**Intermediate Code, Runtime, and Code Optimisation**

1. Indirect triples offer two benefits compared to quadruples and triples. Describe these benefits and explain why they occur.

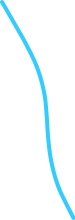
**Answer:**

1. **Moving statements require reordering of list of statements, but no change to the triples**
2. **Save space if same temporary is reused**
3. Briefly explain what the following code optimisation techniques do, and for each technique write down the benefits and the potential risks that compilers should look out for.

• Inlining

• Loop unrolling

**Answer:**



**Inlining:copy the function body to the call site and instantiate with parameters**

**Benefits:Avoid function call overhead**

**Potential risks:**

**1.Code will grow very quick if the inline functions is large.**

**2.Be careful with recursion.**

**Loop unrolling: repeat the loop body if the number of iteration is known.**

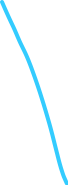
**Benefits:Less jumps, less predicate check**

**Potential risks:**

**1.Code grows significantly in size.**

**2.Code may outgrown the instruction cache, the unrolled version has to be read from memory.**

1. Each live activation has an activation record on the control stack. Name and describe each of the parts of an activation record.



**Answer:**

**Results/Return values: Results of procedure**

**Arguments/Parameters: Arguments of procedure**

**Control link Link to previous AR**

**Return address/ Saved machine status: Address jump to after execution**

**Temporaries: Temporary values used in evaluation**

**Local data: Local variables**

**Access link: Link to additional data needed by the procedure**

1. Dynamic allocation of objects on a heap within memory is widely used in programming languages. Compare and contrast the use of fixed addresses and explicit deallocation versus garbage collection as techniques for heap memory management, giving the advantages and disadvantages of each.

**Answer:**

**Fixed address and explicit deallocation**

**Pros:High efficiency**

**Cons:Memory leek, dangling reference, double free bugs**

**Garbage collection**

**Pros:Relieve the programmer from manually dealing with memory deallocation.Therefore reducing the associated issue.**

**Cons:Consuming additional resources, performance impact, incompatible with manual recourse management.**

5.Write the Intermediate Representation translation scheme for the FOR statement, which is defined as below. Assume that all three expressions (exprl, expr2, expr3) that form the loop header always appear in IR.

for\_stmt .- FOR(exprl; expr2; expr3) block

**Answer:**



6. The following is a Three Address Code (TAC) Intermediate Representation for a register machine: it contains a function called foo, which takes an integer parameter n. Identify and describe what the code does in plain English, and write down the high-level language counterpart in Java-like pseudocode.

foo:

t1=n EQ 1

CJUMP t1 label1

t2=n-1

PARAM t2

t3=CALL foo

a t4=n MULT t3

return t4

label1:



return 1

**Answer:**



7.Provide two benefits and disadvantages of the intermediate Representation for stack machines, compared to that for register machines, with a brief description of each benefit.

**Answer:**

**Benefits:**

**1.Generate more compact code(Frequently the instructions only consist of the opcode)**

**2.Yields simpler compiler, code generation for each instruction is independent from prior/subsequent code**

**3.Yields simpler interpreter, as memory access is centralised and instructions have fewer variations**

**Disadvantages:**

1. **Make more memory address, as variable needs to be pushed back to memory(No temporary register to hold them)**
2. **Common subexpression is hard to factor out**
3. **Instruction order is tightly coupled with how temporary values are stored in the stack -hard to move code, less flexible to optimise**

8. Translate the arithmetic expression d = a [i] - b \* c into the following intermediate representation.

• Abstract Syntax Tree

• Quadruples

• Triples

**Answer:**

**1.**



**2.**



9. Translate the following fragment of code into TAC IR for register machine. Optimise it as much as possible during the translation, and provide a list of all optimisations that have been applied.

i=0

b = 10

while(i < 10) {

if(b > i){

a += b + 2 \* i

}

else{

a \*= i + b

}

i++

}

**Answer:**

10.Briefly explain what the following code optimisation techniques do, and for each technique write down at least one potential risk that compilers should look out for.

• Constant Folding

• Common Subexpression Elimination

**Answer:**

**Constant folding:Evaluates expressions at compile time whenever appropriate, thus decreases number of instructions.**

**Potential risk:**

**Be careful about semantics, especially respect to precisions.**

**Common subexpression elimination:Avoid evaluating the same expressions for more than once.**

**Potential risk:**

**Value of the same expression may change across the whole program.**

11.Write the Intermediate Representation translation scheme for the ternary conditional assignment Exprl = Expr2 ? Expr3 : Expr4. The ternary conditional expression A ? B : C first evaluates the Boolean expression A - the expression evaluates to B if it is true, C if false. For example, after the declaration int a = (l == 0) ? 3 : 5;, variable a is assigned with 5. Use T[] to mark further recursive translation.

