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_____编译原理与技术_____成绩 _____

Problem 1: Instruction Selection (24 points)

1.

我承诺，我将严格遵守考试纪律。

承诺人：_____

题号									
得分									
批阅人(流水阅卷教师签名处)									

2.

Problem 2: Garbage Collection (24 points)

1.

(a)

(b)

2.

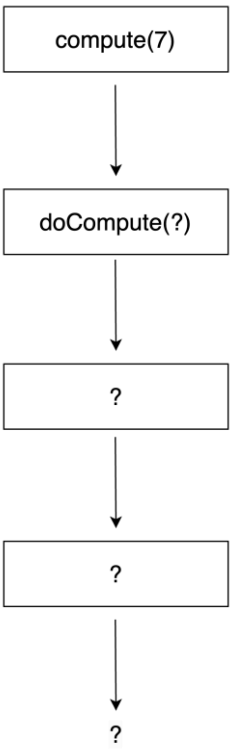
3.

Problem 3: Functional Programming (20 points)

1.

2.

3.



Problem 4: Register Allocation (32 points)

1.

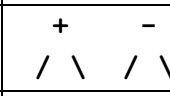
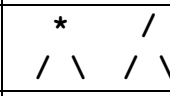
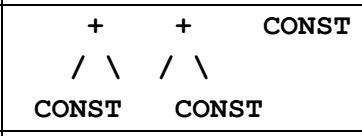
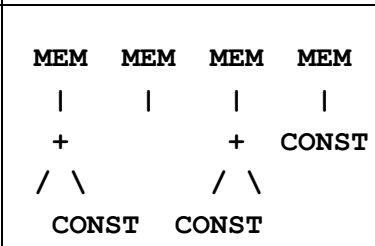
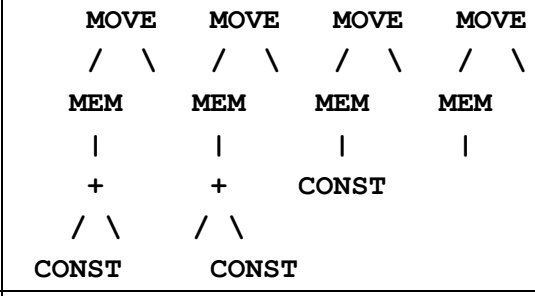
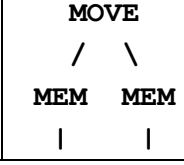
2.

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

3.

4.

Problem 1: Instruction Selection (24 points)

Name	Effect	Trees	Cost
-	ri		1
ADD SUB	ri ← rj + rk ri ← rj – rk		2
MUL DIV	ri ← rj * rk ri ← rj / rk		2
ADDI	ri ← rj + c		3
LOAD	ri ← M[rj + c]		4
STORE	M[rj + c] ← ri		4
MOVEM	M[rj] ← M[ri]		4

Note:

- The notation **M[x]** denotes the memory word at address x.
- The **third column** in the table above is the set of **tiles** corresponding to the machine instruction in the first column of the table
- The **fourth column** in the table above is the **cost** corresponding to the machine instruction in the first column of the table
- In this architecture, a register r_0 always contains zero

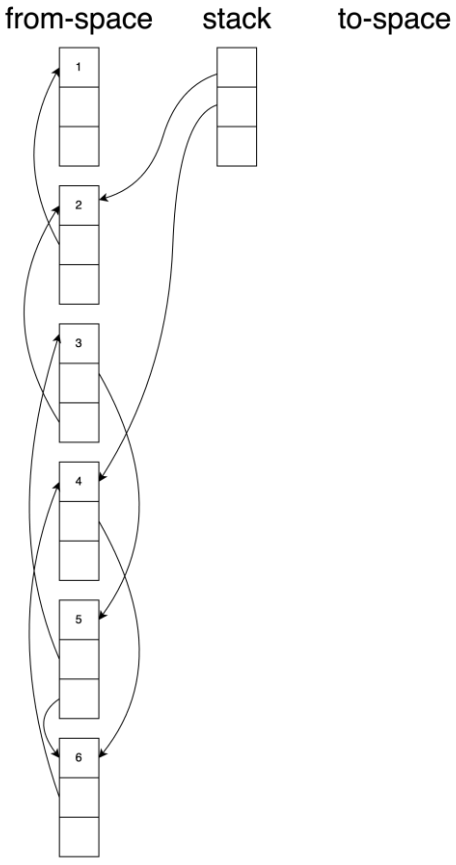
1. For the following expression, Please **draw the IR tree.** (8’)

   MOVE(MEM(*(CONSTa , *(CONSTb , CONSTc))) , MEM(+(/(TEMPi , TEMPj) , +(CONSTd , MEM(+(CONSTf , TEMPk))))))

2. We have learned a **dynamic programming algorithm** to find the **optimum** tiling for the IR tree. Please use the dynamic-programming algorithm to tile **the IR tree in question 1**. **Draw your tiles, write the corresponding instruction sequences** after tiling, and **calculate the cost** of your instruction sequences. (16’)

Problem 2: Garbage Collection (24 points)

1. The Cheney Algorithm is a fundamental design for copying garbage collection. The object reference graph provided below represents the initial state before the garbage collection process begins. Your task is to manually apply the Cheney Algorithm to this reference graph and draw the reference graphs for the following two states:
   a) After processing the GC roots. (3’)
   b) After processing the entire from-space. (3’)



