(Page Left Blank Intentionally)

CSE 401 – Section 10 – Instruction Scheduling and Register Allocation

(Adapted from 19 AU Final)

1. **Instruction Scheduling**

Consider a hypothetical machine that has the following instructions and number of cycles for each instruction to complete:

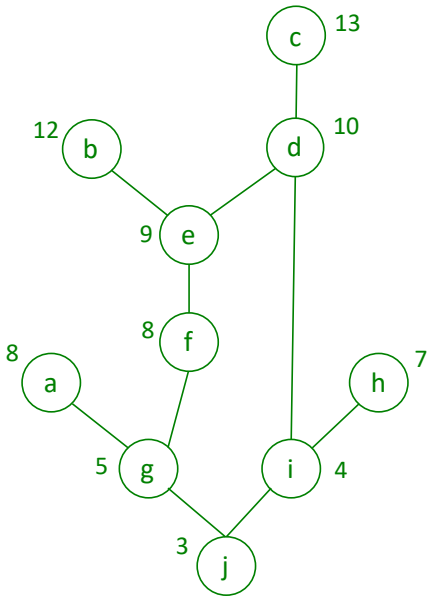
Operation	Cycles
LOAD	3
STORE	3
ADD	1
MULT	2
SHIFT	1

Our compiler's instruction selection algorithm generated the following series of machine instructions for the statement y[i] = a*x[i]. Currently, every value is stored in a temporary register, so we'll need to select actual registers in a later step.

```
a. LOAD t1 <- a          // t1 = a
b. LOAD t2 <- x          // t2 = address of x[] array
c. LOAD t3 <- i          // t3 = i
d. SHIFT t4 <- t3, 3     // t4 = i*8 (shift t3 left 3)
e. ADD t5 <- t2, t4      // t5 = address of x[i]
f. LOAD t6 <- MEM[t5]    // t6 = x[i]
g. MULT t7 <- t1, t6     // t7 = a*x[i]
h. LOAD t8 <- y          // t8 = address of y[] array
i. ADD t9 <- t4, t8      // t9 = address of y[i]
j. STORE MEM[t9] <- t7   // store y[i]
```

Part a.

Draw the precedence graph for the instructions above. The graph should show all the dependencies between different instructions. Every node in the graph should be labeled with the letter of its corresponding instruction (a-j) and annotated with the latency. Latency is defined as the minimum number of machine cycles between the start of the instruction and the end of the last instruction. (Hint: Work from the last instruction up to the first instruction)



Part b.

Using the precedence graph you drew for Part a, present the order of the instructions as they should be chosen by the forward list scheduling algorithm. For each cycle, write the scheduled instruction and its type. You do not need to write the operand(s) for instructions.

Cycle #	Instruction	Operation
1	c.	LOAD i
2	b.	LOAD x
3	a.	LOAD a
4	d.	SHIFT
5	e.	ADD
6	f.	LOAD x[i]
7	h.	LOAD y
8	--	
9	g.	MULT
10	i.	ADD
11	j.	STORE y[i]

2. **Register allocation & graph coloring**

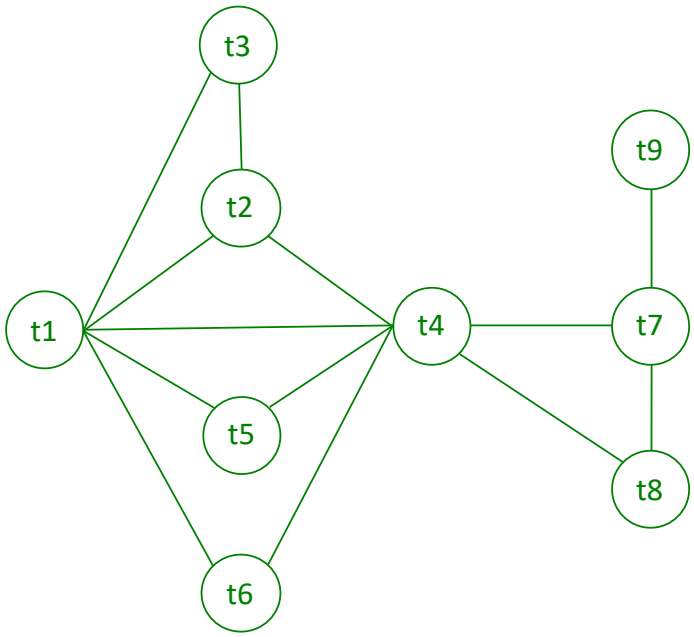
Consider the same scenario from problem 1. You should assume the code is executed in the sequence given and not rearranged before assigning registers.

a. LOAD t1 <- a
b. LOAD t2 <- x
c. LOAD t3 <- i
d. SHIFT t4 <- t3, 3
e. ADD t5 <- t2, t4
f. LOAD t6 <- MEM[t5]
g. MULT t7 <- t1, t6
h. LOAD t8 <- y
i. ADD t9 <- t4, t8
j. STORE MEM[t9] <- t7

Operation	Cycles
LOAD	3
STORE	3
ADD	1
MULT	2
SHIFT	1

Part a.

Draw the interference graph for the temporary variables (t1-t9) in the code.



Part b.

Give an assignment of groups of temporary variables to registers that uses the minimum number of registers possible based on the information in the interference graph. Use R1, R2, R3, … for the register names.

Three registers are needed. Here is one of several possibilities:

R1: t1, t7
R2: t2, t5, t6, t8, t9
R3: t3, t4