Name: _____

CS322 Spring'08: Languages and Compiler Design II 5/6/08

Prof. Jingke Li (FAB 120-06, li@cs.pdx.edu), Classes: TTh 12-1:50pm, EB102, Office Hours: TTh 10-11am.

1. _____
2. _____
3. _____
4. _____
5. _____

Midterm Exam

(Time Limit: 90 minutes)

There are 5 problems in total, each worthes *10 points*. Write your answers in the given spaces, continue on back pages if necessary. This is a **closed book** exam, but you are allowed to use a single sheet of notes.

1. [*10 pts*] Stack machine code is a form of intermediate code. It assumes the presence of an operand stack. Most operations take their operands from the stack and push their results back onto the stack. For example, an integer subtract operation would remove the top two elements from the stack and push their difference onto the stack. Neither the operands nor the result need be referenced explicitly in the subtract instruction. In fact, only a few instructions (e.g. `push` and `pop`) need to reference a *single* operand explicitly. Stack machine code hence is called *one-address code*.

   Assume that we have a one-address code with the following instructions:

`add`, `sub`, `mul`, `div`	pop off the top two elements from the stack, perform the corresponding arithmetic operation (the very top element is the left operand), and push the result back to the stack
`swap`	swap the top two elements of the stack
`push` x	push x onto the stack
`pop` x	pop off the top element from the stack, and assign it to x

   Translate the expression `a - 2 * b + (c + 1) / d` into this one-address code. No additional variables are allowed.

2. [*10 pts*] Translate Boolean expression "`a and (not b or c)`" (with short-circuit semantics) into three-address code, using *only* the following statements:

`label` *lab*	— introduce a label *lab*
`goto` *lab*	— jump to label *lab*
`if0` *v* `goto` *lab*	— if *v* `=false` jump to label *lab*
t `:=` *v*	— assign *v* to *t*

Record the result in a temp. Assume that Boolean type is available for the IR code, so the temp's value should be either `true` or `false`.

Extra Credit: If you use six or less statements, you will receive 2 extra points.

3. [*10 pts*] Consider the following MINI program

```
class test {
  public static void main(String[] dummy) {
    B b = new B();
    System.out.println(b.sum());
  }
}
class A {
  int x = 1;
  public int value(int i) { return x+i; }
}
class B extends A {
  int y;
  public int sum() {
    int z;
    z = this.value(2);
    return x+y+z;
  }
}
```

(a) Fill out the following table for the three methods, `main`, `value`, and `sum`:

method's label			
varCnt			
argCnt			

(b) Translate the `return` statement of method `sum` into an IR tree statement. Be precise.

4. [*10 pts*] Consider the following code fragment.

```
(1)  m := 0
(2)  v := 0
(3)  if (v>=n) goto (15)
(4)  r := v
(5)  s := 0
(6)  if (r<n) goto (9)
(7)  v := v+1
(8)  goto (3)
(9)  x := A[r]
(10) s := s+x
(11) if (s<=m) goto (13)
(12) m := s
(13) r := r+1
(14) goto (6)
(15) return m
```

(a) Break the code into basic blocks and draw a control-flow graph for it.

(b) Identifier natural loops in the CFG. For each loop, list all the basic blocks that belong to it, and indicate its header and tail.

5. [*10 pts*] Recall the value numbering optimization idea: each variable's initial value is assumed to be unique; an expression value is determined based on its variables' values; and each unique value is assigned a distinct number. So `a+b` would correspond to three values: `a => '1'; b => '2'; a+b = '1'+'2' => '3'`.

   Apply the value numbering optimization to the following code:

```
c := a
d := b
u := a+d
v := c+b
w := a*b
x := u*w
y := c*d
```

   (a) Fill out the following table

var	*num*	*expr*	*num*
a		a+d	
b		c+b	
c		a*b	
d		u*w	
u		c*d	
v			
w			
x			
y			

   (b) Simplify the code with the value numbering result (don't delete any statements).