CMSC 430: Introduction to Compilers

Dupe: a duplicity of types

Announcements

- Assignment 3: due 02/27
- Quiz 4: due: 03/02

- Today:
  - Dupe

Adding Boolean Values

```
(if (zero? 1) 2 3)                                  ;=> 3
(if #t 1 2)                                         ;=> 1
(if #f 1 2)                                         ;=> 2
(if 0 1 2)                                          ;=> ?
(zero? (if #t 1 2))                                 ;=> #f
(if (zero? (sub1 1)) (zero? 0) (zero? 1))           ;=> #t
```

Dupe Operational Semantics

(if #t (add1 2) 4) means 3
(if #f (add1 2) 4) means 4
(if 1 (add1 2) 4) means 3

Dupe Syntax

```
(if #t 3 4)
(if #t (add1 2) 4)
(if #f (add1 2) 4)
(if (zero? 0) (add1 2) 4)
(if 1 (add1 2) 4)
```

Dupe AST

(struct If (e1 e2 e3) #:prefab)

- Example:

```
(If (Lit #t) (Lit 1) (Lit 2)) ;;(if #t 1 2)
(If (Prim1 zero? (Lit 0)) (Lit 1) (Lit 2))
    ;;(if (zero? 0) 1 2))
```

Dupe Parser

```
(match s
  [(? datum?) (Lit s)]
  [(list-rest (? symbol? k) sr)
      ['if
        (match sr
            [(list s1 s2 s3)
             (If (parse s1) (parse s2) (parse s3))]
             [_ error …]
       [_ error ….]…
```

Dupe Interpreter

```
(define (interp e)
(match e
  [(Lit d) d]
  [(Prim1 p e)
  (interp-prim1 p (interp e))]
  [(If e1 e2 e3)
    (if (interp e1)
        (interp e2)
        (interp e3))]))
```

Dupe: Adding Boolean Values

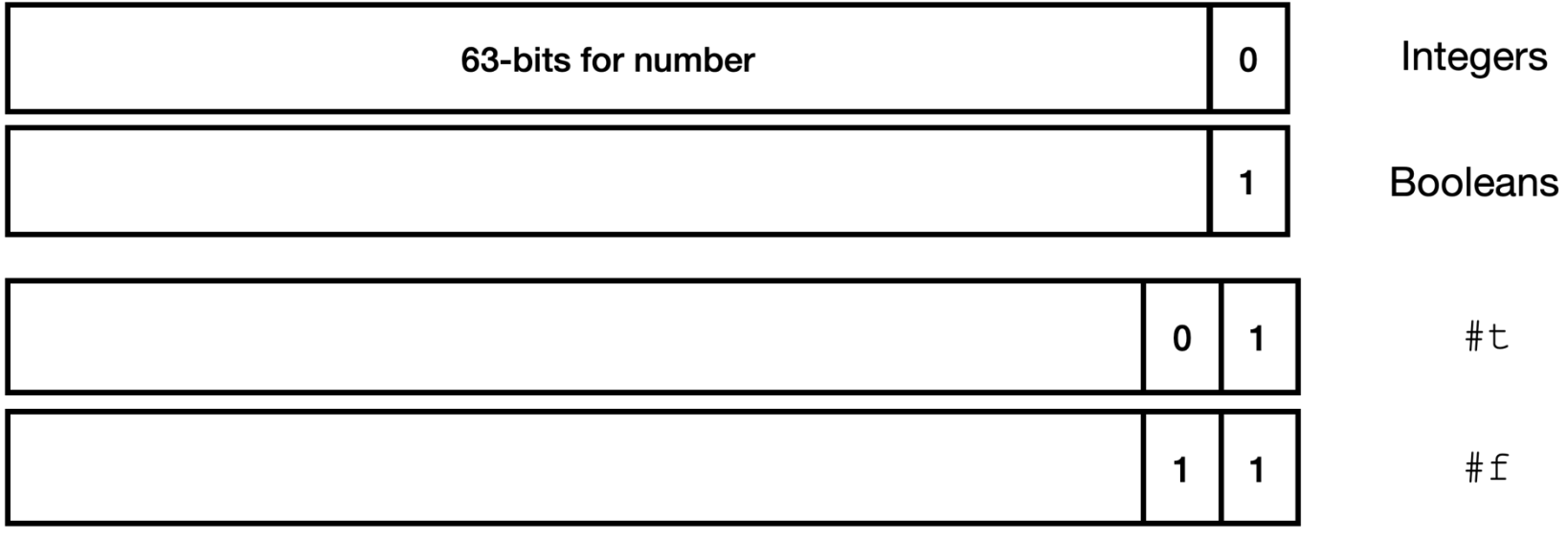
```
(compile-e (Lit 42)) => (Mov rax 42)

(compile-e (Lit #f)) => (Mov rax ?)
```

How do we make Booleans and Integers out of Integers?

Dupe: integers and booleans

- Encoding values in Dupe
  - Type tag in least significant bits

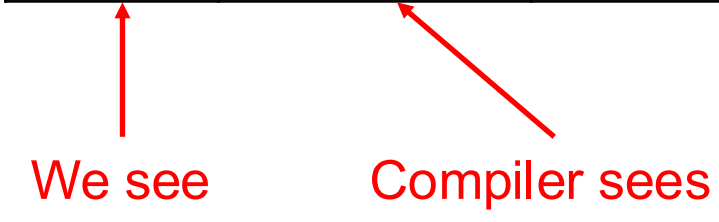


Representing Values with Bits in Dupe

Values	Bits	Decimal
0	0000	0
1	0010	2
2	0100	4
#t	0001	1
#f	0011	3

Representing Values with Bits in Dupe

Values	Bits	Decimal
0	0000	0
1	0010	2
2	0100	4
#t	0001	1
#f	0011	3



Compiler vs Interpreter

```
(asm-interp (compile (parse 1))≠(interp 1)

(asm-interp (compile (parse 1)) == 2
(interp 1) == 1
```

Values vs Bit Representation

```
(define (bits->value b)
  (cond
    [(= b (value->bits #t)) #t]
    [(= b (value->bits #f)) #f]
    [(int-bits? b) (arithmetic-shift b -1)]
    [else (error "invalid bits")]))

(define (value->bits v)
  (cond
    [(eq? v #t) #b01]
    [(eq? v #f) #b11]
    [(integer? v) (arithmetic-shift v 1)]))
```

Compiler vs Interpreter

```
(bits->value (asm-interp
  (compile (parse 1)))) == (interp 1)
```

We convert the result of asm-interp to Dupe values:

```
(bits->value (asm-interp
  (compile (parse 1))) == 1
```

Compiler

```
;; Expr Expr Expr -> Asm
(define (compile-if e1 e2 e3)
  (let ((l1 (gensym 'if))
  (l2 (gensym 'if)))
  (seq
    (compile-e e1)
    (Cmp rax (value->bits #f))
    (Je l1)
    (compile-e e2)
    (Jmp l2)
    (Label l1)
    (compile-e e3)
    (Label l2))))
```