```
| EIf(ifexpr, thenexpr, elseexpr) ->
  let _ = update_depth si in
  let test_bool =
    [ IMov(stackloc si, Reg(EAX));
      IAnd(Reg(EAX), Const(1));
      ICmp(Reg(EAX), Const(0));
      IJne(error_non_bool);
      IMov(Reg(EAX), stackloc si);] in
  let ifexpr = compile_expr ifexpr si env in
  let thenexpr = compile_expr thenexpr si env in
  let elseexpr = compile_expr elseexpr si env in
  let else_lbl = gen_temp "else" in
  let end_lbl = gen_temp "end_if" in
  ifexpr @
  test_bool @
  [ ICmp(Reg(EAX), true_const);
    IJne(else_lbl);] @
  thenexpr @
  [ IJmp(end_lbl);
    ILabel(else_lbl);] @
  elseexpr @
  [ILabel(end_lbl)]
```

```
| EIf(EPrim2(Greater, e1, e2) as ifexpr, thenexpr, elseexpr) ->
```

Instruction Selection

(if x e1 e2)

(if (f x) e1 e2)

⋮

(if (> e1 e2) thn els)

(if (> 1 3) thn els)

Which of these won't let us skip a tag check?

A
B
C
D

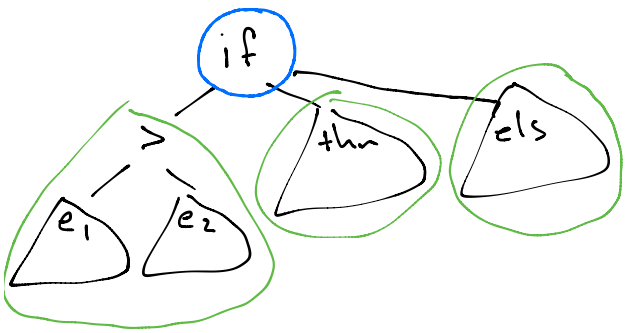
```
| EPrim2(Plus, EApp(_, _) as e1, e2) ->
| EPrim2(Plus, ENum(n), e2) ->
| EPrim2(Plus, EPlus(_, _) as e1, e2) ->
| EPrim2(Plus, EPrim2(Greater, _, _) as e1, e2) ->
```

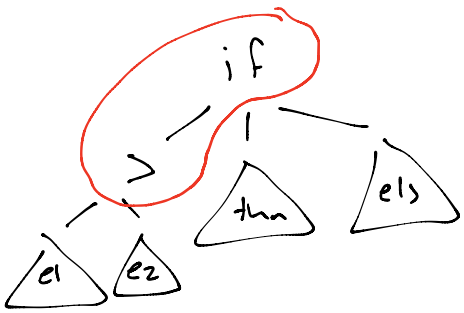
fail with "BAD!"

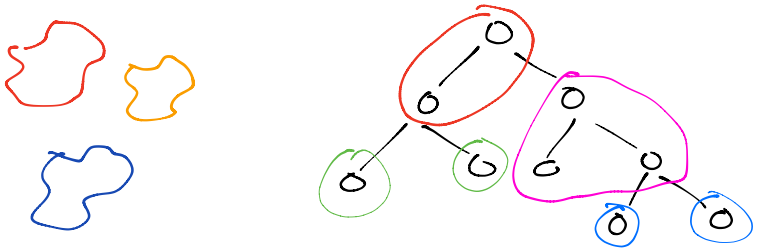
↖ Example program with effects here

Opt: .... Same answer and effects ...

(+ (> (print 1) (print 2)) 3)