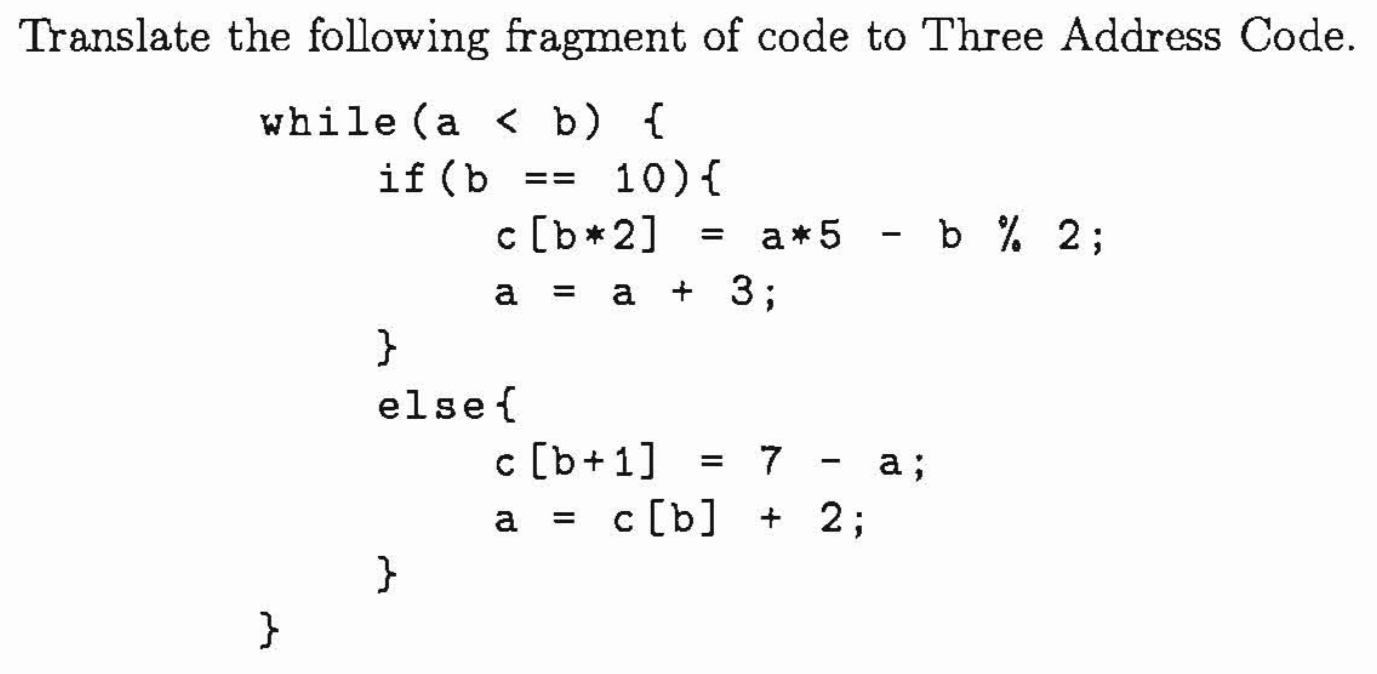
**Answer:**



12.Write down the data structure in which local variables of functions are stored during runtime, and justify the choice of the data structure. Also, describe which address the Frame Pointer (FP) register

points to, and why

**Answer: 1.Activation record. 2.? 3.?**

13.



**Answer:**



14. The following is a Three Address Code (TAC) Intermediate Representation for a register machine: it contains a function called foo, which takes an integer parameter n. Identify and describe what the code does in plain English, and write down the high-level language counterpart in Java-like pseudocode.

foo:

t1 =n EQ 1

C JUMP t1 label1

t2 = n - 1

PARAM t 2

t3 =CALL foo

t4 =n MULT t3

return t4

label1:

return 1

}

**Answer:**



15. Translate the following Java code into TAC IR for register machine. Optimise it as much as possible during the translation, and provide a list of all optimisations that have been applied. To simplify the translation, you can make the following assumptions:

• Java method Math. sin (double x) can be called with the name \_sin.

• Java method Math. cos (double x) can be called with the name \_cos.

• Java static field Math. PI can be accessed with IR variable name PI.

public double bar(int n, double x){



double a = 0;



double b = 0;



int i =n;



while(i > 0){



a += Math.sin(2 \* Math.PI + 4 \* i);



b = Math.cos(2 \* Math.PI + x);



i -= 1;



}

return a + b;

}



**Answer:**

