上 海 交 通 大 学 试 卷（A 卷）

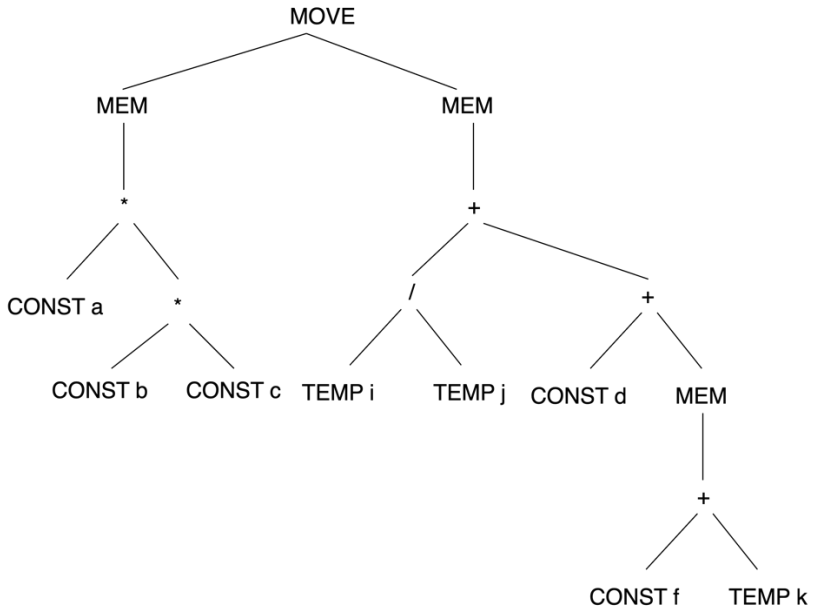
（ 2023 至 2024 学年 第 1 学期 ）

班级号_____ 学号_____ 姓名 _____

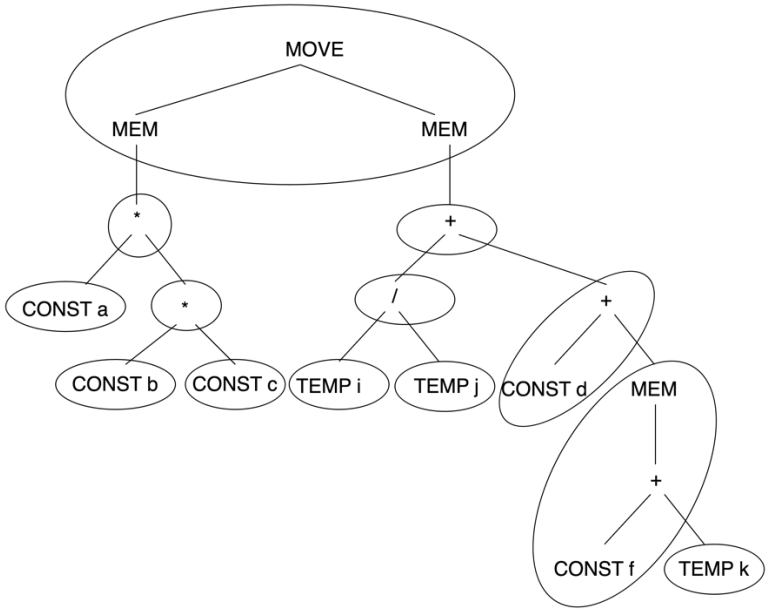
课程名称_____计算机系统基础（汇编）_____成绩 _____

Problem 1: Instruction Selection

1.(8)



2. (16)



instruction:

