Lab 2 Tutorial

Comp 412

Copyright 2018, Keith D. Cooper, Linda Torczon & Zoran Budimlić, all rights reserved.

Students enrolled in Comp 412 at Rice University have explicit permission to make copies of these materials for their personal use.

Faculty from other educational institutions may use these materials for nonprofit educational purposes, provided this copyright notice is preserved.
Software Tools for Lab 2

A number of tools to help you build and debug Lab 2

• An ILOC Simulator
  – Essentially, an interpreter for Lab 2 ILOC
  – Matches the Lab 2 specs
    → Single functional unit, latencies as specified, ILOC Lab 1 subset enforced
  – For code check 1, implements →x command line option

• Lab 2 Reference Implementation
  – A functioning Lab 2 allocator, available as an executable black box
  – Has additional functionality relative to your allocator
  – Runs only on CLEAR

• Various scripts, in the scripts directory

• A library of ILOC programs

All of these tools are found on CLEAR in ~comp412/students/lab2
Software Tools for Lab 2

Lab 2 ILOC Simulator

• Takes as input an ILOC program, interprets it, and produces the results
• Can initialize memory from the command line
• Can use ILOC’s output operation to print values from memory to stdout

% cat ex1.iroc
// add two numbers
loadI 314 => r0
loadI 0 => r1
load r1 => r1
add r0,r1 => r2
loadI 0 => r0
store r2 => r0
output 0
% sim -i 0 14 17 < ex1.iroc
328

Executed 7 instructions and 7 operations in 11 cycles.
Software Tools for Lab 2

Lab  ILOC Simulator

% cat ex1.iiloc
// add two numbers
  loadI 314 => r0
  loadI 0 => r1
  load r1 => r1
  add r0,r1 => r2
  loadI 0 => r0
  store r2 => r0
  output 0
%

% sim -t -i 0 14 17 ex1.iiloc
Interlock settings memory registers branches

0:  [loadI 314 => r0 (314)]
1:  [loadI 0 => r1 (0)]
2:  [load r1 (addr: 0) => r1 (14)]
3:  [ stall ]
4:  [ stall ] *2
5:  [add r0 (314), r1 (14) => r2 (328)]
6:  [loadI 0 => r0 (0)]
7:  [store r2 (328) => r0 (addr: 0)]
8:  [ stall ]
9:  [ stall ] *7
10: [output 0 (328)]

output generates => 328

The Lab 2 Simulator has an undocumented flag, -x, that is used for Code Check 1. The -x flag tracks assignments and reports if registers were multiply defined. For example, on cc1.i, it reports 3 reassignments.

Executed 7 instructions and 7 operations in 11 cycles.
Software Tools for Lab 2

Lab 2 Reference Allocator

• The reference implementation shows you what a reasonably good allocator might produce for a given input block

• lab2_ref implements the algorithms from class
  – Implementation in C, so it runs quickly
  – Uses the linked list data structure shown in class (doubly linked)
  – Forward walk (allocation) and backward walk (renaming) are pointer chasing loops
    → Did backward loop with recursion (as in foldr) in an earlier version
    → Speed and runtime stack space proved problematic, even in C

• lab2_ref produces commented output that may be helpful

You can beat lab2_ref in allocation quality

• Every year, several students beat it on several of the report blocks
• Harder to beat it on allocation speed
Software Tools for Lab 2

The Scripts directory has scripts to run and test an allocator

• AllocAndRunAll
  – Applies an allocator to an entire directory of code and produces the results of running the resulting allocated code in the simulator
  – Relies on the SIM INPUT and OUTPUT comments for test blocks
  – Only run one instance at a time; the scripts use scratch files in your home directory that are naively named
  – Copy the scripts and modify your private copies

• CodeCheck
  – Scripts for the CodeChecks are in /clear/courses/comp412/students/lab2/code_check_1 and /clear/courses/comp412/students/lab2/code_check_2
  – They will run five specific ILOC blocks
  – You should run your allocator inside the scripts to test it

There are other scripts in the wild that you can find and use.
What Matters?

