上 海 交 通 大 学 试 卷（A 卷）

（ 2021 至 2022 学年 第 1 学期 ）

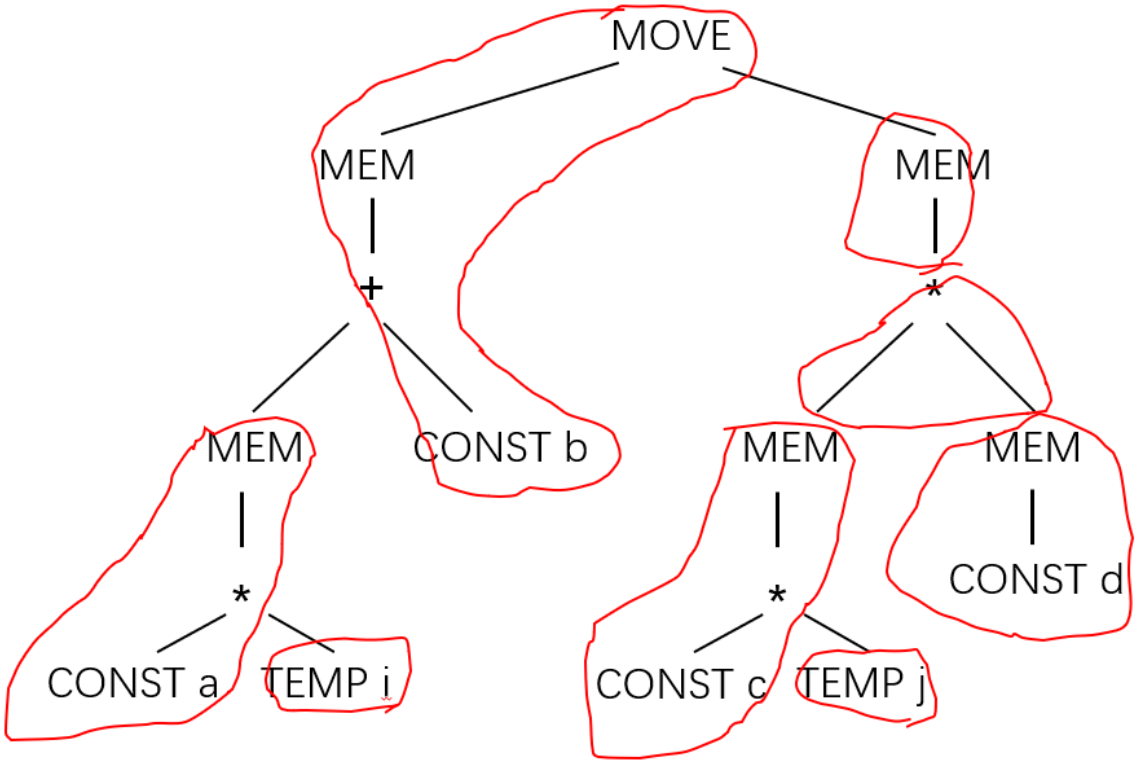
班级号_____ 学号_____ 姓名 _____

课程名称_____计算机系统基础（汇编）_____成绩 _____

Problem 1: Instruction Selection

volume b rename const a,b,c,d to i,j,k,l respectively and rename temp i,j to a,b accordingly.

1.



(3', -1 for each incorrect tile)

