Different Forms of IR

Register Transfer Level (RTL)

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

Example:

IR (RTL style):

```
reg1 := %fp + offset_x
reg2 := *reg1
```

Target:

```
LOAD  [%fp+offset_x],reg
```

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

Different Forms of IR

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

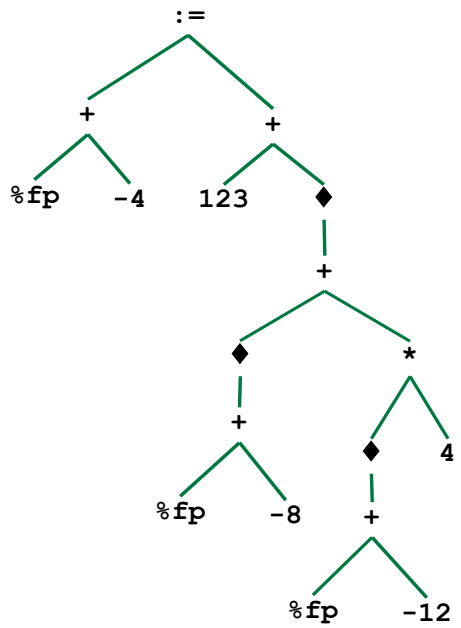
IR Code in Tree-Form

Source Code:

```
x := 123 * a[i];
```

IR Code:

```
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
```



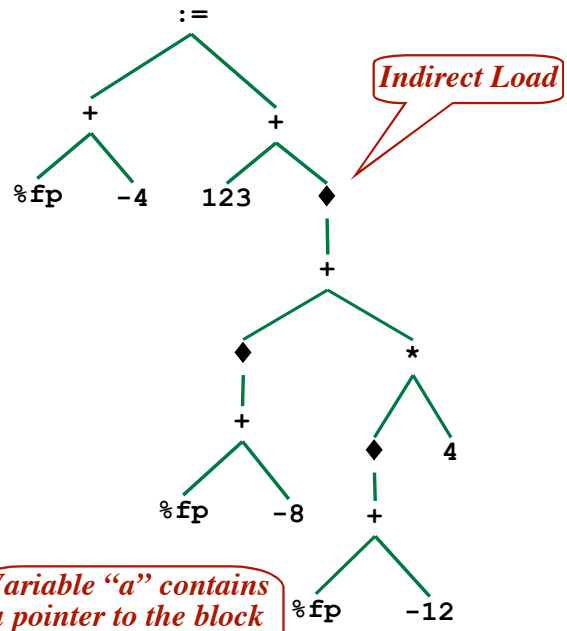
IR Code in Tree-Form

Source Code:

```
x := 123 * a[i];
```

IR Code:

```
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
```



x	%fp-4
a	%fp-8
i	%fp-12

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.

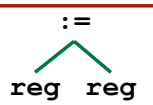
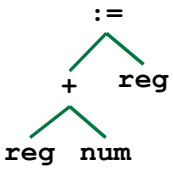
Code Generation via Pattern Matching

Given: A Set of Rules
Pattern → Target Code

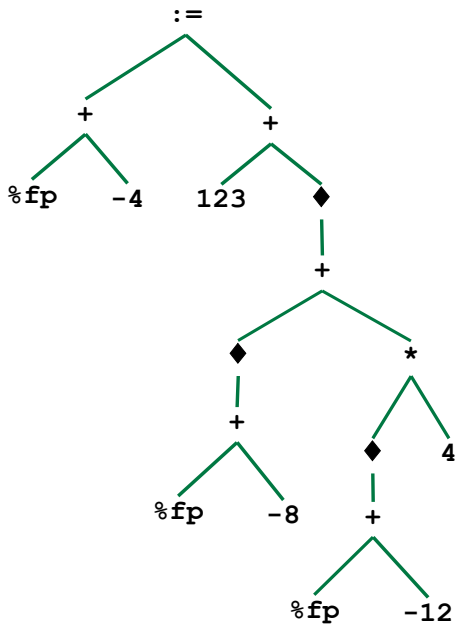
Approach: Match patterns against pieces of the tree.

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

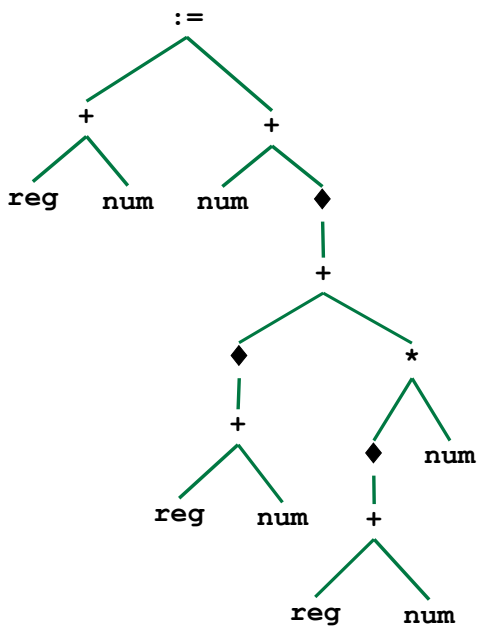
Pattern	Replacement	Code Template
+ (reg, num)	→ reg	ADD reg,num,reg
+ (reg, reg)	→ reg	ADD reg,reg,reg
* (reg, reg)	→ reg	MUL reg,reg,reg
♦ + (reg, num)	→ reg	LOAD [reg+num],reg
♦ reg	→ reg	LOAD [reg],reg
num	→ reg	SET num,reg

Pattern	Replacement	Code Template
	→ done	ST reg,[reg]
	→ done	ST reg,[reg+num]