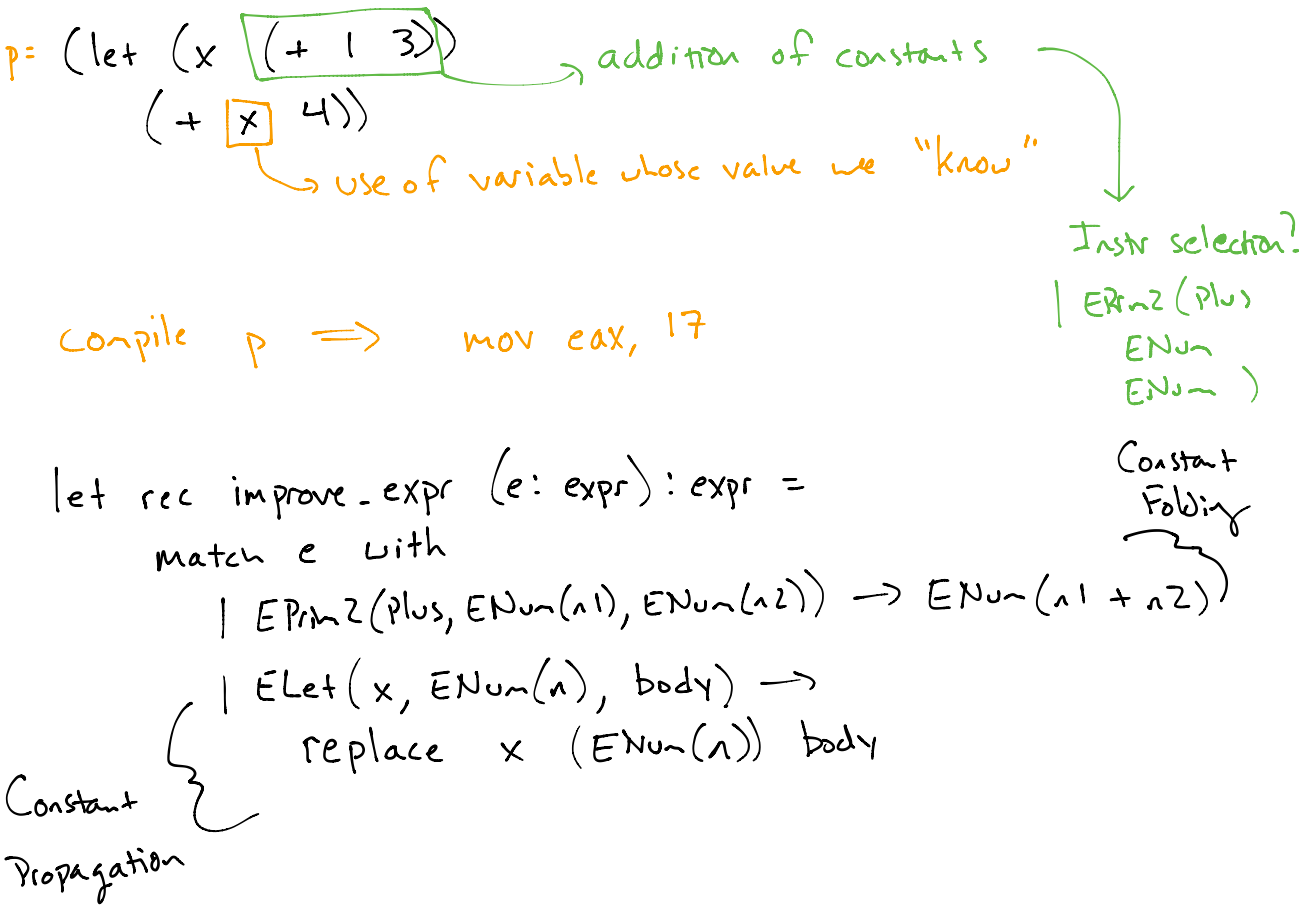


Instruction Selection

```
let rec improve_instrs (is: instr list): instr list =
  match is with
  | (IMov(RegOffset(EBP, n), Reg(EAX))::
     (IMov(Reg(EAX), RegOffset(EBP, n))::
      rest_instrs ->
improve_instrs (IMov(RegOffset(...))::rest_instrs)
  . . .
  . . .
  | i :: rest ->
    i :: (improve_instrs rest)
```

Peephole Optimization

```
f:
  mov ebp, esp
  sub esp, 12
  mov eax, 3
  mov [ebp - 4], eax
  mov eax, [ebp + 0]
  mov [ebp - 8], eax
  and eax, [ebp - 4]
  and eax, 1
  cmp eax, 1
  jne near error_non_int
  mov eax, [ebp - 8]
  and eax, 0xfffffffe
  add eax, [ebp - 4]
  jo near overflow_check
  mov [ebp - 4], eax
  mov eax, [ebp - 4]
  mov [ebp - 8], eax
  mov eax, 3
  mov [ebp - 12], eax
  and eax, [ebp - 8]
  and eax, 1
  cmp eax, 1
  jne near error_non_int
  mov eax, [ebp - 12]
  and eax, 0xfffffffe
  add eax, [ebp - 8]
  jo near overflow_check
  add esp, 16
  ret
```


p = (let (x (+ 1 3))
(+ x 4))
addition of constants
use of variable whose value we "know"
Instr selection?
| EPrim2 (Plus
ENum
ENum)
compile p ⟹ mov eax, 17
let rec improve_expr (e: expr) : expr =
match e with
| EPrim2(Plus, ENum(n1), ENum(n2)) → ENum(n1 + n2)
Constant Folding
| ELet(x, ENum(n), body) →
replace x (ENum(n)) body
Constant Propagation
and replace id e body =
....