CS 152 Final Exam

March 19, 2020

Question	Possible Points	Points Scored
1	15	
2	15	
3	15	
4	25	
5	30	
Total	100	

1. (15 points) Short Questions.

   (a) (5 points) Identify the leaders of basic blocks in the intermediate code given below.

```
1   Read i
2   if i < 0 go to 10
3   Read j
4   if j < 0 go to 8
5   j = j + 1
6   if j > 10 go to 8
7   go to 4
8   i = i - j
9   go to 2
10  Write i,j;
```

(b) (5 points) Give an example of a situation where the need for *backpatching* arises during code generation? Explain why backpatching is needed.

forward branches when branch target is specified using instruction number.
because we don't know the instruction number of target, the target in the branch generated is left blank & backpatched later when we reach/generate the target instruction.

(c) (5 points) Differentiate heap based memory allocation from stack based memory allocation in terms of their *efficiency* and *generality*. Justify your answer.

	Efficiency	Generality	
Heap		✓	→ arbitrary order vs LIFO order
Stack	✓		

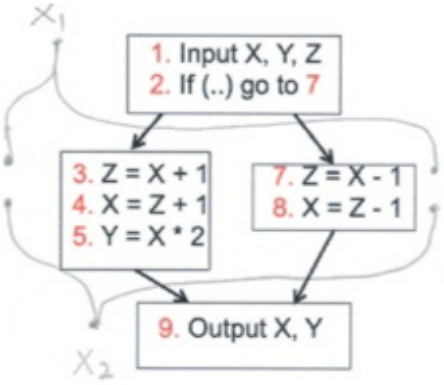
↓

bumping the stack vs search & update free list

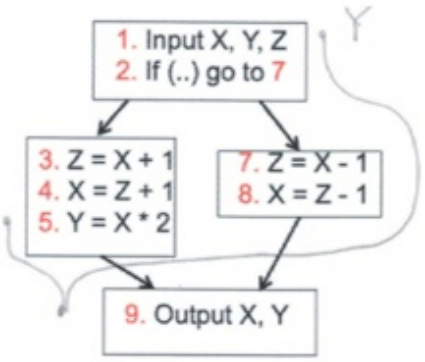
2. (15 points) For the code given below, assuming that none of the variables are live at the end of the code segment:

   (a) (8 points) Draw the live ranges for all the variables (X, Y, Z).

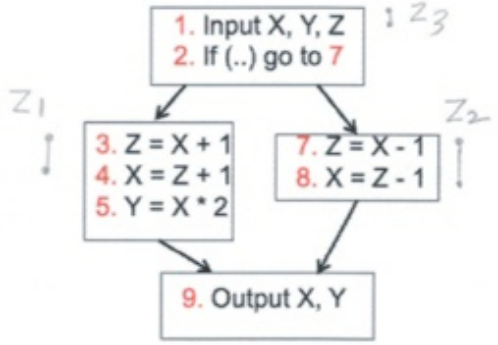
Live Range(s) of X



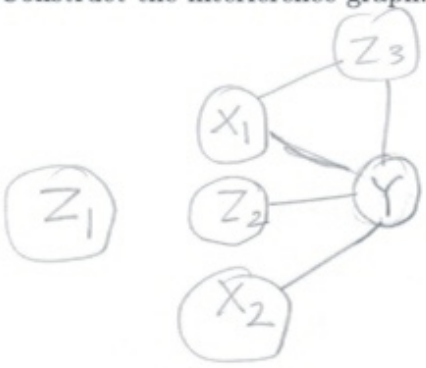
Live Range(s) of Y



Live Range(s) of Z



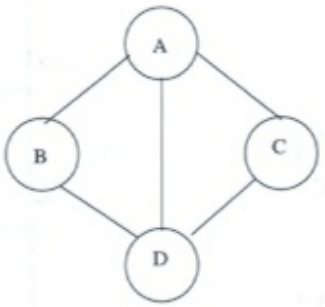
   (b) (7 points) Construct the interference graph.



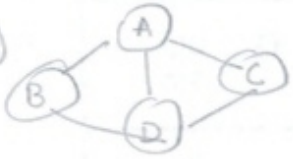
3. (15 points) Using the coloring algorithm discussed in class, for the interference graph given below:

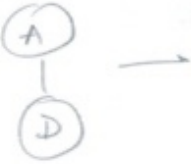
   (a) (7 points) Allocate registers using coloring assuming 3 registers are available.

   (b) (8 points) Allocate registers using coloring assuming 2 registers are available.