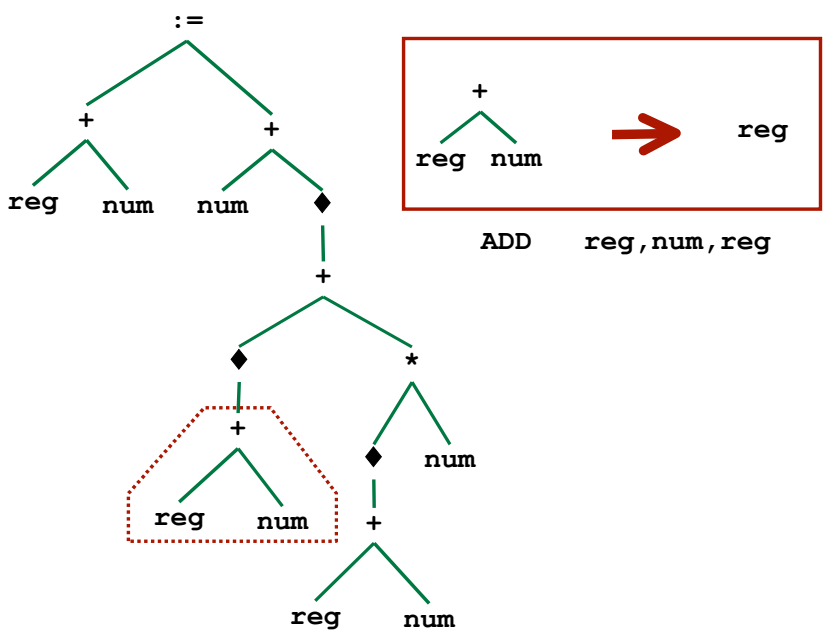
Tiling The Tree



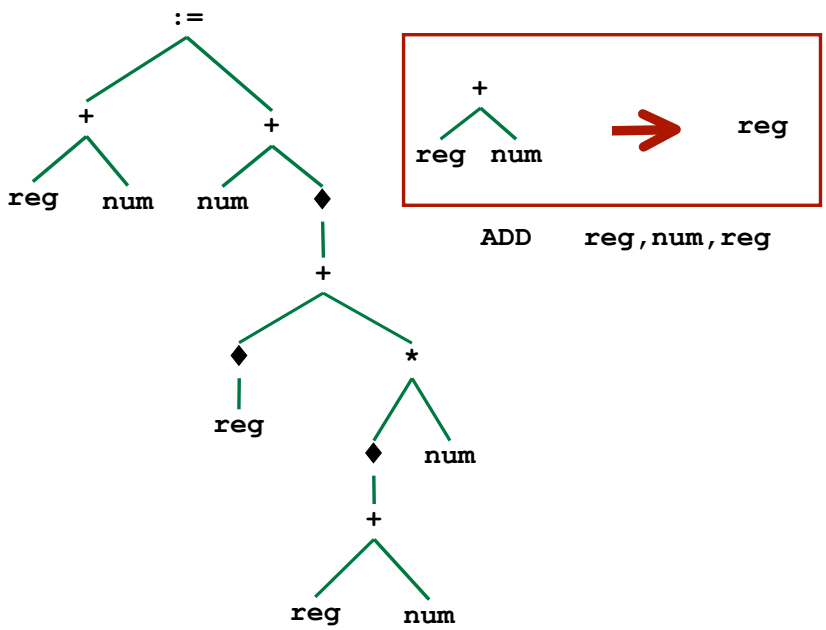
Tiling The Tree



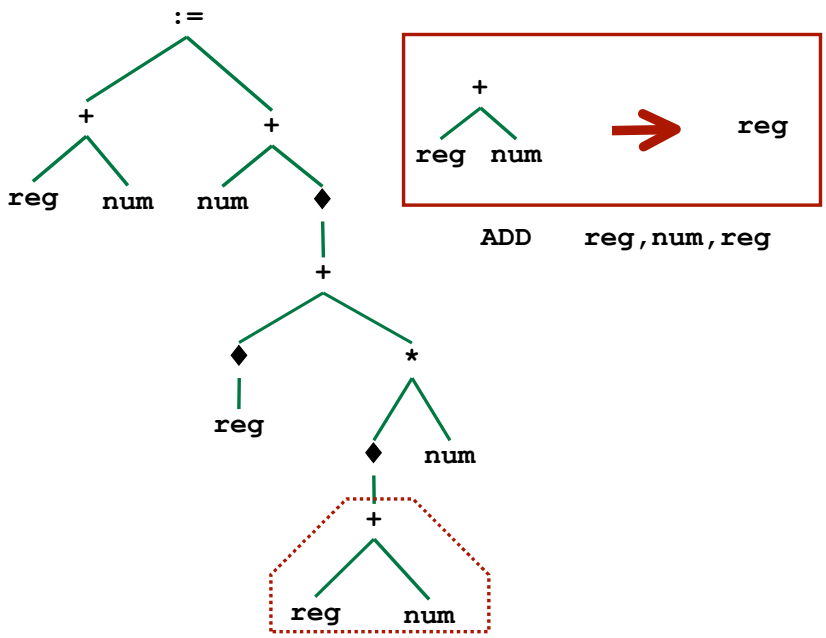
Tiling The Tree



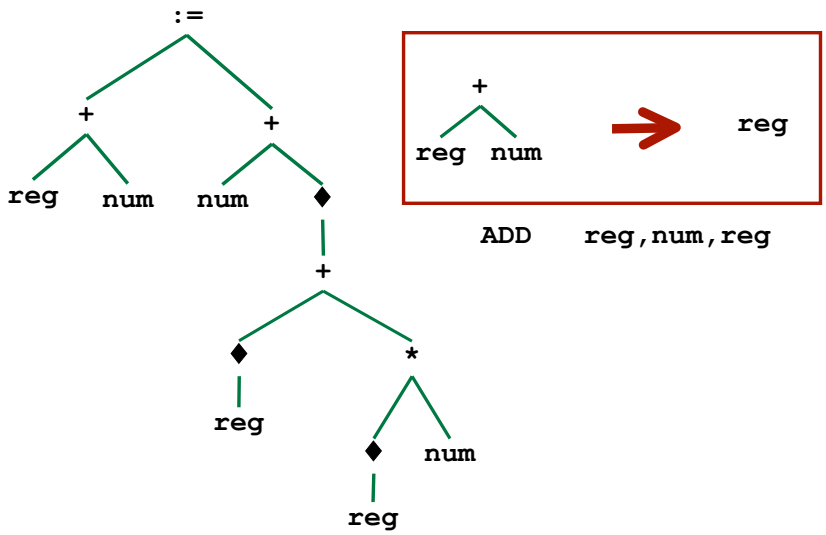
Tiling The Tree



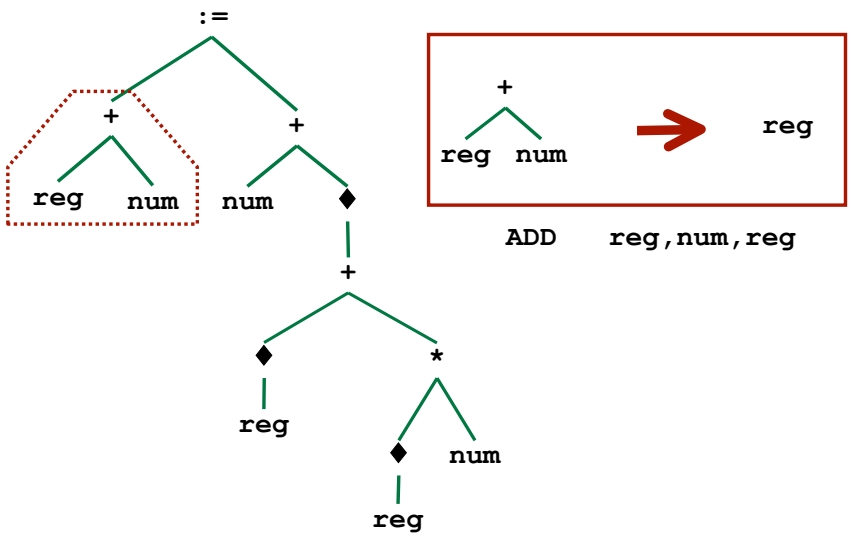
Tiling The Tree



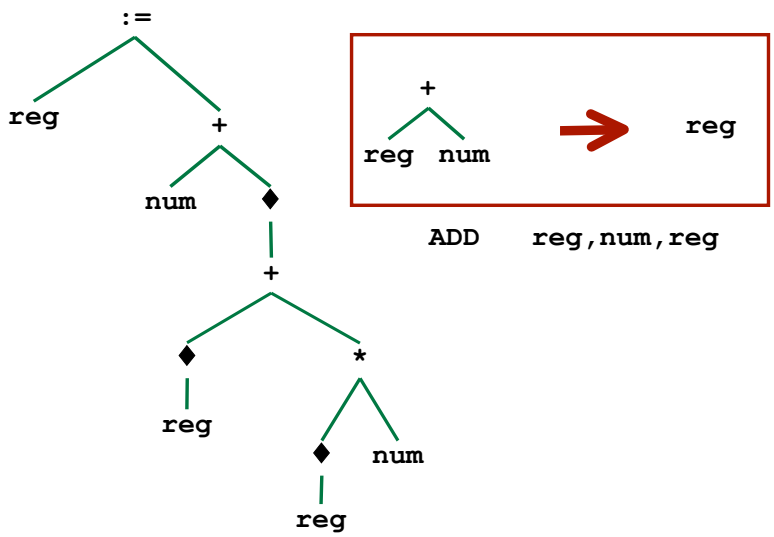
Tiling The Tree



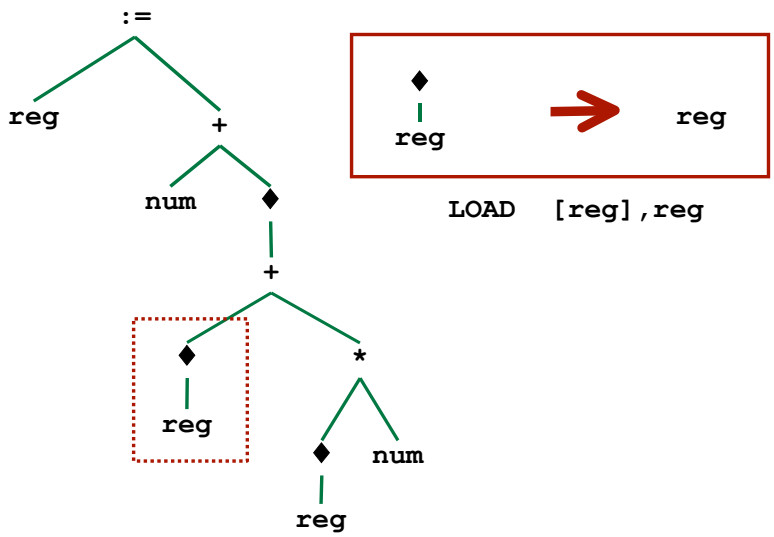
Tiling The Tree



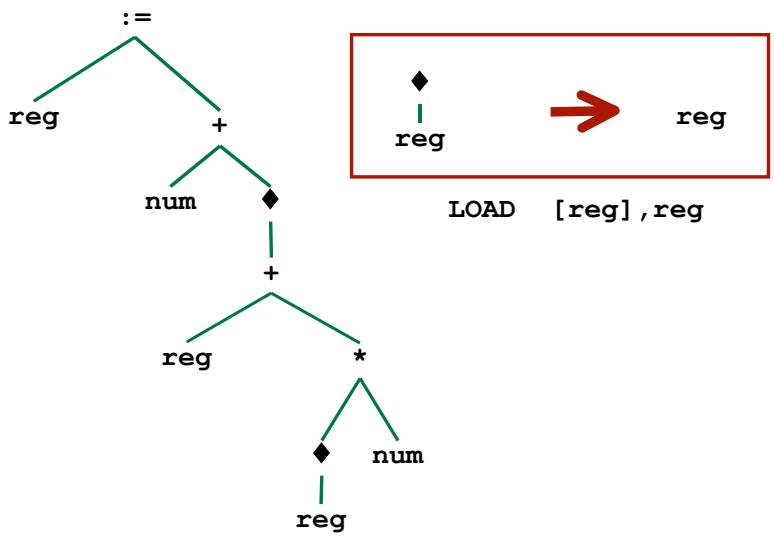
Tiling The Tree



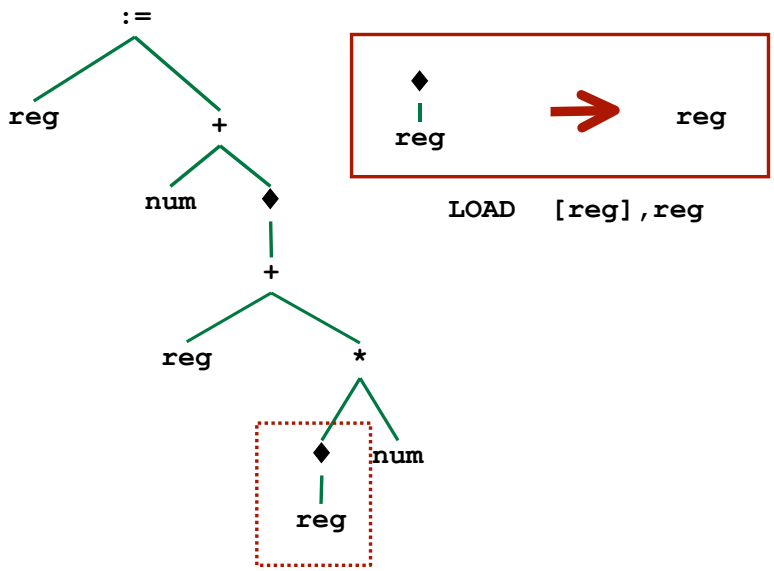
Tiling The Tree



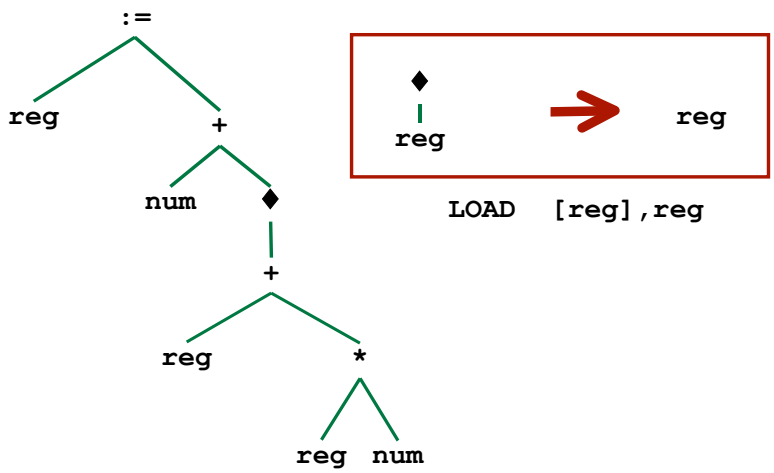
