- mk load (num, num. val).

7b) DAG AND THREE ADDRESS CODE:

Three address code:

It should have almost one operation in addition with the assignmen operation.

Temparary names can be used for storing the intermediate results.

It may have fewer than three operands.

Eg :-

```
prod = 0
  i = 1
while ( i <= 20)
{
  prod = prod + a[i] * b[i]
  i = i+1
}
```

Three address code :-

```
t1 := 4 * i
t2 := a[t1]
t3 := 4 * i
t4 := b[t3]
t5 := t2 * t4
t6 := prod + t5
prod := t6
t7 := i+1
i := t7
if i <= 20 goto (1)
```

DAG (Directed Acyclic Graph) :-

A DAG for an expression identifies the common subexpressions in the expression.

Algorithm:-

Input: A basic block

Output: DAG for basic block
Containing the foll. information

1) A label for each node. for leaves, the label is an identifier. For interior nodes, an operator symbol.

2) For each node a list of attached identifiers to hold the computed values.

Case (i) $x := y \ op \ z$

Case (ii) $x := op \ y$

Case (iii) $x := y$.

DAG for the code:

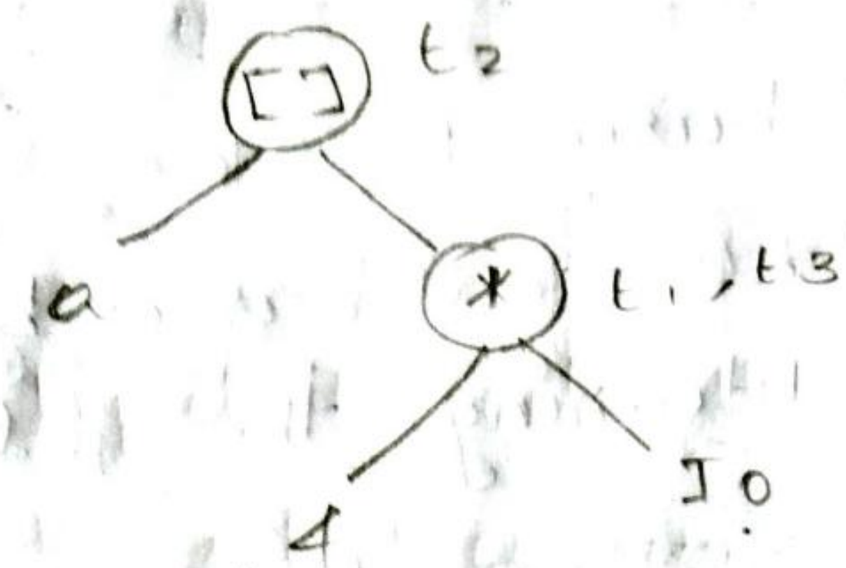
a)

Statement (1)

b)

