

CSCD70 Compiler Optimization

Tutorial #7 Register Allocation

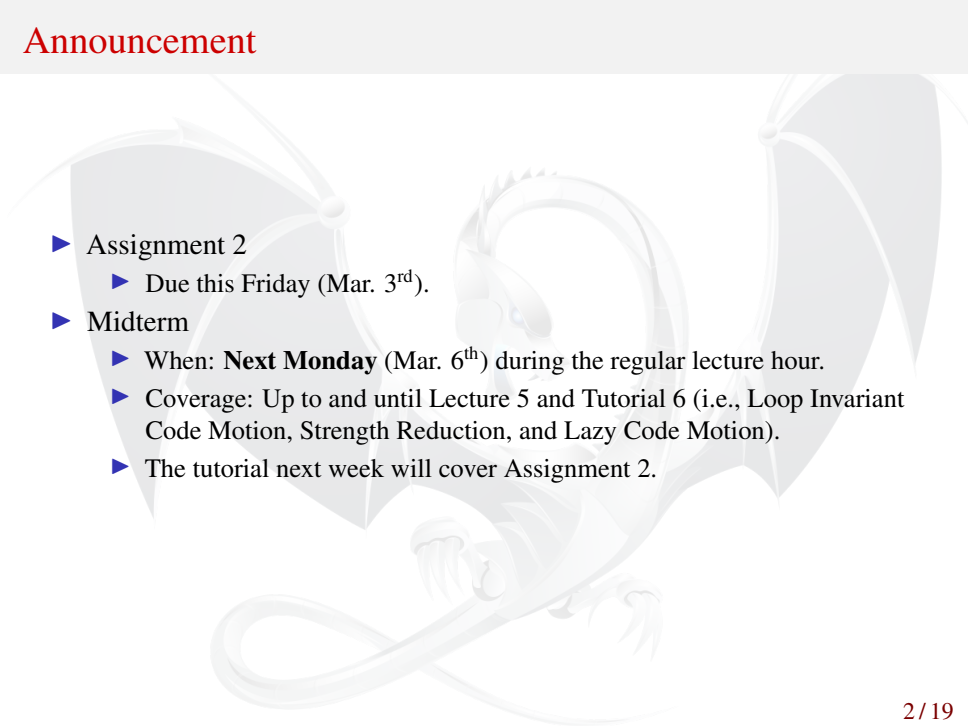
Bojian Zheng

bojian@cs.toronto.edu

Department of Computer Science, University of Toronto

Acknowledgement: Thanks to Professor Gennady Pekhimenko, Professor Nandita Vijaykumar, students from previous offerings of CSCD70, and developers from the LLVM community.

Announcement

- 
- ▶ Assignment 2
 - ▶ Due this Friday (Mar. 3rd).
 - ▶ Midterm
 - ▶ When: **Next Monday** (Mar. 6th) during the regular lecture hour.
 - ▶ Coverage: Up to and until Lecture 5 and Tutorial 6 (i.e., Loop Invariant Code Motion, Strength Reduction, and Lazy Code Motion).
 - ▶ The tutorial next week will cover Assignment 2.

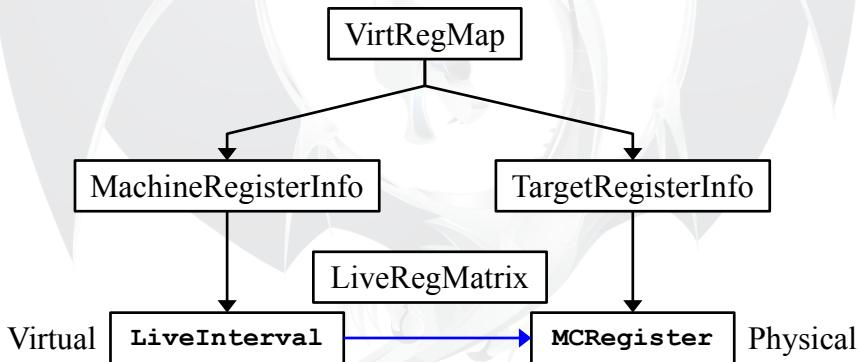
- ▶ How to write a register allocator in LLVM?
- ▶ The Greedy Register Allocator
- ▶ Registers on x86 CPUs



Register Allocator in LLVM

Register Allocator in LLVM

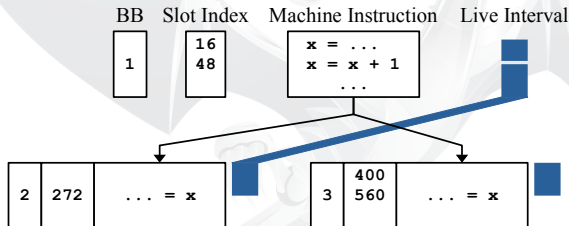
- ▶ Given a machine function, materialize all its **virtual** registers using **physical** registers.
- ▶ Collaboration Diagram:



Live Interval

Each consists of ❶ segments of machine instruction indices (aka. slot indices),
❷ value numbered defsites, and ❸ a spilling weight. E.g.,

❶ $[16r, 48r : 0)[48r, 272r : 1)[400B, 560r : 1)$ ❷ $0@16r, 1@48r$
❸ $\text{weight} : 6.173594e - 03$



High-Level Workflow

Algorithm 1: The Minimal Register Allocator

worklist \leftarrow FIFOQueue();

①

for *each virtual register* $V \in \text{MRI.VirtRegs}()$ **do**

 worklist.push($V.\text{getLiveInterval}()$);

②

while $\text{LI} \leftarrow \text{worklist.pop}()$ **do**

 let P be the first physical register that is available;

 LRM.assign(LI, P);

① Push all virtual registers to the worklist.