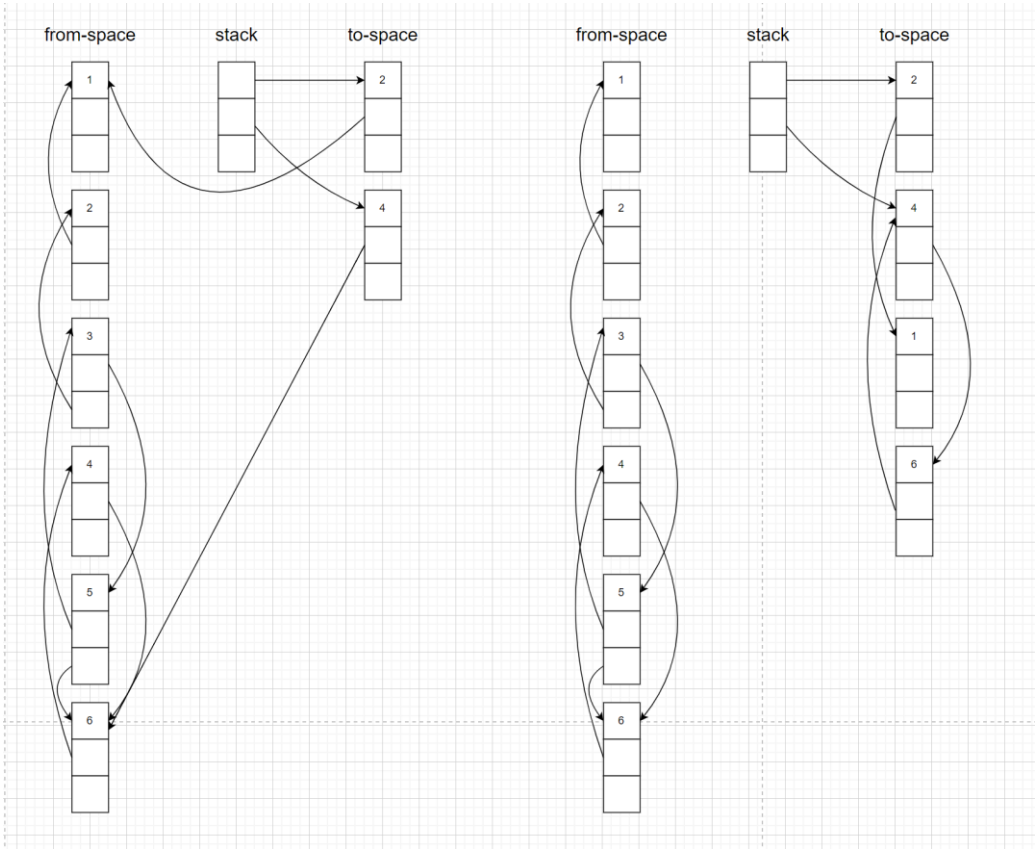
r1 ← r0 + b
r2 ← r0 + c
r3 ← r1 * r2
r4 ← r0 + a
r5 ← r4 * r3
r6 ← ri / rj
r7 ← M[rk + f]
r8 ← r7 + d
r9 ← r6 + r8
M[r5] ← M[r9]

cost:31

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. (6)



2．(6)

因为 Mutator 和 Collector 需要并发处理堆内数据结构，引入了并发开销。

延迟敏感型应用适用于 concurrent GC，因为使用 STW GC 时，与 STW GC 重叠的 request 会出现显著大的时延，导致应用响应速度的尾延迟很大。而使用 concurrent GC 尽管会使吞吐能力有所损失，但却可以让响应时延的分布相对平缓，因此更适合时延敏感型的引用。

3．(12)

Mark&Copy GC 的运行时间与堆中已用空间的尺寸成正比。所以可以用上一次的 GC 速度和当前的堆尺寸估算下次 GC 的运行时长。而 Mutator 运行到 OOM 所需的时间可以用堆的空闲尺寸和分配速度进行估算。当 GC 的预测运行时长接近 OOM 的预测时间，则应该触发 GC，避免在 GC 运行完成前就遭遇 OOM。

