Different Forms of IR

Register Transfer Level (RTL)

The IR Code is at a lower level than the TARGET architecture

Example:

IR (RTL style):

\[
\begin{align*}
\text{reg1} & := \%fp + \text{offset}_x \\
\text{reg2} & := \ast\text{reg1}
\end{align*}
\]

Target:

\[
\text{LOAD } [\%fp+\text{offset}_x],\text{reg}
\]

Can accommodate different CPU architectures. Porting back-end is easier. Used in “gcc”.

The Intermediate Representation

- 3-Address Instructions
  Linear sequence of operations

- Trees
  Will use in “tiling” approach...
  Similar to AST, but...
    - Greater level of detail
    - Closer to target (e.g., specific operations: IADD, FADD)

Variable addressing is explicit (\%fp+offset\_x)
Indirections (to fetch R-Values) are explicit
Source Code:
\[ x := 123 \times a[i]; \]

IR Code:
\[
\begin{align*}
  t1 &:= \%fp + -4 \\
  t2 &:= \%fp + -8 \\
  t3 &:= *t2 \\
  t4 &:= \%fp + -12 \\
  t5 &:= *t4 \\
  t6 &:= t5 * 4 \\
  t7 &:= t3 + t6 \\
  t8 &:= *t7 \\
  t9 &:= 123 * t8 \\
  *t1 &:= t9
\end{align*}
\]

Indirect Load

Variable "a" contains a pointer to the block of elements; Each element is 4 bytes
**Code Generation by Tiling**

**Assumption:**

The Intermediate Representation is in tree-form.

**Code Generation via Pattern Matching**

**Given:** A Set of Rules  
Pattern $\rightarrow$ Target Code

**Approach:** Match patterns against pieces of the tree.

**Goal:** Cover the entire tree with matches.  
(Every matched pattern will indicate what code to generate.)

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Replacement</th>
<th>Code Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>$+$ reg num</td>
<td>$\rightarrow$ reg</td>
<td>ADD reg,num,reg</td>
</tr>
<tr>
<td>$+$ reg reg</td>
<td>$\rightarrow$ reg</td>
<td>ADD reg,reg,reg</td>
</tr>
<tr>
<td>$*$ reg reg</td>
<td>$\rightarrow$ reg</td>
<td>MUL reg,reg,reg</td>
</tr>
<tr>
<td>$+$ reg num</td>
<td>$\rightarrow$ reg</td>
<td>LOAD [reg+num],reg</td>
</tr>
<tr>
<td>$*$ reg</td>
<td>$\rightarrow$ reg</td>
<td>LOAD [reg],reg</td>
</tr>
<tr>
<td>num</td>
<td>$\rightarrow$ reg</td>
<td>SET num,reg</td>
</tr>
</tbody>
</table>
### Pattern Replacement Code Template

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Replacement</th>
<th>Code Template</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>:= reg reg</code></td>
<td><code>done</code></td>
<td><code>ST reg,[reg]</code></td>
</tr>
<tr>
<td><code>:= + reg num</code></td>
<td><code>done</code></td>
<td><code>ST reg,[reg+num]</code></td>
</tr>
</tbody>
</table>

---

**Tiling The Tree**

```
:= + +
%fp -4 123
```

```
+ +
%fp -8 +
```

```
+ *
%fp 4
```

```
+ *
%fp -12
```
\[ \begin{align*} + & \quad + \\
reg & \quad num \quad num \\
+ & \quad + \\
reg & \quad num \quad + \\
reg & \quad num \\
\end{align*} \]
\[
\text{reg} := \text{reg} \text{reg} \vdash \text{num} \vdash + \\
\text{num} \vdash \text{reg} \vdash \text{num} \vdash + \\
\text{reg} \vdash \text{num} \vdash \text{reg}
\]

15

\[
\text{reg} := \text{reg} \vdash \text{num} \vdash \text{reg} \vdash \text{reg} \vdash + \\
\text{reg} \vdash \text{num} \vdash \text{reg} \vdash \text{reg} \vdash \text{reg} \vdash \text{reg}
\]

16
Tiling The Tree

\[
\text{reg} := \text{reg} + \text{num} \cdot \text{num} + \text{reg} \\
\]

LOAD \([\text{reg}], \text{reg}\)
Tiling The Tree

::=  
  reg +  
  num  
  reg *  
  reg num

LOAD [reg],reg

Tiling The Tree

::=  
  reg +  
  num  
  reg *  
  reg num

SET num,reg
Tiling The Tree

\[
\begin{align*}
\text{reg} & := \text{reg} \text{ reg}^* \text{ reg} + \text{num} \text{ num} \text{ reg} \set \text{num,reg}
\end{align*}
\]
Tiling The Tree

```
reg := num ⇒ reg
    set num, reg

reg + reg * reg
    +

reg * reg
    *

reg reg
    *

reg reg
```

Tiling The Tree

```
reg := reg + reg * reg
    +

reg * reg
    *

reg reg
    *

reg reg
```

© Harry H. Porter, 2006
Tiling The Tree

reg := reg
    + reg
    + MUL reg, reg, reg

ADD reg, reg, reg

Tiling The Tree

reg := reg
    + reg
    + reg
    + reg

ADD reg, reg, reg
Tiling The Tree

\[
\text{reg} := \text{reg} + \text{reg} + \text{reg}
\]

ADD \text{reg, reg, reg}

Tiling The Tree

\[
\text{reg} := \text{reg} + \text{reg}
\]

