

CS412/413

Introduction to Compilers
Radu Rugina

Lecture 29: More Instruction Selection
07 Apr 03

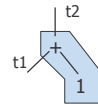
Outline

- Tiles: review
- Maximal munch algorithm
- Some tricky tiles
 - conditional jumps
 - instructions with fixed registers
- Dynamic programming algorithm

Instruction Selection

- Current step: converting low-level intermediate code into abstract assembly
- Implement each IR instruction with a sequence of one or more assembly instructions
- DAG of IR instructions are broken into tiles associated with one or more assembly instructions

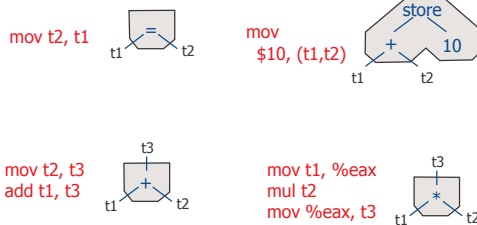
Tiles



mov t1, t2
add \$1, t2

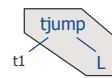
- Tiles capture compiler's understanding of instruction set
- Each tile: sequence of machine instructions that match a subgraph of the DAG
- May need additional move instructions
- Tiling = cover the DAG with tiles

Tiles: Examples

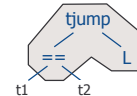


Conditional Branches

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



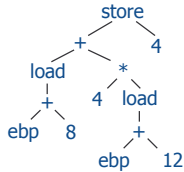
test t1,t1
jnz L



cmp t1,t2
je L

Maximal Munch Algorithm

- Maximal Munch = find largest tiles (greedy algorithm)
- Start from top of tree
- Find largest tile that matches top node
- Tile remaining subtrees recursively



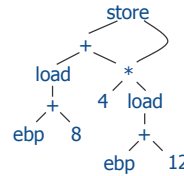
CS 412/413 Spring 2003

Introduction to Compilers

7

DAG Representation

- DAG: a node may have multiple parents
- Algorithm: same, but nodes with multiple parents occur inside tiles only if all parents are in the tile



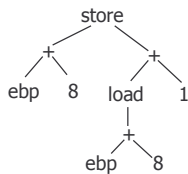
CS 412/413 Spring 2003

Introduction to Compilers

8

Example

`x = x + 1;`



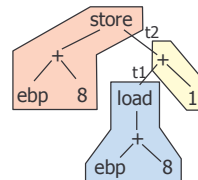
CS 412/413 Spring 2003

Introduction to Compilers

9

Example

`x = x + 1;`



```

mov 8(%ebp), t1
mov t1, t2
add $1, t2
mov t2, 8(%ebp)
    
```

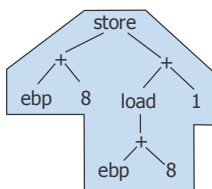
CS 412/413 Spring 2003

Introduction to Compilers

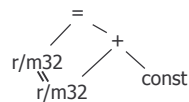
10

Alternate (CISC) Tiling

`x = x + 1;`



```
add $1, 8(%ebp)
```



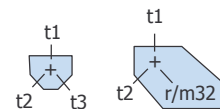
CS 412/413 Spring 2003

Introduction to Compilers

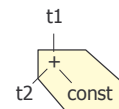
11

ADD Expression Tiles

```
mov t2, t1
add r/m32, t1
```



```
mov t2, t1
add imm32, t1
```



CS 412/413 Spring 2003

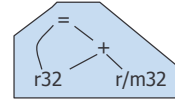
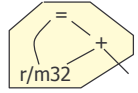
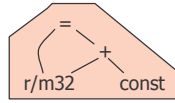
Introduction to Compilers

12

ADD Statement Tiles

Intel Architecture

```
add imm32, %eax
add imm32, r/m32
add imm8, r/m32
add r32, r/m32
add r/m32, r32
```



CS 412/413 Spring 2003

Introduction to Compilers

13

Designing Tiles

- Only add tiles that are useful to compiler
- Many instructions will be too hard to use effectively or will offer no advantage
- Need tiles for all single-node trees to guarantee that every tree can be tiled, e.g.

```
mov t2, t1
add t3, t1
```



CS 412/413 Spring 2003

Introduction to Compilers

14

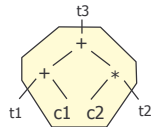
More Handy Tiles

lea instruction computes a memory address

```
lea (t1,t2), t3
```



```
lea c1(t1,t2,c2), t3
```



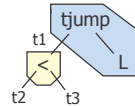
CS 412/413 Spring 2003

Introduction to Compilers

15

Matching Jump for RISC

- As defined in lecture, have
tjump(cond, destination)
fjump(cond, destination)
- Our tjump/fjump translates easily to RISC ISAs that have explicit comparison result



MIPS

```
cmplt t2, t3, t1
br t1, L
```

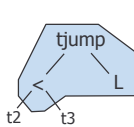
CS 412/413 Spring 2003

Introduction to Compilers

16

Condition Code ISA

- Pentium: condition encoded in jump instruction
- cmp: compare operands and set flags
- jcc: conditional jump according to flags



```
cmp t1, t2
jl L
```

set condition codes

test condition codes

CS 412/413 Spring 2003

Introduction to Compilers

17

Fixed-register instructions

```
mul r/m32
```

Multiply value in register eax

Result: low 32 bits in eax, high 32 bits in edx

```
jecz L
```

Jump to label L if ecx is zero

```
add r/m32, %eax
```

Add to eax

- No fixed registers in low IR except frame pointer
- Need extra move instructions

CS 412/413 Spring 2003

Introduction to Compilers

18

Implementation

- Maximal Munch: start from top node
- Find largest tile matching top node and all of the children nodes
- Invoke recursively on all children of tile
- Generate code for this tile
- Code for children will have been generated already in recursive calls
- How to find matching tiles?