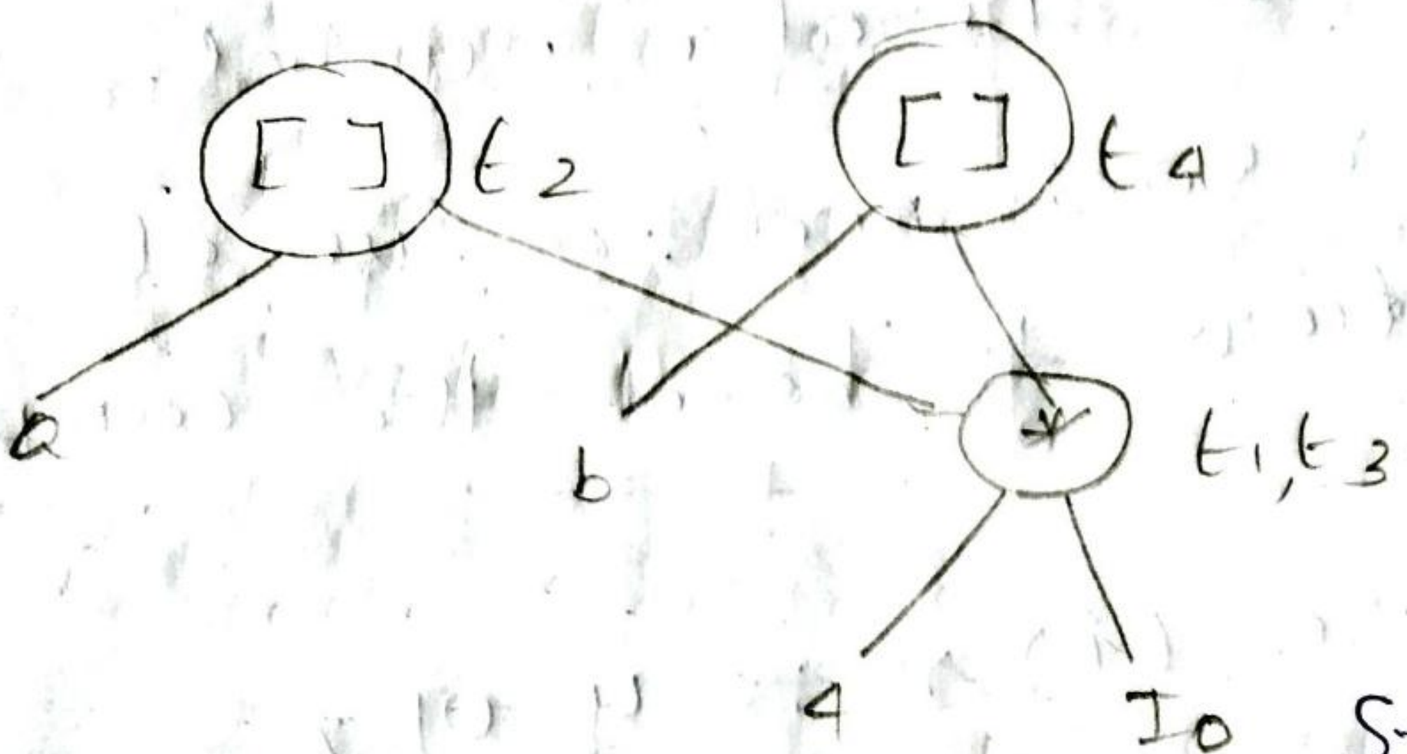
Statement (2)

(c)



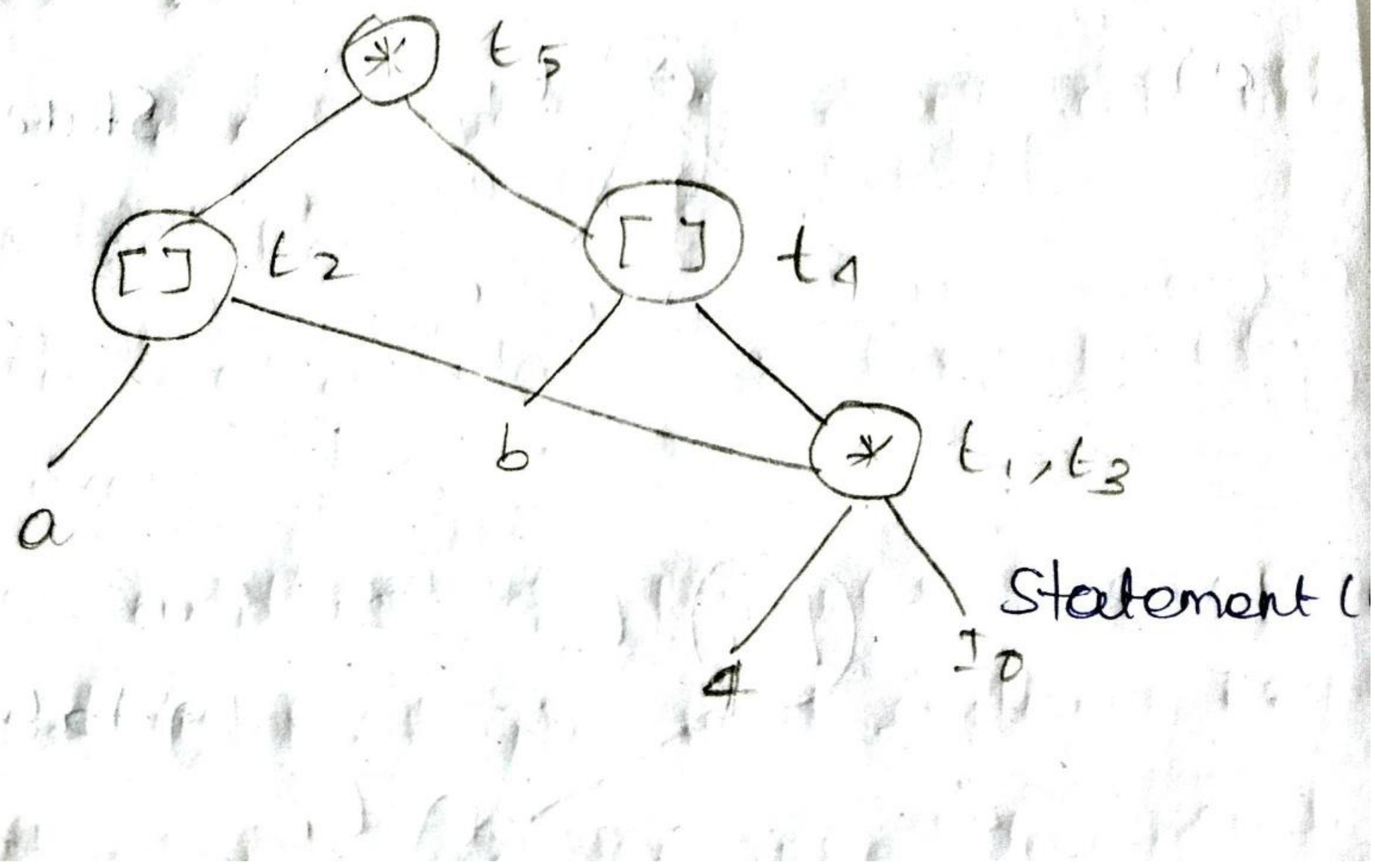
Statement (3)