(a)



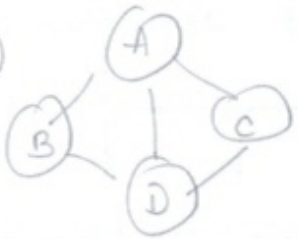


empty

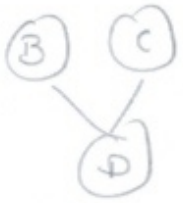
B, C; A, D

R_3 R_3 R_2 R_1

(b)



spill A



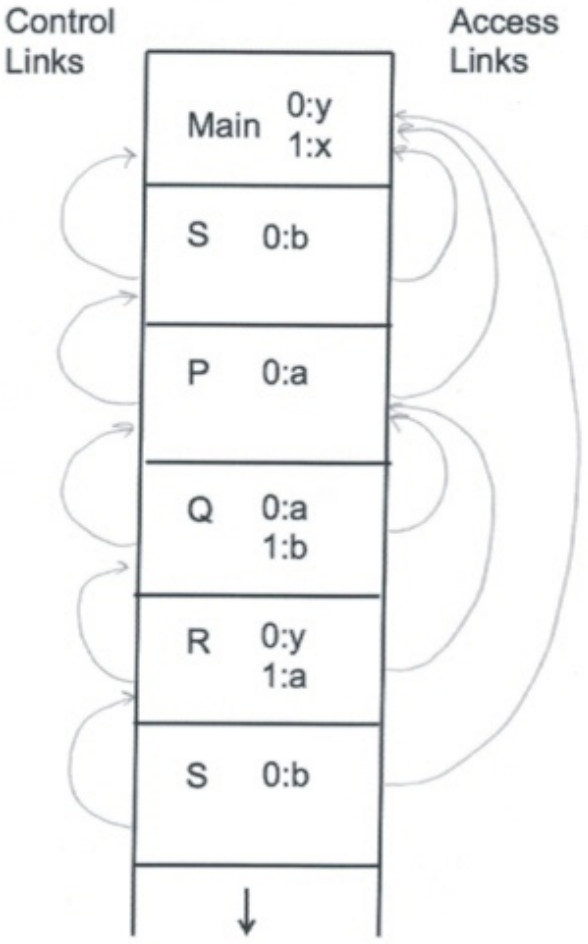
B, C, D

R_2 R_2 R_1

4. (25 points) For the program below and its runtime stack for the call sequence $Main \rightarrow S \rightarrow P \rightarrow Q \rightarrow R \rightarrow S$, answer the following questions.

(a) (15 points) Draw the control and access links for the runtime stack.

```
Main () {
   integer y, x;
   Procedure P () {
      integer a;
      Procedure Q () {
         integer a, b;
         Call R ();
      }
      Procedure R () {
         integer y, a;
         Call S ();
      }
      Call Q ();
   }
   Procedure S () {
      integer b;
      Call P ();
   }
   Call S ();
}
```



(b) (5 points) In the given call sequence when R calls S, show how S's access link is setup.

3 − 1 = 2 R →(al) P →(al) main S →(al) main

(c) (5 points) When inside procedure Q, explain how access links are used to access variable y.

3 − 1 = 2 traverse two links taking to main & then use offset 0 to access y.

5. (30 points) The grammar given below corresponds to a new control construct with the following semantics. When the value of variable corresponding to 'id' is less than zero, equal to zero, or greater than zero, the code in the 'less', 'equal', or 'greater' part is executed.

   (a) (10 points) First show the form of intermediate code that uses comparisons and branches (unconditional and conditional) to implement this construct.

   (b) Second provide the semantic rules to implement this construct – the generated code should be eventually available in the attribute $< S >.code$.

$< S > \rightarrow$ switch id do $< less > < equal > < greater >$
$< less > \rightarrow$ less $< S >$ endless
$< equal > \rightarrow$ equal $< S >$ endequal
$< greater > \rightarrow$ greater $< S >$ endgreater

if id.place < 0 goto Less
if id.place = 0 goto equal
if id.place > 0 goto greater

<less> → less <S> endless { <less>.code = <S>.code; <less>.label = newlabel(); }

<equal> → equal <S> endequal { <equal>.code = <S>.code; <equal>.label = newlabel(); }

<greater> → greater <S> endgreater { <greater>.code = <S>.code; <greater>.label = newlabel(); }

<S> → switch id do <less> <equal> <greater> {
exit = newlabel();
<S>.code = gen(if id.place < 0 goto <less>.label)
|| gen(if id.place = 0 goto <equal>.label)
|| gen(if id.place > 0 goto <greater>.label)
|| gen(<less>.label:) || <less>.code
|| gen(goto exit)
|| gen(<equal>.label:) || <equal>.code
|| gen(goto exit)
|| gen(<greater>.label:) || <greater>.code
|| gen(exit:)

less: <S>
goto exit
equal: <S>
goto exit
greater: <S>
exit: