Constraint-Based Code Generation

Gabriel Hjort Blindell – KTH, SICS
Roberto Castañeda Lozano – SICS
Mats Carlsson – SICS
Frej Drejhammar – SICS
Christian Schulte – KTH, SICS

IOSS 2013

Outline

What is code generation?

What is code generation?

- Target-independent program representation $\rightarrow$ Optimized target-specific assembly code

What is code generation?

- Target-independent program representation $\rightarrow$ Optimized target-specific assembly code
    - One of the oldest computer science problems

What is code generation?

- Target-independent program representation $\rightarrow$ Optimized target-specific assembly code
    - One of the oldest computer science problems
- Set of interdependent NP-complete problems

What is code generation?

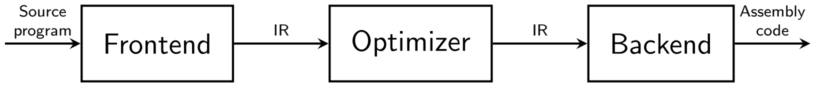
- Target-independent program representation $\rightarrow$ Optimized target-specific assembly code
    - One of the oldest computer science problems
- Set of interdependent NP-complete problems
    - Traditionally solved using non-optimal heuristics

What is code generation?

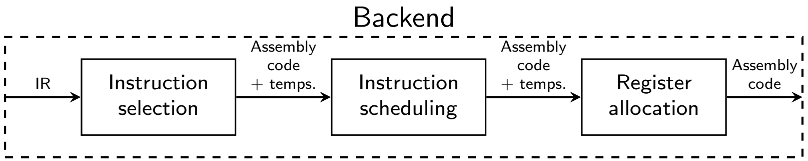
- Target-independent program representation $\rightarrow$ Optimized target-specific assembly code
  - One of the oldest computer science problems
- Set of interdependent NP-complete problems
  - Traditionally solved using non-optimal heuristics
  - Phase ordering

Traditional compiler

Traditional compiler

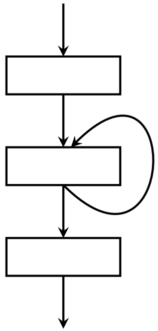


Traditional compiler



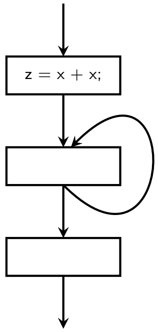
Intermediate representation (IR)

Intermediate representation (IR)



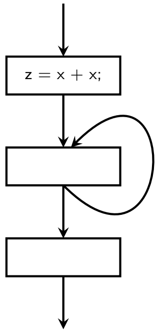
Control flow graph
(CFG), per function

Intermediate representation (IR)

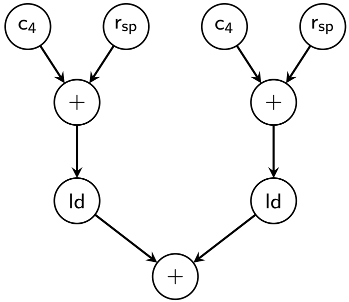


Control flow graph
(CFG), per function

Intermediate representation (IR)

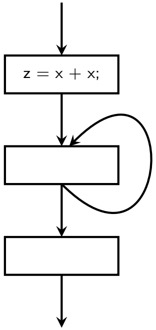


Control flow graph (CFG), per function

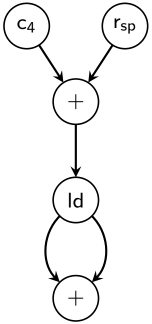


Expression tree, per block

Intermediate representation (IR)



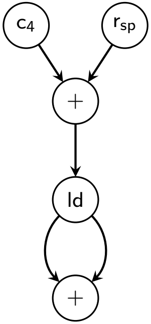
Control flow graph (CFG), per function



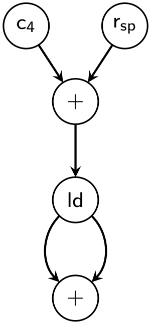
Directed acyclic graph (DAG)

