上 海 交 通 大 学 试 卷（A 卷）

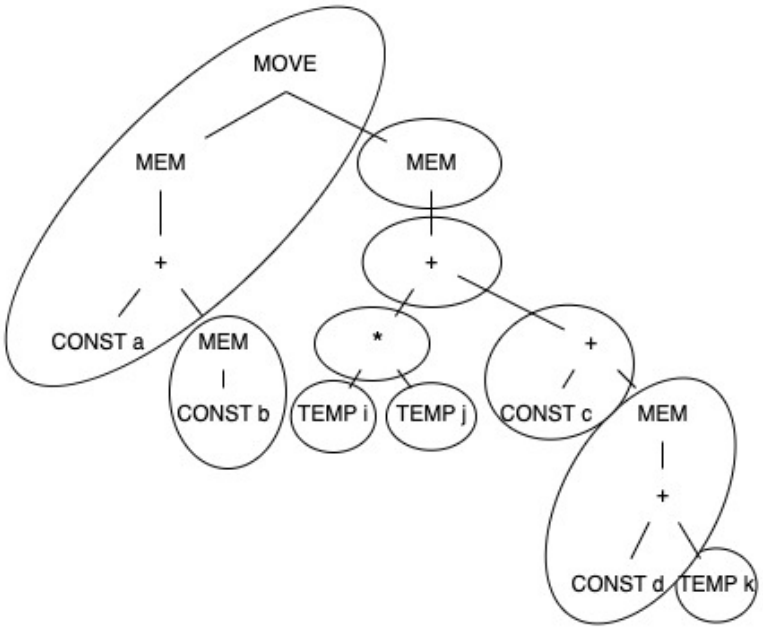
（ 2022 至 2023 学年 第 1 学期 ）

班级号_____ 学号_____ 姓名 _____

课程名称_____计算机系统基础（汇编）_____成绩 _____

Problem 1: Instruction Selection

1.



(4', -1 for each incorrect tile)

instruction:

r1 ← M[rk + d]
r2 ← r1 + c
r3 ← ri * rj
r4 ← r3 + r2
r5 ← M[r4]
r6 ← M[b]
M[r6 + a] ← r5

(7', 1 for each instruction, order is not unique)

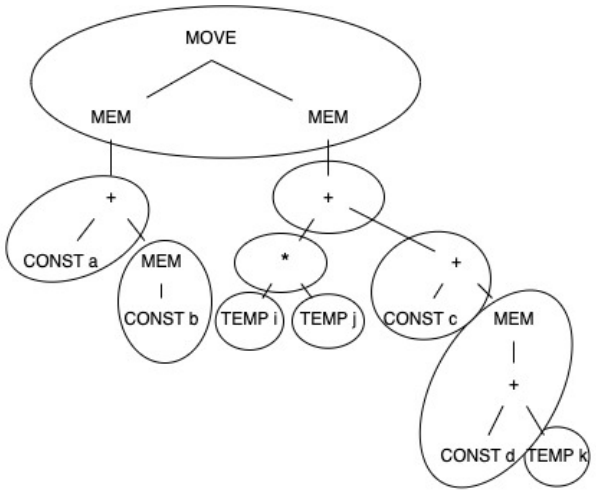
Solution for Volume A:

cost:26(1')

Solution for Volume B:

cost:27(1')

2.



(4′, -1 for each incorrect tile)

instruction:

r1 ← M[rk + d]
r2 ← r1 + c
r3 ← ri * rj
r4 ← r3 + r2
r5 ← M[b]
r6 ← r5 + a
M[r6] ← M[r4]

(7′, 1 for each instruction, order is not unique)

Solution for Volume A:

cost:25(1′)

Solution for Volume A:

cost:26(1′)

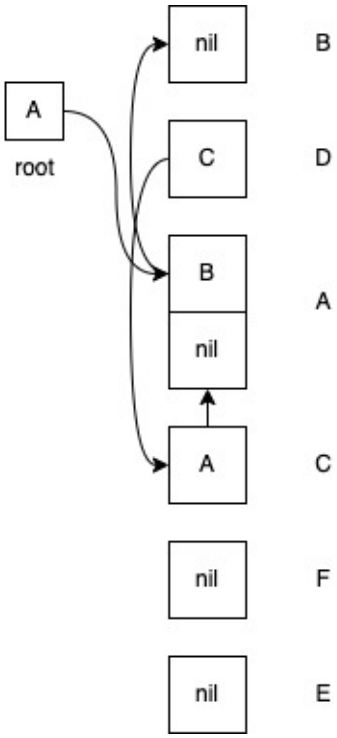
Additional explanation:
1. It's fine to dismiss r0 and add const zero.
2. Instruction must follow the format in effect, like r1 <- r1 + c is not allowed.
3. Directly use i/j/k to represent temp i/j/k is not allowed.

