

Compiler: Assembler Generation

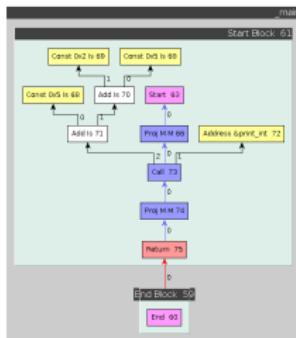
Norman Böwing, Andreas Eberle, Aleksej Frank, Polina Goltsman and Valentin Zickner

KARLSRUHE INSTITUTE OF TECHNOLOGY (KIT)

```
33     public static void createAssembly() {
34         boolean debugRegisterAllocation = false;
35         InterferenceGraph.setDebuggingMode(debugRegisterAllocation);
36
37         final ArrayList<AssemblerOperation> assembler = new ArrayList<>();
38
39         assembler.add(new TextOperation());
40         assembler.add(new P2AlignOperation());
41
42         for (Graph graph : Program.getGraphs()) {
43             BackEdges.enable(graph);
44             graph.walk(new InsertBlockAfterConditionVisitor());
45             BackEdges.disable(graph);
46             assembler.add(new FunctionSpecificationOperation(graph.getEntity().getLdName()));
47         }
48
49         for (Graph graph : Program.getGraphs()) {
50             if (debugRegisterAllocation)
51                 System.out.println(graph.getEntity().getLdName());
52
53             BlockNodesCollectingVisitor collectorVisitor = new BlockNodesCollectingVisitor();
54             graph.walkTopological(collectorVisitor);
55         }
56     }
```

Tasks of Assembler Generation

■ Precondition: Firm-Graph



Tasks of Assembler Generation

- Precondition: Firm-Graph
- Result: x86_64 Assembler



```
.text
.p2align 4,,15
.globl _main
.type _main, @function
_main:
.L61:
    movl $0x2, %ebp # Const Is<0x2>[69:10]
    add $0x5, %ebp # Add Is[70:11]
    add $0x5, %ebp # Add Is[71:12]
    movl %ebp, %edi
    call print_int # Call T[73:14]
    ret
.size _main, .-_main
.L59:
# end node
```

Tasks of Assembler Generation

- Precondition: Firm-Graph
- Result: x86_64 Assembler
- Tasks:
 - Instruction selection
 - Instruction scheduling
 - Naive register allocation: Use stack



```
.text
.p2align 4,,15
.globl _main
.type _main, @function
_main:
.L61:
    movl $0x2, %ebp # Const Is<0x2>[69:10]
    add $0x5, %ebp # Add Is[70:11]
    add $0x5, %ebp # Add Is[71:12]
    movl %ebp, %edi
    call print_int # Call T[73:14]
    ret
.size _main, .-_main
.L59:
# end node
```

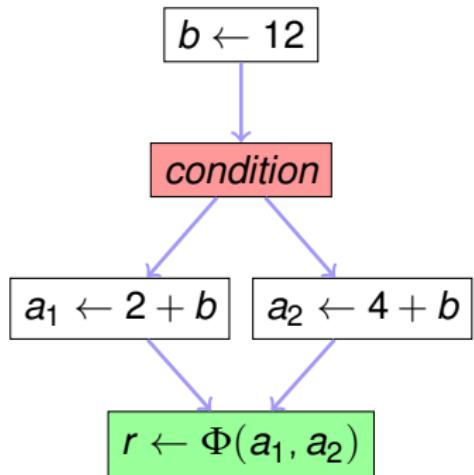
Tasks of Assembler Generation

- Precondition: Firm-Graph
- Result: x86_64 Assembler
- Tasks:
 - Instruction selection
 - Instruction scheduling
 - Naive register allocation: Use stack
- Problems:
 - Memory access is expensive

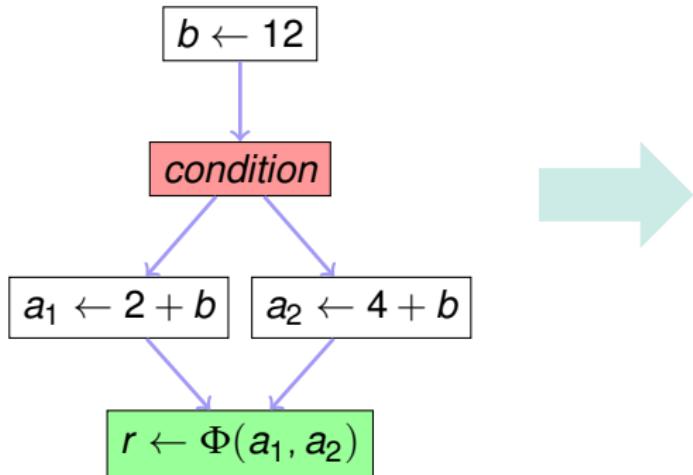


```
.text
.p2align 4,,15
.globl _main
.type _main, @function
_main:
.L61:
    movl $0x2, %ebp # Const Is<0x2>[69:10]
    add $0x5, %ebp # Add Is[70:11]
    add $0x5, %ebp # Add Is[71:12]
    movl %ebp, %edi
    call print_int # Call T[73:14]
    ret
.size _main, .-_main
.L59:
# end node
```

