INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
End-Autumn Semester 2023-24

Date of Examination: 16-11-2023 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. Consider that you have a RISC like processor with three registers R1, R2, R3 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 a 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, respectively.

(a) Now consider the expression $x = (a - b) + (a-c) + (b+c)$. 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) Describe the data structure for the Register descriptor and Address descriptor table. State the content of the Register descriptor and Address descriptor table before and after each machine instruction generation.

(iii) At each step of the translation, apply the 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.

Marks: [1+(3+6+2)]=12

2. Consider the evaluation of the following expression
(m*(t*s)/(u+v))*(n*((b-q)*(c+r)))

(i) Next, construct the expression tree of the aforesaid expression and annotate with the 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 and spill operations in the expression tree.

Marks: [3+5]=8

3.

Addr.	Quad	Addr.	Quad	Addr.	Quad
200	t1=i	211	t10=t9*4	221	t16=t15*4
201	t2=t1*4	212	if a[t7]<a[t10] goto 214	222	t17=i
202	t3=i	213	goto 216	223	t18=t17*4
203	t4=t3+1	214	t11=100	224	a[t18]=a[t16]
204	t5=t4*4	215	temp=t11	225	t19=temp
205	if a[t2]>a[t5] goto 207	216	t12=i	226	t20=i
206	goto 230	217	t13=t12*4	227	t21=t20+1
207	t6=i	218	temp=a[t13]	228	t22=t21*4
208	t7=t6*4	219	t14=i	229	a[t22]=t19
209	t8=i	220	t15=t14+1	230	Return
210	t9=t8+1				

All the variables with naming convention t# (where # is a number) are compiler generated temporaries.

(a) Determine the minimum window size (number of consecutive quad's at a time and sliding over from beginning to end), to maximize peep-hole optimization for the following possible optimizations on the code. Justify your window size by mentioning the consecutive quads and the kind of optimization.

(b) Eliminate local definitions and use of temporary variables. Perform algebraic simplifications and strength reduction.

(c) Identify the unreachable code and specify what other analysis you have to perform to declare it as an unreachable code.

(d) Renumber the quad's, re-target the goto's and rewrite the TAC program after clean-up.

Marks: [3+6+4+2]=15

4. Consider the following grammar with P as the start symbol

P ⇨ S
S ⇨ while (B) S | {L} | L | A
L⇨ LS | S
A ⇨ id=E
B ⇨B or B | B and B | (B) | E relop E | E
E ⇨ E+E | E*E | id | num

(a) Augment the above grammar with suitable marker nonterminals 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 snippet. Note that the semantic actions should handle backpatching to generate the three address codes.

(c) Apply your SDT to translate the following code snippet to the respective three address codes. Assume that the address generation starts from the address 300. Draw the suitably annotated parse tree and clearly show the backpatching steps (say, the ***goto Label*** statements before and after the backpatching).

```
i=1
f=1
while (i ≤ 10 and f>0)
{
        f=f*i
        i=i+1
}
```

Marks: [1+2+7]=10

5.

Addr.	Quad	Addr.	Quad	Addr.	Quad
100	t1=0	112	t6=d*4	124	t14=t11*t13
101	d=t1	113	t7=a[t5]	125	t15=x
102	t2=100	114	t8=100	126	t16=t14+t15
103	n=t2	115	if t7>t8 goto 117	127	t17=d*4
104	if d< n goto 109	116	goto 127	128	t18=a[t17]
105	goto 135	117	t9=20	129	t19=d*4
106	t3=d	118	if d<t9 goto 120	130	t20=b[t19]
107	t3=t3+1	119	goto 127	131	t21=t18+t20
108	d=t3	120	t10=d*4	132	t22=x
109	t4=2*2	121	t11=a[t10]	133	t23=t21*t22
110	t5=t4*2	122	t12=d*4	134	goto 106
111	x=t5	123	t13=b[t12]	135	Return

All the variables with naming convention t# (where # is a number) are compiler generated temporaries.

(a) Construct the CFG, by marking the leader quad's, and identifying the basic blocks.

(b) Using the value-numbering method on DAG's, determine the local common sub-expressions only within the basic blocks that contains quad (122: t12=d*4) and quad (129: t19=d*4). Show the states of the required data structures over this process. Eliminate local common sub-expressions. Redraw the CFG after the local common subexpression elimination.

(c) To determine the common sub-expressions across blocks, formulate a suitable DFA problem for Available Expressions at the entry and exit of every block. Clearly state the definitions for the data-flow direction, confluence operator, initial conditions, and IN, OUT, GEN & KILL sets for the DFA. Outline the algorithm to solve the DFA equations.

(d) Using your formulation, compute the Available Expressions at the entry and exit of every block. Show the iterations for this computation.

(e) Using the Available Expressions, eliminate the common sub-expressions.

(f) In every block where a sub-expression has been eliminated, locally propagate copies.

(g) In every block where copies have been propagated, eliminate unused temporary.

(h) Redraw the CFG after optimization.

Marks: [4+8+5+10+2+2+2+2]=35

6. Consider the following grammar, where S is the start symbol.

S ⇨ id=E | L=E
E ⇨ E+E | id | L
L ⇨ id [E] | L [E]

(a) Design a syntax directed translator to generate three address codes of the above grammar.

(b) Consider an integer matrix A of size **3×5**, and integer array B. Using your SDT, translate the following statement to a three-address code. Show the annotated parse tree in the process.

A[i][j] = B[p+q]

(c) Draw an optimized DAG for the following expressions of a basic block. Show all the steps.

x=a[i]
p=a[i]
a[j]=y
z=a[i]

Marks: [(2+6)+2]=10

7. Consider the following grammar G with start symbol S

S⇨ AaAb | BbBa
A⇨ €
B⇨ €

(a) Justify if the grammar G is in LL(1)

(b) (i) Justify if the grammar G is in SLR(1)
(ii) From the constructed LR(0) automaton, can you identify a viable prefix, which may lead to an **invalid** reduction step? Show the respective stack configuration and the relevant Item Set(s).

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