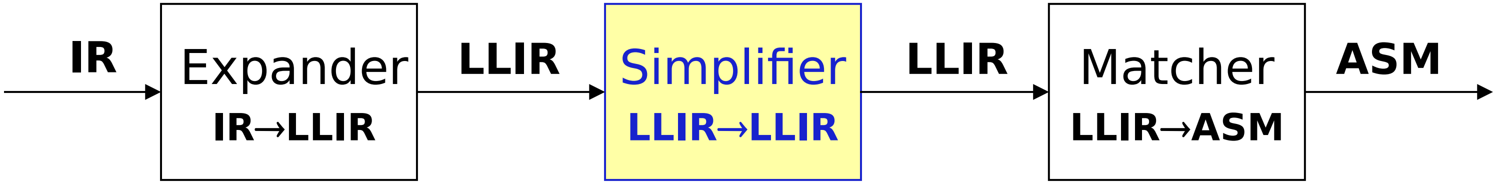


*RTL = Register Transfer Language

Peephole Matching

Simplifier

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

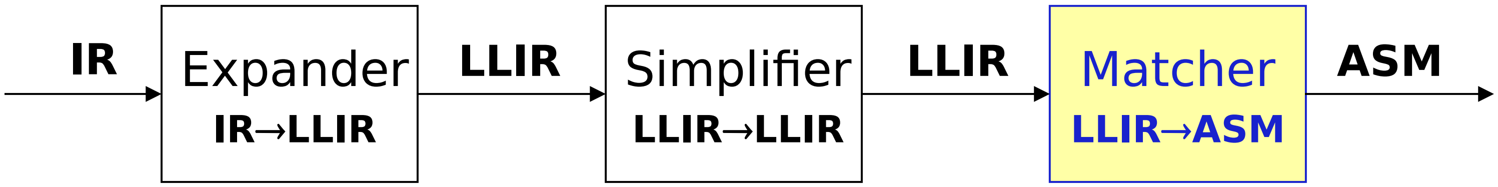


- This is the heart of the peephole system
  - → Benefit of peephole optimization shows up in this step

Peephole Matching

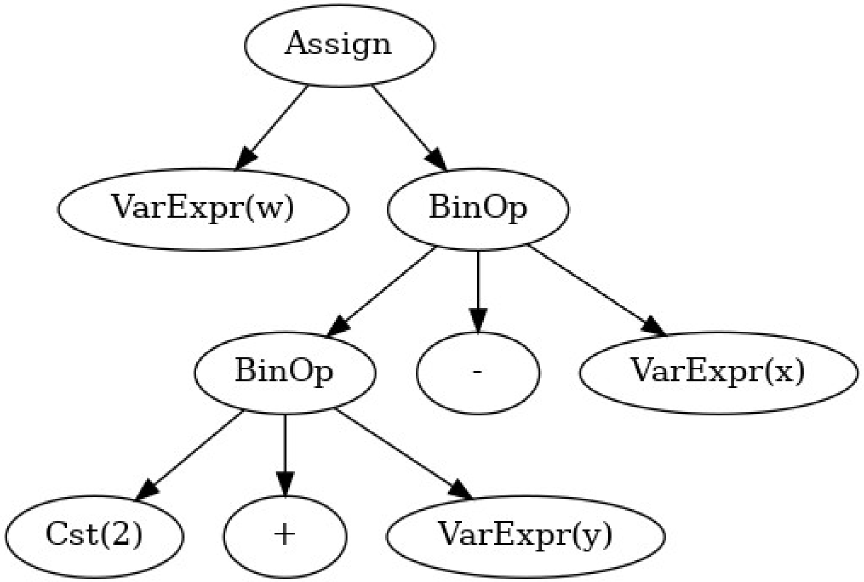
Matcher

- 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 condition code)*
- Generates the assembly code output



