# Lecture 20: Local register allocation

David Hovemeyer

November 18, 2025

601.428/628 Compilers and Interpreters

# Today

- Register allocation
- Local register allocation
- Implementing register allocation

# Register allocation

# Storage for virtual registers

- In Assignment 4: "virtual registers" are used as storage for some local variables and all temporary values
- In generated low-level code, every vreg is allocated memory storage in the stack frame
- This approach works, but
  - Stack frames will be very large
  - Nearly every instruction will access memory (slow!)
- Ideally, we'd like as many low-level instructions as possible to use *machine registers* as operands
- How to do this?

# Register allocation

*Register allocation* is the general problem of allocating and assigning CPU registers as storage locations for variables and values.

This is a classic problem in compilers with many decades of research on techniques.

For Assignment 5, implementing effective register allocation is likely to be the single most important code generation improvement you can make (in terms of the runtime efficiency of the generated code.)

# Register allocation issues

There are lots of important issues to consider, such as

- Which variables and values should be allocated a CPU register?
- There is a limited number of CPU registers: what if we don't have enough?
- What do we do if a location is accessed in multiple basic blocks?

Today we'll consider a simple but effective technique, *bottom-up local register allocation*

# Local register allocation

# Local register allocation

The goal of *local register allocation* is to allocate and assign CPU registers to use for values in a single basic block (hence, "local")

Because the scope is a single basic block, it's not useful for values that are either assigned or used in a different basic block

- Liveness analysis can determine which vregs are live at the beginning or end of a basic block: these should *not* be candidates for register allocation
- The local register allocator will focus on keeping *temporary values* in registers

# Bottom-up local register allocation

Goal allocate and assign a machine register for each virtual register, excluding vregs with live value at beginning or end of block. Annotate vreg operands with the assigned mreg (to convey decisions to the low-level code generator.)

Given set of mregs available for allocation:

**for each** instruction *i*
    **for each** vreg operand *v*[1]
        **if** *v* is spilled
            allocate mreg *m* and restore spilled value to *m*
        **else if** *v* doesn't have an assigned mreg
            allocate an mreg *m*
        annotate operand *v* with mreg *m*
        **if** *v* does not contain a live value after *i*
            return its assigned mreg *m* to the pool of available mregs

[1] Ignoring excluded vregs

# Allocating a machine register

- If a machine register is available, allocate it (easy case)
- If no machine register is available (harder case):
  1. Choose a victim vreg
  2. Allocate a currently-unused spill location, otherwise, use a new spill location
  3. Emit a spill instruction (specifying the vreg, mreg, and spill location)
  4. Use the stolen mreg to satisfy the allocation

# Choosing a victim

When the register allocator runs out of available mregs, but it needs to allocate one, it chooses a victim vreg and "steals" its allocated mreg.

- It *spills* the value in the victim's mreg to a spill location in the stack frame (and keeps track of which location the vreg's value was spilled to)
- When necessary, a spilled value will be *restored* just before it is needed as a source operand

Which vreg should be chosen as victim? In *bottom-up* register allocation, we choose the vreg whose next use is furthest in the future.

# This seems familiar...

The general idea is that a small number of machine registers are temporarily assigned to a (potentially larger) set of storage locations (vregs allocated for temporary values.)

Very much the same idea as virtual memory paging!

| | | |
|---|---|---|
| mreg : temp vreg | :: | physical page : virtual page |
| spill : page out | :: | restore : page in |

LRU eviction of virtual pages assumes that the least recently used page will be the one least likely to be needed soon. But with temp vregs, we actually *know* which one has the furthest next use.

# Returning an mreg to the pool

If a vreg does not contain a live value after the instruction, then its mreg can and should be returned to the pool of available registers.

Note that if a vreg source operand is dead immediately after the instruction in which it appears, its assigned mreg can be returned to the pool before the allocation decision for the destination vreg is considered. So, you can reuse the same mreg for a source vreg operand and the destination vreg operand, as long as the source operand is dead after the instruction. For example:

```
add_l vr25, vr23<%ecx>, vr24<%r8d>  // assume vr23 dead after instruction
```

```
add_l vr25<%ecx>, vr23<%ecx>, vr24<%r8d>  // %ecx used for vr25
```

# Intuition about temporary values

Local register allocation can be surprisingly effective because temporary values tend to be consumed very soon after being computed.