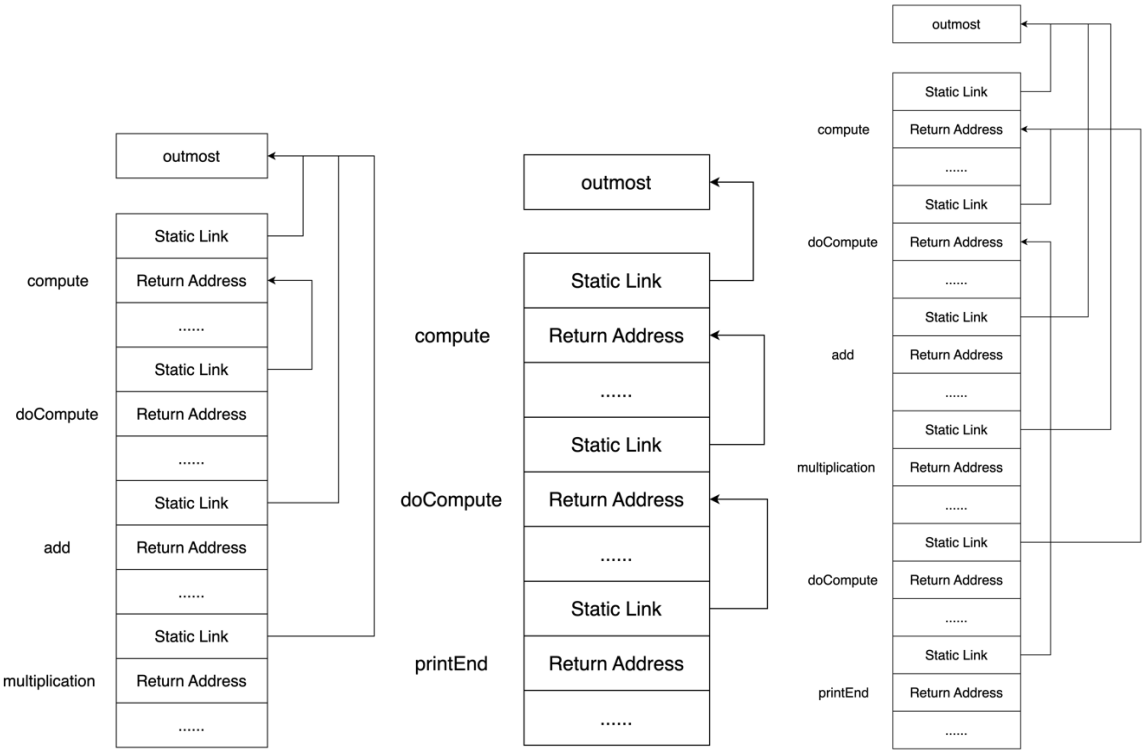
```
def malloc(size):
    heap_free_space -= size
    malloc_total += size;
    malloc_speed = malloc_total / (current_time() – last_gc_done_time);
    if(heap_free_space / malloc_speed < heap_used_space / GC_speed):
        triggerConcurrentGC();
    end if
    return doMalloc(size);
end def
```

```
def startGC():
    gc_start_time = current_time();
    heap_used_space_before_GC = heap_used_space;
    doStartGC();
end def
```

```
def doneGC():
    malloc_total = 0;
    last_gc_done_time = current_time();
    GC_speed = heap_used_space_before_GC / (current_time() - gc_start_time)
    doDoneGC();
end def
```

Problem 3: Functional Programming

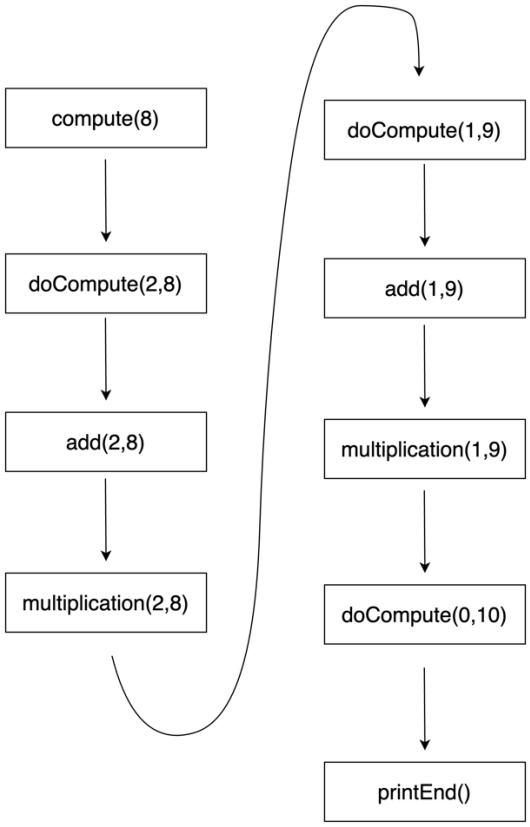
1.



2.

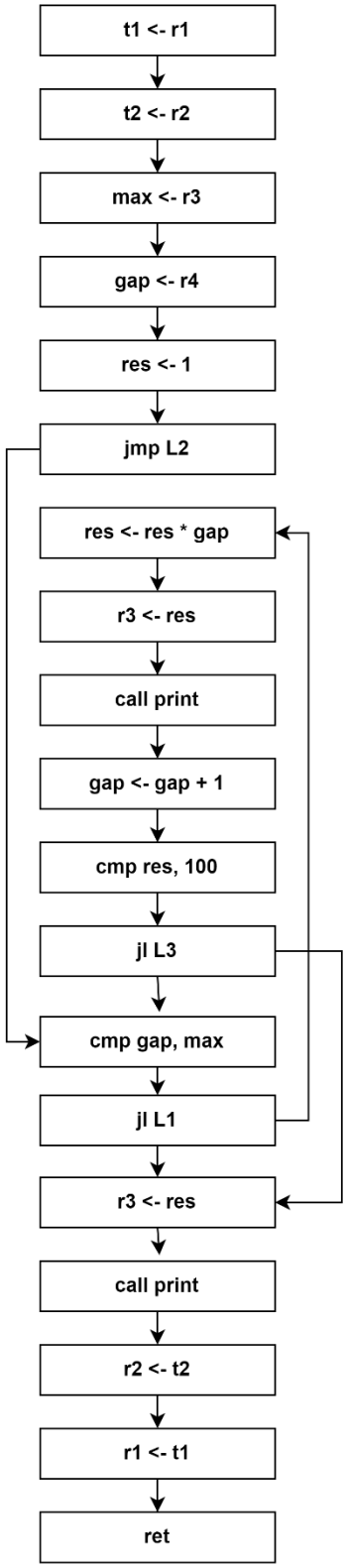
```
tmp[0] = 1
tmp[1] = 2
tmp[2] = 2
tmp[3] = 1
tmp[4] = 1
tmp[5] = 1
tmp[6] = 1
tmp[7] = 1
tmp[8] = 2
tmp[9] = 2
```

