

CS412/413

Introduction to Compilers Radu Rugina

Lecture 26: Instruction Selection 01 Apr 02

Instruction Selection

- Different sets of instructions in low-level IR and in the target machine
- **Instruction selection** = translate low-level IR to assembly instructions on the target machine
- **Straightforward solution**: translate each low-level IR instruction to a sequence of machine instructions
- Example:

$x = y + z$ \Rightarrow `mov y, r1`
`mov z, r2`
`add r2, r1`
`mov r1, x`

CS 412/413 Spring 2002

Introduction to Compilers

2

Instruction Selection

- **Problem**: straightforward translation is inefficient
 - One machine instruction may perform the computation in multiple low-level IR instructions
- Consider a machine with includes the following instructions:
`add r2, r1` $r1 \leftarrow r1+r2$
`mulc c, r1` $r1 \leftarrow r1*c$
`load r2, r1` $r1 \leftarrow [r2]$
`store r2, r1` $[r1] \leftarrow r2$
`movem r2, r1` $[r1] \leftarrow [r2]$
`movex r3, r2, r1` $[r1] \leftarrow [r2+r3]$

CS 412/413 Spring 2002

Introduction to Compilers

3

Example

- Consider the computation: $a[i] = b[j]$
- Assume a, b, i, j are global variables
register ra holds address of a
register rb holds address of b
register ri holds value of i
register rj holds value of j

Low-level IR:

```
t1 = addr b
t2 = j*4
t3 = t1+t2
t4 = [t3]
t5 = addr a
t6 = i*4
t7 = t5+t6
[t7] = t4
```

CS 412/413 Spring 2002

Introduction to Compilers

4

Possible Translation

- Address of $b[j]$: `mulc 4, rj`
`add rj, rb`
- Load value $b[j]$: `load rb, r1`
- Address of $a[i]$: `mulc 4, ri`
`add ri, ra`
- Store into $a[i]$: `store r1, ra`

Low-level IR:

```
t1 = addr b
t2 = j*4
t3 = t1+t2
t4 = [t3]
t5 = addr a
t6 = i*4
t7 = t5+t6
[t7] = t4
```

CS 412/413 Spring 2002

Introduction to Compilers

5

Another Translation

- Address of $b[j]$: `mulc 4, rj`
`add rj, rb`
- Address of $a[i]$: `mulc 4, ri`
`add ri, ra`
- Load and store: `movem rb, ra`

Low-level IR:

```
t1 = addr b
t2 = j*4
t3 = t1+t2
t4 = [t3]
t5 = addr a
t6 = i*4
t7 = t5+t6
[t7] = t4
```

CS 412/413 Spring 2002

Introduction to Compilers

6

Yet Another Translation

- Index value: `mulc 4, rj`
- Address of `a[i]`: `mulc 4, ri`
`add ri, ra`
- Load and store: `movex rj, rb, ra`

Low-level IR:

```
t1 = addr b
t2 = j*4
t3 = t1+t2
t4 = [t3]
t5 = addr a
t6 = i*4
t7 = t5+t6
[t7] = t4
```

CS 412/413 Spring 2002

Introduction to Compilers

7

Issue: Instruction Costs

- Different machine instructions have different *costs*
 - Time cost: how fast instructions are executed
 - Space cost: how much space instructions take

- Example: cost = number of cycles

<code>add r2, r1</code>	cost=1
<code>mulc c, r1</code>	cost=10
<code>load r2, r1</code>	cost=3
<code>store r2, r1</code>	cost=3
<code>movem r2, r1</code>	cost=4
<code>movex r3, r2, r1</code>	cost=5

- Goal: find translation with smallest cost

CS 412/413 Spring 2002

Introduction to Compilers

8

How to Solve the Problem?

- **Difficulty:** low-level IR instruction matched by a machine instructions may not be adjacent

- Example: `movem rb, ra`

- **Idea:** use tree-like representation!
 - Easier to detect matching instructions

Low-level IR:

```
t1 = addr b
t2 = j*4
t3 = t1+t2
t4 = [t3]
t5 = addr a
t6 = i*4
t7 = t5+t6
[t7] = t4
```

CS 412/413 Spring 2002

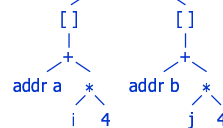
Introduction to Compilers

9

Tree Representation

- **Goal:** determine parts of the tree which correspond to machine instructions

`a[i] = b[j]`



Low-level IR:

```
t1 = addr b
t2 = j*4
t3 = t1+t2
t4 = [t3]
t5 = addr a
t6 = i*4
t7 = t5+t6
[t7] = t4
```

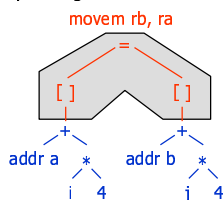
CS 412/413 Spring 2002

Introduction to Compilers

10

Tiles

- **Tile** = tree patterns (subtrees) corresponding to machine instructions



Low-level IR:

```
t1 = addr b
t2 = j*4
t3 = t1+t2
t4 = [t3]
t5 = addr a
t6 = i*4
t7 = t5+t6
[t7] = t4
```

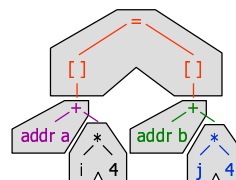
CS 412/413 Spring 2002

Introduction to Compilers

11

Tiling

- **Tiling** = find the set of disjoint tiles that covers the tree



Machine code:

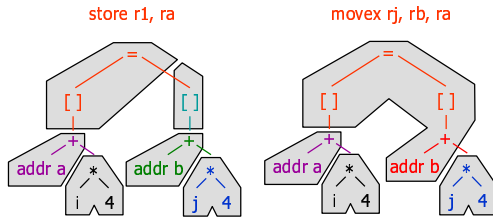
```
mulc 4, rj
add rj, rb
mulc 4, ri
add ri, ra
movem rb, ra
```