"Live range" of a value: span of instructions between where a value is computed and where the value is used.

Live ranges for temporary values are typically short (between 1 and 3 instructions is fairly typical.)

Because live ranges of temp values are usually short, only a small number of values are (typically) live at any single point. As long as the number of available mregs is $\geq$ the number of live temp values, register allocation will succeed without any spills or restores.

# Vregs used for local variables

Values in temp vregs are never alive longer than one statement. (Exception: LVN could cause a computed value to be reused in a subsequent statement, as long as the use is in the same basic block.) That means that vregs used for temp values are never alive at the beginning or end of a basic block.

Vregs used as storage for local variables generally *will* be alive at the beginning and/or end of at least some basic blocks. So, they should be ignored by the local register allocator.

# Dealing with vregs allocated to local variables

**Option #1**: Allocate them in memory in the stack frame, just like in Assignment 4. In theory you shouldn't need to do anything special to allow this to work. (Just don't break the support you implemented for this!)

**Option #2**: Allocate *callee-saved* registers as storage for some local variables. Because their values are stable across function calls, you can *globally* assign them as storage and access to them in all basic blocks. You'll need to generate push and pop intructions in the function prologue and epilogue code to save and restore their original values.

- Interesting question: *which* local variables should be assigned callee-saved registers as their storage?

Because there are a limited number of callee-saved machine registers, using Option #2 exclusively isn't viable.

# Register allocation example

```
localaddr vr41, $4000000
mul_l     vr42, vr13, vr10
add_l     vr43, vr42, vr14
sconv_lq  vr44, vr43
mul_q     vr45, vr44, $8
add_q     vr46, vr41, vr45
mov_q     vr47, (vr46)
localaddr vr48, $2000000

mul_l     vr49, vr15, vr10
add_l     vr50, vr49, vr14
sconv_lq  vr51, vr50
mul_q     vr52, vr51, $8
add_q     vr53, vr48, vr52
mov_q     vr55, (vr53)
mul_q     vr54, vr16, vr55
add_q     vr56, vr47, vr54

mov_q     (vr46), vr56
add_l     vr64, vr14, $1
mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%rcx,%rdx,%r8}

Assigned:

# Register allocation example

```
* localaddr vr41<%rcx>, $4000000
  mul_l    vr42, vr13, vr10
  add_l    vr43, vr42, vr14
  sconv_lq vr44, vr43
  mul_q    vr45, vr44, $8
  add_q    vr46, vr41, vr45
  mov_q    vr47, (vr46)
  localaddr vr48, $2000000

  mul_l    vr49, vr15, vr10
  add_l    vr50, vr49, vr14
  sconv_lq vr51, vr50
  mul_q    vr52, vr51, $8
  add_q    vr53, vr48, vr52
  mov_q    vr55, (vr53)
  mul_q    vr54, vr16, vr55
  add_q    vr56, vr47, vr54

  mov_q    (vr46), vr56
  add_l    vr64, vr14, $1
  mov_l    vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%rdx,%r8}

Assigned:

vr41 → %rcx

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
* mul_l    vr42<%edx>, vr13, vr10
  add_l    vr43, vr42, vr14
  sconv_lq vr44, vr43
  mul_q    vr45, vr44, $8
  add_q    vr46, vr41, vr45
  mov_q    vr47, (vr46)
  localaddr vr48, $2000000

  mul_l    vr49, vr15, vr10
  add_l    vr50, vr49, vr14
  sconv_lq vr51, vr50
  mul_q    vr52, vr51, $8
  add_q    vr53, vr48, vr52
  mov_q    vr55, (vr53)
  mul_q    vr54, vr16, vr55
  add_q    vr56, vr47, vr54

  mov_q    (vr46), vr56
  add_l    vr64, vr14, $1
  mov_l    vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%r8}

Assigned:

vr41 → %rcx

vr42 → %rdx

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
* add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44, vr43
  mul_q     vr45, vr44, $8
  add_q     vr46, vr41, vr45
  mov_q     vr47, (vr46)
  localaddr vr48, $2000000

  mul_l     vr49, vr15, vr10
  add_l     vr50, vr49, vr14
  sconv_lq  vr51, vr50
  mul_q     vr52, vr51, $8
  add_q     vr53, vr48, vr52
  mov_q     vr55, (vr53)
  mul_q     vr54, vr16, vr55
  add_q     vr56, vr47, vr54