Tiling The Tree



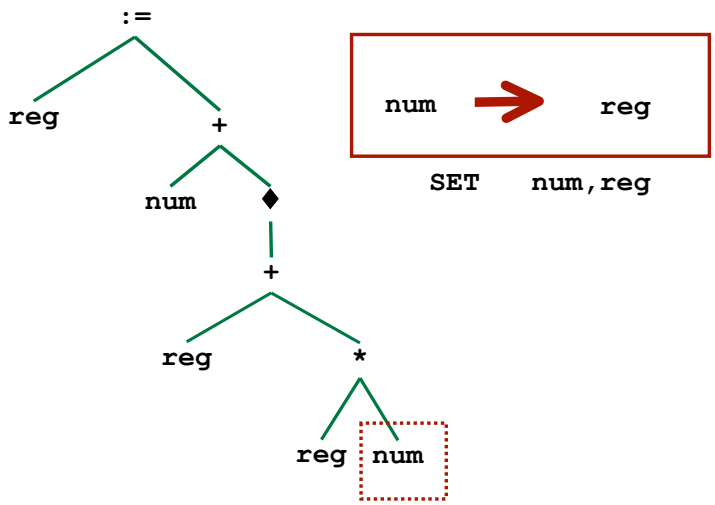
Tiling The Tree



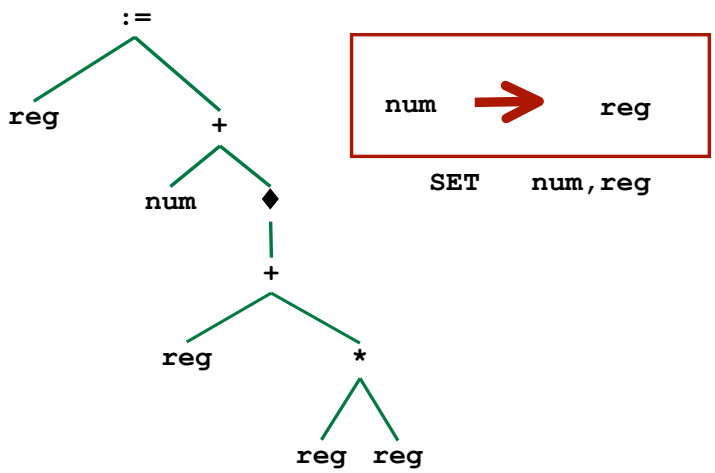
Tiling The Tree



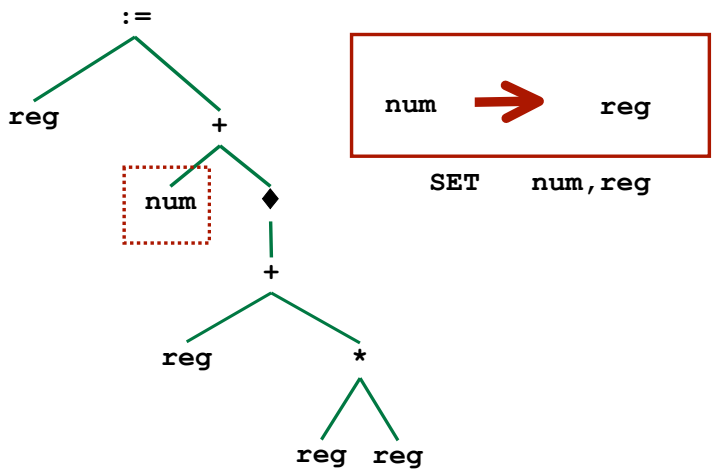
Tiling The Tree



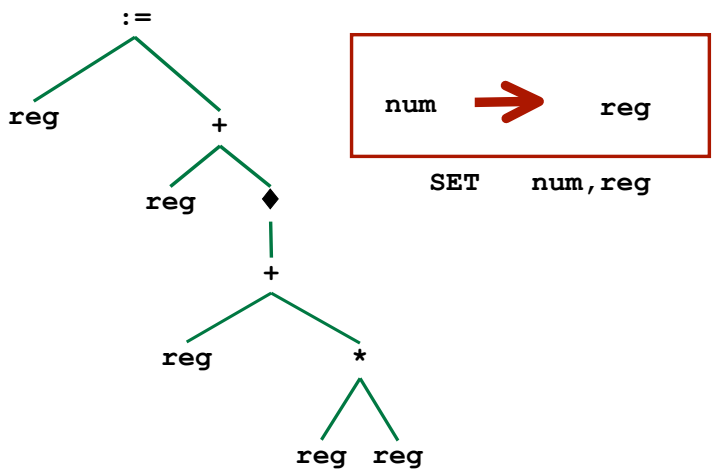
Tiling The Tree



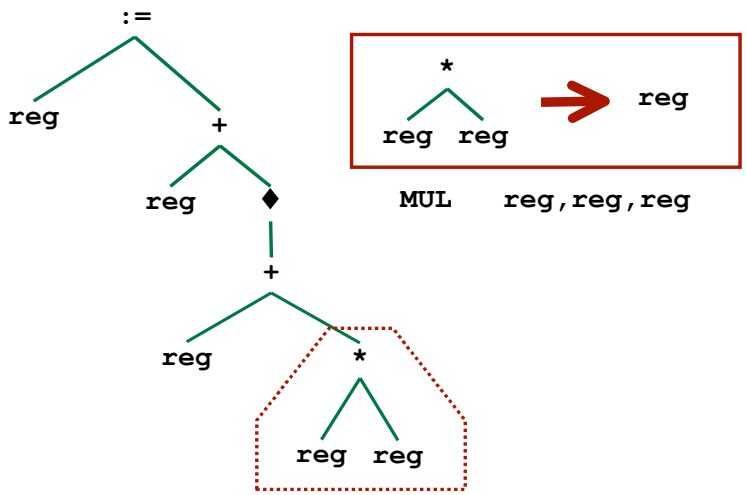
Tiling The Tree



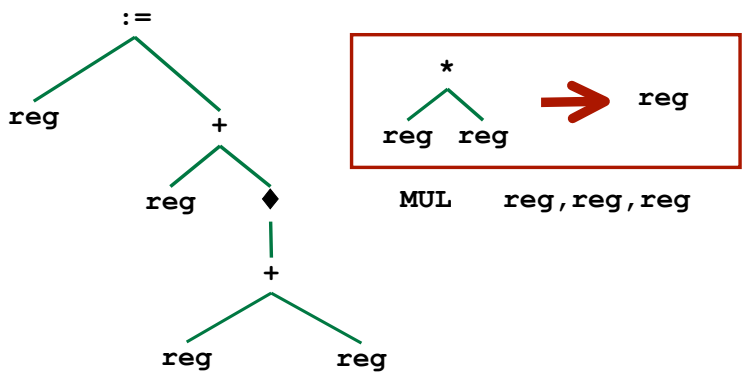
Tiling The Tree



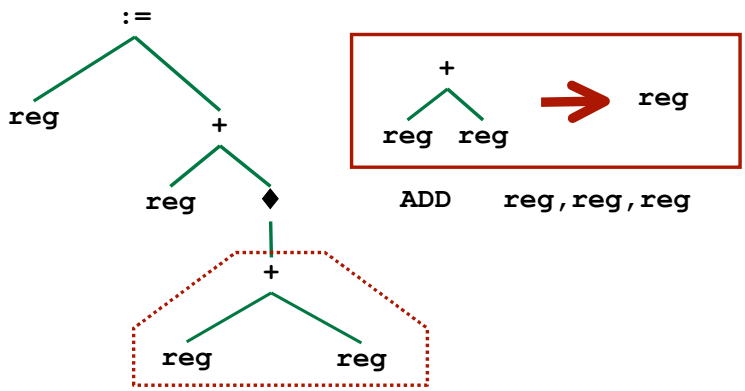
Tiling The Tree



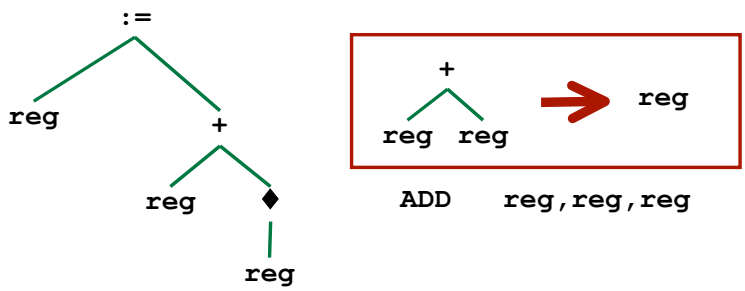
Tiling The Tree



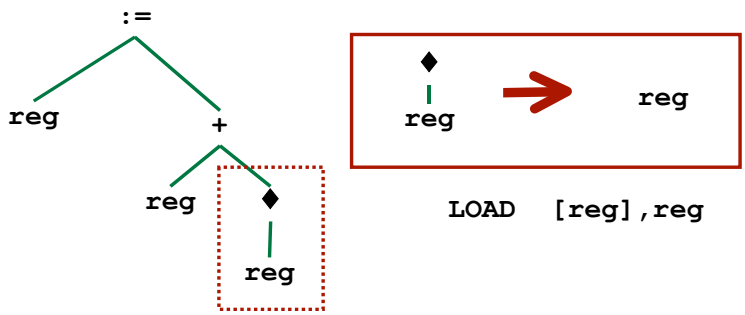
Tiling The Tree



