Compiler Walkthrough

Focus on These Files in gsc/

```
 1024   _host.scm          portability layer
 1434   _utils.scm         miscellaneous utilities
  591   _parms.scm         definition of common symbols
  327   _env.scm           compile time environments
 1635   _source.scm        files -> S-expr (deprecated)
 2816   _ptree1.scm        S-expr -> AST (parse trees)
 2990   _ptree2.scm        AST -> AST transformations
 5513   _prims.scm         AST -> AST transformations
 4235   _front.scm         frontend (AST -> GVM)
 2754   _gvm.scm           GVM -> GVM transformations
  632   _back.scm          interface to backends
 7083   _t-c-*.scm         C backend
12597   _t-cpu-*.scm       Native backend (x86, ...)
18594   _t-univ-*.scm      Universal backend (JS, ...)
  712   _gsclib.scm        compile-file, etc
------
62937 LOC (generates 615401 C LOC)
```

Declarations

- By default Gambit obeys R5RS semantics
- This has an impact on performance
- Declarations allow the programmer to indicate where it is OK to make some assumptions, which enable various optimizations

```
(car x)  ;; 1) read the "car" global variable
         ;; 2) check that it is a function
         ;; 3) call the function
```

```
(declare (standard-bindings))

(car x)  ;; car is known to contain the car
         ;; function so the compiler can inline it
```

Other Declarations

`(block)`	assume global vars defined in this file are not mutated outside it
`(fixnum)`	fixnum arithmetic only
`(flonum)`	flonum arithmetic only
`(not safe)`	assume no type checks fail
`(debug)`	generate debug info
`(not proper-tail-calls)`	turn off TCO

Impact on Performance

```
(define (fib n)
  (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))

(fib 40)
```

MacBook Pro
2.8 GHz Intel Core 2 Duo
4 GB RAM

no declaration (i.e. pure R5RS semantics)	5.68 s
`(declare (standard-bindings))`	4.80 s
`+ (declare (block))`	3.30 s
`+ (declare (fixnum))`	2.70 s
`+ (declare (not safe))`	1.11 s

`gcc -O2 fib40.c`	1.63 s
`sbcl < fib40.lisp` *(no declaration)*	4.24 s

Main Optimizations

- Inlining
  - Primitive functions (**`car`**, **`cons`**, **`map`**, ...)
  - User functions, including recursive functions
    - Speculative inlining of primitive functions (when binding of global var unknown)
- Lambda lifting
- Copy / constant propagation, constant folding

gsc -c -expansion fib1.scm

```
(define fib
  (lambda (n)
    (if (if ('#<procedure #2 ##eq?> < '#<procedure #3 <>)
            (if ('#<procedure #4 ##fixnum?> n)
                ('#<procedure #5 ##fx<> n 2)
                (< n 2))
            (< n 2))
        n
        (let ((temp.9 (fib (if ('#<procedure #2 ##eq?> - '#<procedure #6 ->)
                               (if ('#<procedure #4 ##fixnum?> n)
                                   (let ((temp.7 ('#<procedure #7 ##fx-?>
                                                  n
                                                  2)))
                                     (if temp.7 temp.7 (- n 2)))
                                   (- n 2))
                               (- n 2))))
              (temp.8 (fib (if ('#<procedure #2 ##eq?> - '#<procedure #6 ->)
                               (if ('#<procedure #4 ##fixnum?> n)
                                   (let ((temp.4 ('#<procedure #7 ##fx-?>
                                                  n
                                                  1)))
                                     (if temp.4 temp.4 (- n 1)))
                                   (- n 1))
                               (- n 1)))))
          (if ('#<procedure #2 ##eq?> + '#<procedure #8 +>)
              (if (and ('#<procedure #4 ##fixnum?> temp.9)
                       ('#<procedure #4 ##fixnum?> temp.8))
                  (let ((temp.10 ('#<procedure #9 ##fx+?> temp.8 temp.9)))
                    (if temp.10 temp.10 (+ temp.8 temp.9)))
                  (if (and ('#<procedure #10 ##flonum?> temp.9)
                           ('#<procedure #10 ##flonum?> temp.8))
                      ('#<procedure #11 ##fl+> temp.8 temp.9)
                      (+ temp.8 temp.9)))
              (+ temp.8 temp.9))))))

(fib 40)
```

```
(declare
;;  (standard-bindings)
;;  (block)
;;  (fixnum)
;;  (not safe)
)

(define (fib n)
  (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))

(fib 40)
```