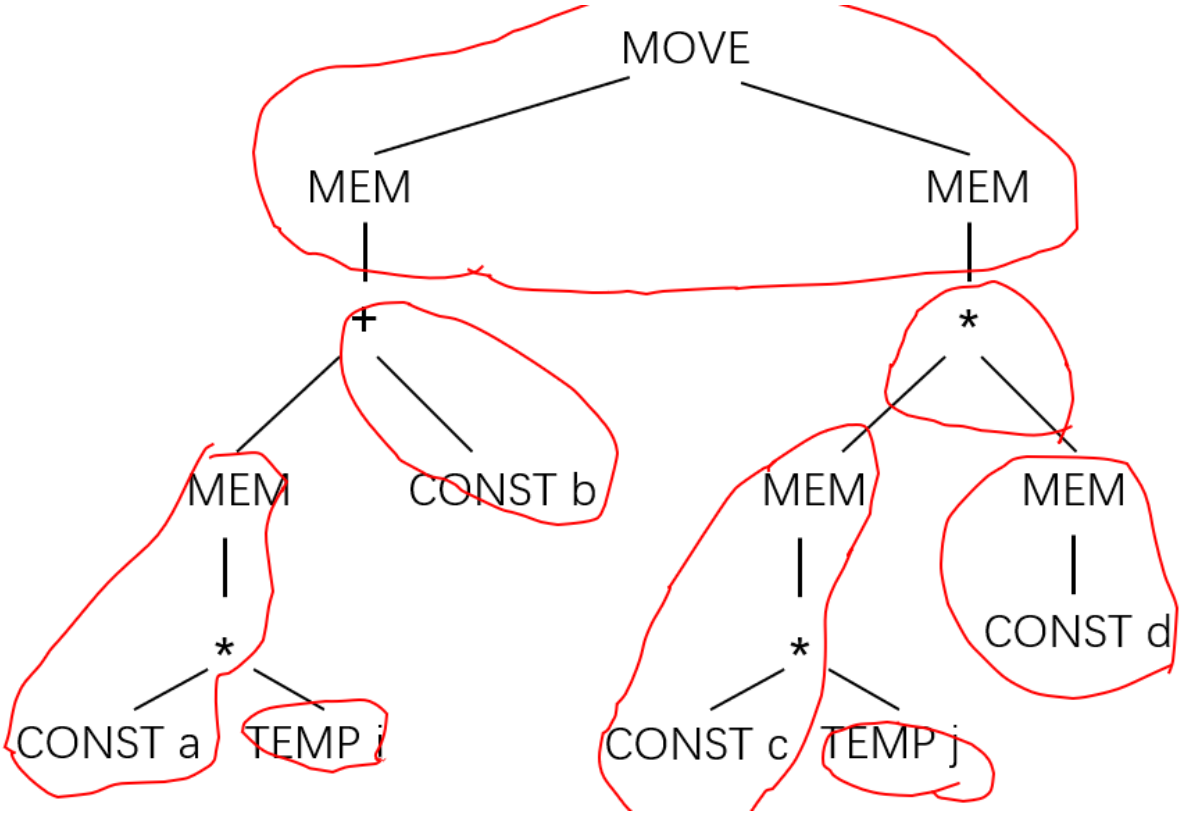
instruction:

r1 ← M[ri * a]
r2 ← M[rj * c]
r3 ← M[r0+d]
r4 ← r2 * r3
r5 ← M[r4]
M[r1 + b] ← r5

(6', 1 for each instruction, order is not unique)

cost:19(2')

2.



(4', -1 for each incorrect tile)

instruction:

r1 ← M[ri * a]
r2 ← r1+b
r3 ← M[rj * c]
r4 ← M[r0+d]
r5 ← r3 * r4
M[r2] ← M[r5]

(6', 1 for each instruction, order and register number is not unique)

cost:18(3')

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 to represent temp i/j is not allowed.

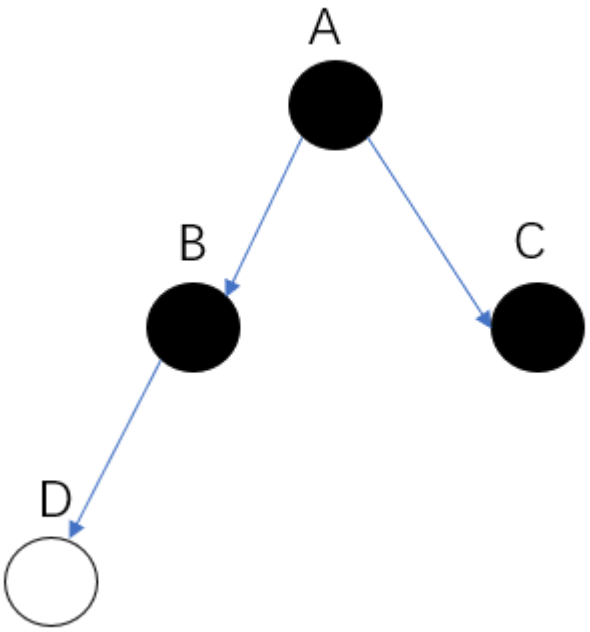
Problem 2: Garbage Collection

1. pros: Short pause time. cons: Extra overhead in runtime.(4', 2' for each)
2. Remark

```
Stop Mutator
for object in New:
  object.color <- black
end for
```

(4', 2' for stop Mutator, 2' for marking)

3. No, suppose we have graph like this, where C initially has a reference to D. And after B is marked black, C set the reference to D to null and B set a new reference to D. In the end, the graph will become like A,B,C in black and D in white, which incurs problem.




```
WriteBarrier(obj, ref):
  if obj.color is black and *ref.color is white:
    *ref.color <- grey
    stack.push(ref)
  end if

Remark
  Stop Mutator
  for object in New:
    object.color <- black
  end for
  while stack is not empty:
    ref <- stack.pop()
    Mark(*ref)
  end while
```

(8′, 2′ for describe problem, 4′ for write barrier, 2′ for Remark)

4.Floating garbage comes from two part: The first part is setting all reference to an object to null after it has been marked black, this object will be dead and still marked black. The second part is garbage in New set.(6′, 3′ for each part) This algorithm cannot fix floating garbage problem unless do the marking job again, which violates the principle of concurrent collection(2′, any answer making sense could score).