3.


compute(8)
doCompute(2,8)
add(2,8)
multiplication(2,8)
doCompute(1,9)
add(1,9)
multiplication(1,9)
doCompute(0,10)
printEnd()

Problem 4: Register Allocation

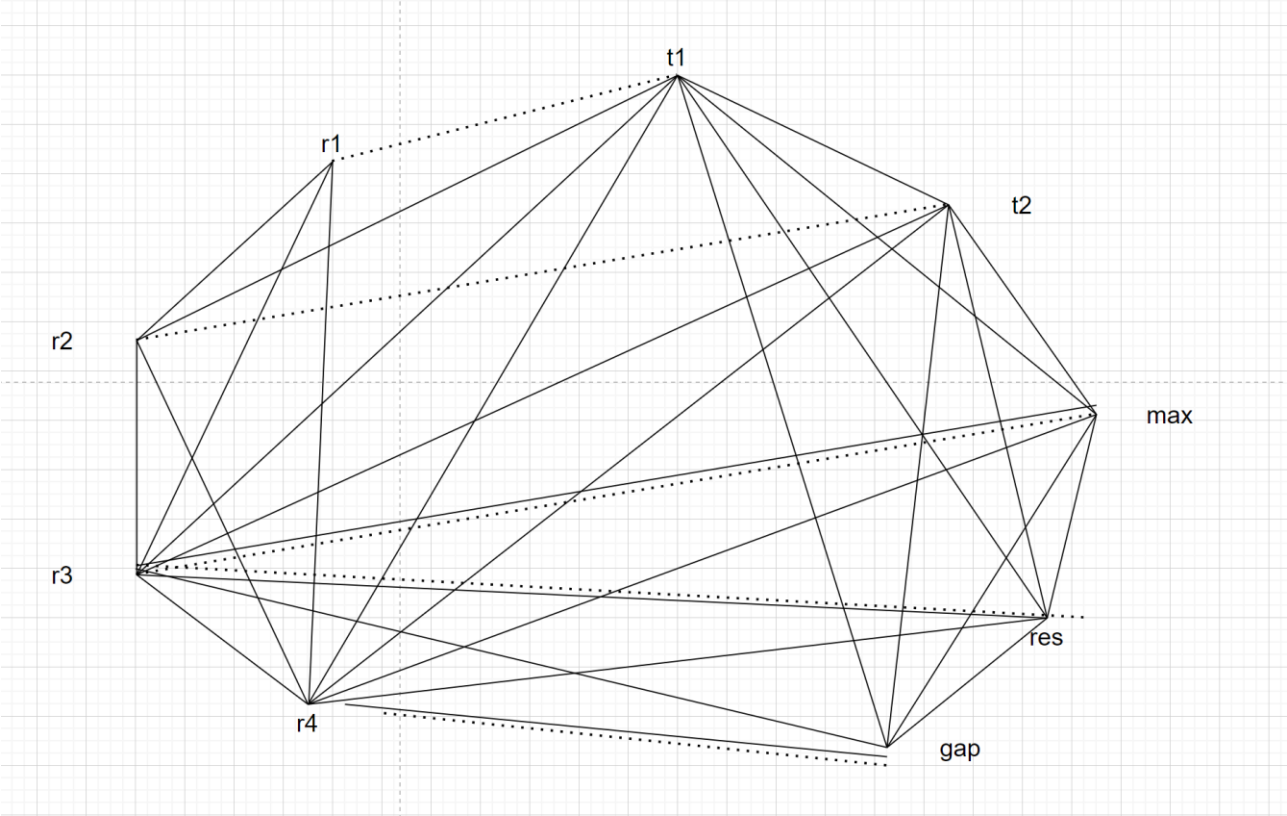
1.



