Compiler Construction
Professor Jeremy Siek

Quiz 1
September 10, 2012

Name: _____

This exam has 3 questions, for a total of 10 points.

1. [3 points] Write down the output of the flatten pass applied to the following P_0 program. Although the output of flatten is an abstract syntax tree, please instead write down the corresponding textual representation of the P_0 program (similar to the below).

```
x = - - input()
print 3 + x
```

Solution:

```
t0 = input()
t1 = - t0
x = - t1
t2 = 3 + x
print t2
```

2. [3 points] Fill in the missing code for `Add` and `UnarySub` in the definition of the following `flatten` function.

```
def flatten(n):
    if isinstance(n, Module):
        return Module(n.doc, flatten(n.node))
    elif isinstance(n, Stmt):
        sss  = [flatten(s) for s in n.nodes]
        return reduce(lambda a,b: a + b, sss, [])
   elif isinstance(n, Assign):
        (rhs,ss) = flatten(n.expr)
        return ss + [Assign(n.nodes, rhs)]
    elif isinstance(n, Name):
        return (n, [])
    elif isinstance(n, Add):









    elif isinstance(n, UnarySub):
```

.

Solution: Here's the code for the `Add` and `UnarySub` cases.

```
elif isinstance(n, Add):
    (left, ss1) = flatten(n.left)
    (right, ss2) = flatten(n.right)
    tmp = generate_name('tmp')
    assign = Assign(nodes=[AssName(name=tmp, flags='OP_ASSIGN')],
                    expr=Add((left, right)))
    return (Name(tmp), ss1 + ss2 + [assign])

elif isinstance(n, UnarySub):
    (expr,ss) = flatten(n.expr)
    tmp = generate_name('tmp')
    assign = Assign(nodes=[AssName(name=tmp, flags='OP_ASSIGN')],
                    expr=UnarySub(expr))
    return (Name(tmp), ss + [assign])
```

3. [4 points] Write down the x86 assembly program that your compiler would produce for the program in question 1.

Solution: The following version is for MacOS X, this solution differs with respect to stack alignment from solutions for Linux, which is fine.

```
.globl _main
_main:
        pushl %ebp
        movl %esp, %ebp
        subl $24, %esp
        call _input
        movl %eax, -16(%ebp)
        movl -16(%ebp), %eax
        negl %eax
        movl %eax, -8(%ebp)
        movl -8(%ebp), %eax
        negl %eax
        movl %eax, -4(%ebp)
        movl $3, %eax
        addl -4(%ebp), %eax
        movl %eax, -12(%ebp)
        subl $12, %esp
        pushl -12(%ebp)
        call _print_int_nl
        addl $16, %esp
        movl $0, %eax
        leave
        ret
```