gsc -c -expansion fib2.scm

```
(declare
    (standard-bindings)
;;  (block)
;;  (fixnum)
;;  (not safe)
)

(define (fib n)
  (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))

(fib 40)
```

```
(define fib
  (lambda (n)
    (if (if ('#<procedure #2 ##fixnum?> n)
            ('#<procedure #3 ##fx<> n 2)
            ('#<procedure #4 <> n 2))
        n
        (let ((temp.9 (fib (if ('#<procedure #2 ##fixnum?> n)
                               (let ((temp.7 ('#<procedure #5 ##fx-?> n 2)))
                                 (if temp.7 temp.7 ('#<procedure #6 -> n 2)))
                               ('#<procedure #6 -> n 2))))
              (temp.8 (fib (if ('#<procedure #2 ##fixnum?> n)
                               (let ((temp.4 ('#<procedure #5 ##fx-?> n 1)))
                                 (if temp.4 temp.4 ('#<procedure #6 -> n 1)))
                               ('#<procedure #6 -> n 1)))))
          (if (and ('#<procedure #2 ##fixnum?> temp.9)
                   ('#<procedure #2 ##fixnum?> temp.8))
              (let ((temp.10 ('#<procedure #7 ##fx+?> temp.8 temp.9)))
                (if temp.10 temp.10 ('#<procedure #8 +> temp.8 temp.9)))
              (if (and ('#<procedure #9 ##flonum?> temp.9)
                       ('#<procedure #9 ##flonum?> temp.8))
                  ('#<procedure #10 ##fl+> temp.8 temp.9)
                  ('#<procedure #8 +> temp.8 temp.9))))))))

(fib 40)
```