Problem 2: Garbage Collection

1. Cycles of garbage cannot be reclaimed by reference counting. (4′)
2. For G0, we can do garbage collection on G0 and G1 together because there might be many newer objects pointing to older objects. For G2, we make the compiled program remember where there are pointers from old objects to new ones (e.g., using remembered list, remembered set, card marking, or page marking). (5′, 2′ for G0, 3′ for G2)
3. Construct the pointer map at compile time to record the registers or frame locations of live pointers in every frame. (5′, any answer related to pointer analysis at compile time

could score)

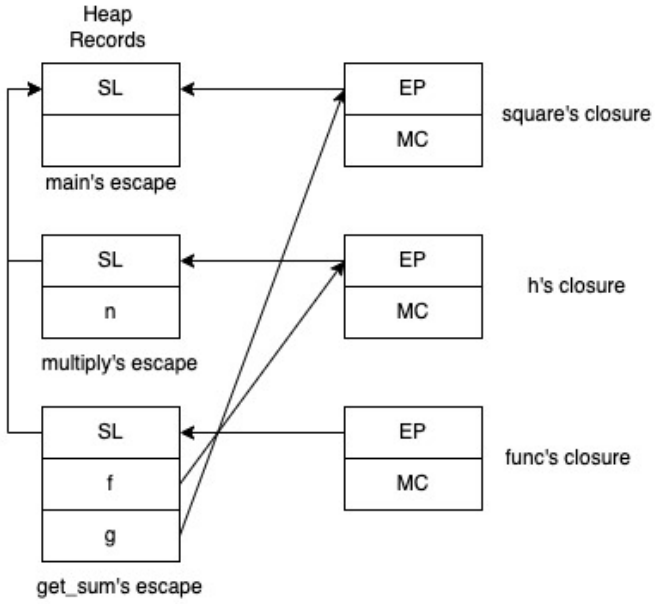
4.



(10’, 1’ for each arrow, 1’ for each field, -1 for each redundant arrow)

Problem 3: Functional Programming

1.



(8’, 3’ for multiply’s escape, 2’ for h’s closure, 1’ for each arrow)

2.

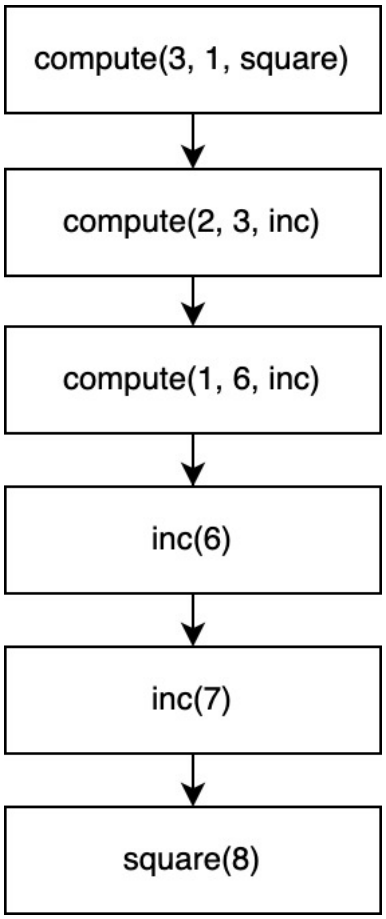
Solution for Volume A:

sum = 3 (4’)

Solution for Volume B:

sum = 10 (4’)

3.