  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%r8}

Assigned:

vr41 → %rcx

vr43 → %rdx

Note that vr42's mreg can be freed and then allocated to vr43

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
* sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45, vr44, $8
  add_q     vr46, vr41, vr45
  mov_q     vr47, (vr46)
  localaddr vr48, $2000000

  mul_l     vr49, vr15, vr10
  add_l     vr50, vr49, vr14
  sconv_lq  vr51, vr50
  mul_q     vr52, vr51, $8
  add_q     vr53, vr48, vr52
  mov_q     vr55, (vr53)
  mul_q     vr54, vr16, vr55
  add_q     vr56, vr47, vr54

  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%r8}

Assigned:

vr41 → %rcx

vr44 → %rdx

vr43's mreg can be freed and allocated to vr44

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
* mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46, vr41, vr45
  mov_q     vr47, (vr46)
  localaddr vr48, $2000000

  mul_l     vr49, vr15, vr10
  add_l     vr50, vr49, vr14
  sconv_lq  vr51, vr50
  mul_q     vr52, vr51, $8
  add_q     vr53, vr48, vr52
  mov_q     vr55, (vr53)
  mul_q     vr54, vr16, vr55
  add_q     vr56, vr47, vr54

  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%r8}

Assigned:

vr41 → %rcx

vr45 → %rdx

vr44's mreg can be freed and allocated to vr45

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
* add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
  mov_q     vr47, (vr46)
  localaddr vr48, $2000000

  mul_l     vr49, vr15, vr10
  add_l     vr50, vr49, vr14
  sconv_lq  vr51, vr50
  mul_q     vr52, vr51, $8
  add_q     vr53, vr48, vr52
  mov_q     vr55, (vr53)
  mul_q     vr54, vr16, vr55
  add_q     vr56, vr47, vr54

  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%r8,%rdx}

Assigned:

vr46 → %rcx

vr41's mreg can be freed and allocated to vr46

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
* mov_q     vr47<%r8>, (vr46<%rcx>)
  localaddr vr48, $2000000

  mul_l     vr49, vr15, vr10
  add_l     vr50, vr49, vr14
  sconv_lq  vr51, vr50
  mul_q     vr52, vr51, $8
  add_q     vr53, vr48, vr52
  mov_q     vr55, (vr53)
  mul_q     vr54, vr16, vr55
  add_q     vr56, vr47, vr54

  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%rdx}

Assigned:

vr46 → %rcx

vr47 → %r8

Note that vr46 still contains a live value at this point!

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
  mov_q     vr47<%rcx>, (vr46<%rcx>)
* localaddr vr48<%rdx>, $2000000

  mul_l     vr49, vr15, vr10
  add_l     vr50, vr49, vr14
  sconv_lq  vr51, vr50
  mul_q     vr52, vr51, $8
  add_q     vr53, vr48, vr52
  mov_q     vr55, (vr53)
  mul_q     vr54, vr16, vr55
  add_q     vr56, vr47, vr54

  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {}

Assigned:

vr46 → %rcx

vr47 → %r8

vr48 → %rdx

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
  mov_q     vr47<%rcx>, (vr46<%rcx>)
  localaddr vr48<%r8>, $2000000
* spill_q   vr46<%rcx>, $0
  mul_l     vr49, vr15, vr10
  add_l     vr50, vr49, vr14
  sconv_lq  vr51, vr50
  mul_q     vr52, vr51, $8
  add_q     vr53, vr48, vr52
  mov_q     vr55, (vr53)
  mul_q     vr54, vr16, vr55
  add_q     vr56, vr47, vr54

  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%rcx}

Assigned:

vr47 → %r8

vr48 → %rdx

Spilled:

vr46 → spill loc 0

Need to spill vr46 to free an mreg to allocate to vr49

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
  mov_q     vr47<%rcx>, (vr46<%rcx>)
  localaddr vr48<%r8>, $2000000
  spill_q   vr46<%rcx>, $0
* mul_l     vr49<%ecx>, vr15, vr10
  add_l     vr50, vr49, vr14
  sconv_lq  vr51, vr50
  mul_q     vr52, vr51, $8
  add_q     vr53, vr48, vr52
  mov_q     vr55, (vr53)
  mul_q     vr54, vr16, vr55
  add_q     vr56, vr47, vr54