(d)



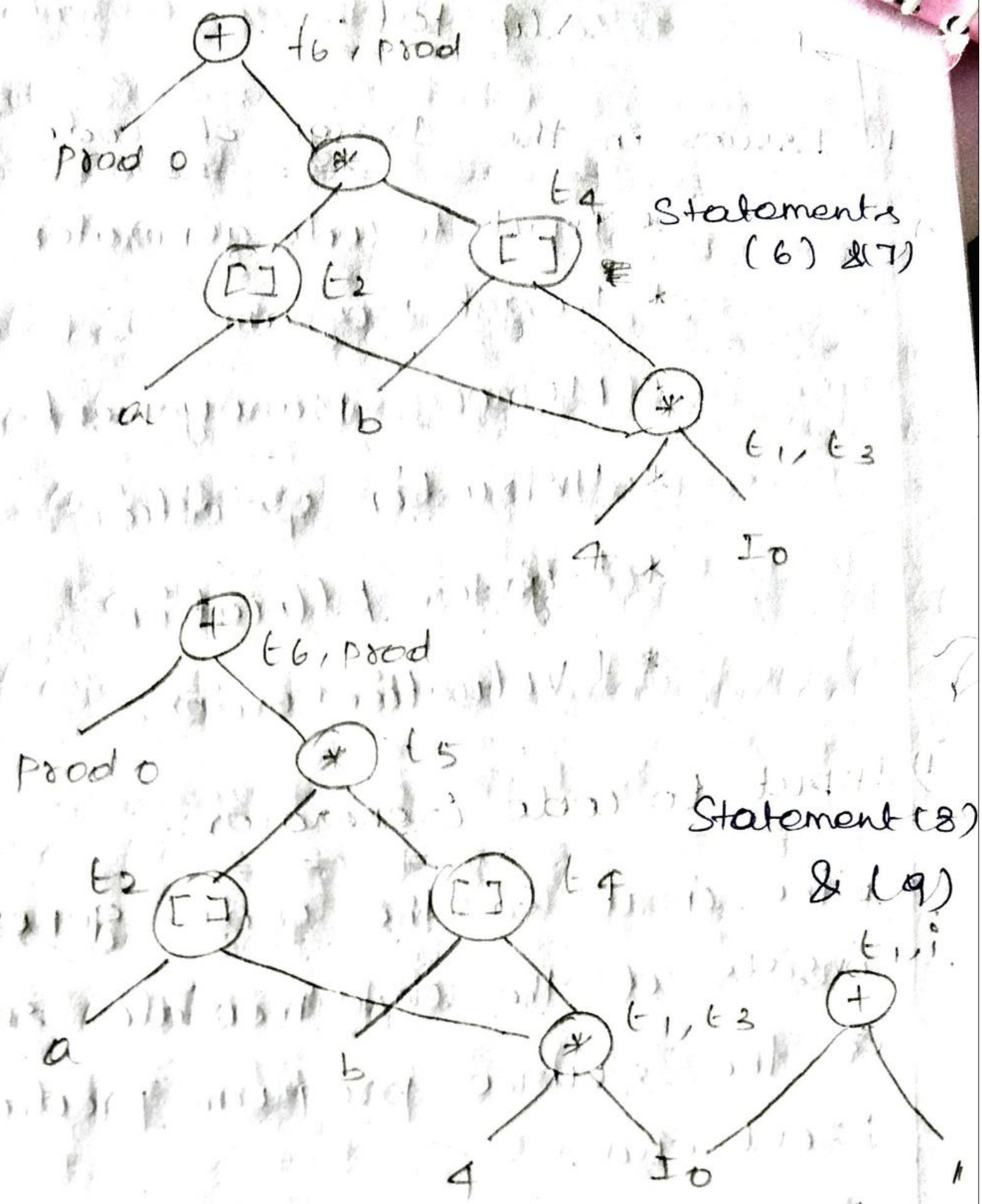
Statement (4)

