# B.Tech. Third Year End-Semester Examination

Course Name: Compilers (Code: CS346)

Full Marks: 100 Time: 3 hours

1) (a) Choose the correct answer.

A. Which of the following code is not three-address code?

a) a = -c
b) a = b + c
c) a = b[i]
d) None of these

B. Reduction in strength means

a) Replacing run time computation by compile time computation
b) Replacing a costly operation by a relatively cheaper one
c) Removing loop invariant computation
d) Removing common sub expression

C. Which of the following is not an intermediate code form?

a) Quadruples
b) Postfix notation
c) Syntax trees
d) Three address codes

D. The optimization technique which is typically applied on loops is

a) Removal of invariant computation
b) Strength Reduction
c) Constant folding
d) All of these

E. Type checking is normally done during

a) Lexical analysis
b) Syntax analysis
c) Semantic Analysis
d) Code optimization

F. A compiler for a high level language that runs on one machine and produce code for different Machine is called

a) Optimizing compiler
b) One pass compiler
c) Cross compiler
d) Multipass compiler

G. Synthesized attribute can be easily simulated by a

a) LL grammar
b) Ambiguous grammar

c) LR grammar
d) None of the above

H. Code can be optimized at
a) Source code
b) Intermediate Code
c) Target Code
d) All of the above

I. In L-attributed Definitions each attribute must be
a) Synthesized
b) Inherited
c) Either Synthesized or Inherited
d) None of the above

J. Reaching definition can be used to identify
a) Dead code elimination
b) Code motion
c) Copy Propagation
d) All of these

(b) Apply dataflow analysis and draw def-use graph for the following code using reaching definitions:

```
n=input;
m = 1;
if ( n>1) {m = m*n; n=n-1}
printf("%d", m);
```

(10 x 1) + (10) = 20

2) (a) Distinguish between local and global optimization? What is peephole optimization? Consider the following sequence of 3-address statements:

(*i*) prod := 0; (*ii*) i := 1; (*iii*) t1 := 4 * i; (*iv*) t2 := a[t1]; (*v*) t3 := 4 * i; (*vi*) t4 := b[t3]; (*vii*) t5 := t2 * t4; (*viii*) t6 := prod + t5; (*ix*) prod := t6; (*x*) t7 := i + 1; (*xi*) i := t7; (*xii*) if i <= 20 goto (*iii*);

Determine various basic blocks in the above segment of code. Draw the control flow graph.

(b) Consider the following code fragment:

```
a = 0;
do{
        b= a+1;
        c=c+b;
        a=b*2;
```

```
} while (a<10);
return c;
```

Apply data-flow analysis and optimize the code by reducing the no. of variables required based on the liveness property of the variables.

(2+2+6) + (10) = 20

3) (a) Why is instruction selection important in code generation? Determine cost for the following instruction sequences (*make usual assumptions*):

(*i*) LD R0, y; LD R1, z; ADD R0,R0,R1;ST x, R0 (*ii*) LD R0,C; LD R1, i; MUL R1,R1,8; ST a(R1), R0

(b) Mention the various steps of register allocation algorithm using graph coloring. Use this algorithm to generate the target code of the following set of instructions assuming only 3 registers are available.

a := b + c, t1 := a * a , b := t1 + a , c := t1 * b, t2 := c + b , a := t2 + t2

(2+6) + (3+9) = 20

4) (a) Define activation record. Distinguish between "static" and "stack" memory allocation. Generate code for the following sequence of instructions assuming stack allocation where SP points to the top of the stack; each action takes 20 bytes of memory, sizes of activation records for procedures m and q are 20 and 60 bytes, respectively; code for m and q start at 100 and 300, respectively; stack starts at 600 (*make other assumptions with proper explanations*).

```
action1    //code for m
call q
action2
halt

action3    //code for q
call q
action6
call q
return
```

(b) Explain with examples the issues of algebraic transformation and dead-code elimination with respect to optimal target code generation

(2+3+11) + (4) = 20

5) (a ) What are the advantages and disadvantages of intermediate code generation phase?

Consider the following Syntax Directed Definition (SDD) that generates three-address code for assignment statements of the form *id = E*, where *id* and *E* represent identifier and arithmetic expression respectively. Attributes ***S. code*** and ***E. code*** denote the three-address code for ***S*** and *E*, respectively. Attribute ***E.addr*** denotes the address that will hold the value of *E*. Function ***top. Get()*** retrieves the symbol table entry corresponding to *id*. A sequence of distinct temporary names ***t1, t2, . . .*** is created by successively executing *new Temp()*.

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $S \rightarrow \textbf{id} = E\ ;$ | $S.code = E.code \ \|\|$ $gen(top.get(\textbf{id}.lexeme)\ '='\ E.addr)$ |
| $E \rightarrow E_1 + E_2$ | $E.addr = \textbf{new}\ Temp()$ <br> $E.code = E_1.code \ \|\| \ E_2.code \ \|\|$ $gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$ |
| $\mid\ - E_1$ | $E.addr = \textbf{new}\ Temp()$ <br> $E.code = E_1.code \ \|\|$ $gen(E.addr\ '='\ '\textbf{minus}'\ E_1.addr)$ |
| $\mid\ ( E_1 )$ | $E.addr = E_1.addr$ <br> $E.code = E_1.code$ |
| $\mid\ \textbf{id}$ | $E.addr = top.get(\textbf{id}.lexeme)$ <br> $E.code = ''$ |

Generate three address code for the statement x= a + - (b+c) by following the semantic rules in the given SDD.

(b) Given the following SDD to generate either a basic type or an array type

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $T \rightarrow B\ C$ | $T.t = C.t$ <br> $C.b = B.t$ |
| $B \rightarrow \textbf{int}$ | $B.t = integer$ |
| $B \rightarrow \textbf{float}$ | $B.t = float$ |
| $C \rightarrow [\ \textbf{num}\ ]\ C_1$ | $C.t = array(\textbf{num}.val,\ C_1.t)$ <br> $C_1.b = C.b$ |
| $C \rightarrow \epsilon$ | $C.t = C.b$ |

where **b** and **t** represent inherited and synthesized attributes respectively. Generate annotated parse-tree and dependence graph for the input string **int[2][3]**.

(2+12) + (4+2) = 20