2. Concurrent garbage collection is extensively used in production environments. Explain why concurrent garbage collection can negatively impact runtime performance. Additionally, discuss the types of applications that are best suited for concurrent garbage collection. (6’)

3. Stop-The-World GC will be triggered when an application runs out of memory. However, concurrent GC operates differently. Discuss the potential consequences of starting concurrent GC either too early or too late. Analyze the trade-offs involved and design a mechanism that triggers **concurrent Mark & Copy GC** based on allocation speed and GC speed. Additionally, provide a **simple** code snippet to demonstrate your approach, including how to calculate allocation speed and GC speed. (12')
   Hint: You can modify **malloc** like this. Feel free to modify any function including **startGC** or **endGC**. Global variables like **heap_free_space** and **current_time** can be seen as given.

```
def malloc(size):
    … …
    // add your code
    if(… …):
        triggerConcurrentGC();
    end if
    return doMalloc(size);
end def
```

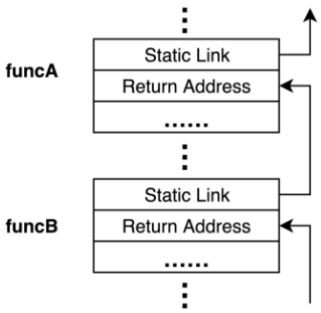
Problem 3: Functional Programming (20 points)

```
1  let
2      type intNum = array of int
3      var num := 10
4      var tmp := intNum [num] of 1
5
6      function add(n: int, m int) =
7          in
8              tmp[n] = tmp[m] + tmp[n]
9          end
10
11     function multiplication(n: int, m: int) =
12         in
13             tmp[m] = tmp[m] * tmp[n]
14         end
15
16     function compute(a: int) =
17         let
18             function doCompute(x: int, y: int) =
19                 let
20                     function printEnd() =
21                         let
22                             var endString = "program end"
23                         in
```

```
24                      print(endString)
25                   end
26              in
27                 if x >= 1 then
28                 (add(x,y)
29                 multiplication(x,y)
30                 doCompute(x-1,y+1))
31                 else
32                    printEnd()
33              end
34       in
35          if a < num / 2
36          then doCompute(a,num-a)
37          else doCompute(num-a,a)
38       end
39
40  in
41     compute(8);
42  end
```

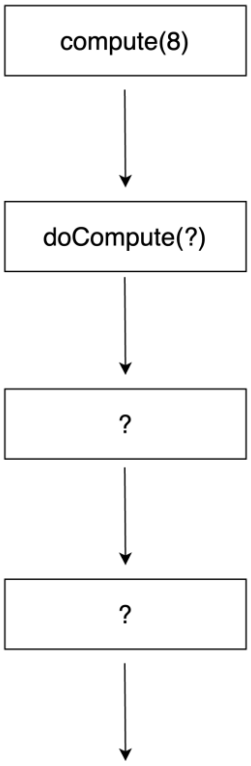
1. Suppose that we implement the nested functions using an access link. Please draw the access link in each of the activation records on the stack based on the following function invocations (9').
   1) compute→ doCompute→ add →multiplication
   2) compute→ doCompute→ printEnd
   3) compute→ doCompute→ add→ multiplication→ doCompute→ printEnd

e.g.



2. What's the actual value of **tmp after line 41**? Please write out **each value of the tmp array** (5')

3. Please complete the following control flow graph for **all the function calls** invoked within **line 41**. You should write the current function name with **the actual value of its arguments**. (6’)



Problem 4: Register Allocation (32 points)

Suppose a compiler has compiled the following function **multiplicative** into instructions on the right:

```
void multiplicative (int max, int gap)
{
    int res = 1;
    while (gap < max) {
        res = res * gap;
        print(res);
        gap = gap + 1;
        if (res > 100)
            break;
    }
    print(res)
}
```

```
multiplicative:
    t1 <- r1
    t2 <- r2
    max <- r3
    gap <- r4
    res <- 1
    jmp L2
L1:
    res <- res * gap
    r3 <- res
    call print
    gap <- gap + 1
    cmp res, 100
    jl L3
L2:
    cmp gap, max
    jl L1
L3:
    r3 <- res
    call print
    r2 <- t2
    r1 <- t1
    ret
```

Several things are worth noting in the instructions:

- There are **four** hardware registers in all.
- The compiler passes parameters using registers. (The first parameter goes to **r3,** The second parameter goes to **r4**)
- **r1,r2** is **callee-saved** while **r3,r4** are **caller-saved**.
- The architecture handles integer divisions by fetching the two operands from **arbitrary** registers. After calculation, it stores the **quotient** and **remainder** to **r3** and **r4** respectively.
- **t1, t2, max, res, gap, i** are temporary registers.

Please answer the following questions according to the instructions.

1. Draw a control flow graph instruction-by-instruction. (4’)

2. Fill up the following def/use/in/out chart. (8’)

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

3. Draw the interference graph for the program. Please use dashed lines for move edges and solid line for real interference edges. (4’)

4. The heuristic to decide which temporary to spill is the same as that in the Tiger book. i.e. The spill priority is calculated as:

```
Spill Priority =
(Uses+Defs-outside-loop + Uses+Defs-within-loop * 10) / Degree
```

Adopt the graph-coloring algorithm to allocate registers for temporaries. Write down the instructions after register allocation. (16’)

NOTE: You must write down **every decision you make** such as simplifying, spilling, and coalescing. And you also need to write down **new instructions** and draw **a new interference graph after each spilling**. If your answer is too simple, you will only get a part of score.