Principal procedure

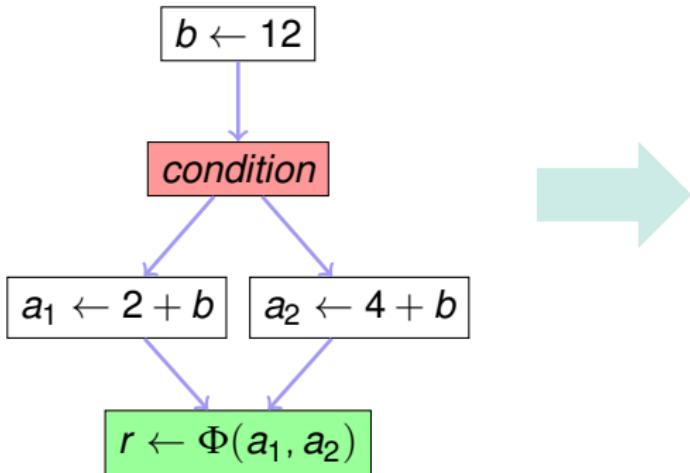


Principal procedure



$R1 \leftarrow 12$
condition
jnz ifcase
jmp elsecase
ifcase : $R2 \leftarrow \text{add } 2, R1$
jmp end
elsecase : $R3 \leftarrow \text{add } 4, R1$
jmp end
end : $R4 \leftarrow \Phi(R2, R3)$

Principal procedure

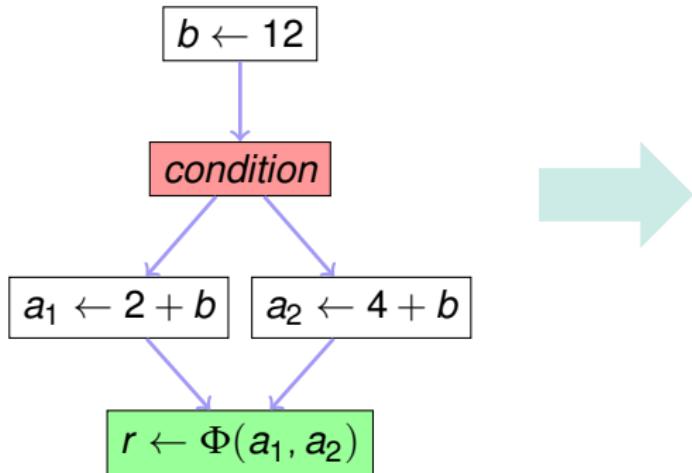


$R1 \leftarrow 12$
condition
jnz ifcase
jmp elsecase
ifcase : $R2 \leftarrow \text{add } 2, R1$
jmp end
elsecase : $R3 \leftarrow \text{add } 4, R1$
jmp end
end : $R4 \leftarrow \Phi(R2, R3)$

Challenges:

- Firm nodes not extendable in jFirm

Principal procedure

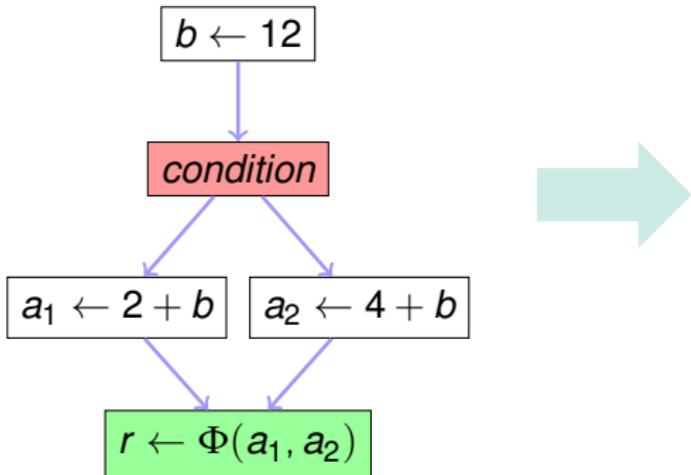


```
R1 ← 12
condition
jnz ifcase
jmp elsecase
ifcase : R2 ← add 2, R1
          jmp end
elsecase : R3 ← add 4, R1
            jmp end
end :    R4 ← Φ(R2, R3)
```

Challenges:

- Firm nodes not extendable in jFirm
- Two operand code in x86_64 assembler:
 $R2 \leftarrow \text{add } 2, R1 \Rightarrow \text{mov } R1, R2; \text{add } 2, R2$

Principal procedure



```
R1 ← 12
condition
jnz ifcase
jmp elsecase
ifcase : R2 ← add 2, R1
          jmp end
elsecase : R3 ← add 4, R1
            jmp end
end :    R4 ← Φ(R2, R3)
```

Challenges:

- Firm nodes not extendable in jFirm
- Two operand code in x86_64 assembler:
 $R2 \leftarrow \text{add } 2, R1 \Rightarrow \text{mov } R1, R2; \text{add } 2, R2$
- Register allocation

Register Allocation: Linear Scan

- Precondition: Linear block order with virtual registers

Register Allocation: Linear Scan

- Precondition: Linear block order with virtual registers

```
1      R1 ← 12
2      condition
3      jnz ifcase
4      jmp elsecase
5  ifcase : R2 ← add 2, R1
6      jmp end
7  elsecase : R3 ← add 4, R1
8      jmp end
9  end :   R4 ← Φ(R2, R3)
```

