INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

End-Autumn Semester 2022-23

Date of Examination: 17-11-2022 Session: AN Duration: 3 hrs Full Marks: 100
Subject No.: CS31003 **Subject: Compilers**
Department/Center/School: Department of Computer Science and Engineering
Specific charts, graph paper, log book etc., required: None
Special Instructions (if any): *(1) Answer all the questions. (2) In case of reasonable doubt, make practical assumptions and write that on your answer script. (3) The parts of each question must answered be together.*

1.
```
void func(int arr[], int n)
{
    int i,j,tmp;
    for (j = 0; j < n - 1; j=j+1){
        for (i = 0; i < n - 1 - j; i=i+1){
            if (arr[i] > arr[i + 1]) {
                tmp = arr[i];
                arr[i] = arr[i+1];
                arr[i+1] = tmp;
            }
        }
    }
}
```

(a) Generate the intermediate representation (IR) as quadruples corresponding to the above source code. Assume the size of *int* is 4 bytes, and the first address location is 100. DO NOT optimize your code while generating. No credit for optimized IR generation.

(b) Draw the symbol table corresponding to the given function as appeared after IR phase of the compilation process.

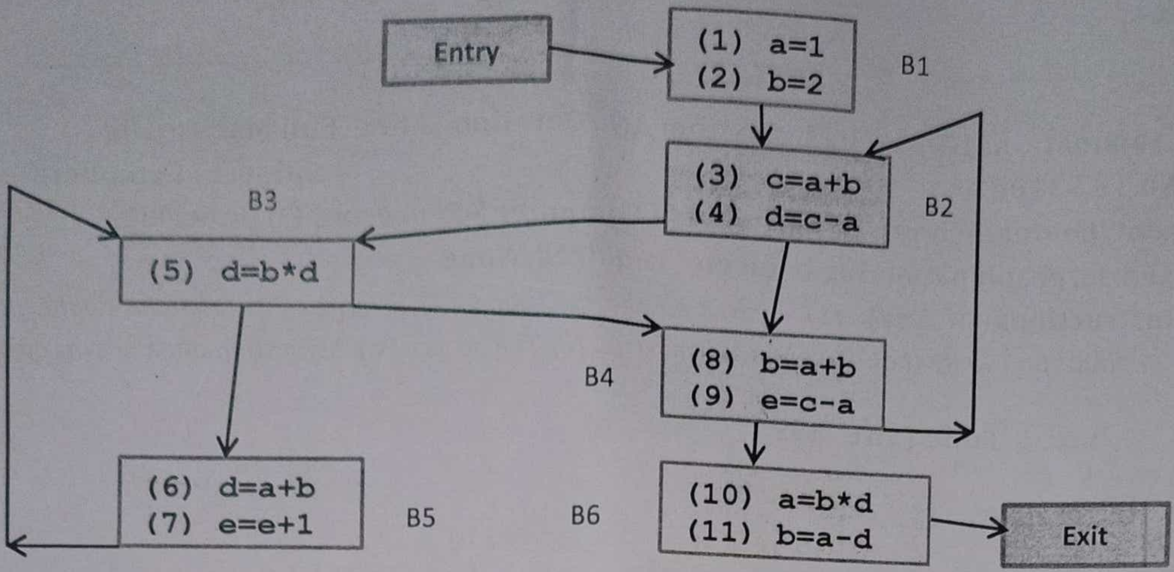
(c) Identify leaders and mark the basic blocks. Draw the control flow graph (CFG). You may use the address locations to mark the basic blocks.

(d) Represent the local expressions of each basic block using value numbering data structure. Next, optimize each basic block by eliminating local common sub-expressions. Analyze all the basic blocks to conclude which block needs further optimization and which is not. Basic blocks with single IR may be ignored from this analysis.

(e) Write down the final optimized IR after previous step.

Marks: [6+6+4+10+4]=30

2.
✓



(a) Formulate a suitable Data Flow Analysis (DFA) problem for (i) reaching definition and (ii) live variable analysis across blocks. Clearly state the definitions for the data-flow direction, confluence operator, initial conditions, and IN, OUT, etc. sets for the DFA.

(b) Perform the reaching definition analysis for each basic block (each rectangle in the above figure is representing one basic block). Show the iterations/passes.

(c) Perform the live variable analysis for each basic block (each rectangle in the above figure is representing one basic block). Show the iterations/passes.