Example: Expander

Original IR



Expand

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

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

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

$r_{19} \leftarrow @w$

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

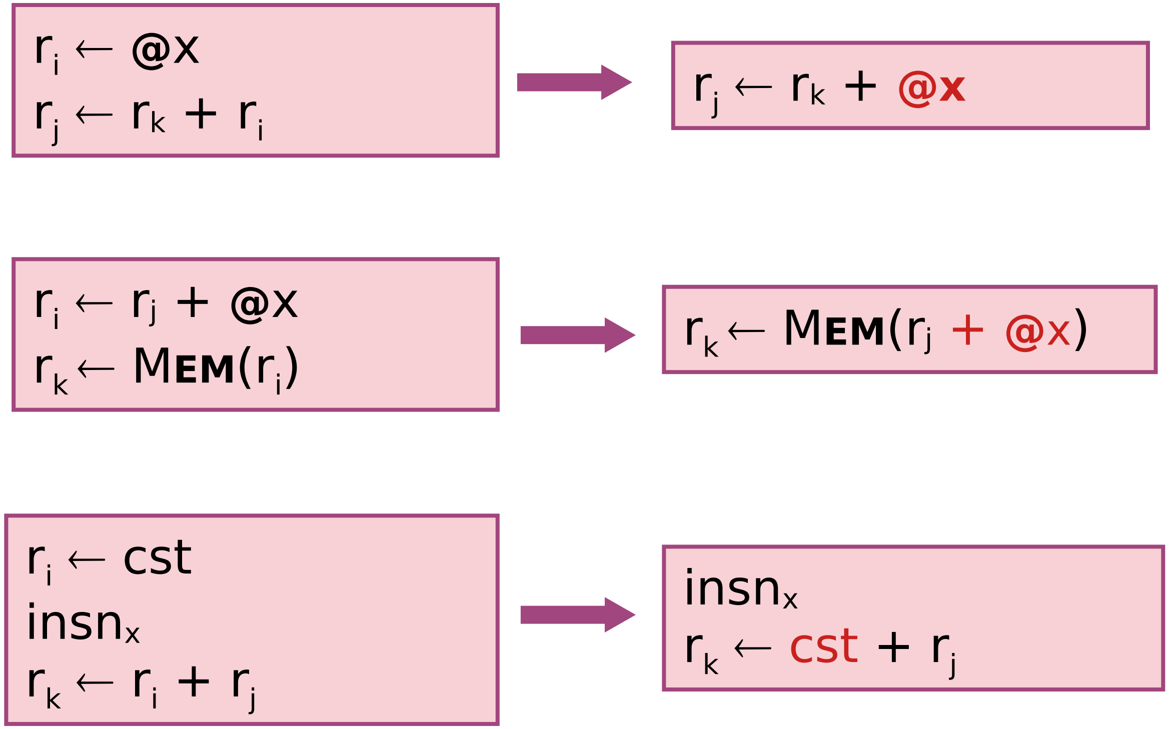
$\mathrm{MEM}(r_{20}) \leftarrow r_{18}$