gsc -c -expansion fib3.scm

```
(define fib
  (lambda (n)
    (if (if ('#<procedure #2 ##fixnum?> n)
            ('#<procedure #3 ##fx<> n 2)
            ('#<procedure #4 <> n 2))
        n
        (let ((temp.9 (let ((n (if ('#<procedure #2 ##fixnum?> n)
                                   (let ((temp.7 ('#<procedure #5 ##fx-?>
                                                  n
                                                  2)))
                                     (if temp.7
                                         temp.7
                                         ('#<procedure #6 -> n 2)))
                                   ('#<procedure #6 -> n 2))))
                        (if (if ('#<procedure #2 ##fixnum?> n)
                                ('#<procedure #3 ##fx<> n 2)
                                ('#<procedure #4 <> n 2))
                            n
                            (let ((temp.9 (fib (if ('#<procedure #2 ##fixnum?>
                                                    n)
                                                   (let ((temp.7 ('#<procedure #5 ##fx-?>
;;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                                                          n
                                                          2)))
;;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                                                     (if temp.7
                                                         temp.7
                                                         ('#<procedure #6 ->
                                                          n
                                                          2)))
                                                   ('#<procedure #6 -> n 2))))
                                       (temp.8 (fib (if ('#<procedure #2 ##fixnum?>
                                                         n)
                                                        (let ((temp.4 ('#<procedure #5 ##fx-?>
;;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                                                              n
                                                              1)))
;;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                                                         (if temp.4
                                                             temp.4
                                                             ('#<procedure #6 ->
                                                              n
                                                              1)))
                                                        ('#<procedure #6 -> n 1)))))
                                   (if (and ('#<procedure #2 ##fixnum?> temp.9)
                                            ('#<procedure #2 ##fixnum?> temp.8))
                                       (let ((temp.10 ('#<procedure #7 ##fx+?>
                                                       temp.8
                                                       temp.9)))
                                         (if temp.10
                                             temp.10
                                             ('#<procedure #8 +> temp.8 temp.9)))
                                       (if (and ('#<procedure #9 ##flonum?> temp.9)
                                                ('#<procedure #9 ##flonum?> temp.8))
                                           ('#<procedure #10 ##fl+> temp.8 temp.9)
                                           ('#<procedure #8 +> temp.8 temp.9)))))))
              (temp.8 (let ((n (if ('#<procedure #2 ##fixnum?> n)
                                   (let ((temp.4 ('#<procedure #5 ##fx-?>
                                                  n
                                                  1)))
                                     (if temp.4
                                         temp.4
                                         ('#<procedure #6 -> n 1)))
                                   ('#<procedure #6 -> n 1))))
                        (if (if ('#<procedure #2 ##fixnum?> n)
                                ('#<procedure #3 ##fx<> n 2)
                                ('#<procedure #4 <> n 2))
                            n
                            (let ((temp.9 (fib (if ('#<procedure #2 ##fixnum?>
                                                    n)
                                                   (let ((temp.7 ('#<procedure #5 ##fx-?>
;;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                                                          n
                                                          2)))
;;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                                                     (if temp.7
                                                         temp.7
                                                         ('#<procedure #6 ->
                                                          n
                                                          2)))
                                                   ('#<procedure #6 -> n 2))))
                                       (temp.8 (fib (if ('#<procedure #2 ##fixnum?>
                                                         n)
                                                        (let ((temp.4 ('#<procedure #5 ##fx-?>
;;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                                                              n
                                                              1)))
;;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                                                         (if temp.4
                                                             temp.4
                                                             ('#<procedure #6 ->
                                                              n
                                                              1)))
                                                        ('#<procedure #6 -> n 1)))))
                                   (if (and ('#<procedure #2 ##fixnum?> temp.9)
                                            ('#<procedure #2 ##fixnum?> temp.8))
                                       (let ((temp.10 ('#<procedure #7 ##fx+?>
                                                       temp.8
                                                       temp.9)))
                                         (if temp.10
                                             temp.10
                                             ('#<procedure #8 +> temp.8 temp.9)))
                                       (if (and ('#<procedure #9 ##flonum?> temp.9)
                                                ('#<procedure #9 ##flonum?> temp.8))
                                           ('#<procedure #10 ##fl+> temp.8 temp.9)
                                           ('#<procedure #8 +> temp.8 temp.9)))))))))
        (if (and ('#<procedure #2 ##fixnum?> temp.9)
                 ('#<procedure #2 ##fixnum?> temp.8))
            (let ((temp.10 ('#<procedure #7 ##fx+?> temp.8 temp.9)))
              (if temp.10 temp.10 ('#<procedure #8 +> temp.8 temp.9)))
            (if (and ('#<procedure #9 ##flonum?> temp.9)
                     ('#<procedure #9 ##flonum?> temp.8))
                ('#<procedure #10 ##fl+> temp.8 temp.9)
                ('#<procedure #8 +> temp.8 temp.9)))))))

(fib 40)
```

```
(declare
    (standard-bindings)
    (block)
;;  (fixnum)
;;  (not safe)
)

(define (fib n)
  (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))

(fib 40)
```