CS 412/413 Spring 2002

Introduction to Compilers

12

Other Possible Tilings



CS 412/413 Spring 2002

Introduction to Compilers

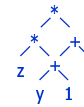
13

Directed Acyclic Graphs

- **Tree representation:** appropriate for instruction selection
 - Tiles = subtrees → machine instructions
- **DAG =** more general structure for representing instructions
 - Common sub-expressions represented by the same node
 - Tile the expression DAG

- **Example:**

```
t = y+1
y = z*t
t = t+1
z = t*y
```



CS 412/413 Spring 2002

Introduction to Compilers

14

Big Picture

- **What the compiler has to do:**
 1. Translate low-level IR code into DAG representation
 2. Then find a good tiling of the DAG
 - Maximal munch algorithm
 - Dynamic programming algorithm

CS 412/413 Spring 2002

Introduction to Compilers

15

DAG Construction

- **Input:** a sequence of low IR instructions in a basic block
- **Output:** an expression DAG for the block
- **Idea:**
 - Label each DAG node with variable which holds that value
 - Build DAG bottom-up
- **Problem:** a variable may have multiple values in a block
- **Solution:** use different variable indices for different values of the variable: $t_0, t_1, t_2,$ etc.

CS 412/413 Spring 2002

Introduction to Compilers

16

Algorithm

```
index[v] = 0 for each variable v
For each instruction I (in the order they appear)
  For each v that I directly uses, with n=index[v]
    if node vn doesn't exist
      create node vn, with label vn
  Create expression node for instruction I, with children
  { vn | v ∈ use[I] }
  For each v ∈ def[I]
    index[v] = index[v] + 1
  If I is of the form x = ... and n = index[x]
    label the new node with xn
```

CS 412/413 Spring 2002

Introduction to Compilers

17

Issues

- **Function calls**
 - May update any global variable
 - def[I] = set of global variables
- **Store instructions**
 - May update any variable
 - If stack addresses are not taken (e.g. Java), def[I] = set of heap variables

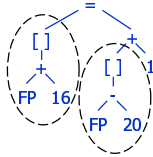
CS 412/413 Spring 2002

Introduction to Compilers

18

Local Variables in DAG

- Use stack pointers to access local variables
- Example: $x = y + 1$



CS 412/413 Spring 2002

Introduction to Compilers

19

Next: DAG Tiling

- **Goal:** find a good covering of DAG with tiles
- **Problem:** need to know what variables are in registers
- **Assume abstract assembly:**
 - Machine with infinite number of registers
 - Temporary variables stored in registers
 - Local/global/heap variables: use memory accesses

CS 412/413 Spring 2002

Introduction to Compilers

20

Problems

- **Classes of registers**
 - Registers may have specific purposes
 - Example: Pentium multiply instruction
 - multiply register eax by contents of another register
 - store result in eax (low 32 bits) and edx (high 32 bits)
 - need extra instructions to move values into eax
- **Two-address machine instructions**
 - Three-address low-level code
 - Need multiple machine instructions for a single tile
- **CISC versus RISC**
 - Complex instruction sets => many possible tiles and tilings
 - Example: multiple addressing modes (CISC) versus load/store architectures (RISC)

CS 412/413 Spring 2002

Introduction to Compilers

21

Pentium ISA

- **Pentium:** two-address CISC architecture
- **General-purpose registers:** eax, ebx, ecx, edx, esi, edi
- **Stack registers:** ebp, esp
- **Typical instruction:**
 - Opcode (mov, add, sub, mul, div, jmp, etc)
 - Destination and source operands
- **Multiple addressing modes:** source operands may be
 - Immediate value: imm
 - Register: reg
 - Indirect address: [reg], [imm], [reg+imm],
 - Indexed address: [reg+reg'], [reg+imm*reg'], [reg+imm*reg'+imm']
- Destination operands = same, except immediate values

CS 412/413 Spring 2002

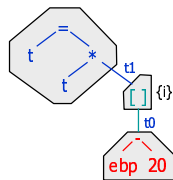
Introduction to Compilers

22

Example Tiling

- Consider: $t = t + i$
 t = temporary variable
 i = local variable
- Need new temporary registers between tiles (unless operand node is labeled with temporary)
- Result code:


```
mov %ebp, t0
sub $20, t0
mov (t0), t1
add t1, t
```
- Note: also compute i , if it is live

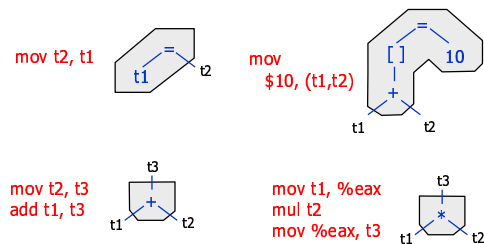


CS 412/413 Spring 2002

Introduction to Compilers

23

Some Tiles



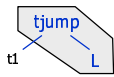
CS 412/413 Spring 2002

Introduction to Compilers

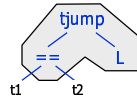
24

Conditional Branches

- How to tile a conditional jump?
- Fold comparison into tile



test t1,t1
jnz L



cmp t1,t2
je L

CS 412/413 Spring 2002

Introduction to Compilers

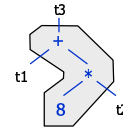
25

Load Effective Address

- Lea instruction computes a memory address
- Doesn't actually load the value from memory



lea (t1,t2), t3



lea (t1,t2,8), t3

CS 412/413 Spring 2002

Introduction to Compilers

26