```
let rec e_to_is e si env (tc: bool) =
                                                                    let compile_def (d : def) =
                                                                      | Def2(name, arg1, arg2, body) ->
 match e with
   | EApp2(name, arg1, arg2) ->
                                                                        let depth = stack_depth body in
     let after_label = gen_tmp "after_call" in
                                                                        let env = _
     let arg1is = e_to_is arg1 si env false in
                                                                        let bodyis = e_to_is body ___
     let arg2is = e_to_is arg2 (si + 1) env false in
                                                                                                      _____ env true in
     let init = if tc then
       [ "mov esp, ebp" ]
                                                                          sprintf "%s:" name;
                                                                          sprintf "sub esp, %d" (depth * 4);
     else
       [ "push ebp"; sprintf "push %s" after_label; ] in
     let after = if tc then [] else
                                                                        @ bodyis @
        [ sprintf "%s:" after_label; "pop ebp"] in
     arg1is @ [ sprintf "mov %s, eax" (stackval si)] @
                                                                          sprintf "mov esp, ebp";
     arg2is @ [ sprintf "mov %s, eax" (stackval (si + 1))] @
                                                                          "ret"
      init @ [
      sprintf "mov eax, %s" (stackval si); "push eax";
      sprintf "mov eax, %s" (stackval (si + 1)); "push eax";
                                                                    type expr =
      "mov ebp, esp";
      "add ebp, 8";
                                                                      | EApp2 of string * expr * expr
       sprintf "jmp %s" name;
                                                                    type def =
      ] @
      after
                                                                      | Def2 of string * string * string * expr
   | EPlus(e1, e2) ->
     let e1is = e_to_is e1 si env false in
     let e2is = e_to_is e2 (si + 1) env false in
                                                                   let rec stack_depth (e : expr) =
     (* code for doing addition *)
                                                                     match e with
   | ELet(x, v, body) ->
                                                                       | ENum(_) | EBool(_) | EId(_) -> 0
     let vis = e_to_is v si env false in
                                                                       | ELet(x, v, body) ->
     let bis = e_{to} is body (si + 1) ((x,si)::env) tc in
                                                                         (max (stack_depth v) ((stack_depth body) + 1))
     (* code for saving var, body *)
                                                                       | EPlus(lhs, rhs) ->
                                                                         (max (stack_depth lhs) ((stack_depth rhs) + 1)) + 1
   | EIf(cond, thn, els) ->
     let condis = e_to_is cond si env false in
                                                                       | EApp2(name, arg1, arg2) ->
     let afterlabel = gen_tmp "after_if" in
     let elslabel = gen_tmp "else" in
                                                                       | EIf(cond, thn, els) ->
                                                                         (max (max (stack_depth cond) (stack_depth thn))
     let thnis = e_to_is thn si env tc in
                                                                               (stack_depth els))
     let elsis = e_to_is els si env tc in
     (* code for checking condition, then, els, etc *)
                          (Note – the "_" argument is just to match
                                                                   (def (sum n sofar)
                                                                                               sum:
   (def (sum n _)
                                                                                               sub esp, 12
                          the EApp2 case, so we just have to look at
                                                                    (if n
    (if n
                          one case for applications)
                                                                     (sum (+ n -1) (+ n sofar))
                                                                                               mov eax, [ebp - 4]
     (+ n (sum (+ n -1) _))
                                                                     sofar))
                                                                                               cmp eax, 0
     0))
                                                                                               je else2
                          sum:
                                                                   (def (our_main input)
                                                                                               mov eax, [ebp - 4]
                          sub esp, 20
   (def (our_main input)
                                                                                               mov [ebp - 12], eax
                                                                    (sum 3 0))
                          mov eax, [ebp - 4]
    (sum 3 0))
                                                                                               mov eax, -1
                          cmp eax, 0
                                                                                               mov [ebp - 16], eax
                          je else2
                                                                                               mov eax, [ebp - 12]
                          mov eax, [ebp - 4]
                                                                                               add eax, [ebp - 16]
                          mov [ebp - 8], eax
                                                                                               mov [ebp - 12], eax
                          mov eax, [ebp - 4]
                                                                                               mov eax, [ebp - 4]
                          mov [ebp - 12], eax
                                                                                               mov [ebp - 16], eax
                          mov eax, -1
                                                                                               mov eax, [ebp - 8]
                          mov [ebp - 16], eax
                                                                                               mov [ebp - 20], eax
                          mov eax, [ebp - 12]
                                                                                               mov eax, [ebp - 16]
                          add eax, [ebp - 16]
                                                                                               add eax, [ebp - 20]
                                                                                               mov [ebp - 16], eax
                          mov ebp, esp
                                                                                               mov ebp, esp
                          push eax
                                                                                               mov eax, [ebp - 12]
                          jmp sum
                                                                                               push eax
                                                                                               mov eax, [ebp - 16]
                                                                                               push eax
                          mov [ebp - 12], eax
                                                                                               jmp sum
                          mov eax, [ebp - 8]
                          add eax, [ebp - 12]
                                                                                               jmp after_if1
                          jmp after_if1
                                                                                               else2:
                          else2:
                                                                                               mov eax, [ebp - 8]
                          mov eax, 0
                                                                                               after_if1:
                          after if1:
                                                                                               mov esp, ebp
                          mov esp, ebp
                                                                                               ret
                          ret