From DAGs to assembly code

From DAGs to assembly code

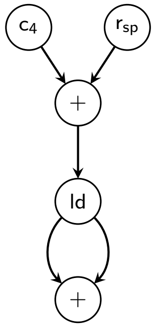


From DAGs to assembly code



- Select which CPU instructions to use

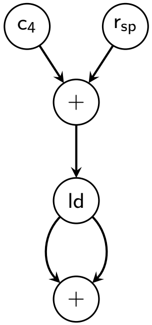
From DAGs to assembly code

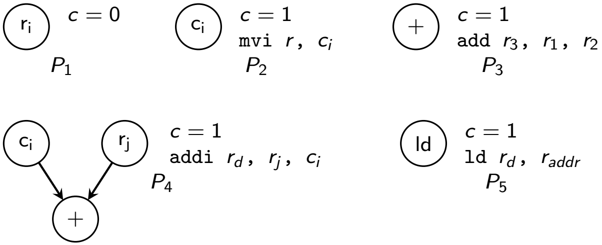


- Select which CPU instructions to use
  - Find covering with least cost

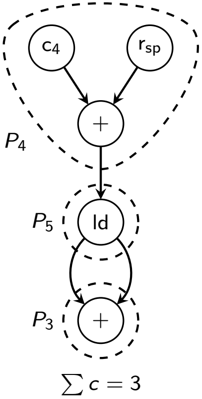
From DAGs to assembly code

- Select which CPU instructions to use
  - Find covering with least cost

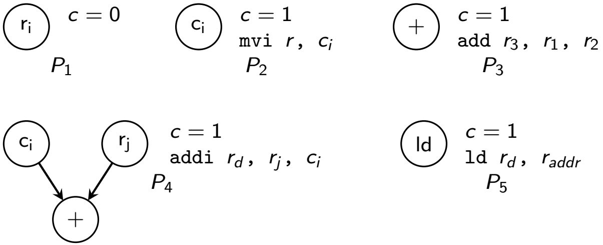




From DAGs to assembly code

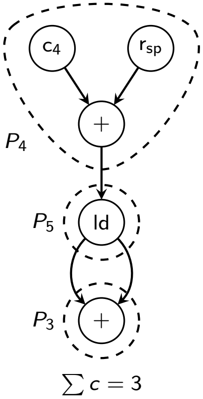


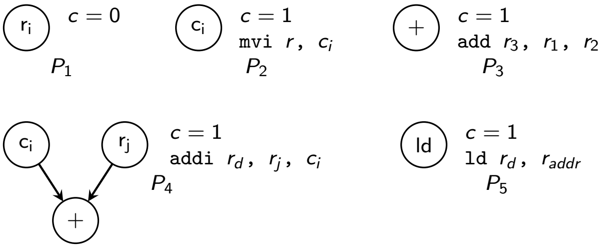
- Select which CPU instructions to use
  - Find covering with least cost



From DAGs to assembly code

- Select which CPU instructions to use
  - Find covering with least cost





$\Longrightarrow$

```
addi t0, rsp, 4
ld t1, t0
add tz, t1, t1
```

From DAGs to assembly code

addi t_0, r_{sp}, 4
ld t_1, t_0
add t_z, t_1, t_1

From DAGs to assembly code

```
addi $t_0$, $r_{sp}$, 4
ld $t_1$, $t_0$
add $t_z$, $t_1$, $t_1$
```

- Schedule instructions

From DAGs to assembly code

```
addi $t_0$, $r_{sp}$, 4
ld $t_1$, $t_0$
add $t_z$, $t_1$, $t_1$
```

- Schedule instructions
- Assign temporaries to registers (or spill to memory)

From DAGs to assembly code

addi t_0, r_{sp}, 4
ld t_1, t_0
add t_z, t_1, t_1

addi r_0, r_{sp}, 4
ld r_1, r_0
add r_2, r_1, r_1

Requires 3 registers

- Schedule instructions
- Assign temporaries to registers (or spill to memory)

From DAGs to assembly code

```
addi $t_0$, $r_{sp}$, 4
ld $t_1$, $t_0$
add $t_z$, $t_1$, $t_1$
```

