INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR

Date 27.11.2006 FN / AN *Time:* 3 Hrs. *Full Marks* **45** *No. of Students:* **56**
Autumn Semester:, 2006 Department: Computer Science and Engineering Sub. No: CS 43001
3^{rd} **Yr. B. Tech.(Hons.)** Sub. Name: Compiler Construction

Instructions : Answer **all** the questions.

1. (a) Generate 3-address code for the following code x = a * b + (c * d + a)
   (b) Explain Reduce-reduce conflict with a small example.
   (c) State 3 tasks of a DAG in code optimization.
   (d) Optimize the following loop : for i = 0 to 99 do
   k = 99 -i
   if a[k] = x then break
   ...
   end loop
   (e) Form a DAG for the following piece of code.
   t_1 = 4 * i
   t_2 = x[t_1]
   (f) Optimize the following piece of code.
   MOV R0,a
   MOV b,R0
   ADD a,R0
   MOV R0,b
   (g) Draw a schematic diagram for *if-then-else* to show the labels as well as the exit points of E.true and E.false.
   (h) Represent the procedure P(x_1, x_2, $\cdots$, x_n) as three address code.
   (i) State the main advantage of static type checking over dynamic type checking.
   (j) Write the full form of the following grammar abbreviations (1).LALR, (2).SLR, (3).LR, (4).LL.
   (k) Define LR(k) items.
   (l) What is lookahead.
   (m) Explain Left-factoring.
   (n) State one difference between macro and inline function.
   (o) State the main difference between Directed Acyclic Graph and Syntax Tree.
   (p) What are the different phases of a compiler?
   (q) Given the following metacharacter "a + b", what does it mean (matches to) in lex.
   (r) State the four types of common error occurring in programs. $\frac{1}{2} \times 18$

2. (a) Eliminate left recursion (both immediate and general) from the following grammar
   S → Aa | b
   A → Ac | Sd | ϵ

(b) Write the **semantic rules** to be specified in the following infix expression grammar such that it generates corresponding postfix equivalents in the process of parsing the infix expression. The rules are
E $\rightarrow$ E + T | E - T | T,
T $\rightarrow$ 0 | 1 | 2 | $\cdots$ | 9.
Also show the **steps of execution** through an **annotated parse tree** by considering the expression "3 + 2 - 4".

(c) Consider the Boolean expression x > y or z > k and not r < s. This expression can be parsed by using the following grammar B $\rightarrow$ B or M B | B and M B | not B | id relop id | true | false and M $\rightarrow$ ϵ. Here the non-terminal M helps as a marker for backpatching. Build up the **parse tree**, generate the **intermediate code** and output the final **B.truelist** and **B.falselist**. You need not write the semantic actions for the individual rules of the grammar. (2 + 3 + 5)

3. (a) Generate intermediate code for the statement X[i] := Y + Z. (2 + 6)

(b) The above code can be parsed using the following grammar
S $\rightarrow$ L := E
E $\rightarrow$ E + E | L
L $\rightarrow$ id[E] | id
Write **semantic actions** for each of the five rules so that the above mentioned intermediate code can be generated. Expand the **parse tree** and show the **steps of generation** of the code.

4. (a) Write down the **intermediate code** for the following code fragment, and show the **basic blocks** and the **control flow** within them.

```
begin
   sum = 0
   term = 1
   while term ≤ 100 do
       sum = sum + term
       term = term + 1
   end
end.
```

(b) Given the code d = (a - b) + (a - c) + (a - c), generate the intermediate code. From the intermediate code generate the target code using a *straight forward code generation* algorithm. Show all the steps of generation in form of a table -

Statements	Code Generated	Reg Descriptor	Addr Descriptor	Comments

(3 + 3)

5. (a) The grammar S $\rightarrow$ *if* S *else* S | *if* S | *a* is ambiguous. Generate a LR parsing table which removes the ambiguity by attaching the else with the nearest if.

(b) Given the grammar
S $\rightarrow$ CC
C $\rightarrow$ cC | d form its **LR(1) parsing table**. In the process show all the **items** and the **goto table** (item no, terminal/nonterminal, item no reached) produced by the grammar. Also draw the **LALR table** from the derived LR(1) parsing table. (5 + 7)