Register Allocation: Linear Scan

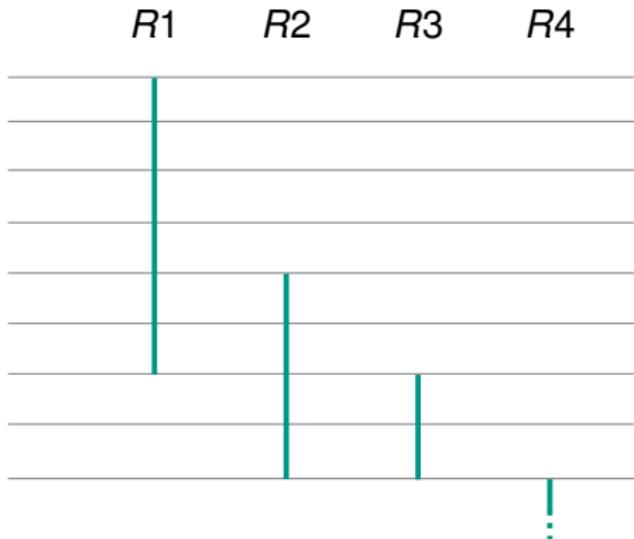
- Precondition: Linear block order with virtual registers

		$R1$	$R2$	$R3$	$R4$
1	$R1 \leftarrow 12$				
2	<i>condition</i>				
3	<i>jnz ifcase</i>				
4	<i>jmp elsecase</i>				
5	<i>ifcase</i> : $R2 \leftarrow \text{add } 2, R1$				
6	<i>jmp end</i>				
7	<i>elsecase</i> : $R3 \leftarrow \text{add } 4, R1$				
8	<i>jmp end</i>				
9	<i>end</i> : $R4 \leftarrow \Phi(R2, R3)$				

Register Allocation: Linear Scan

- Precondition: Linear block order with virtual registers

```
1      R1 ← 12
2      condition
3      jnz ifcase
4      jmp elsecase
5  ifcase : R2 ← add 2, R1
6      jmp end
7  elsecase : R3 ← add 4, R1
8      jmp end
9  end :   R4 ← Φ(R2, R3)
```



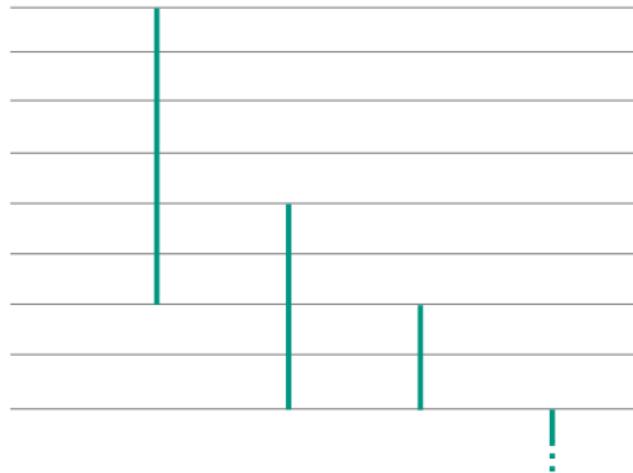
Register Allocation: Linear Scan

- Precondition: Linear block order with virtual registers
- Result: Mapping of Registers to Hardware-Registers



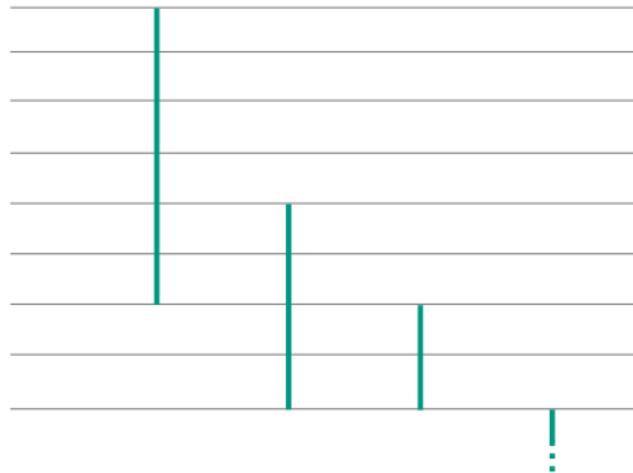
Linear Scan: Extension Graph Coloring

$R1$ $R2$ $R3$ $R4$



Linear Scan: Extension Graph Coloring

$R1$ $R2$ $R3$ $R4$



$R1$

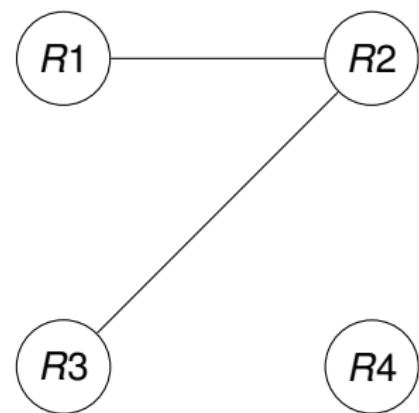
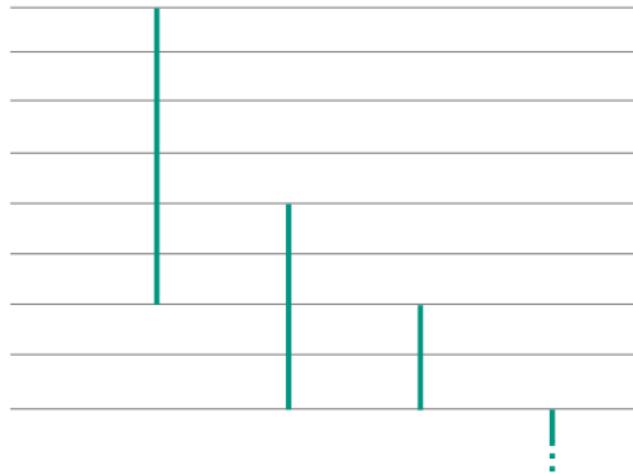
$R2$