- Schedule instructions
- Assign temporaries to registers (or spill to memory)

```
addi $r_0$, $r_{sp}$, 4
ld $r_1$, $r_0$
add $r_2$, $r_1$, $r_1$
```

Requires 3 registers

```
addi $r_0$, $r_{sp}$, 4
ld $r_0$, $r_0$
add $r_0$, $r_0$, $r_0$
```

Requires only 1 register

Our Approach

Our Approach

- Constraint programming

Our Approach

- Constraint programming
    - Optimality

Our Approach

- Constraint programming
    - Optimality*

*Given enough patience and money to burn while waiting

Our Approach

- Constraint programming
  - Optimality*
  - Flexible model

*Given enough patience and money to burn while waiting

Our Approach

- Constraint programming
    - Optimality*
    - Flexible model
    - Integration

*Given enough patience and money to burn while waiting

Our Approach

- Constraint programming
  - Optimality*
  - Flexible model
  - Integration
- Current status

*Given enough patience and money to burn while waiting

Our Approach

- Constraint programming
    - Optimality*
    - Flexible model
    - Integration
- Current status
    - Instruction selection - concepts / ideas

*Given enough patience and money to burn while waiting

Our Approach

- Constraint programming
  - Optimality*
  - Flexible model
  - Integration
- Current status
  - Instruction selection - concepts / ideas
  - Instruction scheduling & register allocation - prototype + paper*

*Given enough patience and money to burn while waiting

**Constraint-based register allocation and instruction scheduling.* CP2012.

Instruction Selection

Which DAG node do we cover by which pattern?

Instruction Selection

Instruction Selection

1. Identify potential use of patterns in the DAG

Instruction Selection

1. Identify potential use of patterns in the DAG
2. Find optimal covering

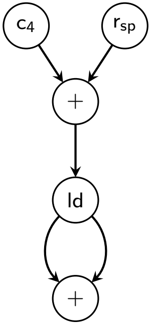
Instruction Selection

1. Identify potential use of patterns in the DAG
   - Use existing $O(n)$ techniques
2. Find optimal covering

Instruction Selection

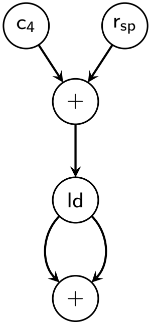
1. Identify potential use of patterns in the DAG
   - Use existing $O(n)$ techniques
2. Find optimal covering
   - Build and solve a constraint model

Instruction Selection

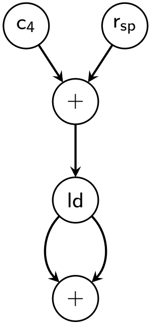


Instruction Selection

- Variables:

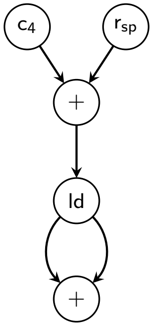


Instruction Selection



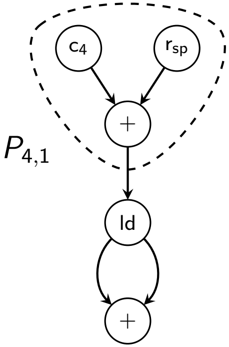
- Variables:
  - One integer variable for each DAG node to decide by which pattern instance is it covered

Instruction Selection



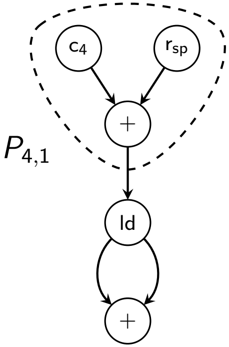
- Variables:
  - One integer variable for each DAG node to decide by which pattern instance is it covered
  - A Boolean variable for each pattern instance to decide whether it is used

Instruction Selection



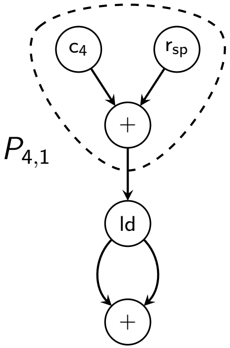
- Variables:
  - One integer variable for each DAG node to decide by which pattern instance is it covered
  - A Boolean variable for each pattern instance to decide whether it is used