gsc -c -expansion fib4.scm

```
(define fib
  (lambda (n)
    (if (if ('#<procedure #2 ##fixnum?> n)
            ('#<procedure #3 ##fx<> n 2)
            ('#<procedure #4 fx<> n 2))
        n
        (let ((temp.9 (let ((n (if ('#<procedure #2 ##fixnum?> n)
                                   (let ((temp.7 ('#<procedure #5 ##fx-?>
                                                  n
                                                  2)))
                                     (if temp.7
                                         temp.7
                                         ('#<procedure #6 fx-> n 2)))
                                   ('#<procedure #6 fx-> n 2))))
                        (if (if ('#<procedure #2 ##fixnum?> n)
                                ('#<procedure #3 ##fx<> n 2)
                                ('#<procedure #4 fx<> n 2))
                            n
                            (let ((temp.9 (fib (if ('#<procedure #2 ##fixnum?>
                                                    n)
                                                   (let ((temp.7 ('#<procedure #5 ##fx-?>
;;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                                                          n
                                                          2)))
;;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                                                     (if temp.7
                                                         temp.7
                                                         ('#<procedure #6 fx->
                                                          n
                                                          2)))
                                                   ('#<procedure #6 fx->
                                                    n
                                                    2))))
                                       (temp.8 (fib (if ('#<procedure #2 ##fixnum?>
                                                         n)
                                                        (let ((temp.4 ('#<procedure #5 ##fx-?>
;;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                                                              n
                                                              1)))
;;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                                                         (if temp.4
                                                             temp.4
                                                             ('#<procedure #6 fx->
                                                              n
                                                              1)))
                                                        ('#<procedure #6 fx->
                                                         n
                                                         1)))))
                                   (if (and ('#<procedure #2 ##fixnum?> temp.9)
                                            ('#<procedure #2 ##fixnum?> temp.8))
                                       (let ((temp.10 ('#<procedure #7 ##fx+?>
                                                       temp.8
                                                       temp.9)))
                                         (if temp.10
                                             temp.10
                                             ('#<procedure #8 fx+> temp.8 temp.9)))
                                       ('#<procedure #8 fx+> temp.8 temp.9))))))
              (temp.8 (let ((n (if ('#<procedure #2 ##fixnum?> n)
                                   (let ((temp.4 ('#<procedure #5 ##fx-?>
                                                  n
                                                  1)))
                                     (if temp.4
                                         temp.4
                                         ('#<procedure #6 fx-> n 1)))
                                   ('#<procedure #6 fx-> n 1))))
                        (if (if ('#<procedure #2 ##fixnum?> n)
                                ('#<procedure #3 ##fx<> n 2)
                                ('#<procedure #4 fx<> n 2))
                            n
                            (let ((temp.9 (fib (if ('#<procedure #2 ##fixnum?>
                                                    n)
                                                   (let ((temp.7 ('#<procedure #5 ##fx-?>
;;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                                                          n
                                                          2)))
;;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                                                     (if temp.7
                                                         temp.7
                                                         ('#<procedure #6 fx->
                                                          n
                                                          2)))
                                                   ('#<procedure #6 fx->
                                                    n
                                                    2))))
                                       (temp.8 (fib (if ('#<procedure #2 ##fixnum?>
                                                         n)
                                                        (let ((temp.4 ('#<procedure #5 ##fx-?>
;;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
                                                              n
                                                              1)))
;;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                                                         (if temp.4
                                                             temp.4
                                                             ('#<procedure #6 fx->
                                                              n
                                                              1)))
                                                        ('#<procedure #6 fx->
                                                         n
                                                         1)))))
                                   (if (and ('#<procedure #2 ##fixnum?> temp.9)
                                            ('#<procedure #2 ##fixnum?> temp.8))
                                       (let ((temp.10 ('#<procedure #7 ##fx+?>
                                                       temp.8
                                                       temp.9)))
                                         (if temp.10
                                             temp.10
                                             ('#<procedure #8 fx+> temp.8 temp.9)))
                                       ('#<procedure #8 fx+> temp.8 temp.9)))))))
          (if (and ('#<procedure #2 ##fixnum?> temp.9)
                   ('#<procedure #2 ##fixnum?> temp.8))
              (let ((temp.10 ('#<procedure #7 ##fx+?> temp.8 temp.9)))
                (if temp.10 temp.10 ('#<procedure #8 fx+> temp.8 temp.9)))
              ('#<procedure #8 fx+> temp.8 temp.9))))))

(fib 40)
```

```
(declare
     (standard-bindings)
     (block)
     (fixnum)
;;   (not safe)
)

(define (fib n)
  (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))

(fib 40)
```

gsc -c -expansion fib5.scm

```
(declare
    (standard-bindings)
    (block)
    (fixnum)
    (not safe)
)

(define (fib n)
  (if (< n 2)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))

(fib 40)
```

```
(define fib
  (lambda (n)
    (if ('#<procedure #2 ##fx<> n 2)
        n
        ('#<procedure #3 ##fx+>
         (let ((n ('#<procedure #4 ##fx-> n 1)))
           (if ('#<procedure #2 ##fx<> n 2)
               n
               ('#<procedure #3 ##fx+>
                (fib ('#<procedure #4 ##fx-> n 1))
                (fib ('#<procedure #4 ##fx-> n 2)))))
         (let ((n ('#<procedure #4 ##fx-> n 2)))
           (if ('#<procedure #2 ##fx<> n 2)
               n
               ('#<procedure #3 ##fx+>
                (fib ('#<procedure #4 ##fx-> n 1))
                (fib ('#<procedure #4 ##fx-> n 2)))))))))

(fib 40)
```