assumes x,y,w stack allocated

Each register is **single use**

@x, @y, @w = offsets from fp

Example: Simplification rules



$r_i \leftarrow @x$
$r_j \leftarrow r_k + r_i$

$\Rightarrow$

$r_j \leftarrow r_k + @x$

$r_i \leftarrow r_j + @x$
$r_k \leftarrow \mathrm{MEM}(r_i)$

$\Rightarrow$

$r_k \leftarrow \mathrm{MEM}(r_j + @x)$

$r_i \leftarrow cst$
$insn_x$
$r_k \leftarrow r_i + r_j$

$\Rightarrow$

$insn_x$
$r_k \leftarrow cst + r_j$

Example: Simplifier

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

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

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

$r_{19} \leftarrow @w$

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

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

Simplify

LLIR Code

$r_{13} \leftarrow \mathrm{MEM}(fp+ @y)$

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

$r_{17} \leftarrow \mathrm{MEM}(fp + @x)$

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

$\mathrm{MEM}(fp + @w) \leftarrow r_{18}$

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

LLIR Code

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

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

LLIR Code

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

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

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

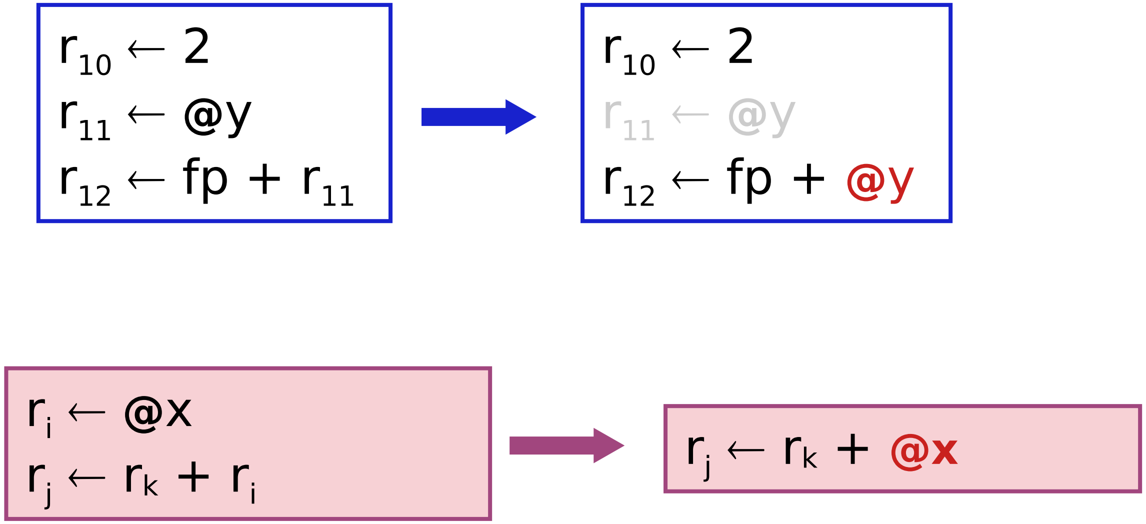
$r_{17} \leftarrow \text{Mem}(r_{16})$

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