LOAD [\text{reg}], \text{reg}
Tiling The Tree

reg := reg + reg

LOAD [reg], reg

Tiling The Tree

reg := reg + reg + reg

ADD reg, reg, reg
Tiling The Tree

reg := reg + reg

ADD reg, reg, reg

Tiling The Tree

reg := reg

ST reg, [reg]

done
Tiling The Tree

\[ := \]

\[ \text{done} \]

\[ \text{ST } \text{reg},[\text{reg}] \]
Tiling The Tree

Do a Post-Order Tree Traversal
Do a Post-Order Tree Traversal

<table>
<thead>
<tr>
<th>Source Code:</th>
<th>IR Code:</th>
</tr>
</thead>
<tbody>
<tr>
<td>x := 123 * a[i];</td>
<td>t1 := %fp + -4</td>
</tr>
<tr>
<td></td>
<td>t2 := %fp + -8</td>
</tr>
<tr>
<td></td>
<td>t3 := *t2</td>
</tr>
<tr>
<td></td>
<td>t4 := %fp + -12</td>
</tr>
<tr>
<td></td>
<td>t5 := *t4</td>
</tr>
<tr>
<td></td>
<td>t6 := t5 * 4</td>
</tr>
<tr>
<td></td>
<td>t7 := t3 + t6</td>
</tr>
<tr>
<td></td>
<td>t8 := *t7</td>
</tr>
<tr>
<td></td>
<td>t9 := 123 * t8</td>
</tr>
<tr>
<td></td>
<td>*t1 := t9</td>
</tr>
</tbody>
</table>

There may be several ways to tile the tree!
Adding Costs to the Patterns

• Several ways to tile the tree.
• Want to choose the “best” tiling.
• Give each pattern a “cost”.
  Based on the instructions to be generated.
  Some instructions may be more costly.
• The cost of tiling the entire tree?
  Sum all costs.

Goal:
Find the lowest-cost tiling.
Cost to get result into “reg” at this point = 2

LOAD [reg],reg
Cost =1

ADD reg,num,reg
Cost=1
Cost to get result into “reg” at this point = 1.5

These two rules are compatible with each other, and can be used together to create a tiling.

Two different rules are used in this tiling.
Dynamic Programming Approaches

Build up the cost, from the bottom, up.

Tiling 1: cost=2, result=reg
Tiling 2: cost=0, result=num

May be several ways to tile...
Dynamic Programming Approaches

May be several ways to tile...

Tiling 1: cost=7, result=reg
Tiling 2: cost=0, result=num

Tiling 1: cost=3, result=reg
Tiling 2: cost=4, result=reg

Tiling 1: cost=2, result=reg
Tiling 2: cost=0, result=num

Keep the lowest cost!
Dynamic Programming Approaches

Tiling 1: cost=3, result=reg

Tiling 1: cost=7, result=reg

Tiling 1: cost=2, result=reg
Tiling 2: cost=0, result=num

Tiling 1: cost=2, result=reg
Tiling 2: cost=0, result=num

Tiling 1: cost=12, result=reg

Tiling 1: cost=3, result=reg

Tiling 1: cost=2, result=reg
Tiling 2: cost=0, result=num

Tiling 1: cost=2, result=reg
Tiling 2: cost=0, result=num
Dynamic Programming Approaches

Tiling 1: cost=7, result=reg
Tiling 2: cost=0, result=num

Tiling 1: cost=2, result=reg
Tiling 2: cost=0, result=num

Tiling 1: cost=12, result=reg
Tiling 2: cost=11, result=reg
Tiling 3: cost=10, result=reg
Tiling 4: cost=9, result=reg

© Harry H. Porter, 2006
Dynamic Programming Approaches

Tiling 1: cost=7, result=reg
Tiling 1: cost=2, result=reg
Tiling 1: cost=2, result=reg
Tiling 2: cost=0, result=num
Tiling 1: cost=2, result=reg
Tiling 1: cost=3, result=reg
Tiling 4: cost=9, result=reg

Ordering Constraints
The resulting sequence of instructions
There are some ordering dependencies.
A "partial-order"
Must do children before their parents.

MULT reg, reg, reg
LOAD [reg], reg
ADD reg, num, reg
SET num, reg

© Harry H. Porter, 2006
Ordering Constraints
The resulting sequence of instructions
There are some ordering dependencies.
A "partial-order"
Must do children
before their parents.
Instruction Scheduling

• Pick an order for the instructions
• Must respect the ordering constraints
• Some sequences may execute faster than others

Example:
Operations that go to memory take a long time.
When a LOAD is executed...
  The CPU will begin the next instruction before LOAD finishes
  When the CPU needs the operand
    The CPU will “stall” (idle clock cycles inserted)

The Idea:
  Execute the LOAD instruction a little sooner
    So the result is available when needed.
Example

ADD %fp,-4,r1
SET 123,r2
ADD %fp,-8,r3
LOAD [r3],r4
ADD %fp,-12,r5
LOAD [r5],r6
SET 4,r7
MULT r6,r7,r8
ADD r4,r8,r9
LOAD [r9],r10
ADD r2,r10,r11
ST r11,[r1]

Problem:
The result of this LOAD is needed in the next instruction
Example

ADD %fp,-4,r1
SET 123,r2
ADD %fp,-8,r3
LOAD [r3],r4
ADD %fp,-12,r5
LOAD [r5],r6
SET 4,r7
MULT r6,r7,r8
ADD r4,r8,r9
LOAD [r9],r10
ADD r2,r10,r11
ST r11,[r1]

Note: The result of this instruction is not needed until much later

Reorder the instructions!