```

```
(Note – the "_" argument is just to match
                                                                    (def (sum n sofar)
                                                                                                sum:
   (def (sum n _)
                          the EApp2 case, so we just have to look at
                                                                    (if n
                                                                                                sub esp, 12
    (if n
                          one case for applications)
                                                                     (sum (+ n -1) (+ n sofar))
                                                                                                mov eax, [ebp - 4]
    (+ n (sum (+ n -1) _))
                                                                                                cmp eax, 0
                                                                     sofar))
    0))
                                                                                                je else2
                          sum:
                                                                    (def (our_main input)
                                                                                                mov eax, [ebp - 4]
                          sub esp, 20
   (def (our_main input)
                                                                    (sum 3 0))
                                                                                                mov [ebp - 12], eax
                          mov eax, [ebp - 4]
    (sum 3 0))
                                                                                                mov eax, -1
                          cmp eax, 0
                                                                                                mov [ebp - 16], eax
                          je else2
                                                                                                mov eax, [ebp - 12]
                          mov eax, [ebp - 4]
                                                                                                add eax, [ebp - 16]
                          mov [ebp - 8], eax
                                                                                                mov [ebp - 12], eax
                          mov eax, [ebp - 4]
                                                                                                mov eax, [ebp - 4]
                          mov [ebp - 12], eax
                                                                                                mov [ebp - 16], eax
                          mov eax, -1
                                                                                                mov eax, [ebp - 8]
                          mov [ebp - 16], eax
                                                                                                mov [ebp - 20], eax
                          mov eax, [ebp - 12]
                                                                                                mov eax, [ebp - 16]
                          add eax, [ebp - 16]
                                                                                                add eax, [ebp - 20]
                                                                                                mov [ebp - 16], eax
                          mov ebp, esp
                                                                                                mov ebp, esp
                          push eax
                                                                                                mov eax, [ebp - 12]
                          jmp sum
                                                                                                push eax
                                                                                                mov eax, [ebp - 16]
                                                                                                push eax
                          mov [ebp - 12], eax
                                                                                                jmp sum
                          mov eax, [ebp - 8]
                          add eax, [ebp - 12]
                                                                                                jmp after_if1
                          jmp after_if1
                                                                                                else2:
                          else2:
                                                                                                mov eax, [ebp - 8]
                          mov eax, 0
                                                                                                after_if1:
                          after_if1:
                          mov esp, ebp
                                                                                                mov esp, ebp
                                                                                                ret
                          ret
let rec e_to_is e si env (tc: bool) =
 match e with
                                                                   let compile_def (d : def) =
   | EApp2(name, arg1, arg2) ->
                                                                      Def2(name, arg1, arg2, body) ->
     let after_label = gen_tmp "after_call" in
                                                                        let depth = stack_depth body in
     let arg1is = e_to_is arg1 si env false in
                                                                        let env =
     let arg2is = e_to_is arg2 (si + 1) env false in
                                                                        let bodyis = e_to_is body __
     let init = if tc then
                                                                                                        ____ env true in
       [ "mov esp, ebp" ]
                                                                          sprintf "%s:" name;
     else
       [ "push ebp"; sprintf "push %s" after_label; ] in
                                                                          sprintf "sub esp, %d" (depth * 4);
     let after = if tc then [] else
     [ sprintf "%s:" after_label; "pop ebp"] in
arg1is @ [ sprintf "mov %s, eax" (stackval si)] @
                                                                        @ bodyis @
     arg2is @ [ sprintf "mov %s, eax" (stackval (si + 1))] @
                                                                          sprintf "mov esp, ebp";
      init @ [
                                                                          "ret"
      sprintf "mov eax, %s" (stackval si); "push eax";
      sprintf "mov eax, %s" (stackval (si + 1)); "push eax";
      "mov ebp, esp";
                                                                    type expr =
      "add ebp, 8";
       sprintf "jmp %s" name;
                                                                      | EApp2 of string * expr * expr
      ] @
                                                                    type def =
      after
   | EPlus(e1, e2) ->
     let e1is = e_to_is e1 si env false in
                                                                      | Def2 of string * string * string * expr
     let e2is = e_to_is e2 (si + 1) env false in
     (* code for doing addition *)
                                                                   let rec stack_depth (e : expr) =
   | ELet(x, v, body) ->
                                                                     match e with
     let vis = e_to_is v si env false in
                                                                       | ENum(_) | EBool(_) | EId(_) -> 0
     let bis = e_{to_i} body (si + 1) ((x,si)::env) tc in
                                                                        | ELet(x, v, body) ->
     (* code for saving var, body *)
                                                                         (max (stack_depth v) ((stack_depth body) + 1))
   | EIf(cond, thn, els) ->
                                                                       | EPlus(lhs, rhs) ->
     let condis = e_to_is cond si env false in
                                                                         (max (stack_depth lhs) ((stack_depth rhs) + 1)) + 1
     let afterlabel = gen_tmp "after_if" in
                                                                       | EApp2(name, arg1, arg2) ->
     let elslabel = gen_tmp "else" in
     let thnis = e_to_is thn si env tc in
                                                                       | EIf(cond, thn, els) ->
     let elsis = e_to_is els si env tc in
                                                                         (max (max (stack_depth cond) (stack_depth thn))
     (* code for checking condition, then, els, etc *)
                                                                               (stack_depth els))
```