Register Allocation by Puzzle Solving
Puzzle Solving

- Create an elementary program.
  * Transform program into SSA form.
    * Transform the SSA program into SSI form.
- Turn the elementary program into a collection of puzzles.
- Perform puzzle solving, spilling, and coalescing.
- Turn the elementary program and the register allocation result into assembly code.
Puzzle Solving

- Create an elementary program.

* Transform program into SSA form.
  * Transform the SSA program into SSI form.

- Turn the elementary program into a collection of puzzles.

- Perform puzzle solving, spilling, and coalescing.

- Turn the elementary program and the register allocation result into assembly code.
Example of elementary program.

```
A = ·
p_1: jump L_2, L_3

L1
A_{01} = ·
p_1: (A_1) = (A_{01})

L2
AL = ·
p_6: c = AL
p_7: jump L_4

L3
AL_{56} = ·
P_6: (A_6, AL_6) = (A_{56}, AL_{56})
C_{67} = AL_6
p_7: (A_7, c_7) = (A_{67}, c_{67})
[(A_8, c_8): L_4] = pi(A_7, c_7)

L4
p_9: (A_9, c_3) = phi([(A_4, c_4): L_2, (A_8, c_8): L_3]
* = c_9, A_9
p_{10}: [(\_): L_{end}] = pi()
```

2008. Fernando Pereira. compilers.cs.ucla.edu/fernando/projects/puzzles/tutorial
Example of elementary program.

Program Point
Point between 2 consecutive instructions

A = ·
p1: jump L2, L3

A01 = ·
p1: (A1) = (A01)
[(A2) : L2, (A3) : L3] = pi (A1)

L1

L2

c = ·
p3: jump L4

p6: c = AL

p7: jump L4

p4:

p8:

L3

L4

joint L2, L3

p9:

p10:

jump L_end

AL = ·

p6: c = AL

p7: jump L4

p4:

p8:

AL56 = ·

p6: (A6, AL6) = (A5, AL56)

C67 = AL6

p7: (A7, C7) = (A6, C67)

[(A8, C8) : L4] = pi (A7, C7)

p5:

p9:

p10:

p11:

[(A9, C9) = phi [[(A4, C4) : L2, (A8, C8) : L3]]

· = C9, A9

p10: [ () : L_end] = pi ()

2008. Fernando Pereira. compilers.cs.ucla.edu/fernando/projects/puzzles/tutorial
Example of elementary program.

1. Define Point
   - \( A = \cdot \)
   - \( p_1 : \text{jump } L_2, L_3 \)

2. Point between 2 consecutive instructions
   - \( c = \cdot \)
   - \( p_3 : \text{jump } L_4 \)

3. Use Point
   - \( A_{L_1} = \)\( A_{L_2} = \)
   - \( p_6 : c = A_L \)
   - \( \phi \) = \( (A_2, c_{23}) \)
   - \( [A_4, c_4] : L_4 = \text{phi}(A_3, c_3) \)

4. Program Point
   - \( p_4 : \text{joint } L_2, L_3 \)
   - \( p_9 : \) = \( c, A \)
   - \( p_{10} : \text{jump } L_{end} \)

5. Use Point
   - \( p_5 : \) = \( (A_2) : L_2, (A_3) : L_3 = \text{phi}(A_1) \)
   - \( p_{11} : \) = \( (A_5, c_{67}) \)
   - \( p_{12} : \) = \( (A_8, c_{67}) \)
   - \( p_{13} : \) = \( (A_9, c_{3}) \)
   - \( (A_8, c_8) : L_4 = \text{phi}(A_7, c_7) \)

2008. Fernando Pereira. compilers.cs.ucla.edu/fernando/projects/puzzles/tutorial
Example of elementary program.

Program Point

Point between 2 consecutive instructions

Define Point

Use Point

**ϕ** function

SSA form

```
A = •
P1: jump L2, L3

AL = P6: c = AL
P7: jump L4

L2

θ

c23 = •
P3: (A3, c3) = (A2, c23)
[(A4, c4) : L4] = pi (A3, c3)

L3

L1

A01 = •
P1: (A1) = (A01)
[(A2) : L2, (A3) : L3] = pi (A1)

AL56 = P6: (A6, AL6) = (A5, AL56)
C67 = AL6
P7: (A7, c7) = (A6, c67)
[(A8, c8) : L4] = pi (A7, c7)

L4

p9: (A9, c3) = phi [(A4, c4) : L2, (A8, c8) : L3]
• = c9, A9
p10: [(()] : Lend] = pi ()

```
The Point of the Program Point

- In an elementary program all live ranges are small.
- Small live ranges means that one puzzle can be defined and solved for each instruction.
- The live range of a variable is the collection of program points where the variable is live.
Puzzle Solving

- Create an elementary program.
  - Transform program into SSA form.
    - Transform the SSA program into SSI form.
- Turn the elementary program into a collection of puzzles.
- Perform puzzle solving, spilling, and coalescing.
- Turn the elementary program and the register allocation result into assembly code.
### Three types of puzzles.

