COL728 Major Exam
Compiler Design
Sem II, 2017-18

Answer all 4 questions Max. Marks: 30

1. Operational semantics: Consider the following statement in a C-like language:

//y == 1 and x == 2 at this point
z = (y = x - y) + (x = x + y);

This is a valid C statement

a. If before this statement is executed, x == 2 and y == 1, then what according to you is the final value of z? [1]

There are two possible answers and both evaluate to 4. Either the first expression may be evaluated first or the second expression may be evaluated first. Either would be correct semantics. Recall that C leaves the order of evaluation of operands “unspecified”. But you were supposed to pick one order and evaluate it in that order.

For example, y = x-y is executed first, then x = x+y and at last z = y+x

Full marks for correct answer, none for wrong.

b. Write the formal operational semantics rules that you applied to arrive at your answer. Your rules should be correct (sound) and complete for evaluating this statement, and should agree with your answer in part a. With each rule, write a few sentences to indicate which rule gets invoked for evaluating which part(s) of the statement above. [7]

$$\frac{E,S \Rightarrow e_1:v_1,\ S_1 \qquad E,S_1 \Rightarrow e_2:v_2,\ S_2}{E,S_2 \Rightarrow e_1+e_2 : v_1+v_2, S_3}$$

Above semantic rule is applied to evaluate x+y and (y=x-y)+(x=x+y)

$$\frac{E,S \Rightarrow e_1:v_1,\ S_1 \qquad E,S_1 \Rightarrow e_2:v_2,\ S_2}{E,S_2 \Rightarrow e_1-e_2 : v_1+v_2, S_3}$$

Above semantic rule is applied to evaluate x-y

$$\frac{E,S \Rightarrow e:v,\ S_1 \qquad E[id] = l_{id} \qquad S_2 = S_1[v / l_{id}]}{E,S \Rightarrow id \leftarrow v,\ S2}$$

Above semantic rule is applied to evaluate assignment to y, x and then to z.

Each rule is of 1.5 marks and 2.5 marks for explanation.

2. Register allocation through graph coloring : Explain what is optimistic coloring using an example program. Write an example program (different from the one that was used in class). Draw its register interference graph. And colour it using the optimistic coloring algorithm. Your example should involve at least one case where the "optimistic" nature of the coloring algorithm yields an advantage. Clearly mention at which step did the optimistic nature of the coloring algorithm help obtain a more efficient solution. [6]

Example Program with its liveness analysis is :

```
                    ---- a
if(a >= 0)
{
                    ---- a
  b = a
                    ---- a,b
  d = a^2 + b
                    ---- d,b
  d = d * b
                    ---- d
}
else
{
                    ---- a
  c =  -a
                    ---- c,a
  d = a^2 + c
                    ---- d,c
  d = d * c
                    ---- d
}
```

Register Interference graph(RIG) for the above program is:

a-b
a-c
b-d
c-d

The register allocation problem can be considered as a graph-coloring problem of its RIG. A coloring of a graph is an assignment of colors to nodes, such that nodes connected by an edge have different colors. A graph is *k-colorable* if it has a coloring with k colors.
Let's say, we want to color our example RIG with 2 colors.

Optimistic Coloring Algorithm:

Step 1: Pick a node with fewer than 2 neighbors.

In our example RIG, all the 4 nodes have 2 neighbors. In this case, pick a node as a candidate for spilling. This temporary variable will "live" in memory.
Remove the picked node and all its edges from the RIG. Let’s say, we pick node “a” for spilling.

Step 2 : After removing a, the remaining graph (b-d, c-d) can be colored with 2 colors.
b -- red
c -- red
d -- black

Step 3 : In optimistic coloring, we now try to add back spilled node and it may be possible that after adding it, the graph is still k colorable.
In our example, since both the neighbors of a, i.e. b,c are red and after adding a, the graph is still 2 colorable, thus providing a more efficient solution.
a -- black
b -- red
c -- red
d -- black

Identifying a correct example and making its register inference graph : 2 marks
Running graph colouring on it : 2 marks
Explanation of optimistic coloring and its role in register allocation : 2 marks
3 marks will be deducted if example does not show benefit of optimistic nature of coloring
No marks if class example is used.

