Section 12 (CS 164 Fall 2025)

All questions from CS 164 midterm Fall 2023

Question 7 (9 points)

7.1 For the block **L1**, perform a *liveness analysis*, listing the temporaries ***in order (t0…tn)*** associated with each space. The final cell has been filled in for you with the implicit return value of this block: **t12**.

```
L1:

t0 <- 10

t1 <- 14

t2 <- t0 + t1

t3 <- t2 + t1

t4 <- read_num()

t5 <- t0 + t4

t6 <- t3 + t1

t7 <- t5 + t3

t8 <- t6 + t5

t9 <- t2 + t7

t10 <- t2 + t9

t11 <- t8 + t7

t12 <- t11 + t9
                              t12
```

7.2 For block **L1**, what is the largest set of temporaries that could be colored the same color/mapped to the same register? If there are multiple sets of the same length, write one of them.

Answer Here:

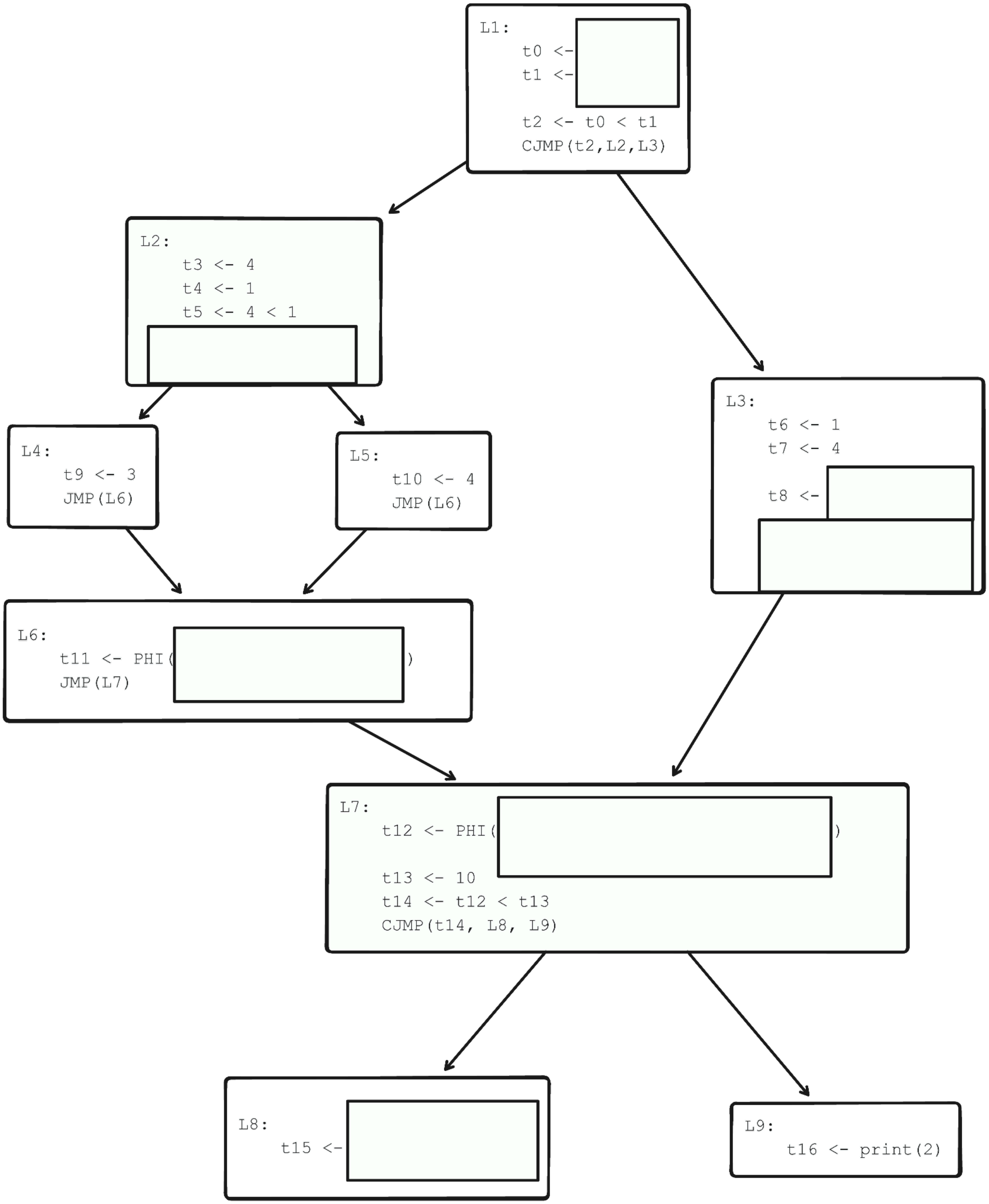
Remember to list the temporaries ***in order (t0…tn)!***