<table>
<thead>
<tr>
<th>Type</th>
<th>Board</th>
<th>Pieces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type-0</td>
<td>[\begin{array}{c} 0 \end{array}] [\begin{array}{c} \cdots \end{array}] [\begin{array}{c} K-1 \end{array}]</td>
<td>[\begin{array}{c} Y \end{array}] [\begin{array}{c} X \end{array}] [\begin{array}{c} Z \end{array}]</td>
</tr>
<tr>
<td>Type-1</td>
<td>[\begin{array}{c} \begin{array}{c} 0 \end{array}\end{array}] [\begin{array}{c} \cdots \end{array}] [\begin{array}{c} \begin{array}{c} K-1 \end{array}\end{array}]</td>
<td>[\begin{array}{c} Y \end{array}] [\begin{array}{c} X \end{array}] [\begin{array}{c} Z \end{array}] [\begin{array}{c} Y \end{array}] [\begin{array}{c} X \end{array}] [\begin{array}{c} Z \end{array}]</td>
</tr>
<tr>
<td>Type-2</td>
<td>[\begin{array}{c} \begin{array}{c} 0 \end{array}\end{array}] [\begin{array}{c} \cdots \end{array}] [\begin{array}{c} \begin{array}{c} K-1 \end{array}\end{array}]</td>
<td>[\begin{array}{c} Y \end{array}] [\begin{array}{c} Y \end{array}] [\begin{array}{c} Y \end{array}] [\begin{array}{c} X \end{array}] [\begin{array}{c} X \end{array}] [\begin{array}{c} X \end{array}] [\begin{array}{c} Z \end{array}] [\begin{array}{c} Z \end{array}] [\begin{array}{c} Z \end{array}]</td>
</tr>
</tbody>
</table>

- **X** --- Can go into any top row
- **Z** --- Can go into any bottom row
- **Y** --- Can go into any column

2008. Fernando Pereira. compilers.cs.ucla.edu/fernando/projects/puzzles/tutorial
What Decides the Board?
The register file determines the puzzle board:

**PowerPC:** 32 general purpose integer registers:

- $R_0$
- $R_1$
- $R_2$
- $R_3$
- $\cdots$
- $R_{31}$

**ARM:** 16 double precision floating-point registers:

- $S_0$
- $S_1$
- $S_2$
- $S_3$
- $S_4$
- $S_5$
- $S_6$
- $S_7$
- $\cdots$
- $S_{30}$
- $S_{31}$

- $D_0$
- $D_1$
- $D_2$
- $D_3$
- $D_{16}$

**SPARC V8:** 8 quad-precision floating-point registers:

- $F_0$
- $F_1$
- $F_2$
- $F_3$
- $F_4$
- $F_5$
- $F_6$
- $F_7$
- $\cdots$
- $F_{28}$
- $F_{29}$
- $F_{30}$
- $F_{31}$

- $D_0$
- $D_1$
- $D_2$
- $D_3$
- $D_{16}$
- $D_{17}$

- $Q_0$
- $Q_1$
What Decides the Board?

The register file determines the puzzle board:

**PowerPC: 32 general purpose integer registers:**

R₀, R₁, R₂, R₃, ..., R₃₁

PowerPC does not support aliasing, so each area has one column that corresponds to one of the 32 registers.

**ARM: 16 double precision floating-point registers:**

S₀, S₁, S₂, S₃, S₄, S₅, S₆, S₇, ..., S₃₀, S₃₁

D₀, D₁, D₂, D₃, D₄, D₅, D₆, D₇, D₈, D₉, D₁₀, D₁₁

**SPARC V8: 8 quad-precision floating-point registers:**

F₀, F₁, F₂, F₃, F₄, F₅, F₆, F₇, ..., F₂₈, F₂₉, F₃₀, F₃₁

Q₀, Q₁, Q₂, Q₃, Q₄, Q₅, Q₆, Q₇, Q₈, Q₉, Q₁₀, Q₁₁, Q₁₂, Q₁₃, Q₁₄, Q₁₅, Q₁₆, Q₁₇, Q₁₈, Q₁₉, Q₂₀, Q₂₁, Q₂₂, Q₂₃, Q₂₄, Q₂₅, Q₂₆, Q₂₇, Q₂₈, Q₂₉, Q₃₀, Q₃₁
What Decides the Board?