Additional explanation:

The second question :

Mark without trimming stack later(1')

Simply set objects black without stop mutator(2')

Mark and trimming the stack later without stop mutator(3')

Stop mutator at the beginning but start it after finish(-1')

The third question:

Correctly describe write barrier in words is OK.

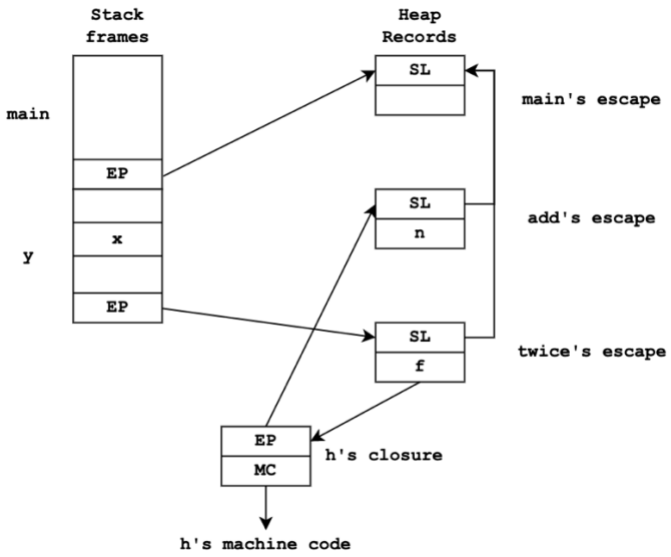
Any answers that make sense can get score.

The forth question:

Root objects died after initial mark is OK.

Problem 3: Functional Programming

1.



(8', 4' for twice's escape, 1' for each arrow, 1' for "f" in twice's escape)

2.

Solution for Volume A:

x = 17 (2')

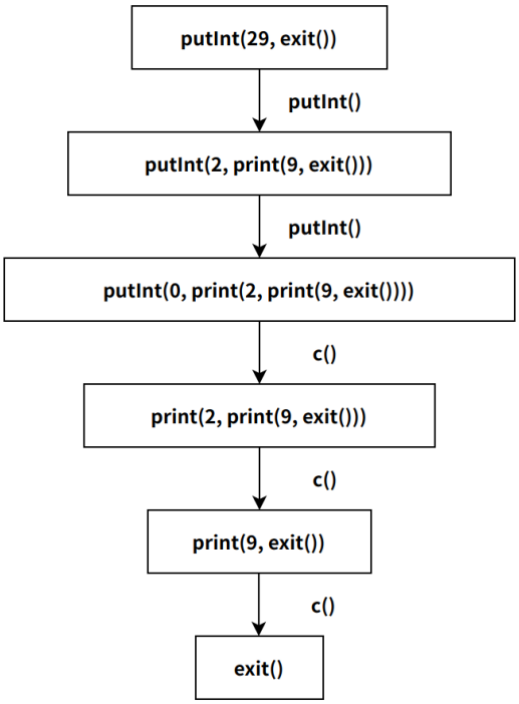
n = 6 (2')

Solution for Volume B:

x = 19 (2’)

n = 5 (2’)

3.

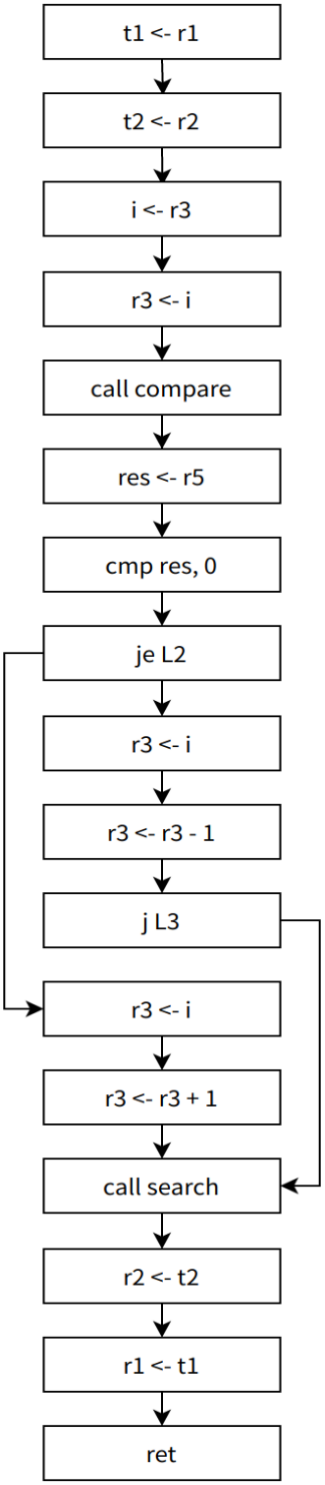


(8’, 2’ for each line)