  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {}

Assigned:

vr47 → %r8
vr48 → %rdx
vr49 → %rcx

Spilled:

vr46 → spill loc 0

%rcx can now be allocated to vr49

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
  mov_q     vr47<%rcx>, (vr46<%rcx>)
  localaddr vr48<%r8>, $2000000
  spill_q   vr46<%rcx>, $0
  mul_l     vr49<%ecx>, vr15, vr10
* add_l     vr50<%ecx>, vr49<%ecx>, vr14
  sconv_lq  vr51, vr50
  mul_q     vr52, vr51, $8
  add_q     vr53, vr48, vr52
  mov_q     vr55, (vr53)
  mul_q     vr54, vr16, vr55
  add_q     vr56, vr47, vr54

  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {}

Assigned:

vr47 → %r8
vr48 → %rdx
vr50 → %rcx

Spilled:

vr46 → spill loc 0

vr49's mreg can be freed and allocated to vr50

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
  mov_q     vr47<%rcx>, (vr46<%rcx>)
  localaddr vr48<%r8>, $2000000
  spill_q   vr46<%rcx>, $0
  mul_l     vr49<%ecx>, vr15, vr10
  add_l     vr50<%ecx>, vr49<%ecx>, vr14
* sconv_lq  vr51<%rcx>, vr50<%ecx>
  mul_q     vr52, vr51, $8
  add_q     vr53, vr48, vr52
  mov_q     vr55, (vr53)
  mul_q     vr54, vr16, vr55
  add_q     vr56, vr47, vr54

  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {}

Assigned:

vr47 → %r8
vr48 → %rdx
vr51 → %rcx

Spilled:

vr46 → spill loc 0

vr50's mreg can be freed and allocated to vr51

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
  mov_q     vr47<%rcx>, (vr46<%rcx>)
  localaddr vr48<%r8>, $2000000
  spill_q   vr46<%rcx>, $0
  mul_l     vr49<%ecx>, vr15, vr10
  add_l     vr50<%ecx>, vr49<%ecx>, vr14
  sconv_lq  vr51<%rcx>, vr50<%ecx>
* mul_q     vr52<%rcx>, vr51<%rcx>, $8
  add_q     vr53, vr48, vr52
  mov_q     vr55, (vr53)
  mul_q     vr54, vr16, vr55
  add_q     vr56, vr47, vr54

  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {}

Assigned:

vr47 → %r8
vr48 → %rdx
vr52 → %rcx

Spilled:

vr46 → spill loc 0

vr51's mreg can be freed and allocated to vr52

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
  mov_q     vr47<%rcx>, (vr46<%rcx>)
  localaddr vr48<%r8>, $2000000
  spill_q   vr46<%rcx>, $0
  mul_l     vr49<%ecx>, vr15, vr10
  add_l     vr50<%ecx>, vr49<%ecx>, vr14
  sconv_lq  vr51<%rcx>, vr50<%ecx>
  mul_q     vr52<%rcx>, vr51<%rcx>, $8
* add_q     vr53<%rdx>, vr48<%rdx>, vr52<%rcx>
  mov_q     vr55, (vr53)
  mul_q     vr54, vr16, vr55
  add_q     vr56, vr47, vr54

  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%rcx}

Assigned:

vr47 → %r8

vr53 → %rdx

Spilled:

vr46 → spill loc 0

vr48's mreg can be freed and allocated to vr53

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
  mov_q     vr47<%rcx>, (vr46<%rcx>)
  localaddr vr48<%r8>, $2000000
  spill_q   vr46<%rcx>, $0
  mul_l     vr49<%ecx>, vr15, vr10
  add_l     vr50<%ecx>, vr49<%ecx>, vr14
  sconv_lq  vr51<%rcx>, vr50<%ecx>
  mul_q     vr52<%rcx>, vr51<%rcx>, $8
  add_q     vr53<%rdx>, vr48<%rdx>, vr52<%rcx>
* mov_q     vr55<%rcx>, (vr53<%rdx>)
  mul_q     vr54, vr16, vr55
  add_q     vr56, vr47, vr54