(e)

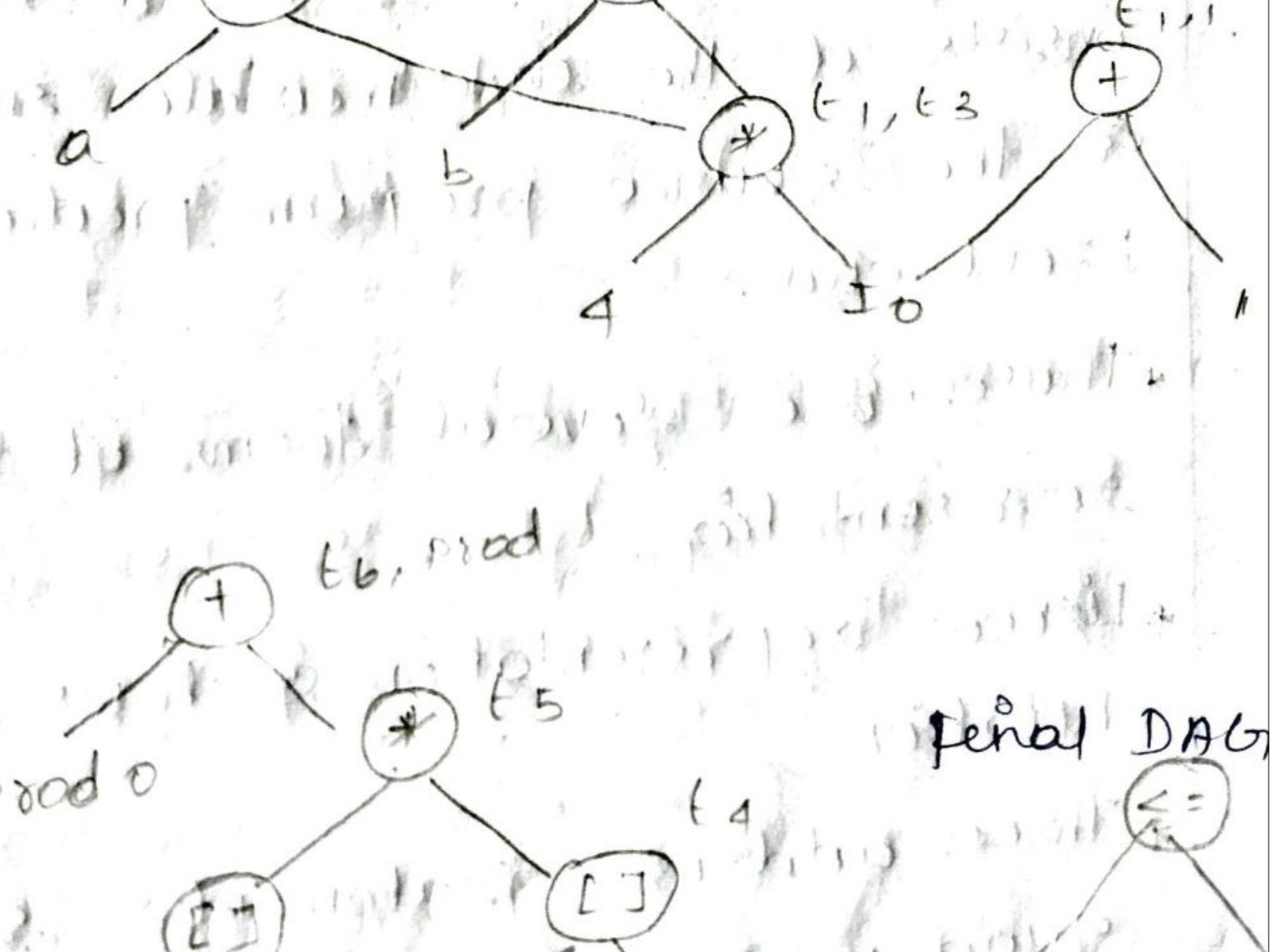