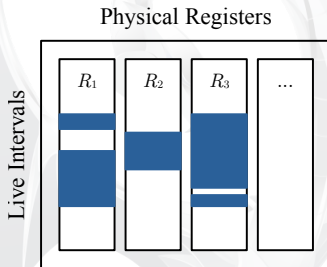
 ▶ What is a better choice for the worklist if the spill weights are known?

② Assign a physical register to each virtual register in the worklist.

 ▶ How to know if one is available? What if none is available?

Live Register Matrix

- ▶ A matrix that keeps track of the register usage in 2D: Live Intervals and Physical Register Units.



- ▶ Each physical register stores a *live interval union* (in blue).

Interference Check

Given a live interval V , query to see if a physical register P is available:

LRM.`checkInterference`(V, P)

- ▶ Free: All clear. Go ahead and assign.
- ▶ VirtReg: Another virtual register U has been assigned to this register. Need to unassign it to proceed.
- ▶ RegUnit: More later ...
- ▶ RegMask

Spilling

- ▶ Move the register's value to a stack slot, and reload it when needed.

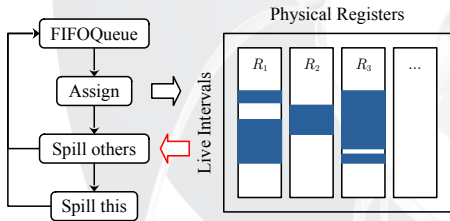


- ▶ The original live interval LI are split into SplitVirtRegs, all of which are appended to the workload.

```
LiveRangeEdit LRE(LI, *SplitVirtRegs, ...);  
SpillerInstance->spill(LRE);
```

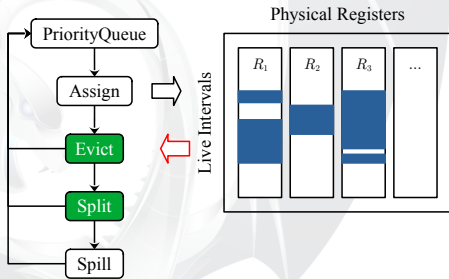
- ▶ Q: How to prevent spilling an already-spilled live interval?

RAMinimal



⇒ Assign ⇐ Unassign

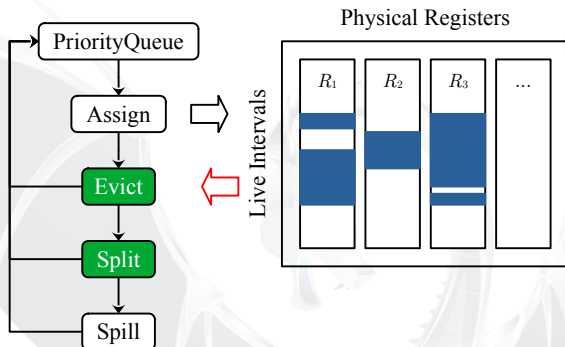
RAGreedy





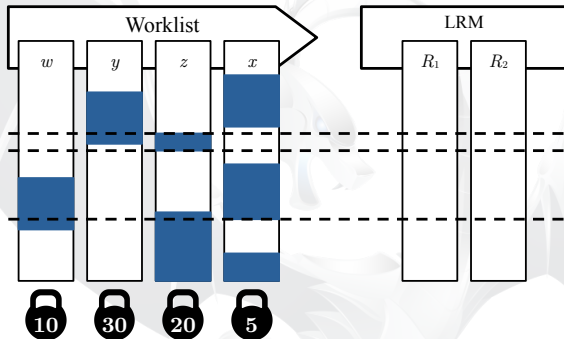
The Greedy Register Allocator

The Greedy Register Allocator

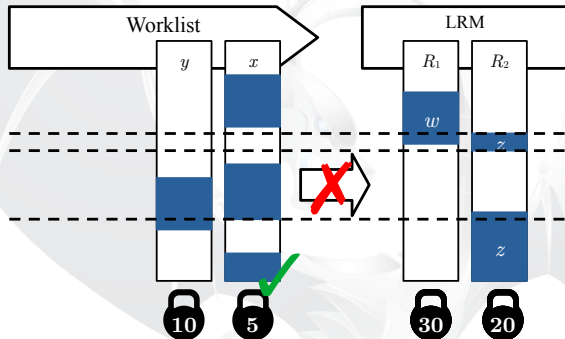


- ▶ Motivation: In RAMinimal, large live intervals are often **spilled**.
- ▶ Key Idea:
 - ▶ Allocate large live intervals first.
 - ▶ Evict and split them if they interfere with small ones.
 - ▶ Spill only if spiltting does not help.
- ▶ A **graduate refinement** process.