Problem 4: Register Allocation

(Volume B rename r5 to r0)

1.



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

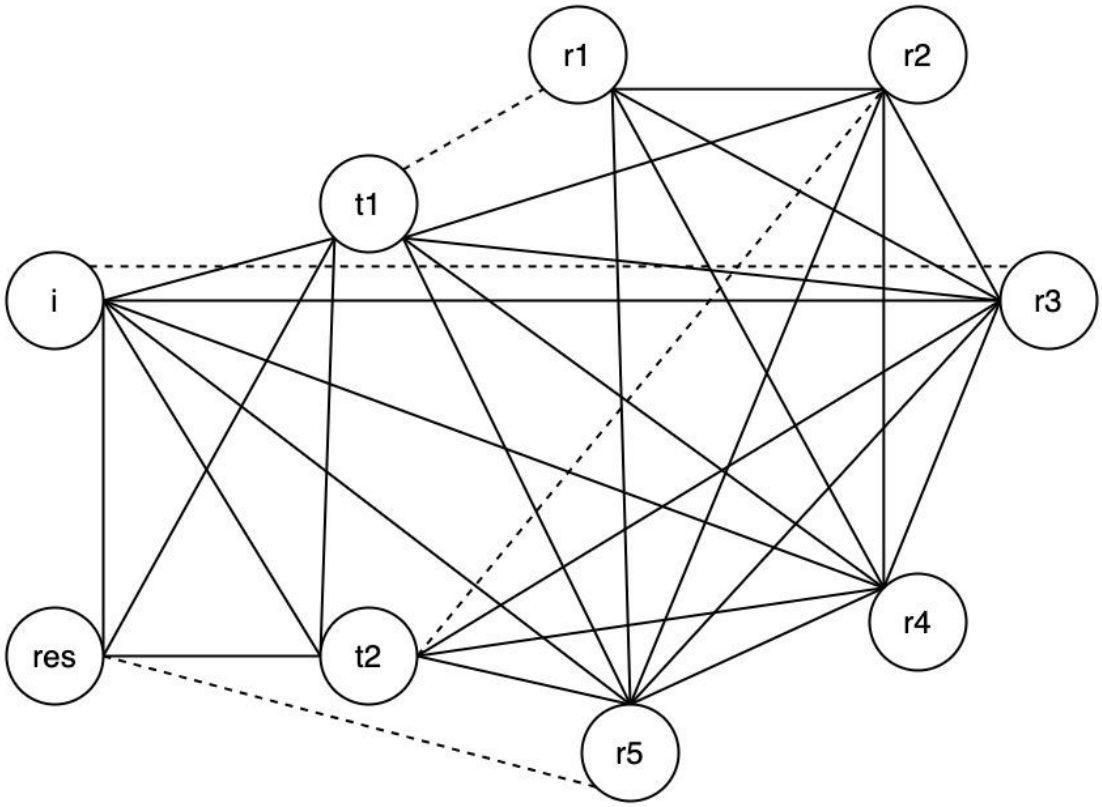
2.

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	r3, t1, t2, i
call compare	r3, r4, r5	r3	r3, t1, t2, i	r5, t1, t2, i
res <- r5	res	r5	r5, t1, t2, i	t1, t2, i, res
cmp res, 0		res	t1, t2, i, res	t1, t2, i
je L2			t1, t2, i	t1, t2, i
r3 <- i	r3	i	t1, t2, i	r3, t1, t2
r3 <- r3 - 1	r3	r3	r3, t1, t2	r3, t1, t2
j L3			r3, t1, t2	r3, t1, t2
r3 <- i	r3	i	t1, t2, i	r3, t1, t2
r3 <- r3 + 1	r3	r3	r3, t1, t2	r3, t1, t2
call deadloop_search	r3, r4, r5	r3	r3, t1, t2	r5, t1, t2
r2 <- t2	r2	t2	r5, t1, t2	r2, r5, t1
r1 <- t1	r1	t1	r2, r5, t1	r1, r2, r5
ret			r1, r2, r5	r1, r2, r5

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

3.