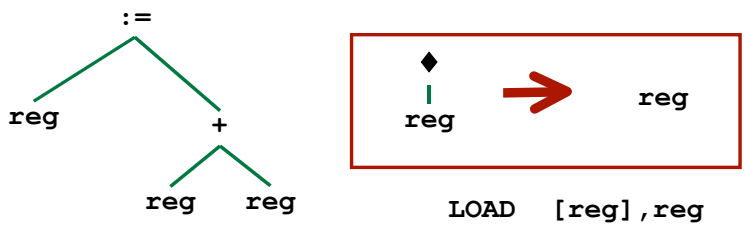
Tiling The Tree



Tiling The Tree

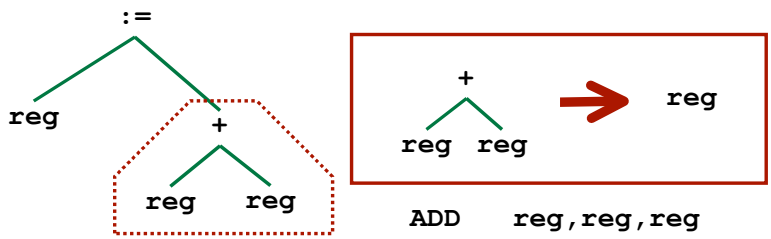


Tiling The Tree



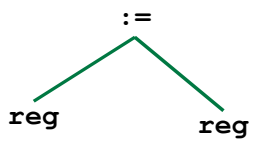

```
LOAD   [reg],reg
```


Tiling The Tree



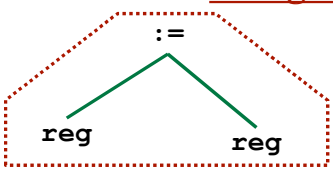

```
ADD    reg,reg,reg
```

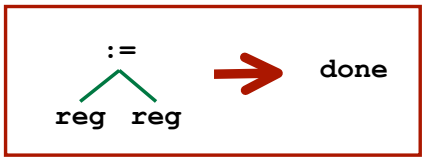
Tiling The Tree




```
ADD    reg,reg,reg
```


Tiling The Tree

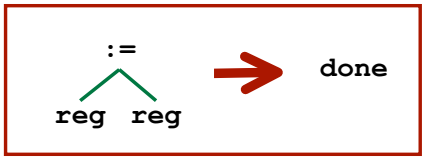





```
ST     reg,[reg]
```

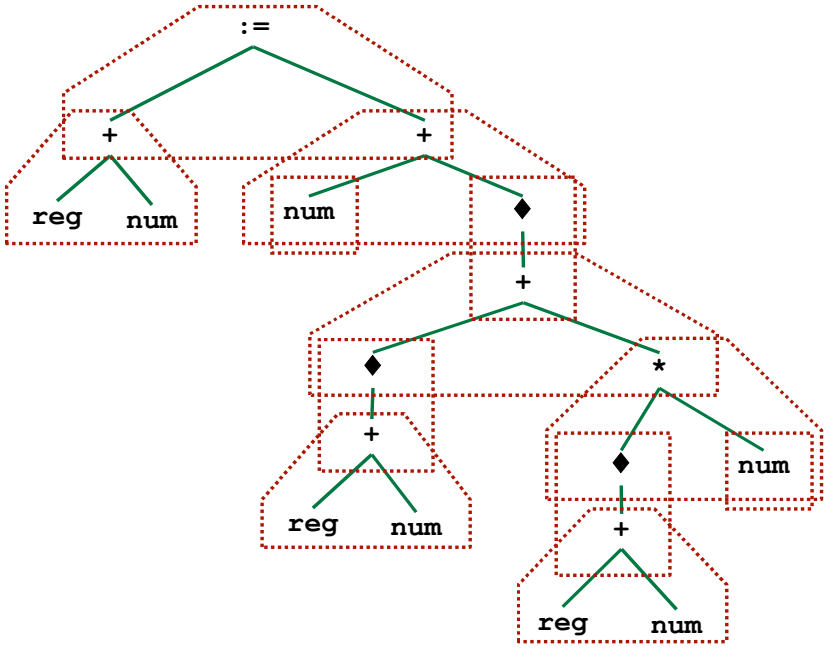
Tiling The Tree

done

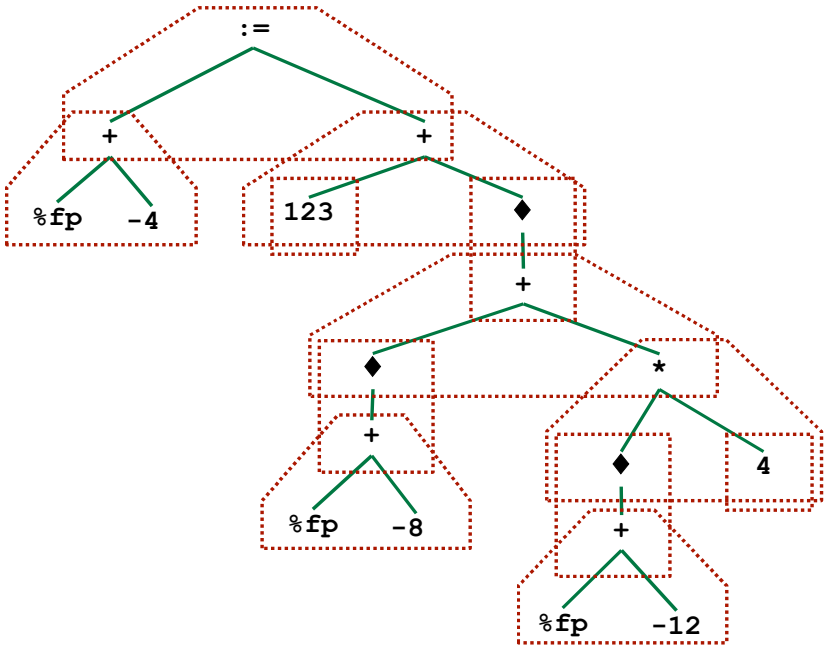


ST reg,[reg]