  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%rdx}

Assigned:

vr47 → %r8

vr55 → %rcx

Spilled:

vr46 → spill loc 0

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
  mov_q     vr47<%rcx>, (vr46<%rcx>)
  localaddr vr48<%r8>, $2000000
  spill_q   vr46<%rcx>, $0
  mul_l     vr49<%ecx>, vr15, vr10
  add_l     vr50<%ecx>, vr49<%ecx>, vr14
  sconv_lq  vr51<%rcx>, vr50<%ecx>
  mul_q     vr52<%rcx>, vr51<%rcx>, $8
  add_q     vr53<%rdx>, vr48<%rdx>, vr52<%rcx>
  mov_q     vr55<%rcx>, (vr53<%rdx>)
* mul_q     vr54<%rdx>, vr16, vr55<%rcx>
  add_q     vr56, vr47, vr54

  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%rcx}

Assigned:

vr47 → %r8

vr54 → %rdx

Spilled:

vr46 → spill loc 0

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
  mov_q     vr47<%rcx>, (vr46<%rcx>)
  localaddr vr48<%r8>, $2000000
  spill_q   vr46<%rcx>, $0
  mul_l     vr49<%ecx>, vr15, vr10
  add_l     vr50<%ecx>, vr49<%ecx>, vr14
  sconv_lq  vr51<%rcx>, vr50<%ecx>
  mul_q     vr52<%rcx>, vr51<%rcx>, $8
  add_q     vr53<%rdx>, vr48<%rdx>, vr52<%rcx>
  mov_q     vr55<%rcx>, (vr53<%rdx>)
  mul_q     vr54<%rdx>, vr16, vr55<%rcx>
* add_q     vr56<%rcx>, vr47<%r8>, vr54<%rdx>

  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%r8,%rdx}

Assigned:

vr56 → %rcx

Spilled:

vr46 → spill loc 0

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
  mov_q     vr47<%rcx>, (vr46<%rcx>)
  localaddr vr48<%r8>, $2000000
  spill_q   vr46<%rcx>, $0
  mul_l     vr49<%ecx>, vr15, vr10
  add_l     vr50<%ecx>, vr49<%ecx>, vr14
  sconv_lq  vr51<%rcx>, vr50<%ecx>
  mul_q     vr52<%rcx>, vr51<%rcx>, $8
  add_q     vr53<%rdx>, vr48<%rdx>, vr52<%rcx>
  mov_q     vr55<%rcx>, (vr53<%rdx>)
  mul_q     vr54<%rdx>, vr16, vr55<%rcx>
  add_q     vr56<%rcx>, vr47<%r8>, vr54<%rdx>
* restore_q vr46<%r8>, $0
  mov_q     (vr46), vr56
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%rdx}

Assigned:

vr56 → %rcx

vr46 → %r8

Restore vr46 from spill location 0 using allocated mreg %r8

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
  mov_q     vr47<%rcx>, (vr46<%rcx>)
  localaddr vr48<%r8>, $2000000
  spill_q   vr46<%rcx>, $0
  mul_l     vr49<%ecx>, vr15, vr10
  add_l     vr50<%ecx>, vr49<%ecx>, vr14
  sconv_lq  vr51<%rcx>, vr50<%ecx>
  mul_q     vr52<%rcx>, vr51<%rcx>, $8
  add_q     vr53<%rdx>, vr48<%rdx>, vr52<%rcx>
  mov_q     vr55<%rcx>, (vr53<%rdx>)
  mul_q     vr54<%rdx>, vr16, vr55<%rcx>
  add_q     vr56<%rcx>, vr47<%r8>, vr54<%rdx>
  restore_q vr46<%r8>, $0
* mov_q     (vr46<%r8>), vr56<%rcx>
  add_l     vr64, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%rdx,%rcx,%r8}

Assigned:

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
  mov_q     vr47<%rcx>, (vr46<%rcx>)
  localaddr vr48<%r8>, $2000000
  spill_q   vr46<%rcx>, $0
  mul_l     vr49<%ecx>, vr15, vr10
  add_l     vr50<%ecx>, vr49<%ecx>, vr14
  sconv_lq  vr51<%rcx>, vr50<%ecx>
  mul_q     vr52<%rcx>, vr51<%rcx>, $8
  add_q     vr53<%rdx>, vr48<%rdx>, vr52<%rcx>
  mov_q     vr55<%rcx>, (vr53<%rdx>)
  mul_q     vr54<%rdx>, vr16, vr55<%rcx>
  add_q     vr56<%rcx>, vr47<%r8>, vr54<%rdx>
  restore_q vr46<%r8>, $0
  mov_q     (vr46<%r8>), vr56<%rcx>
* add_l     vr64<%edx>, vr14, $1
  mov_l     vr14, vr64
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%rcx,%r8}

