FINAL EXAM CS322 2012 - 2013

<u>Question1: Run – time stack</u> [30 marks]

Show the stack with all activation records, including static and dynamic chains, when execution reaches position ***1*** in the following skeletal program. Assume ***Bigsub*** has a static_depth of ***1***.

```
procedure Bigsub is

        procedure A (flag : Boolean) is

                procedure B is

                        …

                        A (false);

                end; -- of B

        begin -- of A

                if flag then B;

                else        C;

                …

        end; -- end of A

        procedure C is

                procedure D is

                        …        ← 1

                end; -- end of D

                …

                D;

        end; -- end of C

begin -- of Bigsub

        …

        A (true);

end; -- of Bigsub
```

The calling sequence for this program for execution to reach ***D*** is ***Bigsub*** calls ***A***, which calls ***B***, which calls ***A***, which calls ***C***, which calls ***D***.

Question 2: Backpatching [30 marks]

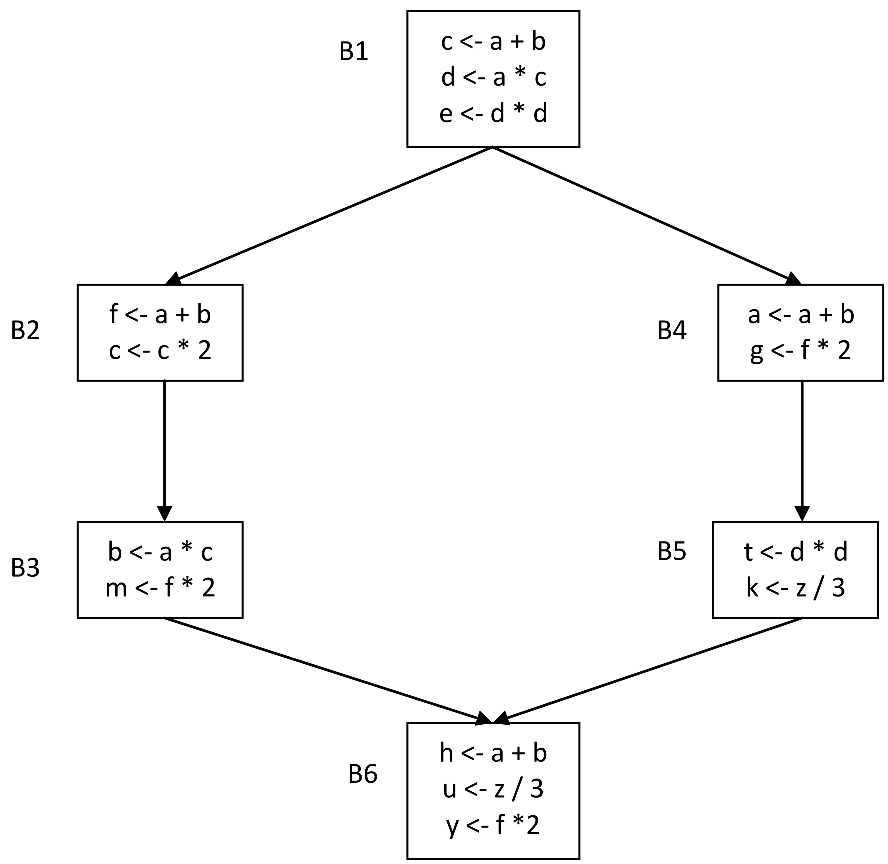
Using the translation scheme for boolean expression, translate the following expression. You may assume the address of the first instruction generated is 100.

a < b || c < d && e < f

Note: Assuming that && (and) has precedence over || (or) and they are left – associative.

Question 3: Code Optimization [40 marks]

Consider the following CFG:



1. Compute DEEXPR, EXPRKILL, and AVAIL sets for the blocks in this CFG.
2. In this CFG, which expressions does the global redundancy elimination algorithm (GRE) find as redundant?

------------------------END-----------------------

REFERENCE

Translation scheme for boolean expressions (Backpatching technique)

B -> B1 || M B2	{	backpatch (B1.falselist, M.instr); B.truelist = merge (B1.truelist, B2.truelist); B.falselist = B2.falselist; }
B -> B1 && M B2	{	backpatch (B1.truelist, M.instr); B.truelist = B2.truelist; B.falselist = merge (B1.falselist, B2.falselist); }
B -> !B1	{	B.truelist = B1.falselist; B.falselist = B1.truelist;}
B -> (B1)	{	B.truelist = B1.truelist; B.falselist = B1.falselist;}
B -> E1 rel E2	{	B.truelist = makelist (nextinstr); B.falselist = makelist (nextinstr+1); emit (‘ifTrue’ E1.addr rel E1.addr ‘goto_’); emit (‘goto_’);}
B -> true	{	B.truelist = makelist (nextinstr); emit (‘goto_’);
B -> false	{	B.falselist = makelist (nextinstr); emit (‘goto_’);}
B -> Ɛ	{	M.instr = nextstr;}

Compute the local sets

VarKill <- Ø

DEExpr(n) <- Ø

for i = k **downto** 1 {

// Assume each operation is of the form “x <- y op z”

VarKill <- VarKill ∪ {x}

If (y ∉ VarKill) **and** (z VarKill)

Add expression “y op z” to DEExpr(n)

}

ExprKill(n) <- Ø

for Each expression *e* in the global scope

    for Each variable *v* ∈ *e*

        if *v* ∈ VarKill
            ExprKill(n) <- ExprKill(n) ∪ {*e*}

Computing AVAIL sets

for i = 0 to h {

    Compute DEExpr(n_i) and ExprKill(n_i)

    Avail(n_i) <- Ø

}

Changed <- true

while (Changed) {

    Changed <- false

    for i = 0 to h {

        OldValue <- Avail (n_i)

        Avail (n_i) = $\bigcap_{m \in pred\ (n_i)}(DEExpr(m) \cup Avail(m) \cap \overline{ExprKill(m)})$}

        If Avail (n_i) $\neq$ OldValue

            Changed <- true

    }

}