$R3$

$R4$

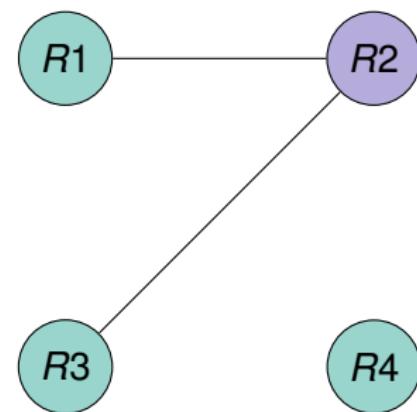
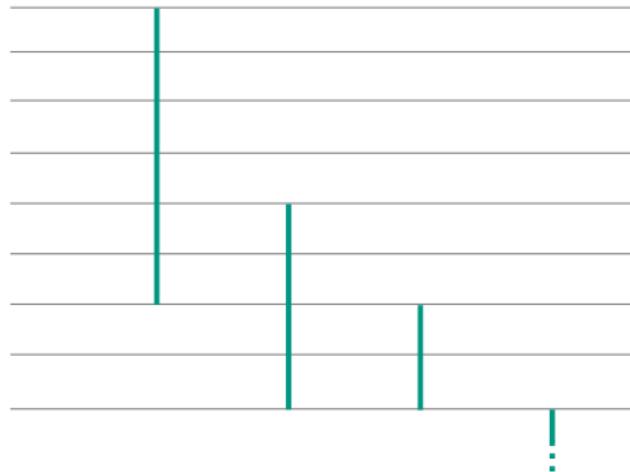
Linear Scan: Extension Graph Coloring

$R1$ $R2$ $R3$ $R4$



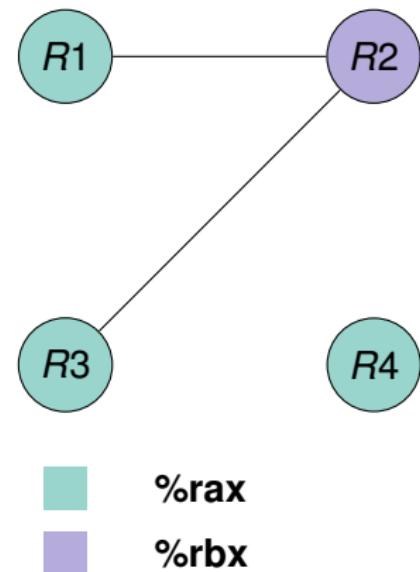
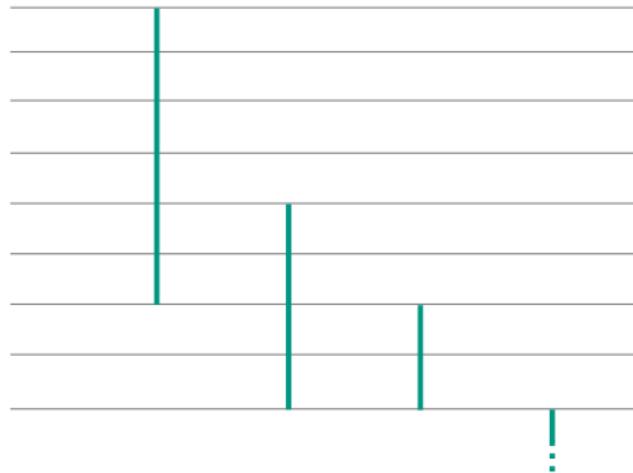
Linear Scan: Extension Graph Coloring

$R1$ $R2$ $R3$ $R4$



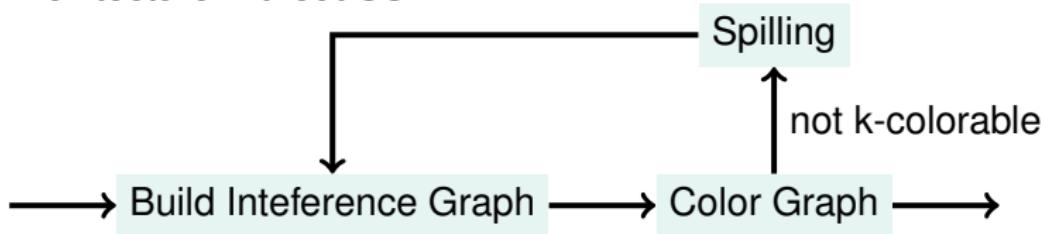
Linear Scan: Extension Graph Coloring

$R1$ $R2$ $R3$ $R4$



Register Allocation with Graph Coloring

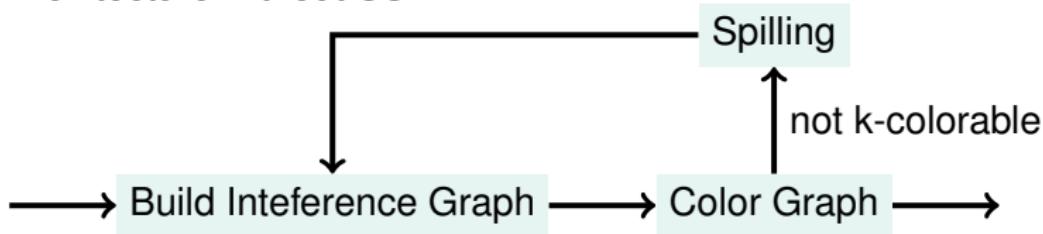
Architecture without SSA



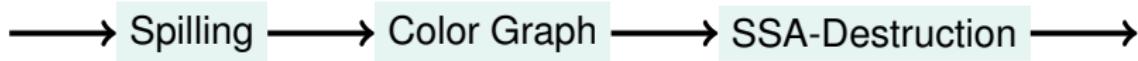
(source: Slides of lecture Compiler, IPD)