Marks: [(3+3)+6+6]=18

3. Consider that you have a RISC like processor with four registers R1, R2, R3 and R4 and binary machine
✓ instruction (ADD, MULT, SUB) is available for each basic operation OP, in the form of **OP reg1, reg2, reg3,** where reg1 is the register, which stores the result after operating on reg2 and reg3. Two dedicated instructions, such as **LD reg, mem** and **ST mem, reg** are available for the load and store operations.

(a) Now consider the expression $d = (a - b) + b*c + (b+c)$. Directly write the corresponding three address intermediate code (you are allowed to do it in one step. Illustration of the annotated parse tree is not required). Assume that these three address statements constitute a single basic block.
Now answer the following two questions (b) and (c).

(b) From this intermediate code, generate the target code using the simple target code generation algorithm. Clearly show the

(i) Machine instructions generated

(ii) State of the Register descriptor and Address descriptor table before and after each machine instruction generation.

(iii) At each step of the translation, apply GetReg(I) algorithm to obtain the required registers for the generation of the machine code. Suitably show, if any spill operation is required. Note that the GetReg() can access the Register and Address descriptor tables. Attach your comments on the register assignment after each step.

(c) (i) Next, construct the expression tree of the aforesaid expression and annotate with Ersov number.

(ii) Considering the availability of only two registers R1 and R2, generate the optimal machine code for the aforesaid expression using the expression tree and the Ersov number. In the machine code generation process, clearly show how do you optimally choose the registers to store the operands and the results. Clearly show each step.

Marks: [2+(3+4+4)+(2+5)]=20

4. Consider the following grammar (Start symbol S)

```
S →  while B do S
     | begin L end
     | A
L →  L S
     | S
A → id = E
E → E + E | E − E
B → E1 relop E2
B → (B)
B → B1 && B2
B → B1 || B2
B → true
B → false
E → id
```

Note that, the **relop** indicates the relational operators, such as <, > etc

(a) Augment the above grammar with suitable marker nonterminal M at suitable places of the production, such that it can handle backpatching.

(b) Write the semantic actions to design a suitable syntax directed translator (SDT) to generate three address code of the above code snippet. Note that the semantic actions should handle backpatching to generate the three address code.

(c) Apply your SDT to translate the following code snippet to the respective three address code. Assume that the address generation starts from the address 500. Draw the suitably annotated parse tree and clearly show the backpatching steps (say, the *goto* ***Label*** statements before and after the backpatching).

```
begin
        while a > b && x< y do
        begin
                x = y + z
                a = a - b
        end
        x = y - z
end
```

Marks: [2+6+6] =14

5. (a) Consider the productions of a CFG below (with start symbol S). Design a LALR parsing table for the following grammar, if possible.

S→AA
A→aA | b

(b) Consider the following production rules (start symbol P), which allow the declaration of variables.

P → D
D→ T **id**; D
T→ B C
T→ **struct** { D }
B→ **int**
B→ **float**
C→ ε
D→ ε

(i) Add suitable semantic actions at the suitable places, such that the syntax directed translator (SDT) enables the grammar to populate the symbol table with various fields (such as variable name, data type, relative address etc). Introduce and invoke the required functions, if necessary.
(ii) Now consider the declaration statements in a piece of source code of **main()** below

```
int x;
struct
{
        int x;
        float y;
} p;
```

Apply your developed SDT and demonstrate how does this code fragment populate the symbol table entries. In this process, suitably annotate the relevant parse tree.

Marks: [4+(3+3)]=10

6. ✓ (a) Consider the following grammar (start symbol: Based_num) which generates the strings of octal and decimal numbers

Based_num→Base_Char Num
Base_Char→**o** | **d**
Num→Num Digit | Digit
Digit→ **0**|**1**|**2**|**3**|**4**|**5**|**6**|**7**|**8**|**9**

This grammar accepts the strings like **o456, d297**.
Propose a Syntax Directed Definition which can evaluate the numerical values of the generated strings (with appropriate base). Your semantic rules must ensure the proper error assignments for the syntactically valid but semantically invalid strings (say o956). Clearly specify the attributes and semantic rules.
(b) Using the above SDD, evaluate the following strings: o496, d297. Show the dependency graph and the flow of attributes.
(c) What is L-attributed SDD? Justify the constraints behind the L-attributed SDD in the light of the parser implementation.

Marks: [(3+3)+2]=8

**************************************END**************************************