$r_{19} \leftarrow @w$

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

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



$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

$r_{12} \leftarrow fp + @y$

$r_i \leftarrow @x$

$r_j \leftarrow r_k + r_i$

$r_j \leftarrow r_k + @x$

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

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

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

$r_{19} \leftarrow @w$

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

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

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

→

$r_{10} \leftarrow 2$

$r_{12} \leftarrow fp + @y$

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

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

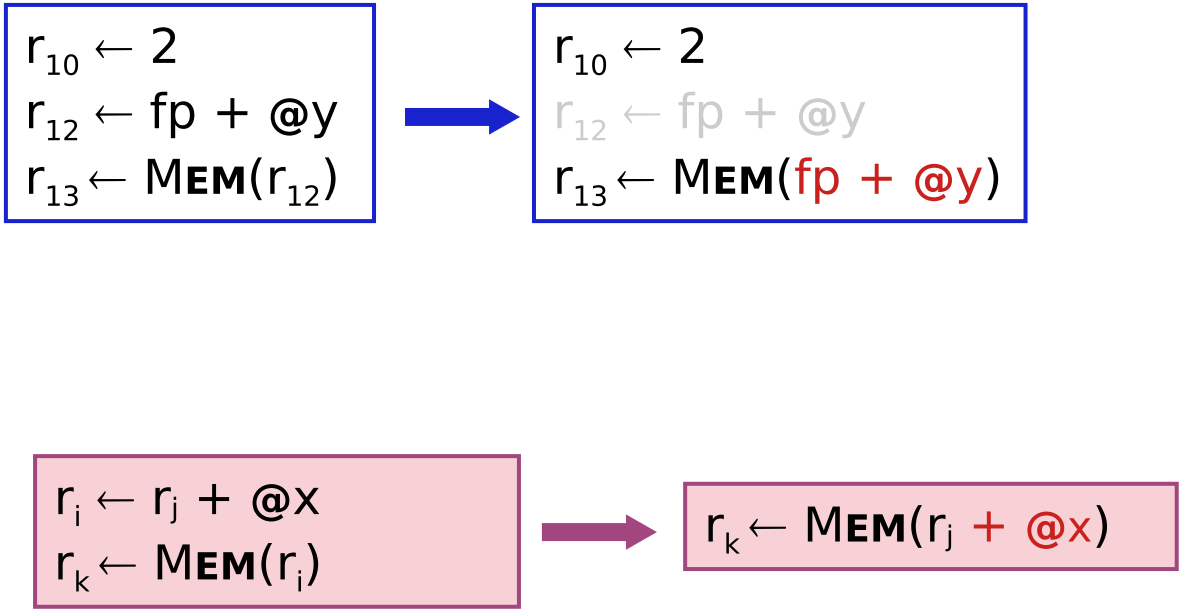
$r_{17} \leftarrow \text{MEM}(r_{16})$

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

$r_{19} \leftarrow @w$

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

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



$r_{10} \leftarrow 2$

$r_{12} \leftarrow fp + @y$

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

$r_{10} \leftarrow 2$

$r_{12} \leftarrow fp + @y$

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

$r_i \leftarrow r_j + @x$

$r_k \leftarrow \text{MEM}(r_i)$

$r_k \leftarrow \text{MEM}(r_j + @x)$

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

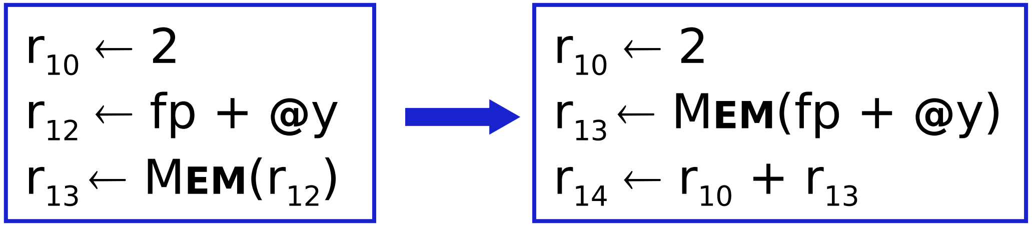
$r_{17} \leftarrow \text{MEM}(r_{16})$