In Lab 2, 75% of the points depend on the correctness of your allocator and the quality of the allocated code that your lab produces.

- We measure performance by looking at the total number of cycles that the allocated code uses in the Lab 2 ILOC Simulator\(^1\)
  - Take the unallocated code & run it
  - Take the allocated code & run it
  - Make sure the answers are identical
  - Difference between the two runs is the cost of spill code

- The objective of Lab 2 is to minimize the cost of the spill code
  - The credit for performance is based on how your lab does relative to the labs of other students & relative to the reference implementation
  - Register allocation can introduce a lot of spill code
  - Attention to detail can reduce spill costs
  - But, you cannot always win

- Allocator speed is part of your report grade

So, focus 1\(^{st}\) on the quality of the allocated code & 2\(^{nd}\) on allocator speed.

\(^1\) Lab 3 will use a different simulator, with different latencies order constraints.
Optimizations

The only optimizations allowed are register allocation and NOP elimination

• No other optimizations allowed!
• Specifically, no constant propagation, value numbering etc. You might be tempted, as some code blocks define the input values. Don’t do it!
• The code you generate must perform all the arithmetic operations that the original code does

```assembly
loadI 412   => r0
loadI 26    => r1
add r0, r1  => r2
loadI 0     => r0
store r2    => r0
output 0
```

```assembly
loadI 438   => r2
loadI 0     => r0
store r2    => r0
output 0
```
Strategy

Think ahead

• Code that works for Code Check 1 may not work for Code Check 2 or the final submission
  – Test your code extensively.

• What works for Lab 2 may not work for Lab 3

• For Lab 2, you need to be able to insert spill and restore instructions

• For Lab 3, you need to be able to move instructions around

• Pointer chasing is much faster than hashtable lookup

• Array lookup is much faster than a hashtable lookup

• Make reasonable assumptions and document them
  – i.e. “We don’t expect the number of registers in the source code to be larger than 1 Million”
Pro Tips for Lab 2

Manage your development process for self-protection

• Test your heuristic widely
  – Report blocks, plus one of the timing blocks (all have the same pattern)
  – Libraries of contributed blocks range from trivial to hard
  – Invent new test blocks & share them with your classmates
Pro Tips for Lab 2

Manage your development process for self-protection

• Test your heuristic widely
  – Report blocks, plus one of the timing blocks (all have the same pattern)
  – Libraries of contributed blocks range from trivial to hard
  – Invent new test blocks & share them with your classmates

• Checkpoint your allocator regularly as you change the heuristics
  – Checkpoint each working version
  – Keep them until after the code due date
Pro Tips for Lab 2

Manage your development process for self-protection

• Test your heuristic widely
  – Report blocks, plus one of the timing blocks (all have the same pattern)
  – Libraries of contributed blocks range from trivial to hard
  – Invent new test blocks & share them with your classmates

• Checkpoint your allocator regularly as you change the heuristics
  – Checkpoint each working version
  – Keep them until after the code due date

• Be careful when copying your data structures that hold instructions
  – Safer to just store a flag indicating a “rematerializable” value and the constant representing the value, then generate a loadI instruction
Pro Tips for Lab 2

Manage your development process for self-protection

• Test your heuristic widely
  – Report blocks, plus one of the timing blocks (all have the same pattern)
  – Libraries of contributed blocks range from trivial to hard
  – Invent new test blocks & share them with your classmates

• Checkpoint your allocator regularly as you change the heuristics
  – Checkpoint each working version
  – Keep them until after the code due date

• Be careful when copying your data structures that hold instructions
  – Safer to just store a flag indicating a “rematerializable” value and the constant representing the value, then generate a loadI instruction

• Post your questions to Piazza
  – The “community effect” is powerful