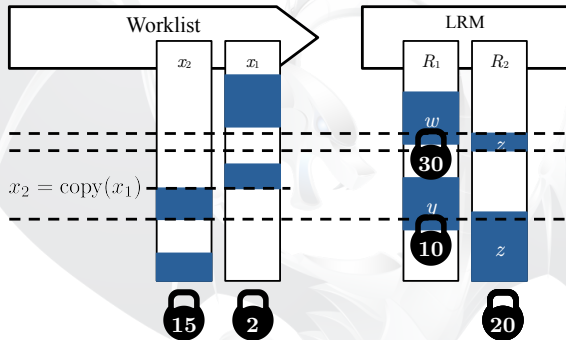
Example Walkthrough



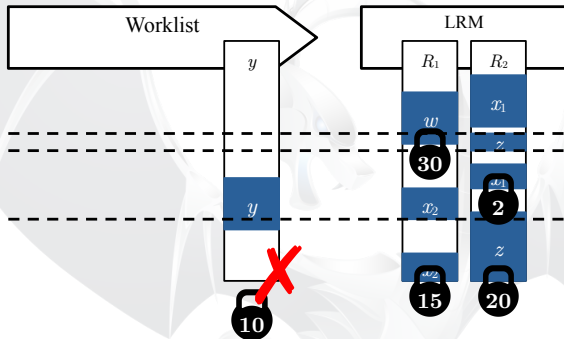
Example Walkthrough



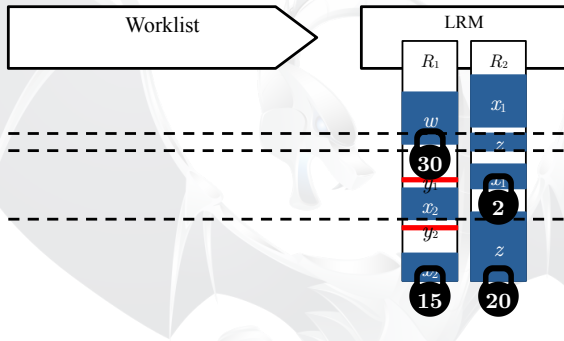
Example Walkthrough



Example Walkthrough



Example Walkthrough





x86 Registers

Register Overlapping

- ▶ Different registers can share the same storage underneath.



- ▶ Need to be check for aliasing when doing register assignment:

```
TargetRegisterInfo::regsOverlap(RegA, RegB);
```

Caller- vs. Callee-Saved Registers

- ▶ Subroutines need to use registers as well.
- ▶ The caller and the callee have an agreement on who to preserve which registers: Caller- vs. Callee-Saved Registers.

- ▶ aka. Call-Clobbered vs. Call-Preserved

E.g., EAX, ECX, EDX are caller-saved whereas EBX is callee-saved. Therefore, upon calling printf:

```
CALL64pcrel32 target-flags(x86-plt) @printf, <regmask ...  
    $ebx ...>
```

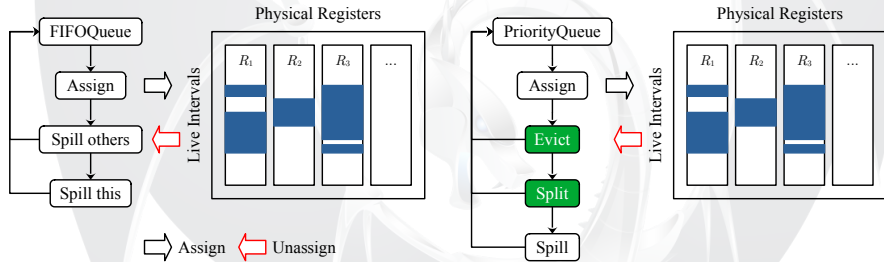
Recall

Free, VirtReg, RegUnit, RegMask from
LRM. $\text{checkInterference}(V, P)$

- ▶ RegUnit: A fixed live range is in the way, typically arguments for a call.
- ▶ RegMask: The live range is crossing an instruction with a regmask operand that does NOT preserve P .

Recap

In this tutorial, we have discussed:



► x86 Registers

- Register Overlapping
- Caller- vs. Callee-Saved Registers

👉 Checkout the [minimal register allocator](#) for details.

References

- [1] <https://blog.llvm.org/2011/09/greedy-register-allocation-in-llvm-30.html>
- [2] https://llvm.org/devmtg/2018-04/talks.html#Talk_11
- [3] https://en.wikibooks.org/wiki/X86_Assembly/X86_Architecture
- [4] <https://www.eecg.utoronto.ca/~amza/www.mindsec.com/files/x86regs.html>
- [5] <https://ece568.ca/Lectures/Section%202%20-%20Software%20Code%20Vulnerabilities/04%20-%20Buffer%20overflows.pdf>
- [6] <https://www.cs.virginia.edu/~evans/cs216/guides/x86.html>