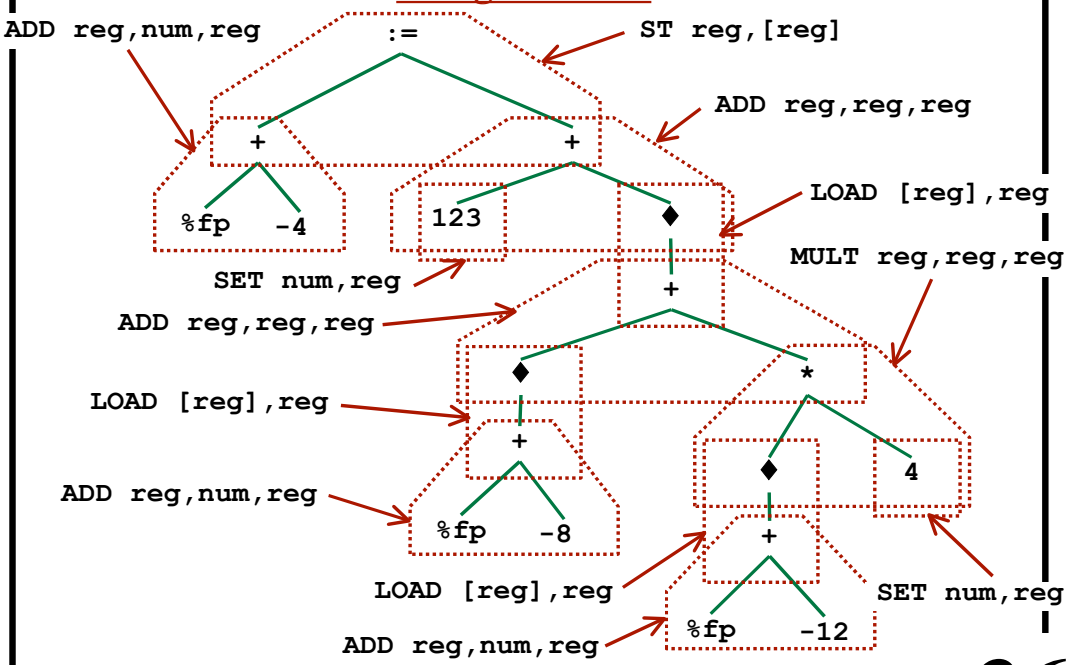
Tiling The Tree



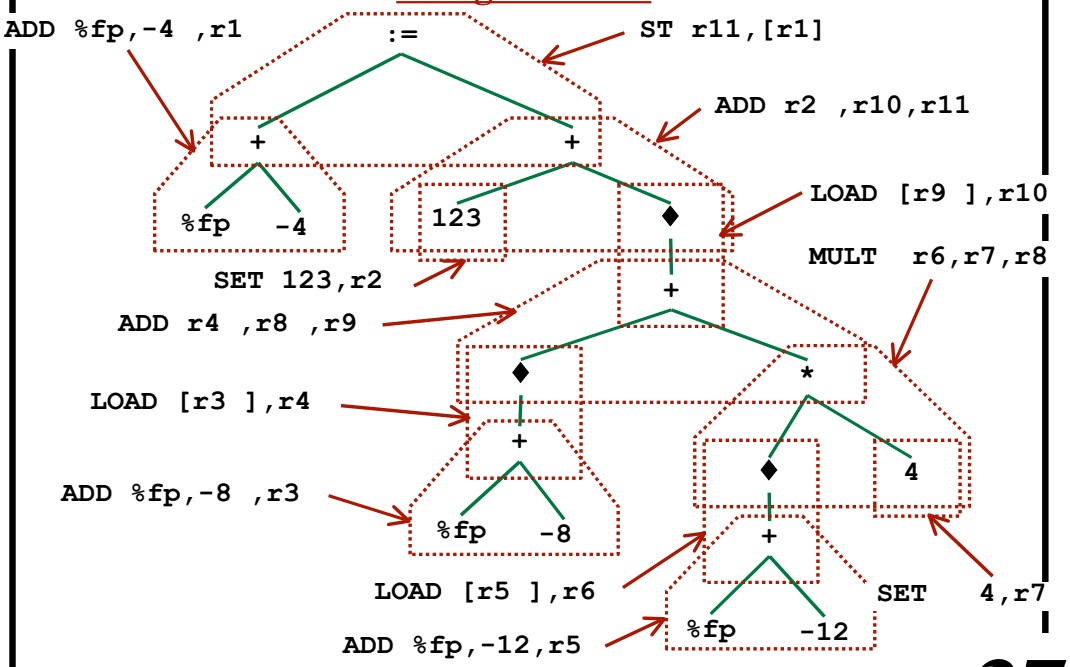
Tiling The Tree



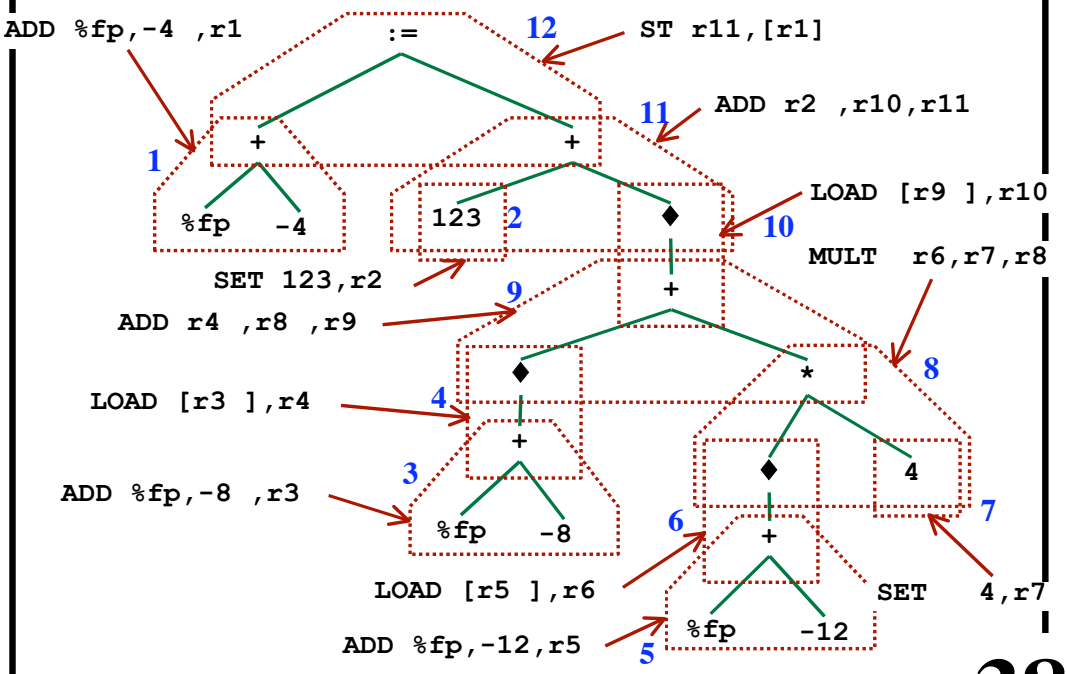
Tiling The Tree



Tiling The Tree



Do a Post-Order Tree Traversal



Do a Post-Order Tree Traversal

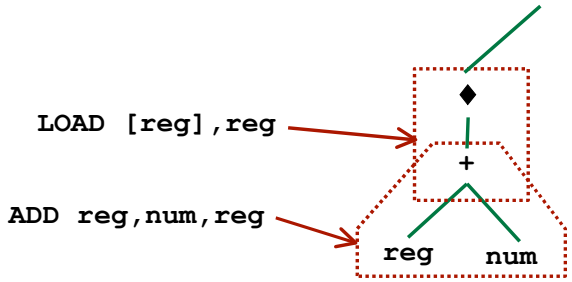
```
ADD    %fp,-4,r1
SET    123,r2
ADD    %fp,-8,r3
LOAD   [r3],r4
ADD    %fp,-12,r5
LOAD   [r5],r6
SET    4,r7
MULT   r6,r7,r8
ADD    r4,r8,r9
LOAD   [r9],r10
ADD    r2,r10,r11
ST     r11,[r1]
```

Source Code:

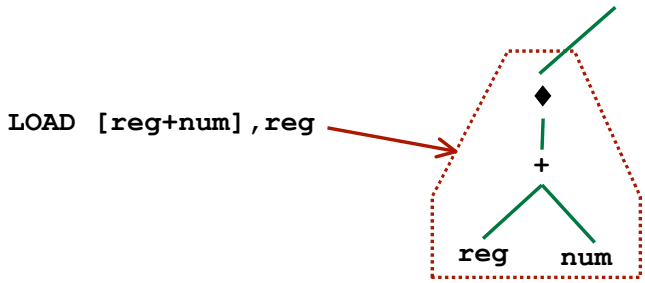
```
x := 123 * a[i];
```

IR Code:

```
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
```

There may be several ways to tile the tree!