More Declarations

```
;; Partial-evaluation declarations:
;;
;; (constant-fold)                          can constant-fold primitives
;; (not constant-fold)                      can't constant-fold primitives
;;
;; Lambda-lifting declarations:
;;
;; (lambda-lift)                            can lambda-lift user procedures
;; (not lambda-lift)                        can't lambda-lift user procedures
;;
;; Inlining declarations:
;;
;; (inline)                                 compiler may inline user procedures
;; (not inline)                             no user procedure will be inlined
;;
;; (inline-primitives)                      can inline all primitives
;; (inline-primitives <var1> ...)           can inline primitives <var1> ...
;; (not inline-primitives)                  can't inline any primitives
;; (not inline-primitives <var1> ...)       can't inline primitives <var1> ...
;;
;; (inlining-limit n)                       inlined user procedures must not be
;;                                          bigger than 'n'
```

More Declarations

```
;; Compilation strategy declarations:
;;
;; (block)      global vars defined are only mutated by code in the current file
;; (separate)   global vars defined can be mutated by other code
;;
;; (core)       toplevel expressions and definitions must be compiled to code
;; (not core)   toplevel expressions and definitions belong to another module
;;
;; Global variable binding declarations:
;;
;; (standard-bindings)                    compiler can assume standard bindings
;; (standard-bindings <var1> ...)         assume st. bind. for vars specified
;; (not standard-bindings)                can't assume st. bind. for any var
;; (not standard-bindings <var1> ...)     can't assume st. bind. for vars spec.
;;
;; (extended-bindings)                    compiler can assume extended bindings
;; (extended-bindings <var1> ...)         assume ext. bind. for vars specified
;; (not extended-bindings)                can't assume ext. bind. for any var
;; (not extended-bindings <var1> ...)     can't assume ext. bind. for vars spec.
;;
;; (run-time-bindings)                    should check bindings at run-time
;; (run-time-bindings <var1> ...)         check at run-time for vars specified
;; (not run-time-bindings)                should not check bindings at run-time
;; (not run-time-bindings <var1> ...)     don't check at run-time for vars specified
```

More Declarations

```
;; Code safety declarations:
;;
;; (safe)                                runtime errors won't crash system
;; (not safe)                            assume program doesn't contain errors
;;
;; (warnings)                            show warnings
;; (not warnings)                        suppress warnings
;;
;; Interrupt checking declarations:
;;
;; (interrupts-enabled)                  allow interrupts
;; (not interrupts-enabled)              disallow interrupts
;;
;; Environment map declarations:
;;
;; (environment-map)                     generate environment maps
;; (not environment-map)                 don't generate environment maps
;;
;; Proper tail calls declarations:
;;
;; (proper-tail-calls)                   generate proper tail calls
;; (not proper-tail-calls)               don't generate proper tail calls
```

More Declarations

```
;; Proper procedure identity declarations:
;;
;; (generative-lambda)                    generate closures even when no free vars
;; (not generative-lambda)                don't generate closures when no free vars
;;
;; Optimizing dead local variables declarations:
;;
;; (optimize-dead-local-variables)     optimize dead local variables
;; (not optimize-dead-local-variables) don't optimize dead local variables
;;
;; Optimizing dead definitions declarations:
;;
;; (optimize-dead-definitions)                 compiler can remove dead defs.
;; (optimize-dead-definitions <var1> ...)      only for these var defs.
;; (not optimize-dead-definitions)             can't remove dead defs.
;; (not optimize-dead-definitions <var1> ...) only for these var defs.
```

gsc -c -expansion -gvm dead.scm

```
(define radix 10)

(define radix-set!
  (lambda (x) (set! radix x)))

(define num
  (lambda (n)
    (println (number->string n radix))))

(println (number->string 42 radix))
```

```
**** #<primitive dead#> =
#1 fs=0 entry-point nparams=0 ()
  global[radix] = '10
  frame[1] = r0
  r2 = global[radix]
  r1 = '42
  jump/poll fs=4 #2
#2 fs=4
  jump/safe fs=4 global[number->string] r0=#3 nargs=2
#3 fs=4 return-point
  r0 = frame[1]
  jump/poll fs=4 #4
#4 fs=4
  jump/safe fs=0 global[println] nargs=1
```

```
(declare
 (block)
 (optimize-dead-definitions)
)

(define radix 10)

(define (radix-set! x)
  (set! radix x))

(define (num n)
  (println
   (number->string n radix)))

(num 42)
```