2.

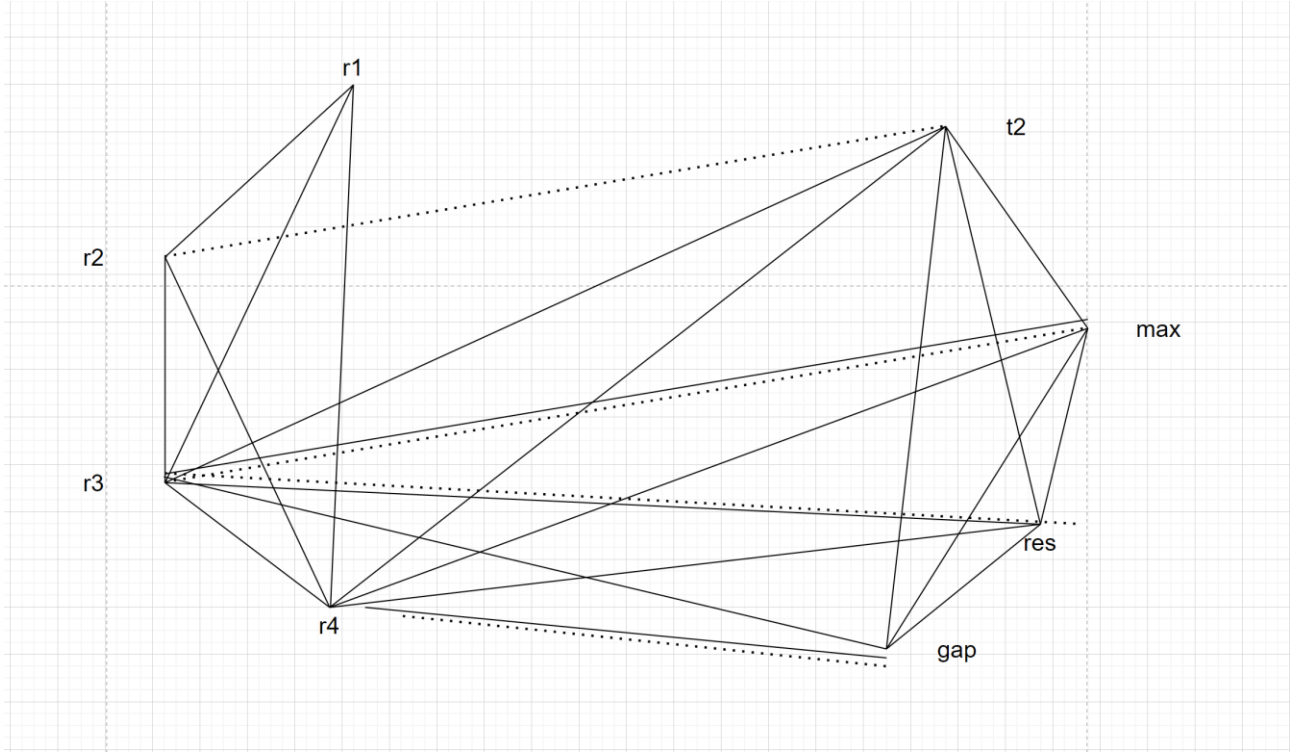
instr	def	use	in	out
t1 <- r1	t1	r1	r1, r2, r3, r4	t1, r2, r3, r4
t2 <- r2	t2	r2	t1, r2, r3, r4	t1,t2, r3, r4
max <- r3	max	r3	t1,t2, r3, r4	t1,t2, max, r4
gap <- r4	gap	r4	t1,t2, max, r4	t1,t2, max, gap
res <- 1	res		t1,t2, max, gap	t1,t2, max, gap, res
jmp L2			t1,t2, max, gap, res	t1,t2, max, gap, res
L1:				
res <- res * gap	res	res,gap	t1,t2, max, gap, res	t1,t2, max, gap, res
r3 <- res	r3	res	t1,t2, max, gap, res	t1,t2, max, gap, res, r3
call print	r3,r4	r3	t1,t2, max, gap, res, r3	t1,t2, max, gap, res
gap <- gap + 1	gap	gap	t1,t2, max, gap, res	t1,t2, max, gap, res
cmp res, 100		res	t1,t2, max, gap, res	t1,t2, max, gap, res
jl L3			t1,t2, max, gap, res	t1,t2, max, gap, res
L2:				
cmp gap, max		Gap,res2	t1,t2, max, gap, res	t1,t2, max, gap, res
jl L1			t1,t2, max, gap, res	t1,t2, max, gap, res
L3:				
r3 <- res	r3	res	t1,t2, res	t1,t2, r3
call print	r3,r4	r3	t1,t2, r3	t1,t2
r2 <- t2	r2	t2	t1,t2	t1,r2
r1 <- t1	r1	t1	t1,r2	r1,r2
ret			r1,r2	r1,r2

3.