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.

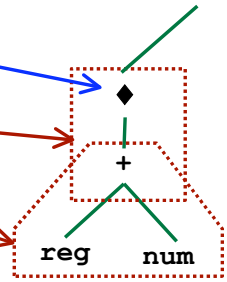
+ (reg num)	→	reg	ADD	reg,num,reg	Cost=1
♦ (reg)	→	reg	LOAD	[reg],reg	Cost=1
♦ + (reg num)	→	reg	LOAD	[reg+num],reg	Cost=1.5

+ (reg num)	→	reg	ADD	reg,num,reg	Cost=1
♦ (reg)	→	reg	LOAD	[reg],reg	Cost=1
♦ + (reg num)	→	reg	LOAD	[reg+num],reg	Cost=1.5

Cost to get result into "reg" at this point = 2

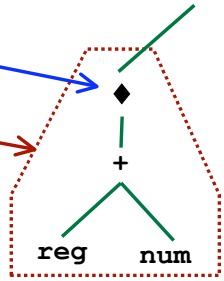
LOAD [reg],reg
Cost =1

ADD reg,num,reg
Cost=1



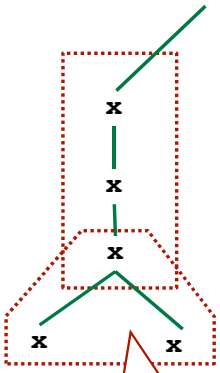
Cost to get result into "reg" at this point = 1.5

`LOAD [reg+num],reg`
Cost=1.5



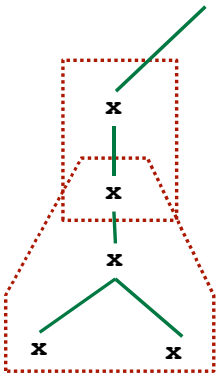
Different Ways to Tile a Tree

Tiling #1



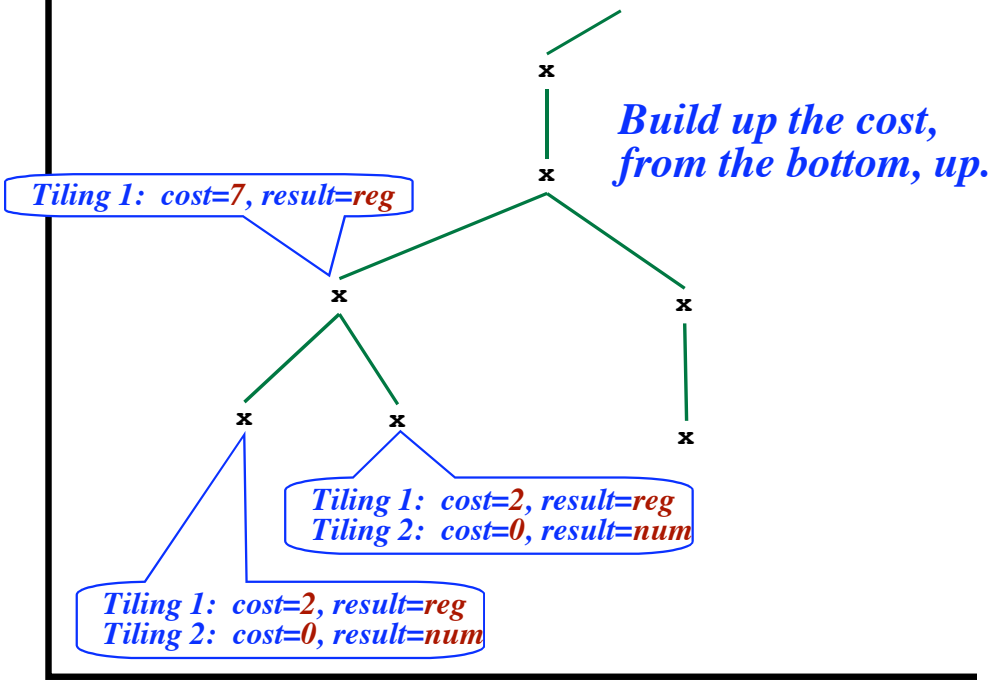
These two rules are compatible with each other, and can be used together to create a tiling.

Tiling #2

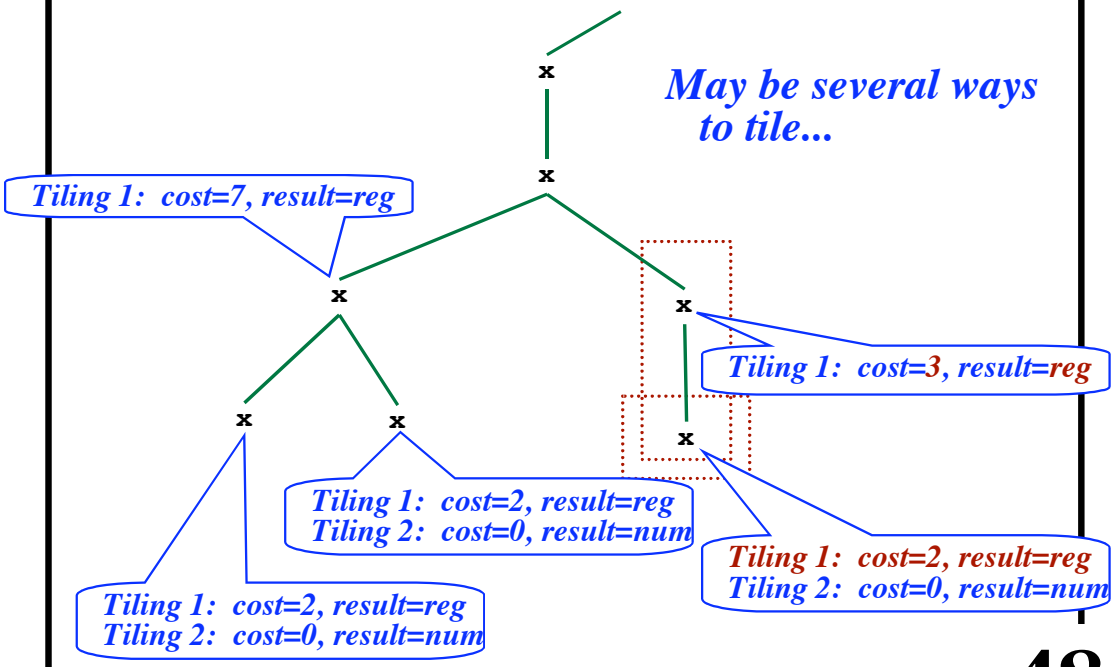


Two different rules are used in this tiling.