Statement (

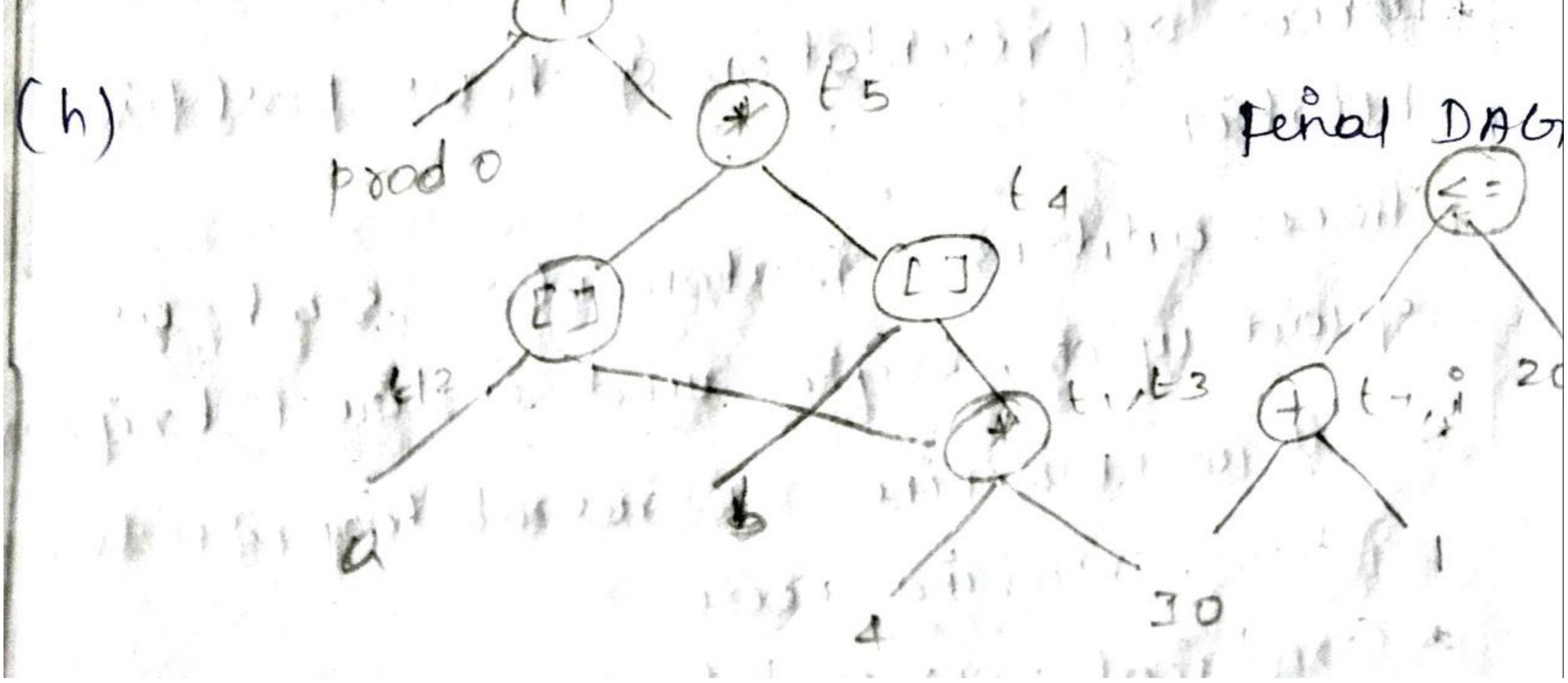
(f)