Register Allocation with Graph Coloring

Architecture without SSA

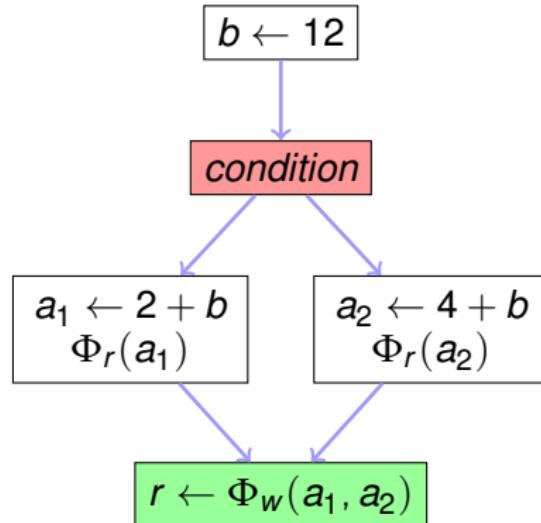


Architecture with SSA



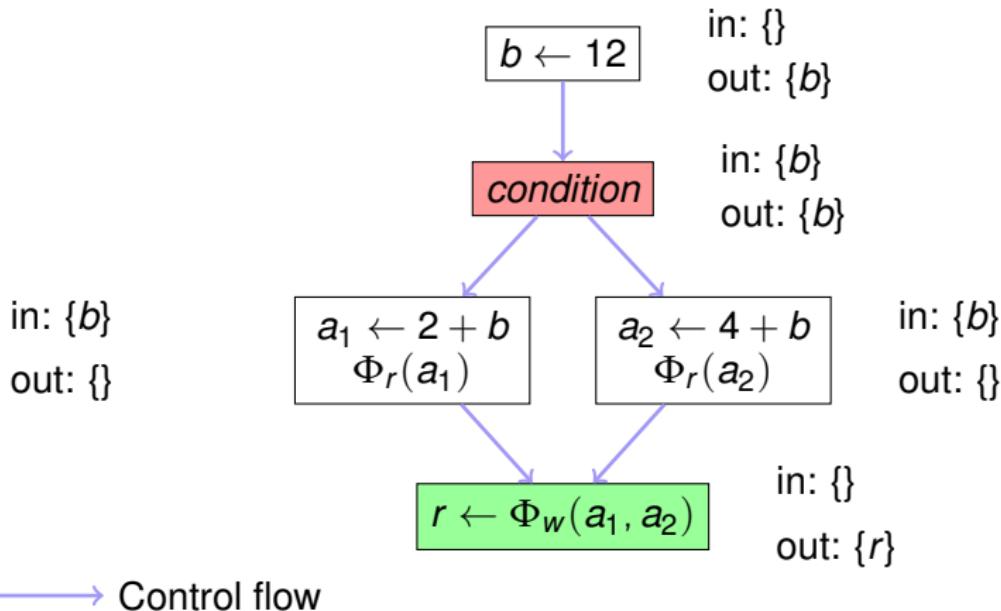
(source: Slides of lecture Compiler, IPD)