Instruction Selection



- Variables:
  - One integer variable for each DAG node to decide by which pattern instance is it covered
  - A Boolean variable for each pattern instance to decide whether it is used
- Constraints:

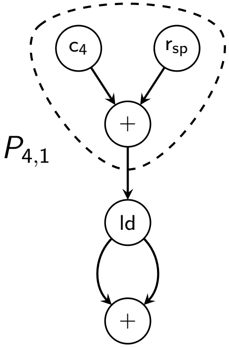
Instruction Selection



- Variables:
  - One integer variable for each DAG node to decide by which pattern instance is it covered
  - A Boolean variable for each pattern instance to decide whether it is used
- Constraints:

$$D(\textcircled{c_4}) = P_{4,1} \iff D(\textcircled{+}) = P_{4,1}$$
$$D(\textcircled{r_{sp}}) = P_{4,1} \iff D(\textcircled{+}) = P_{4,1}$$

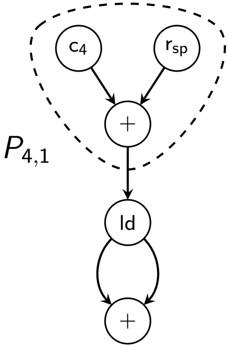
Instruction Selection



- Variables:
    - One integer variable for each DAG node to decide by which pattern instance is it covered
    - A Boolean variable for each pattern instance to decide whether it is used
- Constraints:

$$D(\textcircled{c_4}) = P_{4,1} \iff D(\textcircled{$+$}) = P_{4,1}$$
$$D(\textcircled{r_{sp}}) = P_{4,1} \iff D(\textcircled{$+$}) = P_{4,1}$$
$$D(\textcircled{$+$}) = P_{4,1} \iff D(B_{P_{4,1}}) = 1$$

Instruction Selection

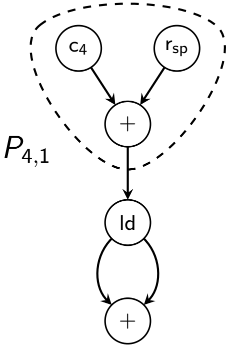


- Variables:
    - One integer variable for each DAG node to decide by which pattern instance is it covered
    - A Boolean variable for each pattern instance to decide whether it is used
- Constraints:

$$D(\textcircled{c_4}) = P_{4,1} \iff D(\textcircled{$+$}) = P_{4,1}$$
$$D(\textcircled{r_{sp}}) = P_{4,1} \iff D(\textcircled{$+$}) = P_{4,1}$$
$$D(\textcircled{$+$}) = P_{4,1} \iff D(B_{P_{4,1}}) = 1$$

- Objective:

Instruction Selection

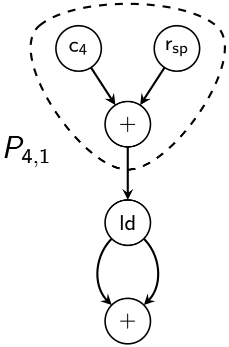


- Variables:
  - One integer variable for each DAG node to decide by which pattern instance is it covered
  - A Boolean variable for each pattern instance to decide whether it is used
- Constraints:

  $$D(\enclose{circle}{c_4}) = P_{4,1} \iff D(\enclose{circle}{+}) = P_{4,1}$$
  $$D(\enclose{circle}{r_{sp}}) = P_{4,1} \iff D(\enclose{circle}{+}) = P_{4,1}$$
  $$D(\enclose{circle}{+}) = P_{4,1} \iff D(B_{P_{4,1}}) = 1$$
- Objective:

  $\min \sum_{p,i \in P_{p,i}} c_p B_{P_{p,i}}$

Instruction Selection



- Pattern restrictions can simply be added as constraints

Instruction Scheduling

in which cycle is each instruction issued?