(g)



(h)



qxp test

1. Issues in the Design of code Generator

* Input to code generator
* Target program
* Memory Management
* Instruction selection
* Register Allocation
* Evaluation order.

i) Input to code Generator:-

* The input to the code generator consists of the intermediate representation of the source program produced by the Front end.
* There are several forms of intermediate representation.
* Linear representation such as postfix notation
* Three address statements such as quadruples, triples and indirect triples
* Virtual machine representation such as stack machine code.
* Graphical representation such as syntax trees and dags

ii) Target program:-

* The output of the code generator is the target program
* The target program may take on a variety of forms.
  * Absolute machine language.
  * Relocatable machine Language
  * Assembly language

* Producing an absolute machine language Program as output has the advantage that it can be placed in a fixed location in memory and immediately executed.

* Producing a relocatable machine language program as output allows subprograms to be compiled separately.
* Producing an assembly language Program as output makes the process of code generation easier.

iii) Memory management:-

* Mapping names in source program to addresses of data objects in runtime memory is done by the front end of

a compiler and the code generator.

* A name in a three address statement refers to a symbol-table entry for the name.

* Static allocation and stack allocation are used to convert the intermediate representation into addresses in the target code.

iv) Instruction selection :-

* The factors do instruction selection are

* uniformity
* completeness
* machine idioms
* instruction speeds.

* Every three address statement of the form $x = y + z$ can be translated into the code sequence.

```
MOV  y  R0
ADD  z  R0
MOV  R0 x.
```

* Unfortunately, this code generation after poor code.

d = a + c

would be translated into

```
Mov b R0
ADD c R0
Mov R0 a
```

Fig 2 : INC a

can be written shorter than

```
Mov a R0
ADD #1 R0
```

or

```
INC R0 a
```

v) Register Allocation

* Instructions involving register operands are shorter and faster than those involving operands in memory.
* Therefore efficient utilization of register is important in generating good code.
* The use of register is divided into

* During register allocation
  → set of variables that will reside in register are selected
* During register assignment
  → the specific register that a variable will reside in is picked
* The multiplication instruction of the form.

MUL x, y.

Where x → multiplicand is the even register of an odd / even register pair
y → multiplier is a single register

* The division instruction of the form

DW x y.

- 64 bit dividend occupies an odd/even register pair, whose even register is x y is a divisor.
- After division even register holds the remainder and odd register holds the quotient.

* The target computer for producing The target code is a byte addressable machine with four bytes to a word and n- general purpose registers

$R_0, R_1, \ldots, R_{N-1}$

It has two address instructions of the form.

op source destination

the address modes together with their assembly language forms and associated costs are as follows.

Mode	form	Address	Added cost
Absolute	M	M	1
Register	R	R	0
indexed	c(R)	c + contents (R)	1
indirect reg	*R	contents(R)	0
indirect indexed	*c(R)	contents(c + contents(R))	1
literal	#c	c	1

Instruction cost:

Instruction cost = 1 + cost associated with source and destination

Eg:

Mov b R0 : cost = 1 + 1 + 0 = 2

ADD c R0 : cost = 1 + 1 + 0 = 2

Mov R0 a : cost = 1 + 0 + 1 = 2

cost = 6