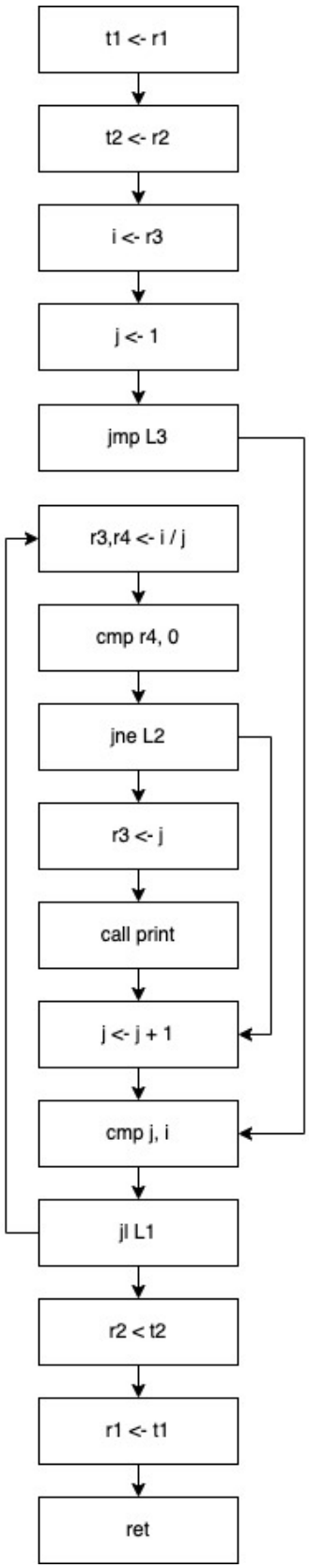


(8’)

Problem 4: Register Allocation

(Volume B switched i and j)

1.



(4', -1 for every instruction that has incorrect incoming/outgoing arrows).

2.

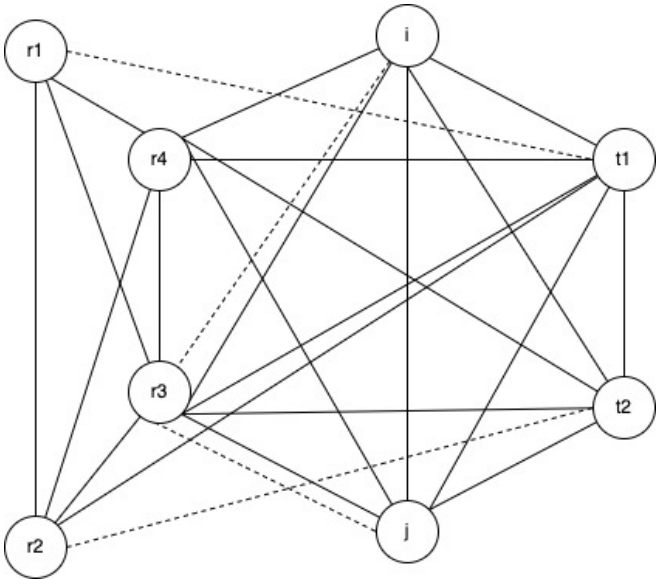
instr	def	use	in	out
t1 <- r1	t1	r1	r1, r2, r3	r2, r3, t1
t2 <- r2	t2	r2	r2, r3, t1	r3, t1, t2
i <- r3	i	r3	r3, t1, t2	t1, t2, i
j <- 1	j		t1, t2, i	t1, t2, i, j
jmp L3			t1, t2, i, j	t1, t2, i, j
r3,r4 <- i / j	r3, r4	i, j	t1, t2, i, j	r4, t1, t2, i, j
cmp r4, 0		r4	r4, t1, t2, i, j	t1, t2, i, j
jne L2			t1, t2, i, j	t1, t2, i, j
r3 <- j	r3	j	t1, t2, i, j	r3, t1, t2, i, j
call print	r3, r4	r3	r3, t1, t2, i, j	t1, t2, i, j
j <- j + 1	j	j	t1, t2, i, j	t1, t2, i, j
cmp j, i		i, j	t1, t2, i, j	t1, t2, i, j
jl L1			t1, t2, i, j	t1, t2, i, j
r2 <- t2	r2	t2	t1, t2	r2, t1
r1 <- t1	r1	t1	r2, t1	r1, r2
ret			r1, r2	**r1, r2**

(8', 2' for each column, -1 for every incorrect cell).

Additional explanation:

1. ret's live-out must be filled in
2. ret's use should not be filled

3.



(4', -1 for every node with incorrect line).

4.

(1) Spill: t1 (2')

```
Calculate spill priority:
    i = (1 + 10 * 2) / 5 = 4.2
    j = (1 + 10 * 5) / 5 = 10.2
    t1 = (2 + 10 * 0) / 6 = 0.33
    t2 = (2 + 10 * 0) / 5 = 0.4
```

Spill t1, because it's spill priority is the lowest.