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

$r_{19} \leftarrow @w$

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

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



$r_{10} \leftarrow 2$

$r_{12} \leftarrow fp + @y$

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

$r_{10} \leftarrow 2$

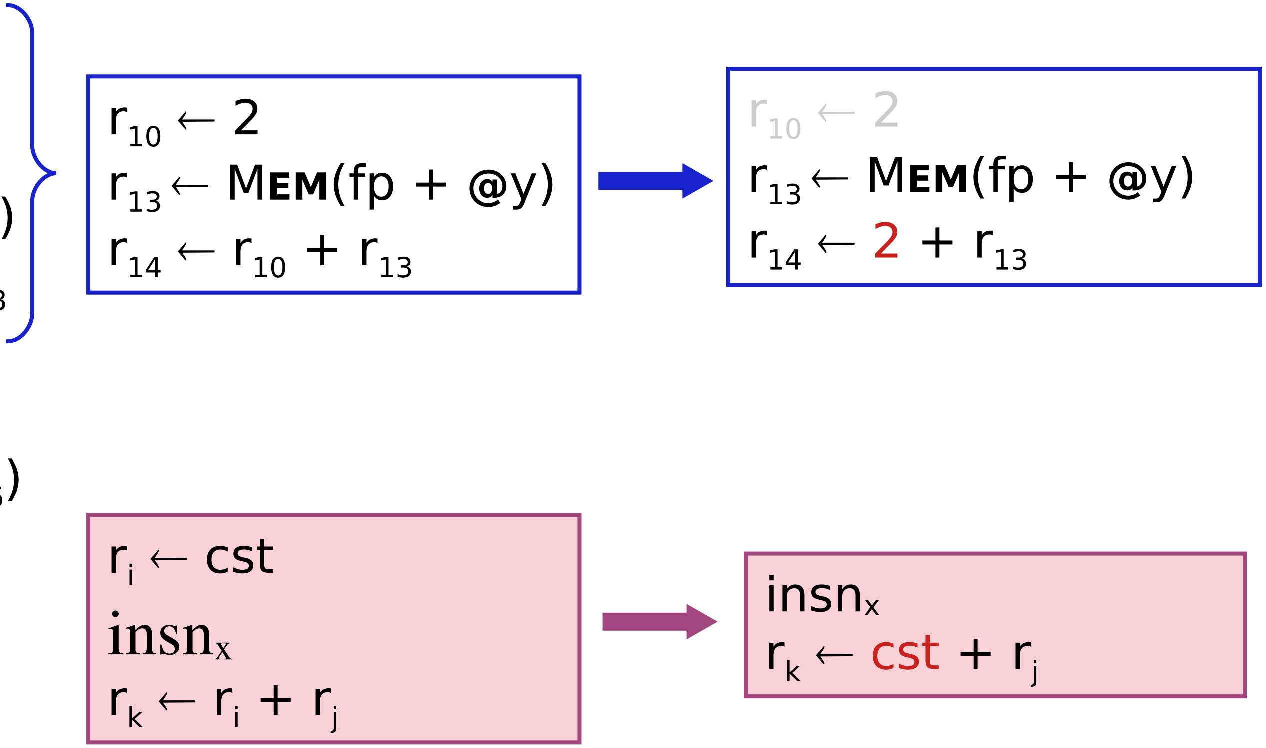
$r_{13} \leftarrow \text{MEM}(fp + @y)$

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

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

LLIR Code

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



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

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

$r_i \leftarrow cst$
$insn_x$
$r_k \leftarrow r_i + r_j$

$insn_x$
$r_k \leftarrow cst + r_j$

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

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

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

$r_{19} \leftarrow @w$

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

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

$r_{10} \leftarrow 2$

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

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

$\Rightarrow$

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

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

$r_{15} \leftarrow @x$

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

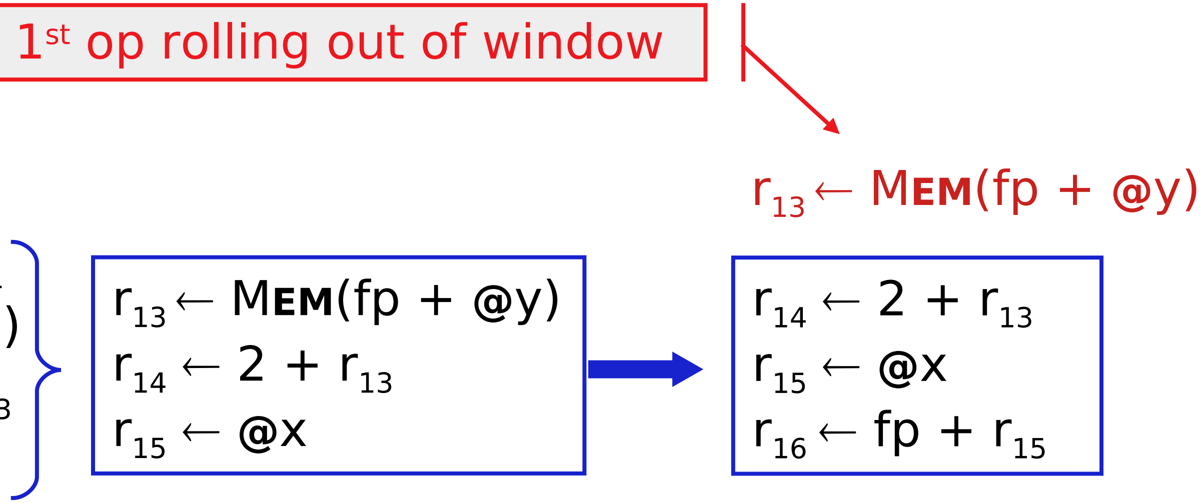
$r_{17} \leftarrow \textsc{Mem}(r_{16})$