The register file determines the puzzle board:

**PowerPC: 32 general purpose integer registers:**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R₀</td>
<td>R₁</td>
<td>R₂</td>
<td>R₃</td>
<td>...</td>
<td>R₃₁</td>
</tr>
</tbody>
</table>

PowerPC does not support aliasing, so each area has one column that corresponds to one of the 32 registers.

**ARM: 16 double precision floating-point registers:**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S₀</td>
<td>S₁</td>
<td>S₂</td>
<td>S₃</td>
<td>S₄</td>
<td>S₅</td>
</tr>
<tr>
<td>D₀</td>
<td>D₁</td>
<td>D₂</td>
<td>D₃</td>
<td>S₆</td>
<td>S₇</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S₃₀</td>
<td>S₃₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D₁₆</td>
<td>D₁₇</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ARM has 32 single precision registers that can be combined into 16 pairs of double precision registers. Each column corresponds to the two registers that can be paired.

**SPARC V8: 8 quad-precision floating-point registers:**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F₀</td>
<td>F₁</td>
<td>F₂</td>
<td>F₃</td>
</tr>
<tr>
<td>D₀</td>
<td>D₁</td>
<td>D₂</td>
<td>D₃</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F₂₈</td>
<td>F₂₉</td>
<td>F₃₀</td>
<td>F₃₁</td>
</tr>
<tr>
<td>D₂₄</td>
<td>D₂₅</td>
<td>D₂₆</td>
<td>D₂₇</td>
</tr>
</tbody>
</table>
What Decides the Board?

The register file determines the puzzle board:

PowerPC: 32 general purpose integer registers:

```
R_0  R_1  R_2  R_3  ...  R_31
```

PowerPC does not support aliasing, so each area has one column that corresponds to one of the 32 registers.

ARM: 16 double precision floating-point registers:

```
S_0  S_1  S_2  S_3  ...  S_31
```

```
D_0  D_1  D_2  D_3  ...  D_16
```

ARM has 32 single precision registers that can be combined into 16 pairs of double precision registers. Each column corresponds to the two registers that can be paired.

SPARC V8: 8 quad-precision floating-point registers:

```
F_0  F_1  F_2  F_3  ...  F_31
```

```
D_0  D_1  D_2  D_3  ...  D_16
```

```
Q_0  Q_1  Q_2  Q_3  ...  Q_16
```

SPARC supports two levels of register aliasing. Two 32-bit floating-point registers can be combined to make a 64-bit register and two 64-bit registers can be combined to make a 128-bit register. Every area has 4 columns corresponding to the 4 registers that can be combined.
A variable is **live-in** if its live range contains a program point that precedes it.

A variable is **live-out** if its live range contains a program point that succeeds it.

| Variables | $p_x: (C, d, E, f) = (C', d', E', f')$  

$A, b = C, d, E$  

$p_{x+1}: (A'', b'', E'', f'') = (A, b, E, f)$ |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Live Ranges</td>
<td><img src="chart.png" alt="Diagram showing program points" /></td>
</tr>
<tr>
<td>Pieces</td>
<td><img src="chart.png" alt="Diagram showing puzzle pieces" /></td>
</tr>
</tbody>
</table>
The live ranges that **end** at the instruction become **X** pieces.

The live ranges that **begin** at the instruction become **Z** pieces.

The live ranges that **cross** the instruction become **Y** pieces.

---

From elementary programs to puzzle pieces.

<table>
<thead>
<tr>
<th>Variables</th>
<th>$p_x: (C, d, E, f) = (C', d', E', f')$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, b = C, d, E</td>
<td>Instruction</td>
</tr>
<tr>
<td>$p_{x+1}: (A'', b'', E'', f'') = (A, b, E, f)$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Live Ranges</th>
<th>A</th>
<th>b</th>
<th>C</th>
<th>d</th>
<th>E</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_x$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{x+1}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pieces</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
</tbody>
</table>

2008. Fernando Pereira. compilers.cs.ucla.edu/fernando/projects/puzzles/tutorial
Example

Program

\[ a = \cdot \]
\[ B = \cdot \]
\[ c = \cdot \]
\[ d = B \]
\[ E = c \]
\[ \cdot = a, d, E \]