Assigned:

vr64 → %rdx

# Register allocation example

```
  localaddr vr41<%rcx>, $4000000
  mul_l     vr42<%edx>, vr13, vr10
  add_l     vr43<%edx>, vr42<%edx>, vr14
  sconv_lq  vr44<%rdx>, vr43<%edx>
  mul_q     vr45<%rdx>, vr44<%rdx>, $8
  add_q     vr46<%rcx>, vr41<%rcx>, vr45<%rdx>
  mov_q     vr47<%rcx>, (vr46<%rcx>)
  localaddr vr48<%r8>, $2000000
  spill_q   vr46<%rcx>, $0
  mul_l     vr49<%ecx>, vr15, vr10
  add_l     vr50<%ecx>, vr49<%ecx>, vr14
  sconv_lq  vr51<%rcx>, vr50<%ecx>
  mul_q     vr52<%rcx>, vr51<%rcx>, $8
  add_q     vr53<%rdx>, vr48<%rdx>, vr52<%rcx>
  mov_q     vr55<%rcx>, (vr53<%rdx>)
  mul_q     vr54<%rdx>, vr16, vr55<%rcx>
  add_q     vr56<%rcx>, vr47<%r8>, vr54<%rdx>
  restore_q vr46<%r8>, $0
  mov_q     (vr46<%r8>), vr56<%rcx>
  add_l     vr64<%edx>, vr14, $1
* mov_l     vr14, vr64<%edx>
```

Exclude: {vr10,vr13,vr14,vr15,vr16}

Avail: {%rcx,%r8,%rdx}

Assigned:

# Limitations of this approach

Annotating your high-level IR with mreg assignments should work out reasonably well.

The low-level code generator will need to keep one or two mregs (probably `%r10` and `%r11`) in reserve for code gen: they will generally be needed when converting from three-operand high-level instructions to two-operand x86-64 instructions.

The idioms generated by this approach tend to use "unnecessary" machine registers. In many cases, these unnecessary mregs can be eliminated by peephole optimization.

- But: this means that some mregs assigned by the local register allocator won't actually be needed!

# A better approach?

Sketch of an approach that might make better use of machine registers:

1. An initial translation from highlevel code to lowlevel code freely uses "fake" machine registers, as many as needed.
2. The peephole optimizer cleans up the low-level instructions, reducing the number of mregs needed.
3. The local register allocator runs after peephole optimization, allocating real mregs to replace the "fake" mregs. Spills and restores might be needed at this point.

I'm not suggesting that you do this for Assignment 5, though, since it would require some fundamental changes to the IR.

# Implementing local register allocation

# Which registers to make available?

When allocating mregs for temp values, caller-saved registers make the most sense.

- You could use callee-saved registers, but these might be better used as storage for local variables, especially loop variables

Suggestion: make any argument registers (`%rdi`, `%rsi`, `%rdx`, etc.) not needed to pass arguments to a function call available to the local register allocator.

- You will need to be able to look at the basic block and know whether there is a function call, and if so, how many arguments it takes
- The control flow graph builder will always place `call` instructions at the end of the basic block

# Considerations

When allocating a machine register from the pool of available registers, it should not matter which one you pick. Common strategies: use a stack or queue.

You will need to keep track of the total number spill locations needed by each basic block. The maximum over all basic blocks is the number needed for the overall function.

- Place storage area for spills somewhere in the stack frame
- Low-level code generator will need to determine an offset into the storage area for each spill and restore

# Register allocator state

Information to keep track of as allocator progresses through instructions in basic block:

- Collection of available machine registers (stack or queue)
- Map of virtual register numbers to assigned machine register
- Collection of available spill locations (stack or queue)
- Map of virtual register numbers to spill locations

# Dealing with procedure calls

The compiler must assume that a call to a procedure could change the value of any caller-save register! (e.g., %rcx, %rdx, %r10, etc.)

Should not be a huge problem in practice:

- If a basic block has a `call` instruction, it will be the last instruction
- Local register allocation should only assign machine registers to vregs used for temp values: these values will be dead at the end of the basic block[2]
- The register allocator should avoid allocating machine registers that will be needed to pass arguments

[2] In general, don't allocate registers to vregs that aren't dead at the beginning or end of the block