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

$r_{19} \leftarrow @w$

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

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



1st op rolling out of window

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

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

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

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

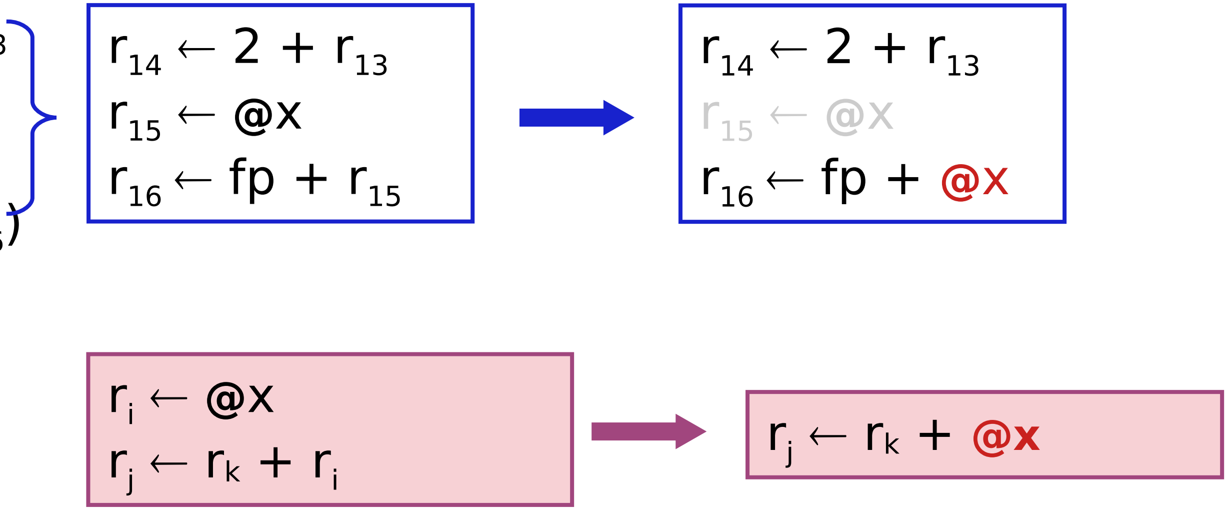
$r_{17} \leftarrow \text{MEM}(r_{16})$

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

$r_{19} \leftarrow @w$

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

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



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

$r_{15} \leftarrow @x$

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

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

$r_{15} \leftarrow @x$

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

$r_i \leftarrow @x$

$r_j \leftarrow r_k + r_i$

$r_j \leftarrow r_k + @x$

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

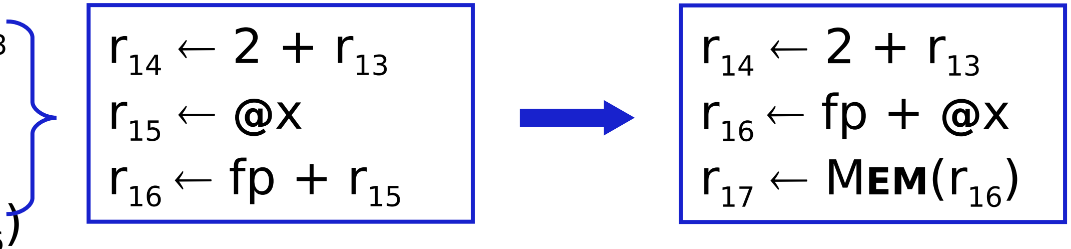
$r_{17} \leftarrow \mathrm{MEM}(r_{16})$

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

$r_{19} \leftarrow @w$

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

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



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

$r_{15} \leftarrow @x$

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

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

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

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

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

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

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

$r_{19} \leftarrow @w$

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

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

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

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

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

$\Rightarrow$

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

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

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

$r_i \leftarrow r_j + @x$

$r_k \leftarrow \text{MEM}(r_i)$

$\Rightarrow$

$r_k \leftarrow \text{MEM}(r_j + @x)$

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

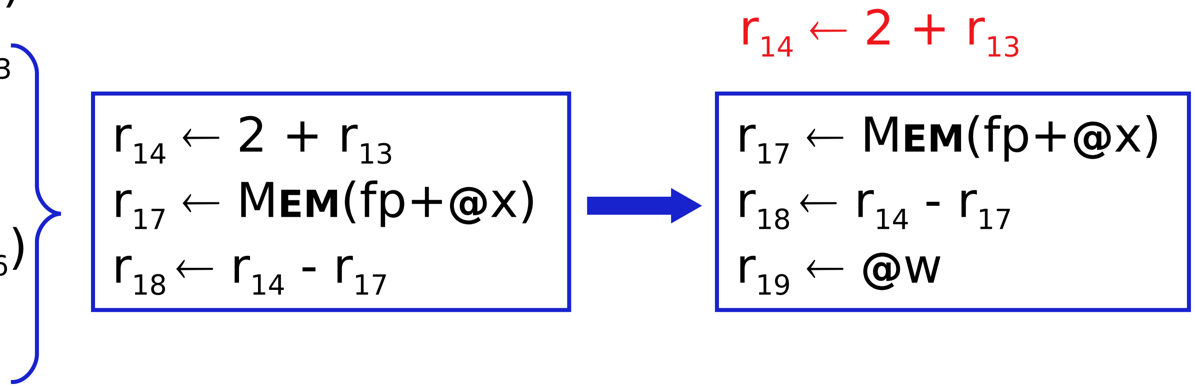
$r_{17} \leftarrow \mathrm{MEM}(r_{16})$

