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

1. **3 points** Draw the abstract syntax tree that Python’s builtin parser would produce for the following program.
   
   ```python
   print - input() + input()
   ```

   **Solution:** The following is a textual version of the solution. A graphical drawing is also OK as long as it contains the key information.

   ```python
   Module(None, Stmt([
       Printnl([Add((UnarySub(CallFunc(Name('input'), []), None, None)),
                CallFunc(Name('input'), [], None, None))],
       None)])
   ```

2. **3 points** Fill in the missing code for `Stmt` in the definition of the following `flatten` function.

   ```python
def flatten(n):
    if isinstance(n, Module):
        return Module(n.doc, flatten(n.node))
    elif isinstance(n, Stmt):
        # ...
    ...
    elif isinstance(n, Name):
        return (n, [])
    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])
    ...
    ```

   **Solution:** Here’s the code for the `Stmt` case.

   ```python
   sss = [flatten(s) for s in n.nodes]
   return reduce(lambda a, b: a + b, sss, [])
   ```
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, so your solution may differ with respect to stack alignment.

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