SSA-Based Register Allocation

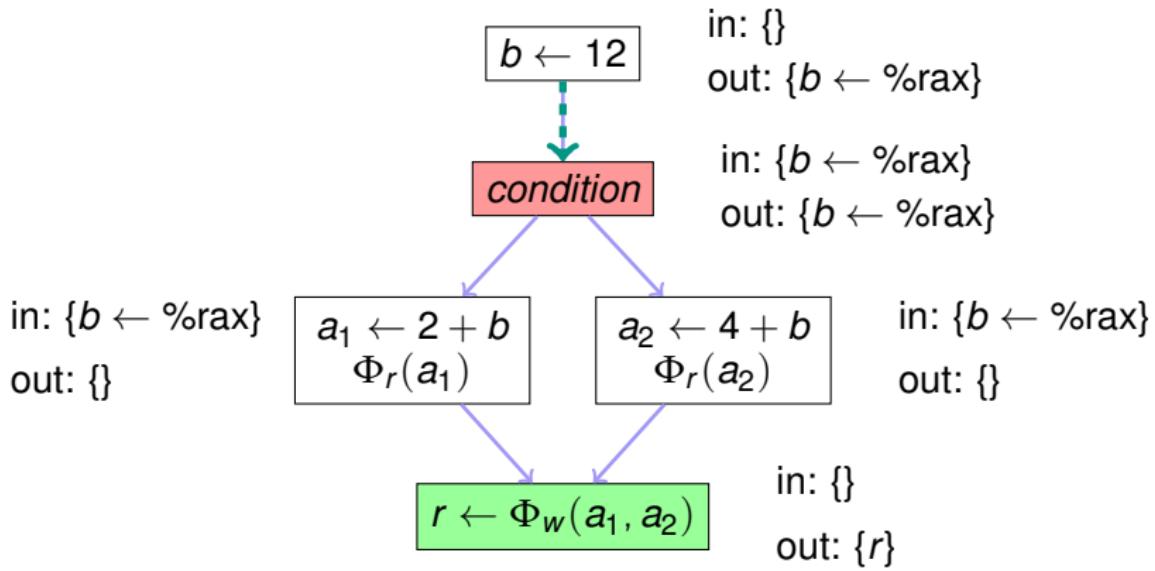


→ Control flow

SSA-Based Register Allocation

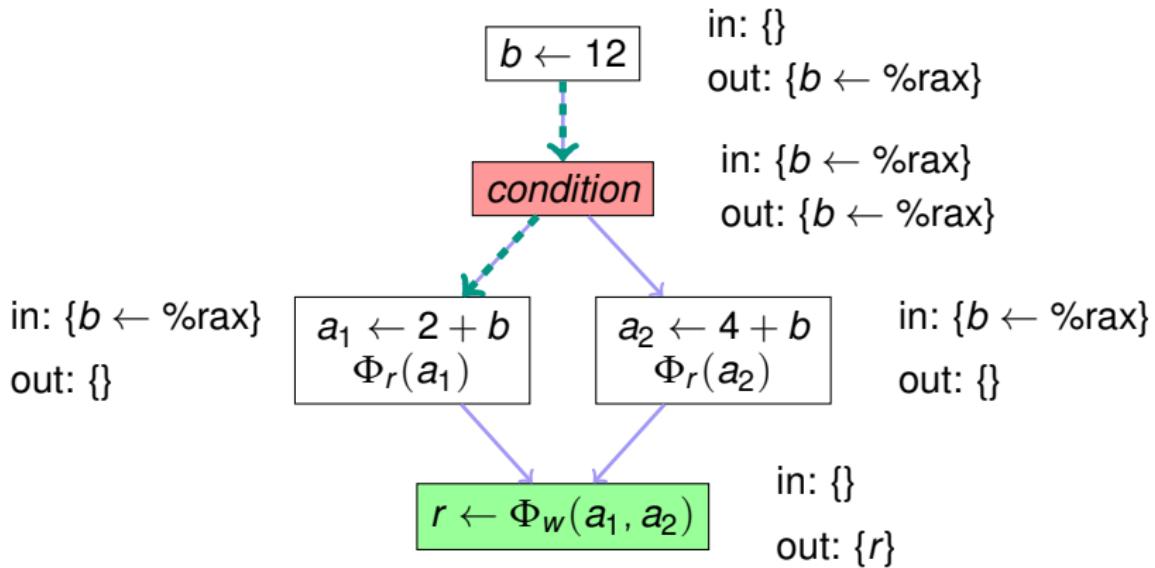


SSA-Based Register Allocation



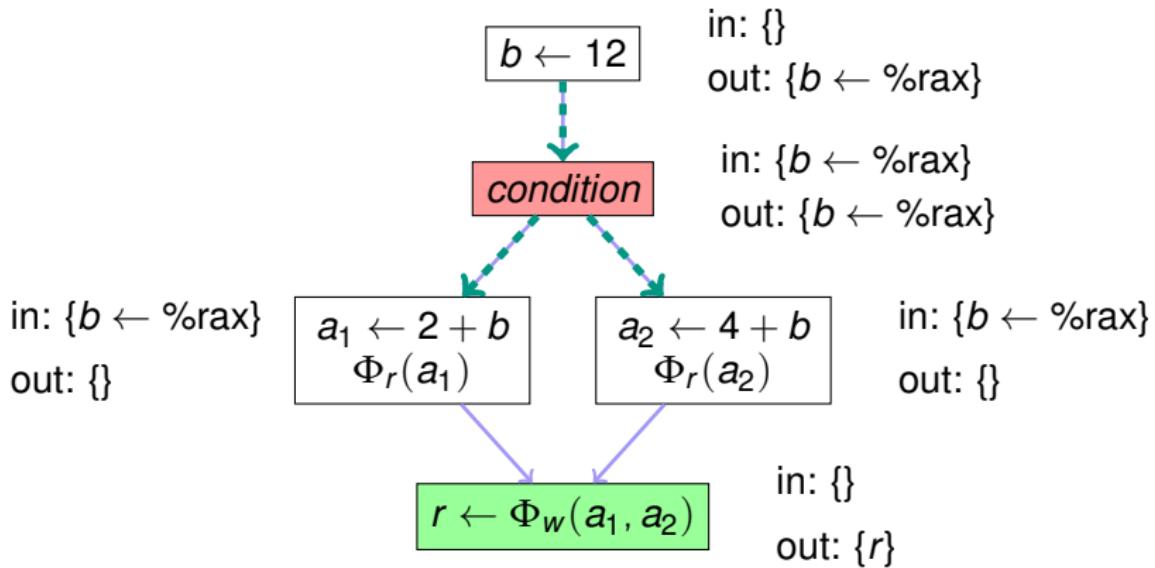
→ Control flow
→ Dominance

SSA-Based Register Allocation

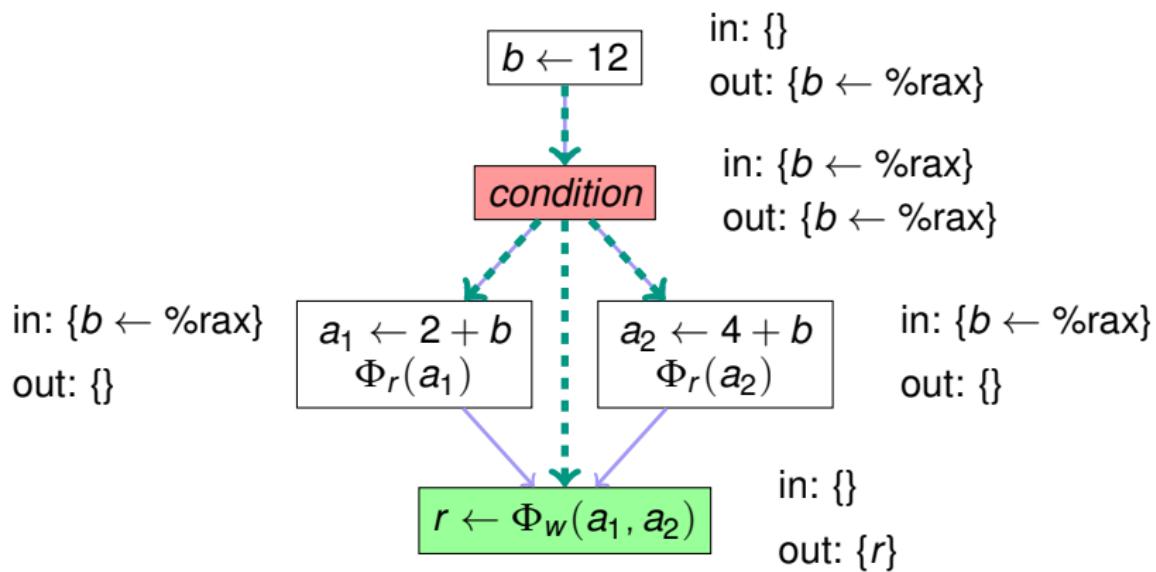


→ Control flow
→ Dominance

SSA-Based Register Allocation

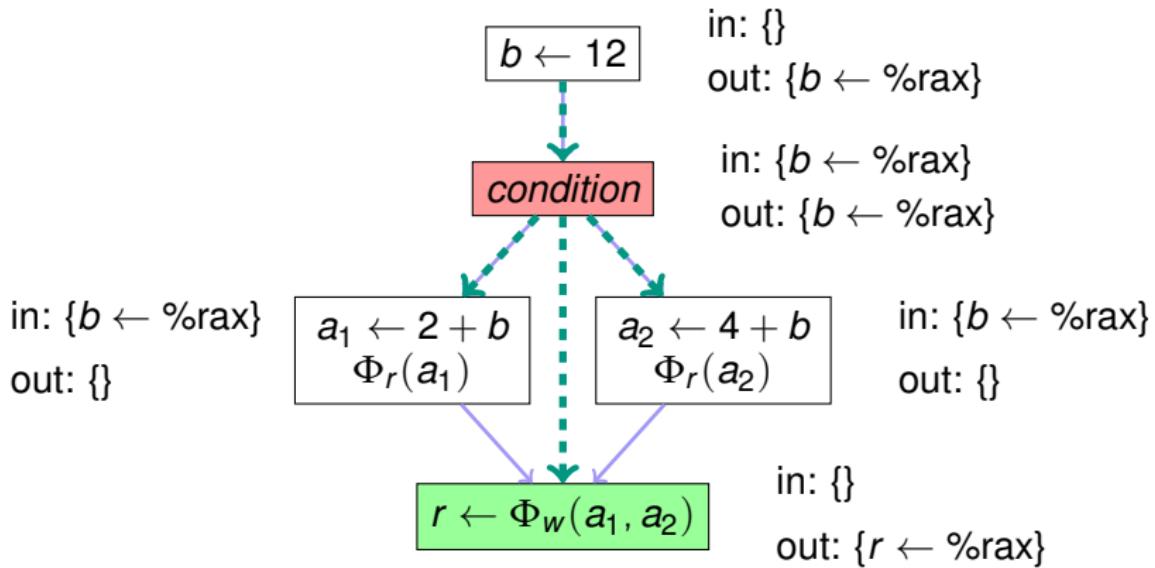


SSA-Based Register Allocation



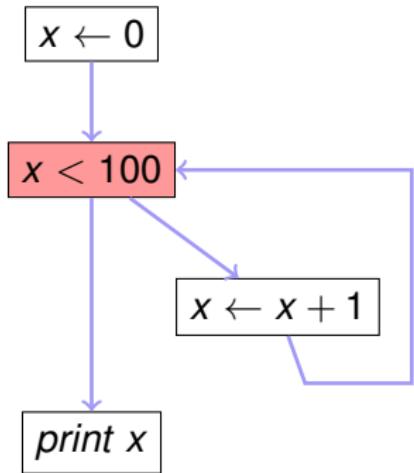
→ Control flow
→ Dominance

SSA-Based Register Allocation

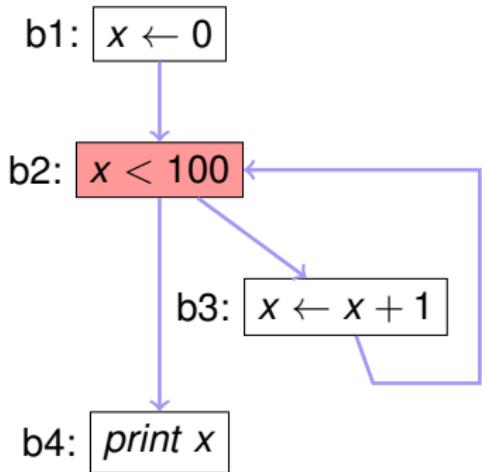


→ Control flow
→ Dominance

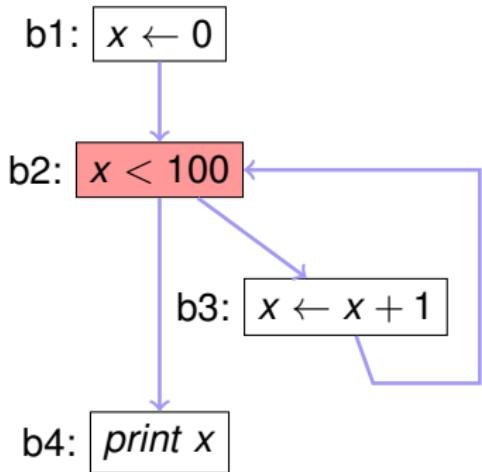
Block Ordering



Block Ordering



Block Ordering



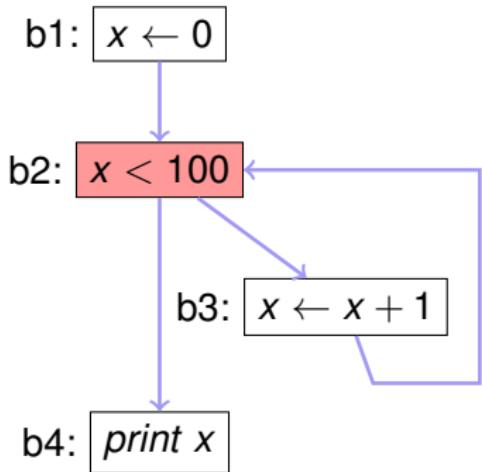
b1 : $x \leftarrow 0$
jmp b2

b2 : $\text{cmp } x, 100$
jl b3
jmp b4

b3 : $x \leftarrow x + 1$
jmp b2

b4 : $\text{print } x$

Block Ordering



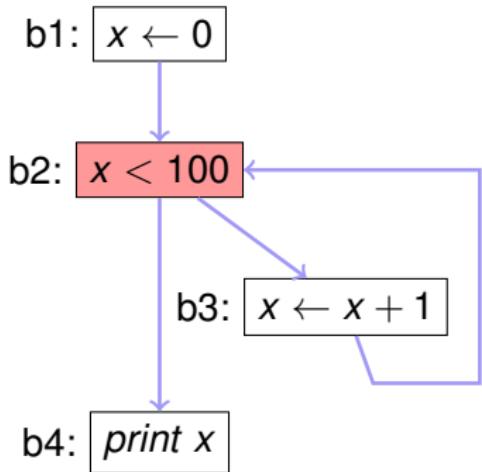
```
