322 Compilers
Why take this course?

• Understanding tools better; what does the compiler really do?

• Computer Engineering & Architecture people: the compiler is your lens to the world

• Phil Greenpun’s 10th rule of programming:

  Any sufficiently complicated C or Fortran program contains an ad hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp.
Interpreters vs Compilers

**interpreter**: program $\rightarrow$ answer

**compiler**: program $\rightarrow$ program

// no answer!

no interpreter $\Rightarrow$ programs don’t run
Why Compile?

Performance. That’s the *only* reason.
Why Compile?: Interpreter overhead

```
addl %rax,%rbx
```

vs

```
(define (interp exp)
  (type-case FAE exp
    [num (n) (num n)]
    [add (lhs rhs)
      (let ([lv (interp lhs)]
        [rv (interp rhs)])
        (type-case FAE-Value lv
          [numV (ln)
            (type-case FAE-Value rv
              [numV (rn) (+ ln rn)]
              [else (error 'interp)])]
          [else (error 'interp)]))]
    ...
  ))
```
Why Compile?: Automate Transformations

- Bad maintenance practices, yet profitable transformations

For example unrolling loops; when the chip sees straight-line code it can go faster:

It can “look ahead” and thus make good guesses about what is going to happen next,

filling in caches early, keeping the pipeline full, etc
Why Compile?: Automate Transformations

• Lower-level details are exposed in destination language

For example, variables might live on the stack or in registers; want to use registers as much as possible
Goalposts

Build a compiler accepting a language (L5) that has:

• Higher-order functions
• Safe, mutable arrays
• Arithmetic on (bounded) integers
• Recursive binding form
• Conditionals

and producing x86-64 assembly

Front End ⇒ Middle End ⇒ Back End
Goalposts

Build a compiler accepting a language (L5) that has:

- Higher-order functions
- Safe, mutable arrays
- Arithmetic on (bounded) integers
- Recursive binding form
- Conditionals

and producing x86-64 assembly

Front End $\Rightarrow$ Middle End $\Rightarrow$ Back End

this course $\Rightarrow$ Simone’s course $\Rightarrow$ this course
Fib in L5

(letrec ([fib
    (lambda (n)
      (if (= n 0)
        0
        (if (= n 1)
          1
          (+ (fib (- n 1))
            (fib (- n 2))))))])

(print (fib 10)))
Fib in L4

no more higher-order functions

```scheme
((print (:fib 10))
 (:fib
  (n)
  (if (= n 0)
   0
   (if (= n 1)
    1
    (+ (:fib (- n 1))
      (:fib (- n 2)))))))))
```
Fib in L3

every intermediate result has a name

```
(((let ([fibten (:fib 10)]))
  (print fibten))
 (:fib
  (n)
  (let ([niszero (= n 0)])
    (if niszero
      0
      (let ([nisone (= n 1)])
        (if nisone
          1
          (let ([n1 (- n 1)])
            (let ([fn1 (:fib n1)])
              (let ([n2 (- n 2)])
                (let ([fn2 (:fib n2)])
                  (+ fn2
                     fn1))))))))))))
```
Fib in L2

no more nested expressions

(:main
  0
  0
  (rdi <- 10)
  ((mem rsp -8) <- :fr)
  (call :fib 1)
  :fr
  (rdi <- rax)
  (rdi *= 2)
  (rdi += 1)
  (call print 1)
  (return))

(:fib
  1
  0
  (cjump rdi = 0 :zero :nonzero)
  :zero
  (rax <- 0)
  (return)
  :nonzero
  (cjump rdi = 1 :one :recur)
  :one
  (rax <- 1)
  (return)
  :recur
  (n <- rdi)
  (rdi -= 1)
  ((mem rsp -8) <- :for)
  (call :fib 1)
  :for
  (result <- rax)
  (n -= 2)
  (rdi <- n)
  ((mem rsp -8) <- :ftr)
  (call :fib 1)
  :ftr
  (rax += result)
  (return))
Fib in L1
no more variables (just registers)

(:main
  0
  0
  (rdi <- 10)
  ((mem rsp -8) <- :fr)
  (call :fib 1)
  :fr
  (rdi <- rax)
  (rdi *= 2)
  (rdi += 1)
  (call print 1)
  (return))

(:fib
  1
  2
  (cjump rdi = 0 :zero :nonzero)
  :zero
  (rax <- 0)
  (return)
  :nonzero
  (cjump rdi = 1 :one :recur)
  :one
  (rax <- 1)
  (return)
  :recur
  ((mem rsp 0) <- r12)
  ((mem rsp 8) <- r13)
  (r12 <- rdi)
  (rdi -= 1)
  ((mem rsp -8) <- :for)
  (call :fib 1)
  :for
  (r13 <- rax)
  (r12 -= 2)
  (rdi <- r12)
  ((mem rsp -8) <- :ftr)
  (call :fib 1)
  :ftr
  (rax += r13)
  (r12 <- (mem rsp 0))
  (r13 <- (mem rsp 8))
  (return))
Implementation/Project overview

L5 $\rightarrow$ L4 $\rightarrow$ L3 $\rightarrow$ L2, each one step

L2 $\rightarrow$ L1, multiple steps:
  - spilling
  - graph coloring
  - graph construction
  - liveness analysis

Speed test

2 assignments per step: tests & implementation

There is no real “late” code

Use any PL you want (learn a new one!)
Want to pair program? Give me a **written** note (one from each member), with the promise on the web page

More admin details, including grading rubric, on website; read it

https://www.eecs.northwestern.edu/~robb/ courses/322-2016-spring/