Instruction Scheduling

- Classic scheduling model with:
  - precedences among instructions

Instruction Scheduling

- Classic scheduling model with:
  - precedences among instructions

if i defines a value used by j:

i must be issued before j

Instruction Scheduling

- Classic scheduling model with:
  - precedences among instructions

    if i defines a value used by j:

    i must be issued before j

  - resource constraints

Instruction Scheduling

- Classic scheduling model with:
  - precedences among instructions

    if *i* defines a value used by *j*:

    i must be issued before *j*

  - resource constraints

    if *i* and *j* use the same functional unit:

    i and *j* must be issued in different cycles

Register Allocation

- Several problems
  - register assignment

Register Allocation

- Several problems
    - register assignment
    - spilling

Register Allocation

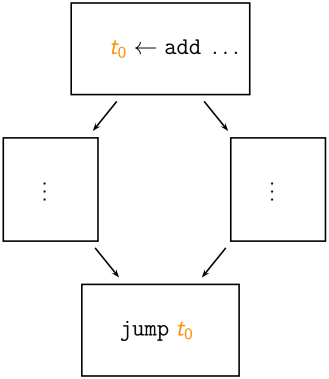
- Several problems
  - register assignment
  - spilling
  - coalescing

Register Allocation

- Several problems
  - register assignment
  - spilling
  - coalescing
- Extra challenge: whole function

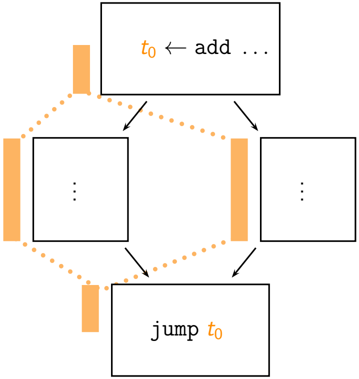
Liveness and Interference

- A temp is live while it might still be used:



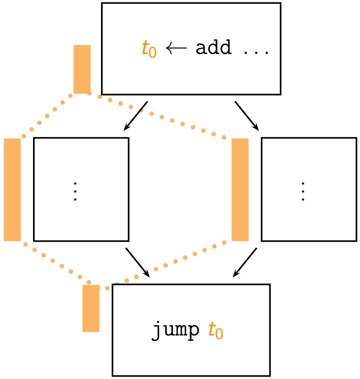
Liveness and Interference

- A temp is live while it might still be used:



Liveness and Interference

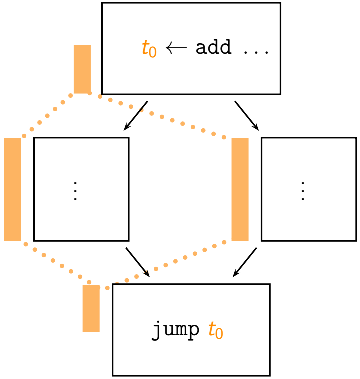
- A temp is live while it might still be used:



- Two temps interfere if they are live simultaneously

Liveness and Interference

- A temp is live while it might still be used:



- Two temps interfere if they are live simultaneously
    - non-interfering temps can share registers

Linear Static Single Assignment Form (LSSA)

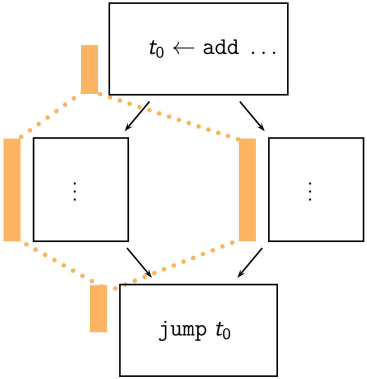
- t_0 is *global*: live in multiple blocks
- How to model interference of global temps?

Linear Static Single Assignment Form (LSSA)

- t_0 is *global*: live in multiple blocks
- How to model interference of global temps?
- LSSA: decompose global temps into multiple local temps

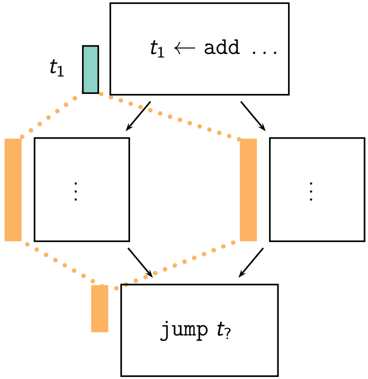
Linear Static Single Assignment Form (LSSA)