Question 9 (4 points)

Below is the Control-Flow Graph constructed from this 164Lang Program:

```
(let ((x (if (< 3 5)(if (< 4 1) 3 4)(- 1 4))))(if (< x 10)(print x)(print 2))
```

Your task is to **fill in the holes in the basic blocks**:



```
L1:
    t0 <-
    t1 <-

    t2 <- t0 < t1
    CJMP(t2,L2,L3)
```

```
L2:
    t3 <- 4
    t4 <- 1
    t5 <- 4 < 1
```

```
L3:
    t6 <- 1
    t7 <- 4

    t8 <-
```

```
L4:
    t9 <- 3
    JMP(L6)
```

```
L5:
    t10 <- 4
    JMP(L6)
```

```
L6:
    t11 <- PHI(                    )
    JMP(L7)
```

```
L7:
    t12 <- PHI(                    )

    t13 <- 10
    t14 <- t12 < t13
    CJMP(t14, L8, L9)
```

```
L8:
    t15 <-
```

```
L9:
    t16 <- print(2)
```

Question 12 (7.5 points)

For the following subquestions, assume we will run common subexpression elimination (CSE) once and function inlining once, but we have the option to vary the order. For function inlining, assume that we inline every possible function.

(a)

```
(define (foo) 35)
(+ (foo) (foo))
```

Please fill in the box next to the correct end of this sentence: For the program above...

□ ...the AST will be smaller if we run common subexpression elimination **before** function inlining.
□ ...the AST will be smaller if we run common subexpression elimination **after** function inlining.
□ ...the AST will be the same whether we run common subexpression elimination or function inlining first.

(b)

```
(define (foo) (+ 35 (+ 49 (+ 70 (+ 1 56)))))
(+ (foo) (foo))
```

Please fill in the box next to the correct end of this sentence: For the program above...

□ ...the AST will be smaller if we run common subexpression elimination **before** function inlining.
□ ...the AST will be smaller if we run common subexpression elimination **after** function inlining.
□ ...the AST will be the same whether we run common subexpression elimination or function inlining first.

(c)

```
(define (bar x) (- (read-num) x))
(define (foo) (+ 35 (bar 4))
(+ (foo) (foo))
```

Please fill in the box next to the correct end of this sentence: For the program above...

□ ...the AST will be smaller if we run common subexpression elimination **before** function inlining.
□ ...the AST will be smaller if we run common subexpression elimination **after** function inlining.
□ ...the AST will be the same whether we run common subexpression elimination or function inlining first.

For the following subquestions, assume we will run common subexpression elimination (CSE) once and constant propagation once, but we have the option to vary the order.

(d)

```
(define (a) (+ 2 (+ 3 (+ 4 (+ 5 6)))))
(define (b) (+ 2 (+ 3 (+ 4 (+ 5 6)))))
(+ (a) (b))
```

Please fill in the box next to the correct end of this sentence: For the program above...

□ ...the AST will be smaller if we run CSE **before** constant propagation.
□ ...the AST will be smaller if we run CSE **after** constant propagation.
□ ...the AST will be the same whether we run CSE or constant propagation first.

(e)

```
(let ((x 3))
(let ((f (lambda (a) (+ a x))))
(let ((x 9))
(let ((f (lambda (a) (+ a x))))
(f 27)
))))
```

Please fill in the box next to the correct end of this sentence: For the program above...

□ ...the AST will be smaller if we run CSE **before** constant propagation.
□ ...the AST will be smaller if we run CSE **after** constant propagation.
□ ...the AST will be the same whether we run CSE or constant propagation first.