b1 : x ← 0
      jmp b2
b2 : cmp x, 100
      jl b3
      jmp b4
b3 : x ← x + 1
      jmp b2
b4 : print x
```

The code on the right shows the transformed assembly-like code corresponding to the blocks:

- b1 :** $x \leftarrow 0$
- b2 :** $\text{cmp } x, 100$
 $\text{jl } b3$
 $\text{jmp } b4$
- b3 :** $x \leftarrow x + 1$
 $\text{jmp } b2$
- b4 :** $\text{print } x$

- Move condition to the end of the loop

Block Ordering



b1 : $x \leftarrow 0$
jmp b2

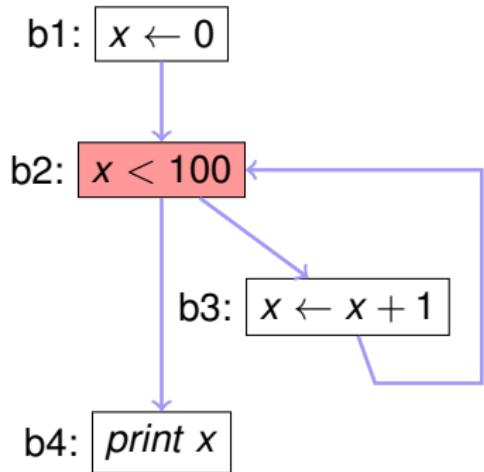
b3 : $x \leftarrow x + 1$
jmp b2

b2 : $cmp\ x, 100$
jl b3
jmp b4

b4 : *print\ x*

- Move condition to the end of the loop

Block Ordering



*b1 : $x \leftarrow 0$
~~jmp b2~~*
*b3 : $x \leftarrow x + 1$
~~jmp b2~~*
*b2 : cmp $x, 100$
jl b3
~~jmp b4~~*
b4 : print x

- Move condition to the end of the loop
- Peephole-Optimizations: Eliminate unnecessary jumps

Implemented Optimizations

- Common Subexpression Elimination
- Constant Folding
- Control Flow Optimization
- Load Store Optimization
- Local Optimization
- Loop Fusion
- Loop Invariant Code Motion
- Loop Unrolling
- Method Inlining
- Normalization
- Peephole Optimizations
- Procedure specialization and removal of side-effect free calls
- Strength Reduction

Facts and Numbers

- Lines of code: 30765
- Classes: 201
- Active days: 107 (93.86%)
- Commits: 1973
 - average 18.4 commits per active day
 - 17.3 per all days
- Craziest commit time: 5am
- Hours spend: way to much
- Longest work streak: 36/48 hours

Conclusion

- Firm makes life easier (and more complicated)
 - Especially on Windows
- Testing is important
- GIT is nice, but using it can be tricky for beginners
- Team coordination essential

Conclusion

- Firm makes life easier (and more complicated)
 - Especially on Windows
- Testing is important
- GIT is nice, but using it can be tricky for beginners
- Team coordination essential



Figure: Source: www.real-drive.de

Thank you for your attention!

Questions?