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

$r_{19} \leftarrow @w$

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

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



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

$r_{17} \leftarrow \mathrm{MEM}(fp+@x)$

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

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

$r_{17} \leftarrow \mathrm{MEM}(fp+@x)$

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

$r_{19} \leftarrow @w$

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

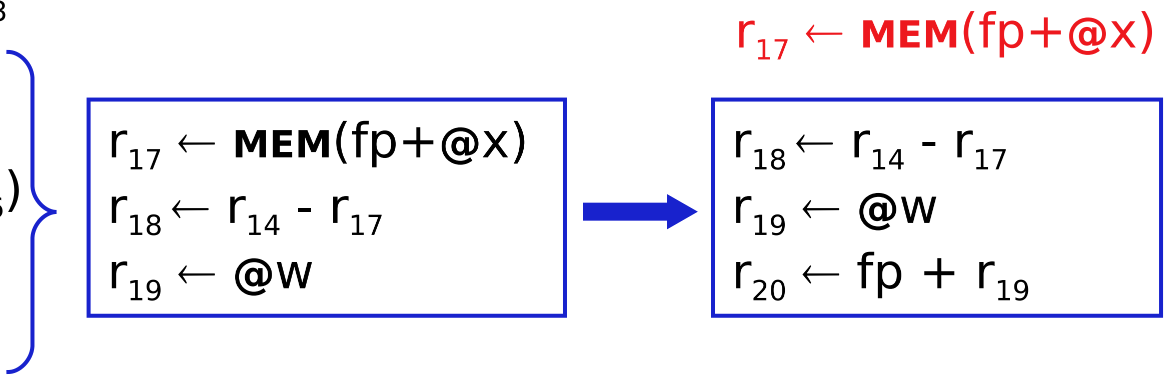
$r_{17} \leftarrow \text{MEM}(r_{16})$

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

$r_{19} \leftarrow @w$

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

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



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

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

$r_{19} \leftarrow @w$

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

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

$r_{19} \leftarrow @w$

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

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

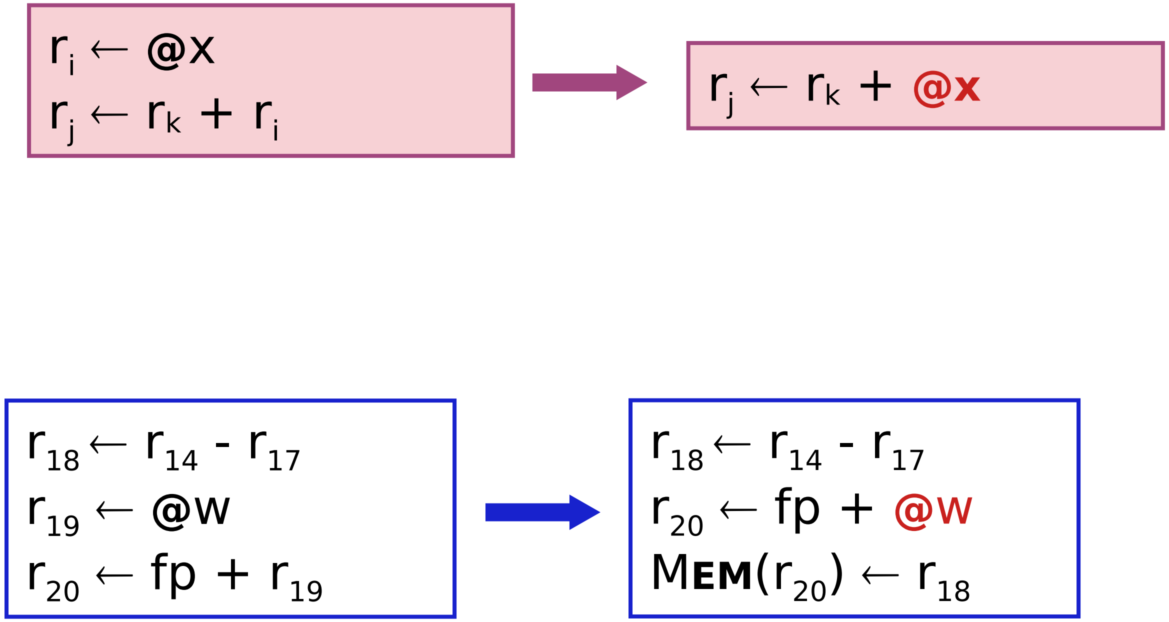
$r_{17} \leftarrow \text{MEM}(r_{16})$

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

$r_{19} \leftarrow @w$

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

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



$r_i \leftarrow @x$

$r_j \leftarrow r_k + r_i$

$\rightarrow$

$r_j \leftarrow r_k + @x$

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

$r_{19} \leftarrow @w$

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

$\rightarrow$

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

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

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

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