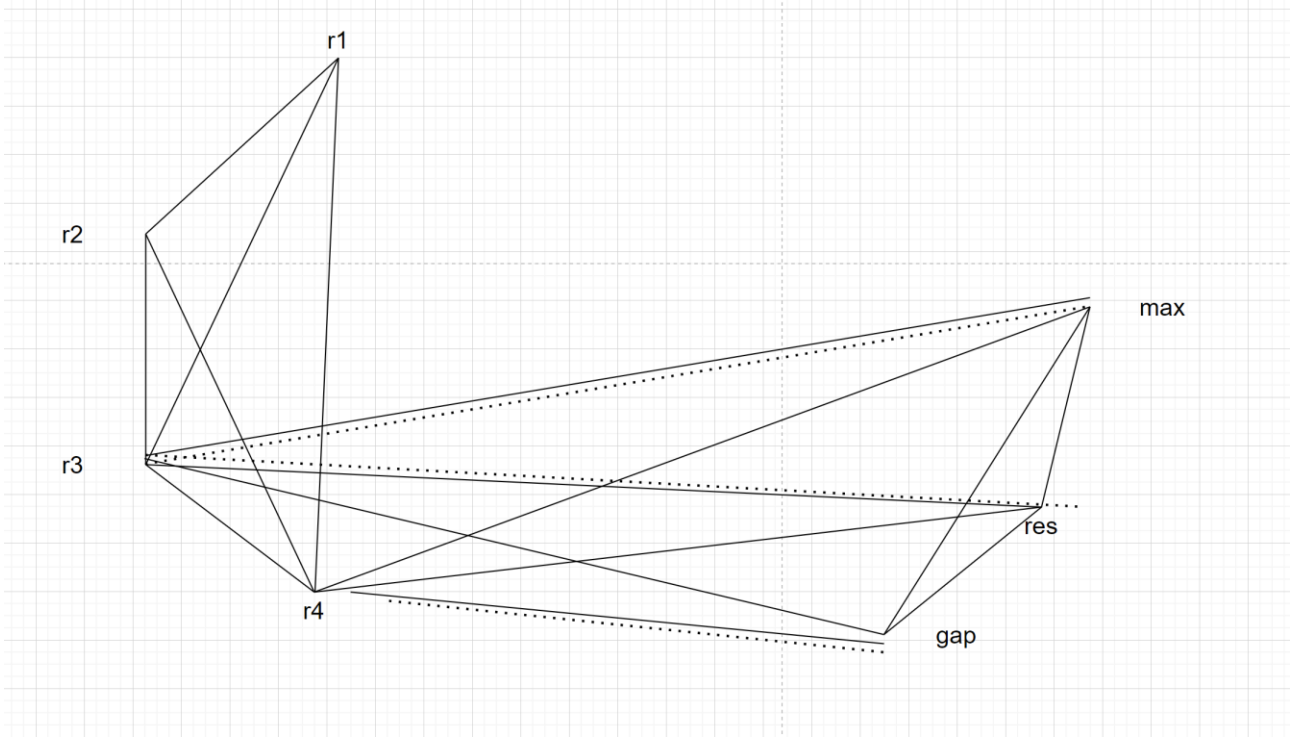


4.