3. Dataflow analysis : copy propagation

Recall that copy propagation refers to a transformation pass where occurrences of targets of copy-assignments are replaced with their RHS values and the copy-assignment is eliminated. E.g.,

```
x = y
z = x + 3
w = y + z
```

Will be replaced to:

```
z = y + 3
w = y + z
```

In this example: x has been replaced with y through copy-propagation.

Specify an algorithm that implements global copy-propagation. Your algorithm should involve a dataflow analysis followed by a transformation that makes use of the results of the dataflow analysis. Clearly specify the values that you compute using copy propagation. Specify the direction, meet operator, and the transfer functions. If your transfer functions need to know the Use/Kill sets for an instruction, please specify them clearly with examples. Also, mention the boundary conditions. Finally, describe the transformation pass that will use the results of this analysis. [6]

For implementing Copy-propagation we will run data flow analysis passes until fixpoint on the instructions with parameters defined below. Our aim is to remove unnecessary assignment instructions of form x=y where x and y are variables by replacing x with y on subsequent uses of x if this is only assignment instruction to x before that use.

[0.5]Direction : Forward
[0.5]Meet Operator : Intersection
Ordering / Lattice :

Complete domain of form (x=y)

$a_1=a_2$ $\quad$ $a_2=a_3$ $\quad$ ------ $\quad$ $a_x=a_y$

Empty

[0.5]Values Computed : Available substitutions of the form x=y means substitute all occurrences of x with y. (where x and y are variables)

[1]Transfer functions :

For any statement s_i
$In[S_i] = \cap(Out[P_j])$ for all j such that P_j is a predecessor of S_i
$Out[S_i] = Use\ \mathit{U}\ (In[S_i] - Kill)$

[0.5]Use : Any available substitution of the form x=y (where x and y are variables) present in the instruction will add substitution x=y to available substitutions.
[0.5]Kill : Any statement of the form x=e where e is some expression will kill all substitutions from the available substitutions set in which x is present in LHS or RHS
E.g.
instruction is x = y then its use set is {x=y}
And kill set will be ay expression having x in LHS or RHS e.g. {y=x, x=z}

[0.5]Boundary Conditions : At entry instructions initialize with empty set.

Initialization: Initialise all instructions with set x=y for all possible variables x and y

[1]Transformation :
After reaching a fixpoint in computing set of available substitutions before each instruction, we can start doing substitutions. We can define a global order on the variables, such that the variables appearing later in that order can be substituted with the variables earlier in that order.

At any instruction that uses y, and where there exists a substituting condition x=y (or y=x) where x occurs earlier in the order than y (i.e., x < y), we can replace y with x. We can keep doing so, until a fixpoint is reached. Notice that it is possible that after replacing y with x, x is later replaced by w.

After this we can run liveness analysis on variables and dead code elimination to remove all assignment instructions which are useless

[1]You can run this algorithm on any program using a block diagram or a list of instructions or describe how above all these parameters work together in dataflow algorithm.

x = y	Available set = {}
z = x + 3	{(x=y)}
x = x+2	{(x=y)}
y=z	{}
w = y + x	{(y=z)}

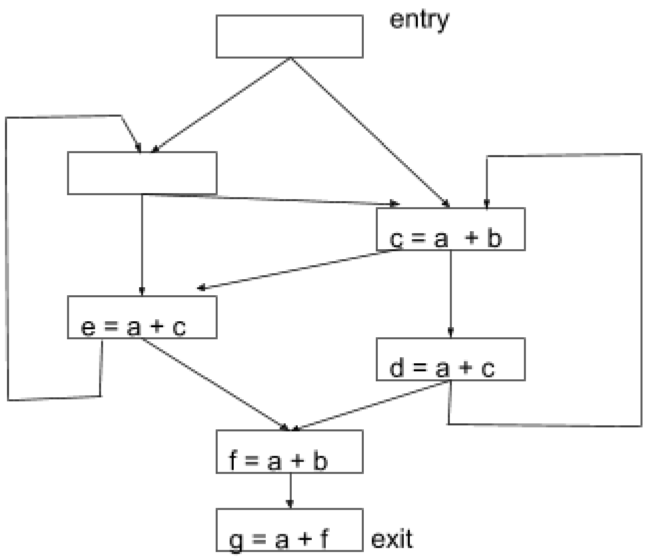
After substitution pass:
x=y
z=y+3
x= y+2
y=z
w=z+x
After dead code removal and liveness analysis
z=y+3
x=y+2

w=z+x

Marks for each part is specified.