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

4.

1. Coalesce (George): res - r5 (1')
2. Spill: t1 (2', 1' for spilling t1, 1' for spill priority calculation)

```
Calculate spill priority:
    i = 4 / 5 = 0.8
    t1 = 2 / 6 = 0.33
    t2 = 2 / 5 = 0.4
```

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

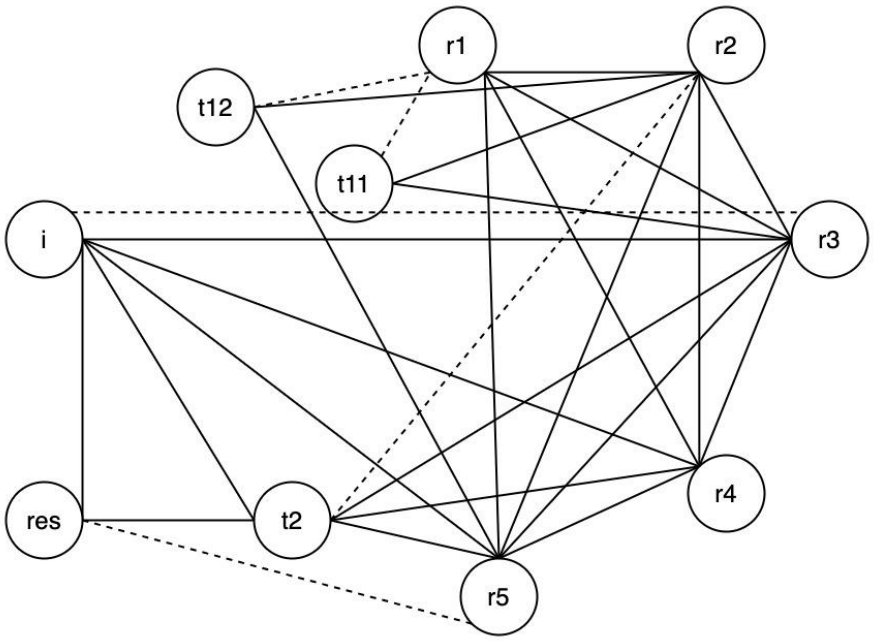
3. Simplify: i (1')
4. Coalesce: t2 - r2 (1')
5. Select: actual spill t1 (1')
6. Rewrite program (3', 2' for new instructions, 1' for interference graph)

```
Assembly language
deadloop_search:
    t11 <- r1
    M[t1loc] <- t11
    t2 <- r2
    i <- r3
    call compare
    res <- r5
    cmp res, 0
    je L2
```

```
L1:
    r3 <- i
    r3 <- r3 - 1
    j L3
L2:
    r3 <- i
    r3 <- r3 + 1
L3:
    call deadloop_search
    r2 <- t2
    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
t2 <- r2	t2	r2	r2, r3	r3, t2
i <- r3	i	r3	r3, t2	r3, t2, i
call compare	r3, r4, r5	r3	r3, t2, i	r5, t2, i
res <- r5	res	r5	r5, t2, i	t2, i, res
cmp res, 0		res	t2, i, res	t2, i
je L2			t2, i	t2, i
r3 <- i	r3	i	t2, i	r3, t2
r3 <- r3 - 1	r3	r3	r3, t2	r3, t2
j L3			r3, t2	r3, t2
r3 <- i	r3	i	t2, i	r3, t2
r3 <- r3 + 1	r3	r3	r3, t2	r3, t2
call deadloop_search	r3, r4, r5	r3	r3, t2	r5, t2
r2 <- t2	r2	t2	r5, t2	r2, r5
t12 <- M[t1loc]	t12		r2, r5	r2, r5, t12

r1 <- t12	r1	t12	r2, r5, t12	r1, r2, r5
ret			r1, r2, r5	r1, r2, r5



7. Coalesce: res - r5, t11 - r1, t12 - r1 (3', 1' for each coalescing)
8. Simplify: i (1')
9. Coalesce: t2 - r2 (1')
10. Select (2', 1' for color assignment, 1' for new instructions)

Node	Color
t11	r1
t12	r1
t2	r2
i	r1
res	r5

```
Assembly language
deadloop_search:
    M[t1loc] <- r1
    r1 <- r3
    call compare
    cmp r5, 0
    je L2
L1:
```

```
    r3 <- r1
    r3 <- r3 - 1
    j L3
L2:
    r3 <- r1
    r3 <- r3 + 1
L3:
    call deadloop_search
    r1 <- M[t1loc]
    ret
```