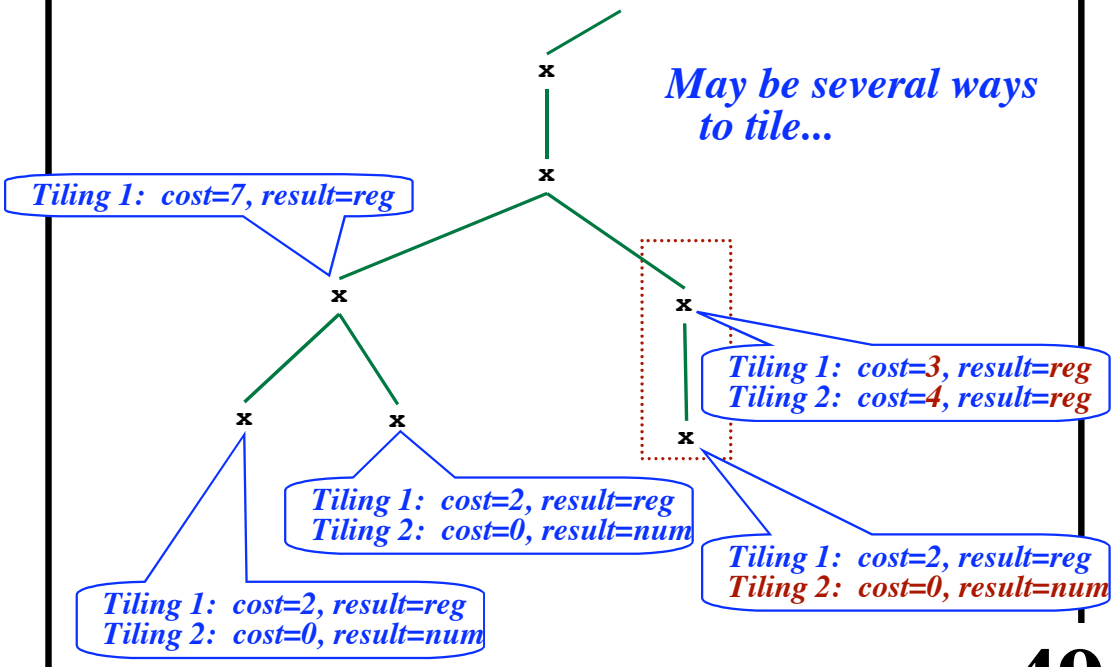
Dynamic Programming Approaches



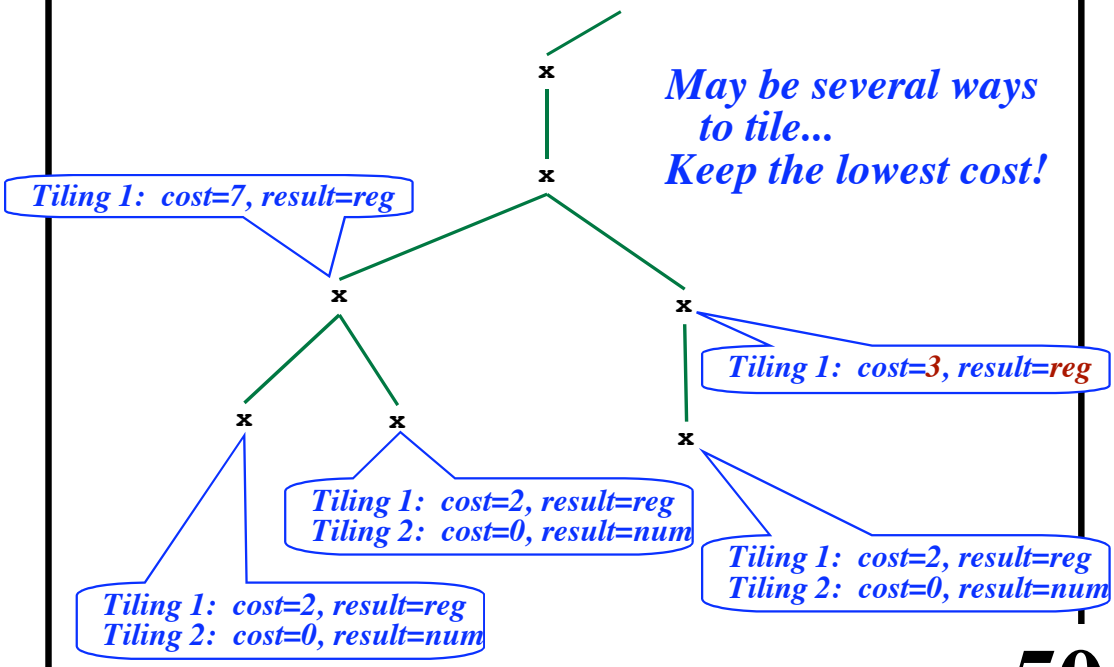
Dynamic Programming Approaches



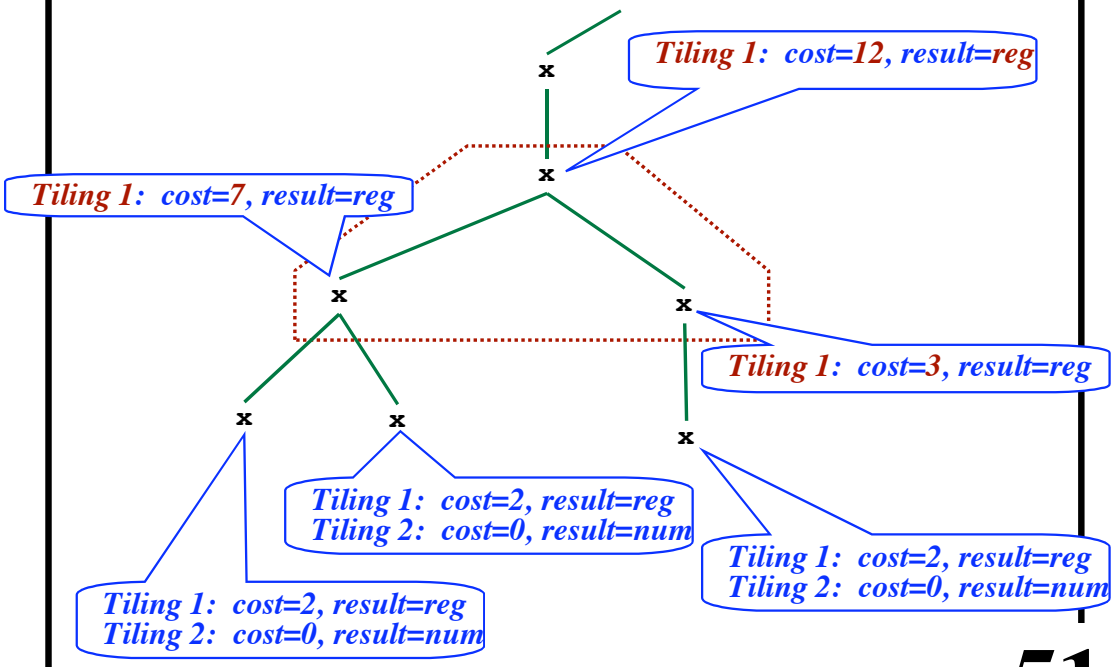
Dynamic Programming Approaches



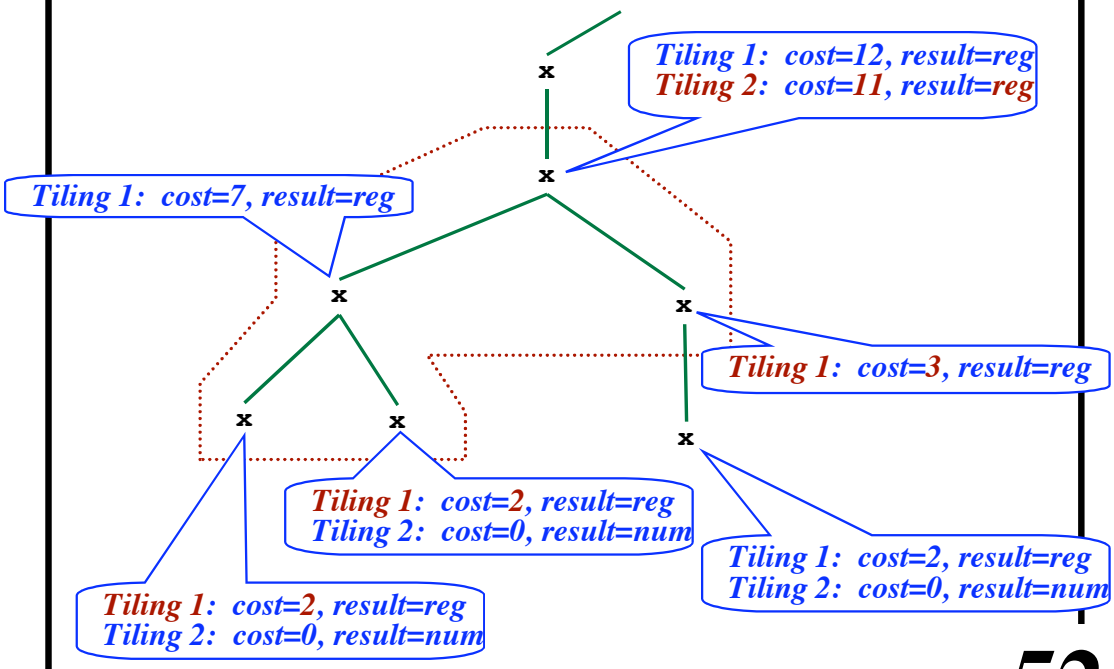
Dynamic Programming Approaches



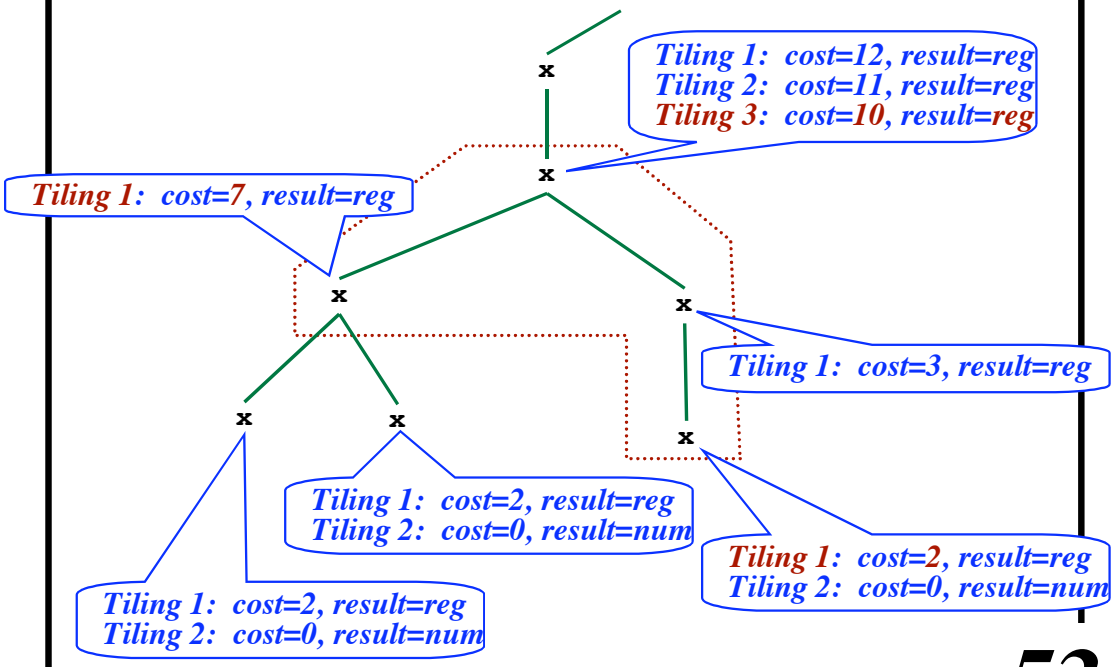
Dynamic Programming Approaches



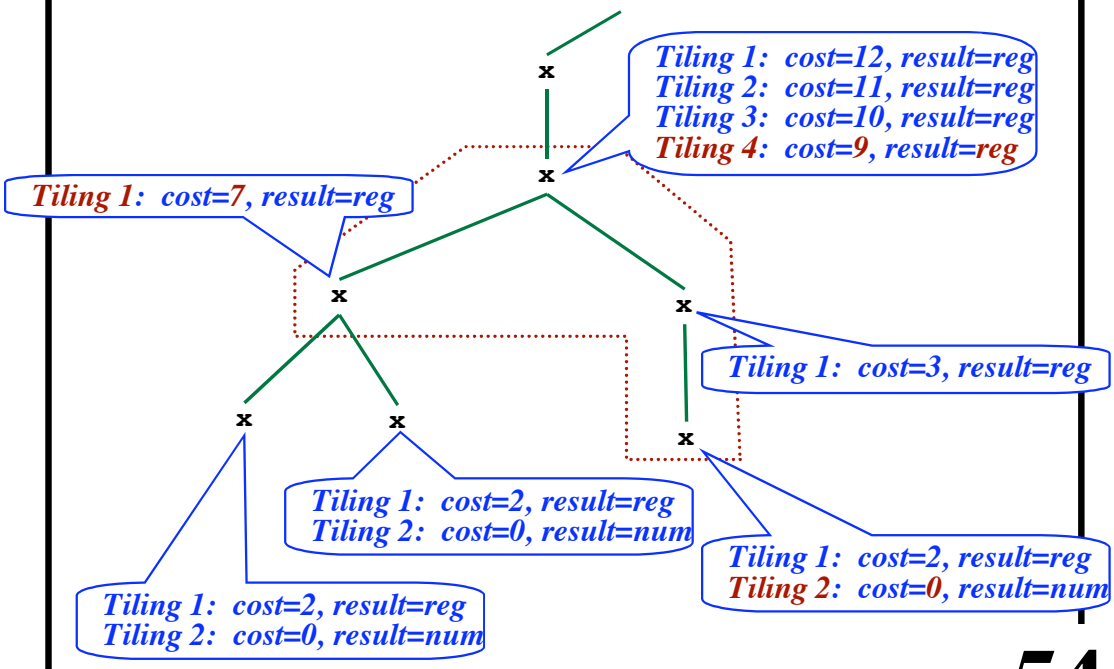
Dynamic Programming Approaches



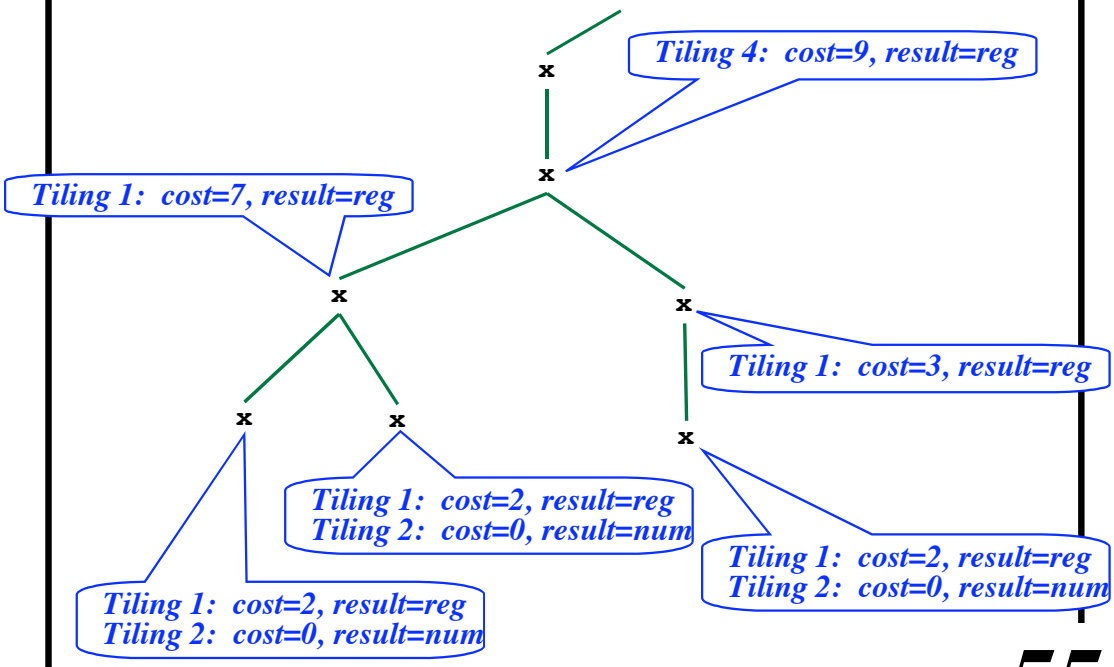
Dynamic Programming Approaches



Dynamic Programming Approaches

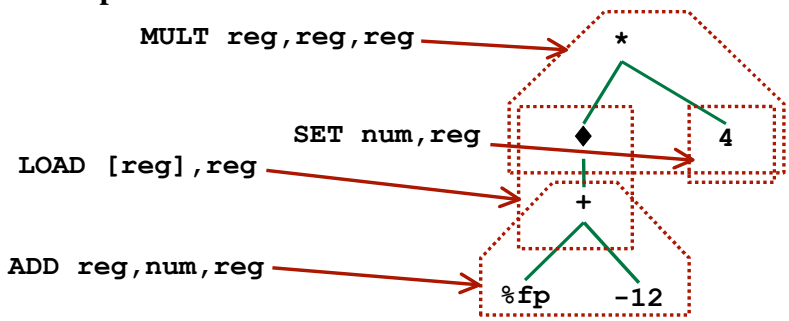


Dynamic Programming Approaches



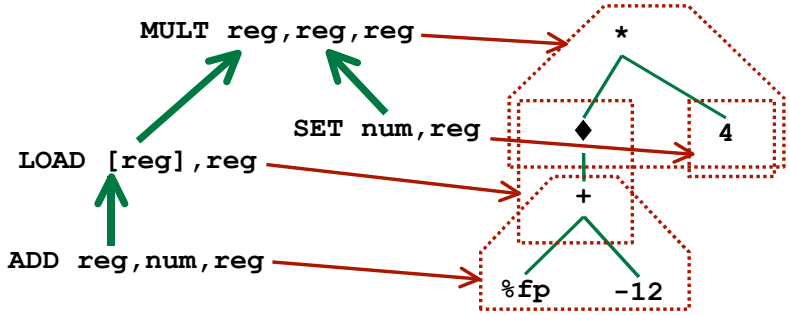
Ordering Constraints

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



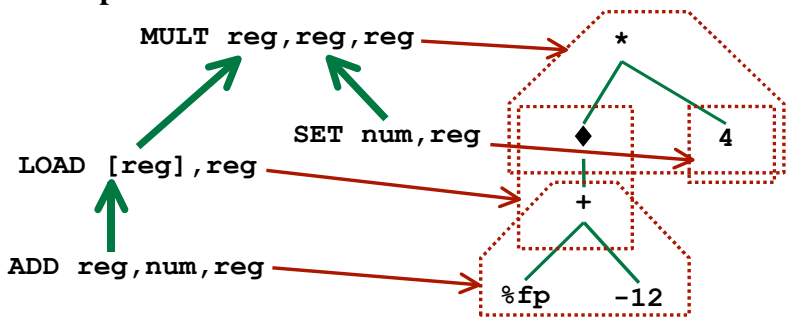
Ordering Constraints

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



Ordering Constraints

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




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

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

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

Instruction Scheduling

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

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]
```

A LOAD is done here

The result is needed here

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

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]
```

```
ADD    %fp,-4,r1
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
SET    123,r2
ADD    r2,r10,r11
ST     r11,[r1]
```

Reorder the instructions!