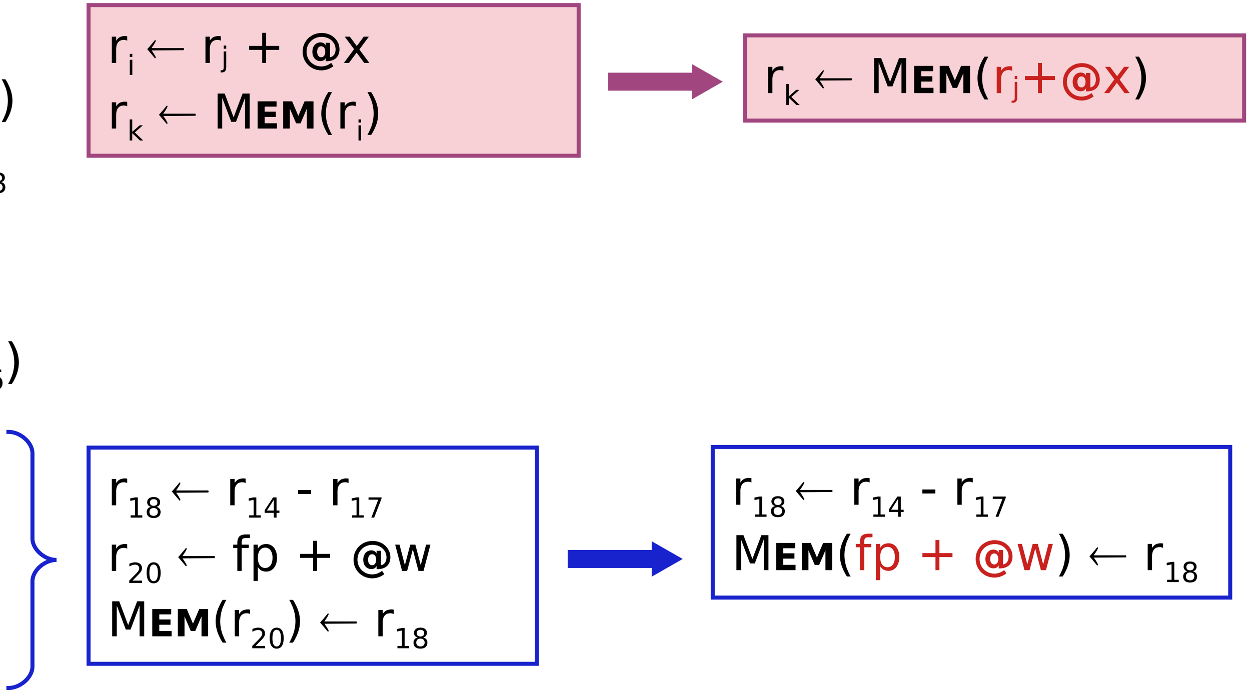
$r_{17} \leftarrow \text{MEM}(r_{16})$

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

$r_{19} \leftarrow @w$

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

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



$r_i \leftarrow r_j + @x$

$r_k \leftarrow \text{MEM}(r_i)$

$r_k \leftarrow \text{MEM}(r_j + @x)$

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

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

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

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

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

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

LLIR Code

$r_{10} \leftarrow 2$

$r_{11} \leftarrow @y$

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

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

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

$r_{15} \leftarrow @x$

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

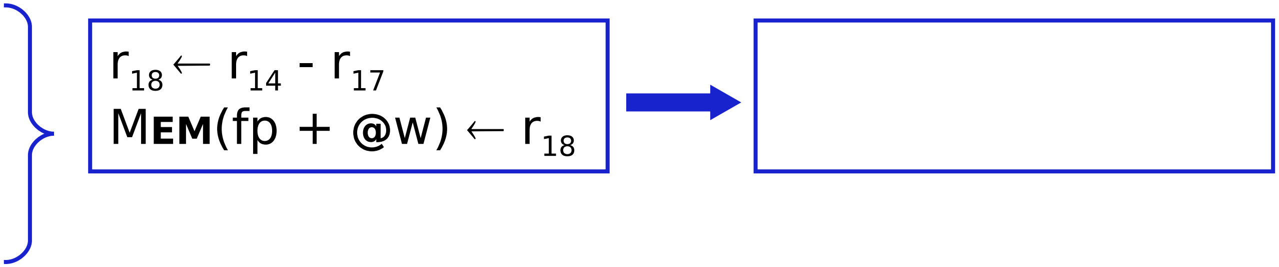
$r_{17} \leftarrow \textsc{Mem}(r_{16})$

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

$r_{19} \leftarrow @w$

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

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



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

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

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

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

Example

LLIR Code

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

Simplify

LLIR Code

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

Example: Matcher

LLIR Code

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

Match

MIPS Code

LW	r_{13} , @y($fp)
ADDI	r_{14} , r_{13} , 2
LW	r_{17} , @x($fp)
SUB	r_{18} , r_{14} , r_{17}
SW	r_{18} , @w($fp)

- Produces pretty good code

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 (LLVM)