Live Ranges

\[ a \]
\[ c \]
\[ d \]
\[ B \]
\[ E \]

_registers

\[ X_H \]
\[ X_L \]
\[ Y_H \]
\[ Y_L \]
\[ Z_H \]
\[ Z_L \]

2008. Fernando Pereira. compilers.cs.ucla.edu/fernando/projects/puzzles/tutorial
Example

Program

\[ x_H = \bullet \]

\[ y = \bullet \]

\[ x_L = \bullet \]

\[ y_H = y \]

\[ z = x_L \]

\[ \bullet = x_H, y_H, z \]

Live Ranges

\[ a \]

\[ b \]

\[ c \]

\[ d \]

Registers

\[ x_H \]

\[ x_L \]

\[ y_H \]

\[ y_L \]

\[ z_H \]

\[ z_L \]

2008. Fernando Pereira. compilers.cs.ucla.edu/fernando/projects/puzzles/tutorial
Example

Program

\[ a_1 = \bullet \]
\[(a_2) = (a_1) \]
\[ B_2 = \bullet \]
\[(a_3, B_3) = (a_2, B_2) \]
\[ c_3 = \bullet \]
\[(a_4, B_4, c_4) = (a_3, B_3, c_3) \]
\[ d_4 = B_4 \]
\[(a_5, c_5, d_5) = (a_4, c_4, d_4) \]
\[ E_5 = c_5 \]
\[(a_6, c_6, E_6) = (a_5, c_5, E_5) \]
\[ \bullet = a_6, d_6, E_6 \]

Live Ranges

\[ a \]
\[ c \]
\[ d \]
\[ E \]

Registers

\[ X_H \]
\[ X_L \]
\[ Y_H \]
\[ Y_L \]
\[ Z_H \]
\[ Z_L \]
Implementing So Far

Capital letters denote variables that must be stored into a pair of registers.

Lowercase letters denote variables that must be stored into a single register.

AL denotes pre-colored registers.

2008. Fernando Pereira. compilers.cs.ucla.edu/fernando/projects/puzzles/tutorial
Puzzle Solving

- Create an elementary program.
  * Transform program into SSA form.
    * Transform the SSA program into SSI form.
- Turn the elementary program into a collection of puzzles.
- Perform puzzle solving, spilling, and coalescing.
- Turn the elementary program and the register allocation result into assembly code.
Our Puzzle Solving Program

Do examples

2008. Fernando Pereira. compilers.cs.ucla.edu/fernando/projects/puzzles/tutorial
## Finishing the Earlier Example

<table>
<thead>
<tr>
<th>Program</th>
<th>Live Ranges</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_H = \cdot$</td>
<td>a</td>
<td>$X_H$</td>
</tr>
<tr>
<td>$Y = \cdot$</td>
<td></td>
<td>$X_L$</td>
</tr>
<tr>
<td>$X_L = \cdot$</td>
<td>c</td>
<td>$Y_H$</td>
</tr>
<tr>
<td>$Y_H = Y$</td>
<td>d</td>
<td>$Y_L$</td>
</tr>
<tr>
<td>$Y_L = X_H$</td>
<td></td>
<td>$Z_H$</td>
</tr>
<tr>
<td>$X = X_L$</td>
<td></td>
<td>$Z_L$</td>
</tr>
<tr>
<td>$\cdot = Y_L, Y_H, X$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2008. Fernando Pereira. compilers.cs.ucla.edu/fernando/projects/puzzles/tutorial
Spilling and local Coalescing

- $S = \text{empty}$
- For each puzzle $p$, in a pre-order traversal of the dominator tree of the program:
  - While $p$ is not solvable:
    - Choose and remove a piece $s$ from $p$, and for every subsequent puzzle $p'$ that contains a variable $s'$ in the family of $s$, remove $s'$ from $p'$.
    - $S' = \text{a solution of } p$, guided by $S$
    - $S = S'$
Puzzle Solving

- Create an elementary program.
  * Transform program into SSA form.
    * Transform the SSA program into SSI form.
- Turn the elementary program into a collection of puzzles.
- Perform puzzle solving, spilling, and coalescing.
- Turn the elementary program and the register allocation result into assembly code.
Putting everything together.

L1
AX = ·
P1:
jump L2, L3

L2
BL = ·
P3:
xcng BX,AX
jump L4

L3
BX = AX
AL = AL
P7:
jump L4

L4
joint L2, L3
P9:
= BL, AX
P10:
jump Lend

2008. Fernando Pereira. compilers.cs.ucla.edu/fernando/projects/puzzles/tutorial