1. Spil:t1



2. Spill t2

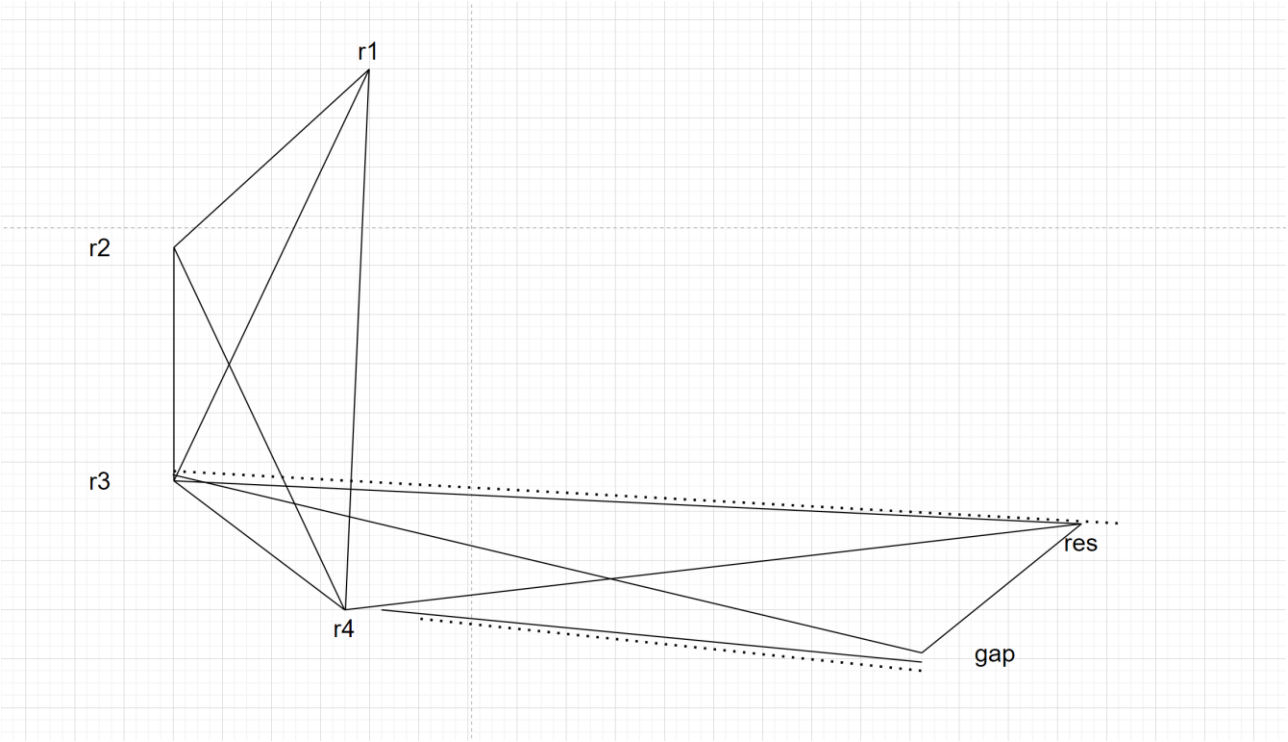


3. Spill：Max

Max=(1+10*1)/4

Res=(2+10*4)/4

Gap=(1+10*4)/4



4. **Simplify gap(r2), res(r1)**

5. **Select gap, res**

6. **Actually spill max, t2, t1**

```
multiplicative:
    t11 <- r1
    M[t1_loc] <- t11
    t21 <- r2
    M[t2_loc] <- t21
    max1 <- r3
    M[loc_max] <- max1
    gap <- r4
    res <- 1
    jmp L2
L1:
    res <- res * gap
    r3 <- res
    call print
    gap <- gap + 1
    cmp res, 100
    jl L3
L2:
    max2 <- M[loc_max]
    cmp gap, max2
    jl L1
L3:
    r3 <- res
```