- t_0 is *global*: live in multiple blocks
- How to model interference of global temps?
- LSSA: decompose global temps into multiple local temps



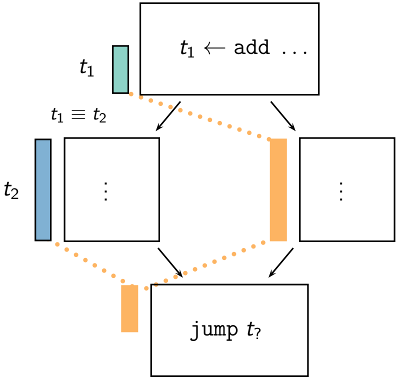
Linear Static Single Assignment Form (LSSA)

- t_0 is *global*: live in multiple blocks
- How to model interference of global temps?
- LSSA: decompose global temps into multiple local temps



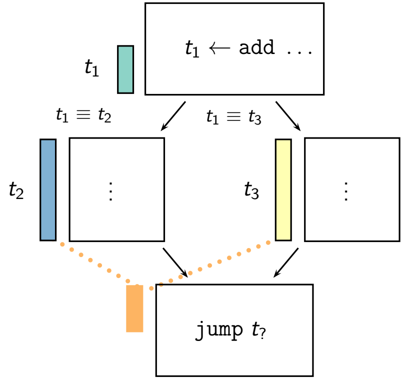
Linear Static Single Assignment Form (LSSA)

- t_0 is *global*: live in multiple blocks
- How to model interference of global temps?
- LSSA: decompose global temps into multiple local temps



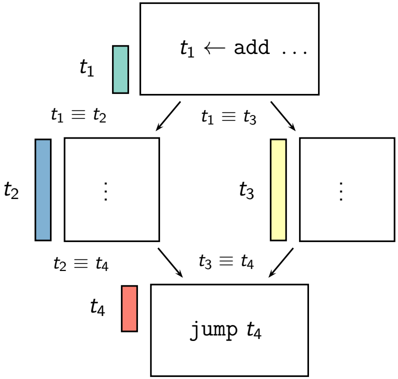
Linear Static Single Assignment Form (LSSA)

- t_0 is *global*: live in multiple blocks
- How to model interference of global temps?
- LSSA: decompose global temps into multiple local temps



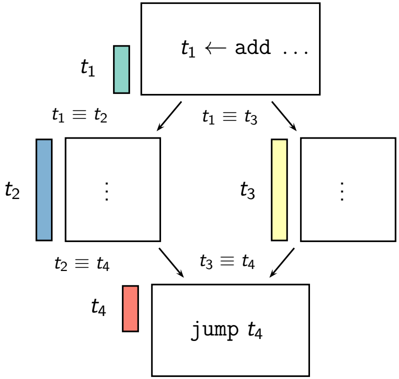
Linear Static Single Assignment Form (LSSA)

- t_0 is *global*: live in multiple blocks
- How to model interference of global temps?
- LSSA: decompose global temps into multiple local temps



Linear Static Single Assignment Form (LSSA)

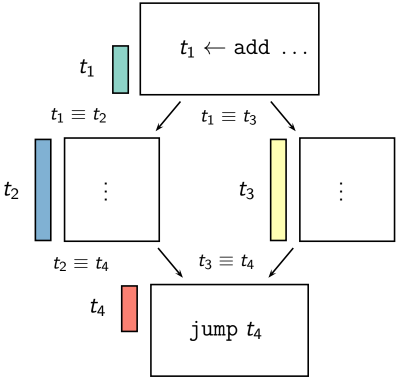
- t_0 is *global*: live in multiple blocks
- How to model interference of global temps?
- LSSA: decompose global temps into multiple local temps



- Invariant: all temps are local

Linear Static Single Assignment Form (LSSA)

- t_0 is *global*: live in multiple blocks
- How to model interference of global temps?
- LSSA: decompose global temps into multiple local temps



- Invariant: all temps are local $\rightarrow$ simple interference model

Register Assignment

to which register do we assign each temporary?