(2) Spill: t2 (2')

```
Calculate spill priority:
    i = (1 + 10 * 2) / 4 = 5.25
    j = (1 + 10 * 5) / 4 = 12.75
    t2 = (2 + 10 * 0) / 4 = 0.5
```

Spill t2, because it's spill priority is the lowest.

(3) Simplify i (the move of r3&i is constrained) (1')

(4) Simplify j (the move of r3&j is constrained) (1')

(5) Select: actual spill t1 and t2 (1')

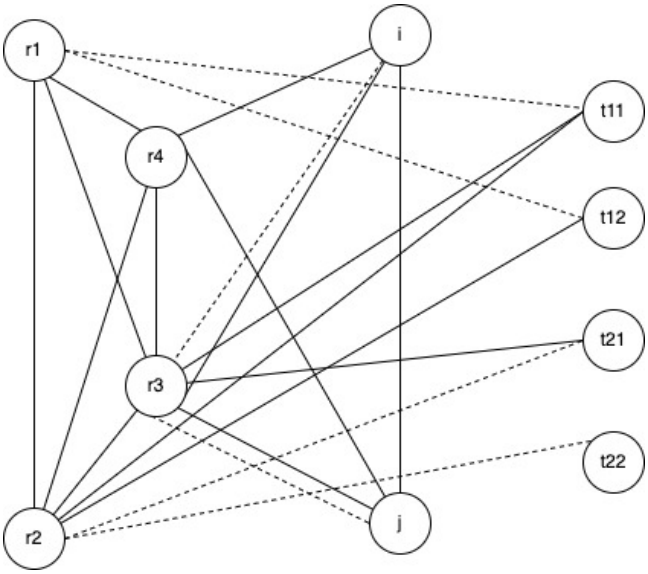
(6) Rewrite program: Rewrite program (3')

```
find_factor:
    t11 <- r1
    M[t1loc] <- t11
    t21 <- r2
    M[t2loc] <- t21
    i <- r3
    j <- 1
    jmp L3
```

```
L1:
   r3,r4 <- i / j
   cmp r4, 0
   jne L2
   r3 <- j
   call print
L2:
   j <- j + 1
L3:
   cmp j, i
   jl L1
   t22 <- M[t2loc]
   r2 <- t22
   t12 <- M[t1loc]
   r1 <- t12
   ret
```

instr	def	use	in	out
`t11 <- r1`	t11	r1	r1, r2, r3	r2, r3, t11
`M[t1loc] <- t11`		t11	r2, r3, t11	r2, r3
`t21 <- r2`	t21	r2	r2, r3	r3, t21
`M[t2loc] <- t21`		t21	r3, t21	r3
`i <- r3`	i	r3	r3	i
`j <- 1`	j		i	i, j
`jmp L3`			i, j	i, j
`r3,r4 <- i / j`	r3, r4	i, j	i, j	r4, i, j
`cmp r4, 0`		r4	r4, i, j	i, j
`jne L2`			i, j	i, j
`r3 <- j`	r3	j	i, j	r3, i, j
`call print`	r3, r4	r3	r3, i, j	i, j
`j <- j + 1`	j	j	i, j	i, j
`cmp j, i`		i, j	i, j	i, j

jl L1			i, j	i, j
t22 <- M[t2loc]	t22			t22
r2 <- t22	r2	t22	t22	r2
t12 <- M[t1loc]	t12		r2	r2, t12
r1 <- t12	r1	t12	r2, t12	r1, r2
ret			r1, r2	r1, r2



(7) Simplify i (the move of r3&i is constrained) (1')
(8) Simplify j (the move of r3&j is constrained) (1')
(9) Coalesce (George): r1 – t11, r1 – t12, r2 – t21, r2 – t22 (2')
(10) Select (2')

Node	Color
t11	r1
t12	r1
t21	r2
t22	r2
i	r1
j	r2

```
find_factor:
   M[t1loc] <- r1
```

```
    M[t2loc] <- r2
    r1 <- r3
    r2 <- 1
    jmp L3
L1:
    r3,r4 <- r1 / r2
    cmp r4, 0
    jne L2
    r3 <- r2
    call print
L2:
    r2 <- r2 + 1
L3:
    cmp r2, r1
    jl L1
    r2 <- M[t2loc]
    r1 <- M[t1loc]
    ret
```

Note that there can't apply Coalesce Rule at the very begin of RA. Because YOU SHOULD NOT coalesce a physical reg to a temp reg. (but you can coalesce a temp reg to a physical reg).