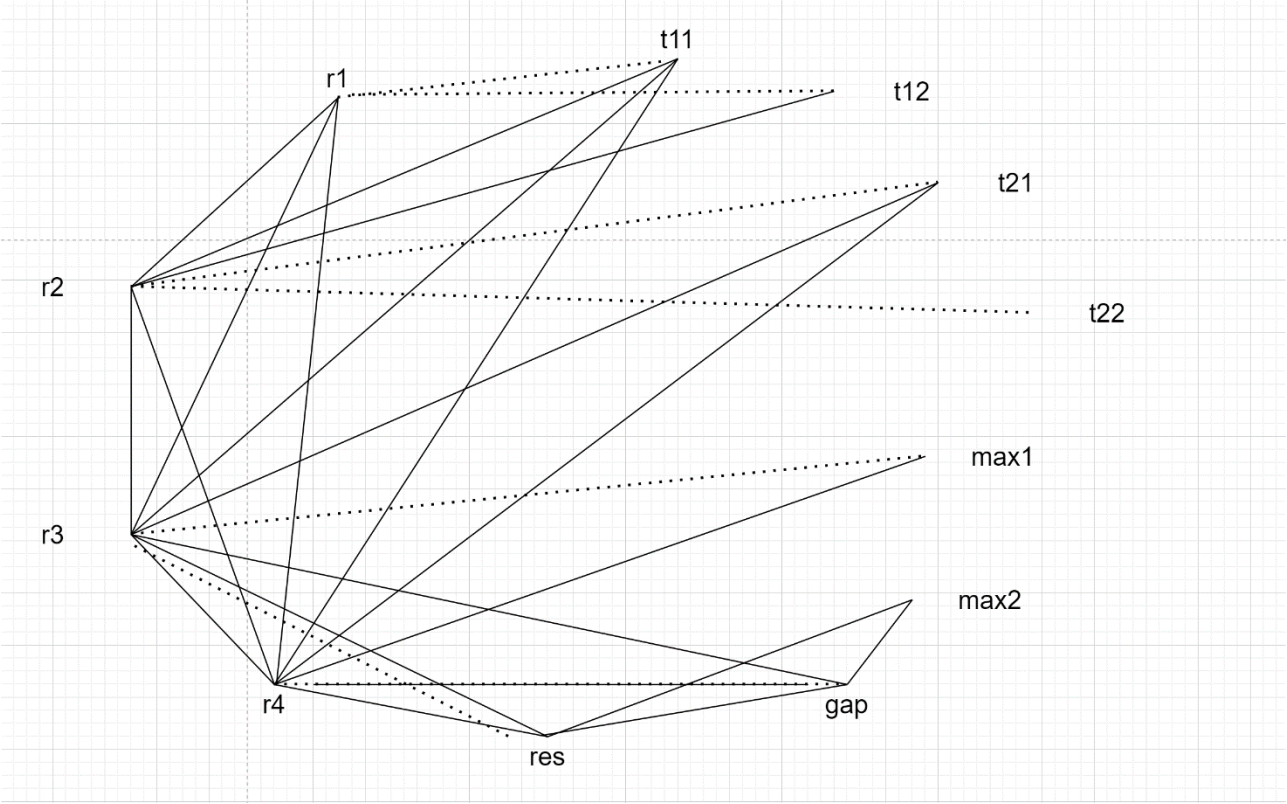
```
call print
t22 <- M[loc_t2]
r2 <- t22
t12 <- M[loc_t1]
r1 <- t12
ret
```

7. Def-use

instr	def	use	in	out
t11 <- r1	t11	r1	r1, r2, r3, r4	t11, r2, r3, r4
M[t1_loc] <- t11		t11	t11, r2, r3, r4	r2, r3, r4
t21 <- r2	t21	r2	r2, r3, r4	t21, r3, r4
M[t2_loc] <- t21		t21	r21, r3, r4	r3, r4
Max1 <- r3	max1	r3	r3, r4	max1, r4
M[max] <- max1		max1	max1, r4	r4
gap <- r4	gap	r4	r4	gap
res <- 1	res		gap	gap, res
jmp L2			gap, res	gap, res
L1:				
res <- res * gap	res	res,gap	gap, res	gap, res
r3 <- res	r3	res	gap, res	gap, res,r3
call print	r3,r4	r3	gap, res,r3	gap, res
gap <- gap + 1	gap	gap	gap, res	gap, res
cmp res, 100		res	gap, res	gap, res
jl L3			gap, res	gap, res
L2:				
max2 <- M[loc_max]	max2		gap, res	gap, res,max2
cmp gap, max2		gap,max2	gap, res,max2	gap, res
jl L1			gap, res	gap, res
L3:				
r3 <- res	r3	res	res	r3
call print	r3,r4	r3	r3	
t22 <- M[loc_t2]	t22			t22

r2 <- t22	**r2**	**t22**	**t22**	**r2**
t12 <- M[loc_t1]	**t12**		**r2**	**r2, t12**
r1 <- t12	**r1**	**t12**	**r2, t12**	**r1,r2**
ret			**r1,r2**	**r1,r2**

8. Graph



9. Coalesce t22 -> r2

10. Coalesce t12 -> r1

11. Coalesce t21-> r2

12. Coalesce t11 -> r1

13. Coalesce max1 -> r3

14. Select res -> r2

15. Select gap -> r1

16. Select max2 -> r4

17.

```
multiplicative:
    M[t1_loc] <- r1
    M[t2_loc] <- r2
    M[loc_max] <- r3
    r1 <- r4
```

```
    r2 <- 1
    jmp L2
L1:
    r2 <- r2 * r1
    r3 <- r2
    call print
    r1 <- r1 + 1
    cmp r2, 100
    jl L3
L2:
    r4 <- M[loc_max]
    cmp r1, r4
    jl L1
L3:
    r3 <- r2
    call print
    r2 <- M[loc_t2]
    r1 <- M[loc_t1]
    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).