Register Assignment as Rectangle Packing

Register Assignment

Rectangle Packing

Register Assignment as Rectangle Packing

Register Assignment	Rectangle Packing
temp live ranges	rectangles

Register Assignment as Rectangle Packing

Register Assignment	Rectangle Packing
temp live ranges	rectangles
temp size	rectangle width

Register Assignment as Rectangle Packing

Register Assignment	Rectangle Packing
temp live ranges	rectangles
temp size	rectangle width
interfering temps cannot share registers	rectangles cannot overlap

Register Assignment as Rectangle Packing

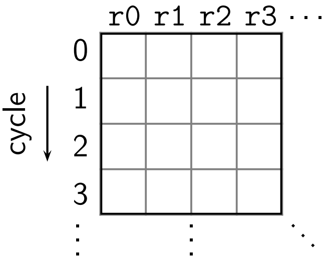
Register Assignment	Rectangle Packing
temp live ranges	rectangles
temp size	rectangle width
interfering temps cannot share registers	rectangles cannot overlap

$\rightarrow$ based on (Pereira *et al.*, 2008)

Register Assignment as Rectangle Packing

Register Assignment	Rectangle Packing
temp live ranges	rectangles
temp size	rectangle width
interfering temps cannot share registers	rectangles cannot overlap

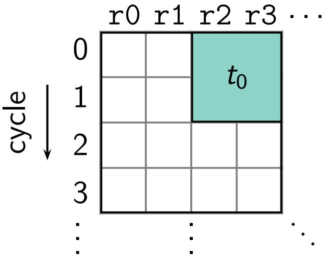
$\rightarrow$ based on (Pereira *et al.*, 2008)



Register Assignment as Rectangle Packing

Register Assignment	Rectangle Packing
temp live ranges	rectangles
temp size	rectangle width
interfering temps cannot share registers	rectangles cannot overlap

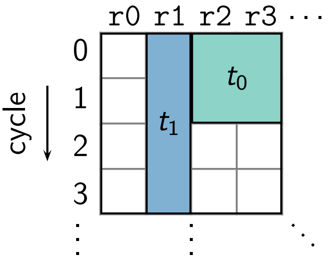
$\rightarrow$ based on (Pereira *et al.*, 2008)



Register Assignment as Rectangle Packing

Register Assignment	Rectangle Packing
temp live ranges	rectangles
temp size	rectangle width
interfering temps cannot share registers	rectangles cannot overlap

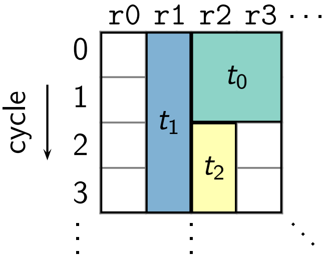
→ based on (Pereira *et al.*, 2008)



Register Assignment as Rectangle Packing

Register Assignment	Rectangle Packing
temp live ranges	rectangles
temp size	rectangle width
interfering temps cannot share registers	rectangles cannot overlap

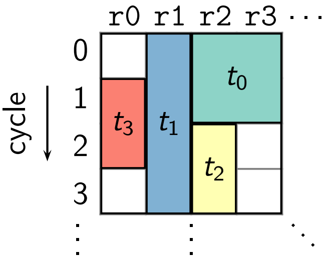
$\rightarrow$ based on (Pereira *et al.*, 2008)



Register Assignment as Rectangle Packing

Register Assignment	Rectangle Packing
temp live ranges	rectangles
temp size	rectangle width
interfering temps cannot share registers	rectangles cannot overlap

→ based on (Pereira *et al.*, 2008)



Register Allocation: Other Problems

- Spilling
  - consider memory locations as registers too

Register Allocation: Other Problems

- Spilling
  - consider memory locations as registers too
  - optional copies to transfer temps

Register Allocation: Other Problems

- Spilling
  - consider memory locations as registers too
  - optional copies to transfer temps
  - new variables to decide on copy implementation
    - special case of instruction selection

Register Allocation: Other Problems

- Spilling
  - consider memory locations as registers too
  - optional copies to transfer temps
  - new variables to decide on copy implementation
    - special case of instruction selection
- Coalescing
  - assign copy source and destination to same register