CS 412/413 Spring 2003

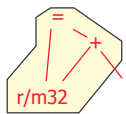
Introduction to Compilers

19

Matching Tiles

```

abstract class LIR_Stmt {
    Assembly munch();
}
class LIR_Assign extends LIR_Stmt {
    LIR_Expr src, dst;
    Assembly munch() {
        if (src instanceof IR_Plus &&
            ((IR_Plus)src).lhs.equals(dst) &&
            is_regmem32(dst)) {
            Assembly e = ((LIR_Plus)src).rhs.munch();
            return e.append(new AddIns(dst,
                e.target()));
        }
        else if ...
    }
}
    
```



CS 412/413 Spring 2003

Introduction to Compilers

20

Tile Specifications

- Previous approach simple, efficient, but hard-codes tiles and their priorities
- Another option: explicitly create data structures representing each tile in instruction set
 - Tiling performed by a generic tree-matching and code generation procedure
 - Can generate from instruction set description: code generator generators
 - For RISC instruction sets, over-engineering

CS 412/413 Spring 2003

Introduction to Compilers

21

How Good Is It?

- Very rough approximation on modern pipelined architectures: execution time is number of tiles
- Maximal munch finds an optimal but not necessarily optimum tiling
- Metric used: tile size

CS 412/413 Spring 2003

Introduction to Compilers

22

Improving Instruction Selection

- Because greedy, Maximal Munch does not necessarily generate best code
 - Always selects largest tile, but not necessarily the fastest instruction
 - May pull nodes up into tiles inappropriately – it may be better to leave below (use smaller tiles)
- Can do better using dynamic programming algorithm

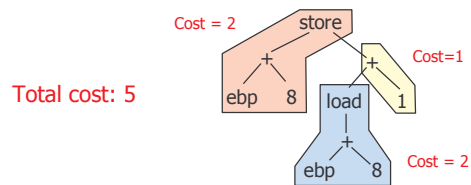
CS 412/413 Spring 2003

Introduction to Compilers

23

Timing Cost Model

- Idea: associate cost with each tile (proportional to number of cycles to execute)
 - may not be a good metric on modern architectures
- Total execution time is sum of costs of all tiles



CS 412/413 Spring 2003

Introduction to Compilers

24

Finding optimum tiling

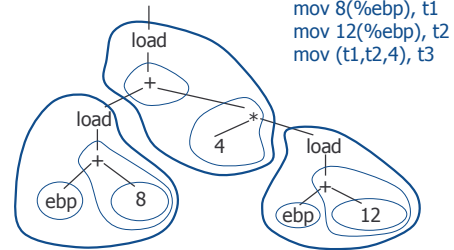
- **Goal:** find minimum total cost tiling of DAG
- **Algorithm:** for every node, find minimum total cost tiling of that node and sub-graph
- **Lemma:** once minimum cost tiling of all nodes in subgraph, can find minimum cost tiling of the node by trying out all possible tiles matching the node
- **Therefore:** start from leaves, work upward to top node

CS 412/413 Spring 2003

Introduction to Compilers

25

Dynamic Programming: a[i]



CS 412/413 Spring 2003

Introduction to Compilers

26

Recursive Implementation

- Dynamic programming algorithm uses memoization
- For each node, record best tile for node
- Start at top, recurse:
 - First, check in table for best tile for this node
 - If not computed, try each matching tile to see which one has lowest cost
 - Store lowest-cost tile in table and return
- Finally, use entries in table to emit code

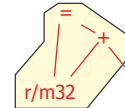
CS 412/413 Spring 2003

Introduction to Compilers

27

Memoization

```
class IR_Move extends IR_Stmt {
    IR_Expr src, dst;
    Assembly best; // initialized to null
    int optTileCost() {
        if (best != null) return best.cost();
        if (src instanceof IR_Plus &&
            ((IR_Plus)src).lhs.equals(dst) && is_regmem32(dst)) {
            int src_cost = ((IR_Plus)src).rhs.optTileCost();
            int cost = src_cost + CISC_ADD_COST;
            if (cost < best.cost())
                best = new AddIns(dst, e.target);
            ...consider all other tiles...
            return best.cost();
        }
    }
}
```



CS 412/413 Spring 2003

Introduction to Compilers

28

Problems with Model

- Modern processors:
 - execution time not sum of tile times
 - instruction order matters
 - Processors pipeline instructions and execute different pieces of instructions in parallel
 - bad ordering (e.g. too many memory operations in sequence) stalls processor pipeline
 - processor can execute some instructions in parallel (super-scalar)
 - cost is merely an approximation
 - instruction scheduling needed

CS 412/413 Spring 2003

Introduction to Compilers

29

Summary

- Can specify code generation process as a set of tiles that relate low IR trees (DAGs) to instruction sequences
- Instructions using fixed registers problematic but can be handled using extra temporaries
- Maximal Munch algorithm implemented simply as recursive traversal
- Dynamic programming algorithm generates better code, can be implemented recursively using memoization
- Real optimization will also require instruction scheduling

CS 412/413 Spring 2003

Introduction to Compilers

30