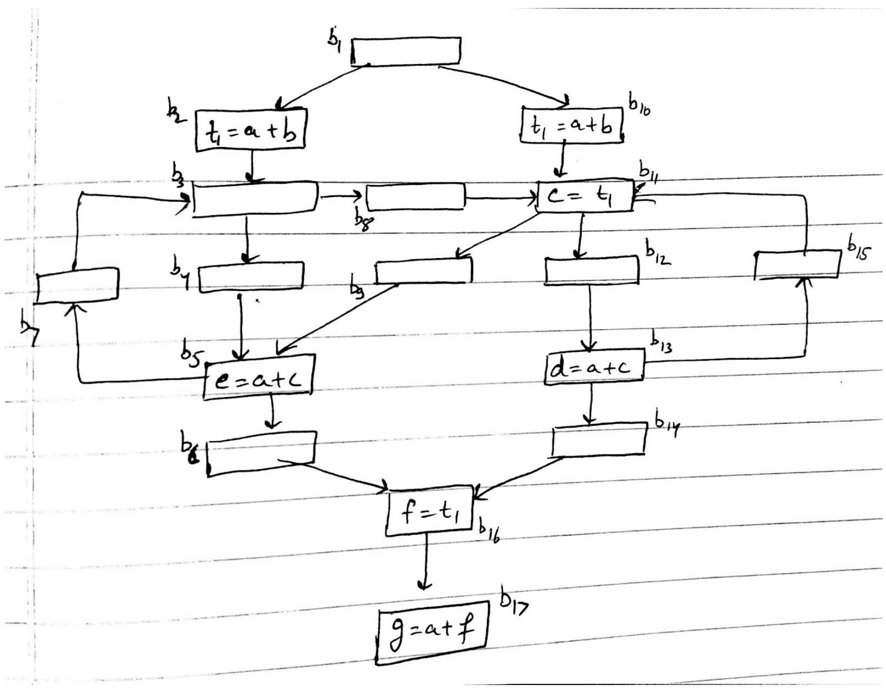
4. Lazy code motion

-



a. Apply the lazy code motion algorithm described in class to the program above. Show the result of the optimization --- it is not necessary to show any intermediate steps. (Introduce new basic blocks as necessary). [6]

Answer for this program with all critical edges removed is shown below:



Expression a+b is needed in : IN : block 1 to 16 OUT : block 1 to 15

Expression a+c is needed in : IN : 4,5,9,12,13   OUT : 4,9,11,12
Expression a+f is needed in : IN : 17   OUT : 16
Expression a+b is missing in : IN : 1
Expression a+c is missing in : IN : 1,2,3,4,8,9,10,11,12   OUT : 1,2,3,8,10,11
Expression a+f is missing in : IN : 1 to 17   OUT : 1 to 16

Expression a+b is postponed in : IN : 2,10   OUT : 1, 2, 10
Expression a+c is postponed in : IN : 5,13   OUT : 4, 9, 12
Expression a+f is postponed in :

Full marks for correct answer. Partial marking if result is correct according to some passes.

b. Are there redundant operations remaining after the optimization? If not, explain why not. If yes, explain why redundant operations were left even after applying the transformation. [4]

Yes there are some redundancies still left e.g. loop 3-4-5-7-3 will calculate a+c again and again though it is not needed. This is because of edge 11-9-5 which also joins this loop and this path kills expression c and so a+c must be recalculated.

This redundancy can also be reduced if one of the two paths among 3-4-5 or 11-12-13 is replicated, but that is not supported by the lazy-code motion algorithm.

2 marks for identifying redundancies. 2 for explanation.