Register Allocation: Other Problems

- Spilling
    - consider memory locations as registers too
    - optional copies to transfer temps
    - new variables to decide on copy implementation
        - special case of instruction selection
- Coalescing
    - assign copy source and destination to same register
- Global
    - decomposed temps are assigned to same register

Overcome Register Allocation Limitations

$$\vdots$$

$t_1 \leftarrow$ `store ...`

Overcome Register Allocation Limitations

$$\vdots$$

$t_1 \leftarrow$ `store ...`

$$\vdots$$

Overcome Register Allocation Limitations

$$\begin{aligned}
&\vdots \\
t_1 &\leftarrow \texttt{store} \ \ldots \\
&\vdots \\
t_2 &\leftarrow \texttt{load} \ t_1 \\
\ldots &\leftarrow \texttt{neg} \ t_2
\end{aligned}$$

Overcome Register Allocation Limitations

$$
\begin{aligned}
&\vdots \\
t_1 &\leftarrow \texttt{store}\ \ldots \\
&\vdots \\
t_2 &\leftarrow \texttt{load}\ t_1 \\
\ldots &\leftarrow \texttt{neg}\ t_2 \\
&\vdots
\end{aligned}
$$

Overcome Register Allocation Limitations

$$
\begin{aligned}
&\vdots \\
t_1 &\leftarrow \texttt{store} \ \ldots \\
&\vdots \\
t_2 &\leftarrow \texttt{load}\ t_1 \\
\ldots &\leftarrow \texttt{neg}\ t_2 \\
&\vdots \\
t_3 &\leftarrow \texttt{load}\ t_1 \\
\ldots &\leftarrow \texttt{inc}\ t_3 \\
&\vdots
\end{aligned}
$$

Overcome Register Allocation Limitations

$$\begin{array}{rl}
 & \vdots \\
t_1 \leftarrow & \texttt{store} \ \ldots \\
 & \vdots \\
t_2 \leftarrow & \texttt{load} \ t_1 \\
\ldots \leftarrow & \texttt{neg} \ t_2 \\
 & \vdots \\
t_3 \leftarrow & \texttt{load} \ t_1 \\
\ldots \leftarrow & \texttt{inc} \ \{t_2, t_3\} \\
 & \vdots
\end{array}$$

Full Integration

- Ultimate goal: fully unified code generation

Full Integration

- Ultimate goal: fully unified code generation
- Problem for scalability?

Full Integration

- Ultimate goal: fully unified code generation
- Problem for scalability? not necessarily!

Full Integration

- Ultimate goal: fully unified code generation
- Problem for scalability? not necessarily!
    - many dominant instruction selection decisions

Full Integration

- Ultimate goal: fully unified code generation
- Problem for scalability? not necessarily!
  - many dominant instruction selection decisions

    why use `mult` and `add` when you can use `mac`?

Full Integration

- Ultimate goal: fully unified code generation
- Problem for scalability? not necessarily!
  - many dominant instruction selection decisions

    why use `mult` and `add` when you can use `mac`?
  - most instruction selection decisions are local

Full Integration

- Ultimate goal: fully unified code generation
- Problem for scalability? not necessarily!
  - many dominant instruction selection decisions

    why use `mult` and `add` when you can use `mac`?
  - most instruction selection decisions are local
- Work in progress . . .

Scaling to Huge Functions

- Inlining and other optimizations can blow function size
- Not usual, but we cannot just ignore them!

Scaling to Huge Functions

- Inlining and other optimizations can blow function size
- Not usual, but we cannot just ignore them!
- Possible strategies:
  1. progressiveness

Scaling to Huge Functions

- Inlining and other optimizations can blow function size
- Not usual, but we cannot just ignore them!
- Possible strategies:
  1. progressiveness
  2. local search, large neighborhood search, etc.

Scaling to Huge Functions

- Inlining and other optimizations can blow function size
- Not usual, but we cannot just ignore them!
- Possible strategies:
  1. progressiveness
  2. local search, large neighborhood search, etc.
  3. if everything else fails, resort to greedy algorithms
     - *decent* polynomial solutions always available