Compilers (Python)
CSCI P423/523, Fall 2021

Final

Name: ______________________

This exam has 12 questions, for a total of 100 points.

1. $\boxed{\text{4 points}}$ What is the output of the following Python program?

```
a = [[0], 1]
b = a[0]
c = a
c[0] = [1]
print(b[0])
```

2. $\boxed{\text{4 points}}$ What is the output of the following Python program?

```
a = [[0], 1]
b = a[0]
c = a
c[0][0] = 1
print(b[0])
```

3. $\boxed{\text{4 points}}$ What is the output of the following Python program?

```
def f(x : int) -> None:
    x = 0

y = 1
f(y)
print(y)
```

4. $\boxed{\text{4 points}}$ Why does our compiler spill variables of `tuple` type to the root stack instead of the regular procedure call stack?

5. $\boxed{\text{4 points}}$ Why must the prelude of a function push the contents of the `rbp` register to the procedure call stack?

6. $\boxed{\text{10 points}}$ Given the following program, what would be the output of the Expose Allocation pass? Recall that you may used the new AST nodes `GlobalValue`, `Allocate`, and `Collect`.

```
print( (42,)[0] )
```

7. [12 points] Given the input program on the left, fill in the blanks in the output of Select Instructions on the right.

```
_start:
    init.321 = 42
    tmp.322 = free_ptr
    tmp.323 = tmp.322 + 16
    tmp.324 = fromspace_end
    if tmp.323 < tmp.324:
      goto _block.328
    else:
      goto _block.329

_block.328:
    goto _block.327

_block.329:
    collect(16)
    goto _block.327

_block.327:
    alloc.320 = allocate(1,tuple[int])
    alloc.320[0] = init.321
    tmp.325 = alloc.320
    tmp.326 = tmp.325[0]
    print(tmp.326)
    return 0
```

```
_start:
    movq $42, init.321
    movq ___(a)___, tmp.322
    movq tmp.322, tmp.323
    addq $16, tmp.323
    movq ___(b)___, tmp.324
    cmpq tmp.324, tmp.323
    jl _block.328
    jmp _block.329

_block.328:
    jmp _block.327

_block.329:
    movq %r15, %rdi
    movq $16, %rsi
    ___(c)___
    jmp _block.327

_block.327:
    movq _free_ptr(%rip), %r11
    ___(d)___
    movq $3, 0(%r11)
    movq %r11, alloc.320
    movq alloc.320, %r11
    ___(e)___
    movq alloc.320, tmp.325
    movq tmp.325, %r11
    ___(f)___
    movq %r11, tmp.326
    movq tmp.326, %rdi
    callq _print_int
    movq $0, %rax
    jmp _conclusion
```

8. $\boxed{\text{12 points}}$ Draw the interference graph for the following program fragment by adding edges between the nodes below. You do not need to include edges between two registers. The live-after set for each instruction is given to the right of each instruction and the types of each variable is listed below.

Recall that the caller-saved registers are

```
rax rcx rdx rsi rdi r8 r9 r10 r11
```

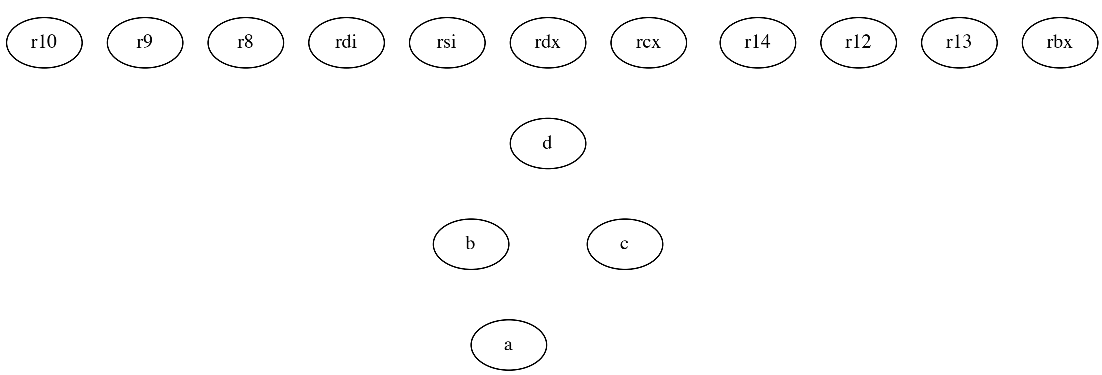
and the callee-saved registers are

```
rsp rbp rbx r12 r13 r14 r15

a : NoneType, b : tuple[int], c : tuple[int], d : tuple[int]

block1:              { r15 d }
    movq %r15, %rdi  { rdi d }
    movq $16, %rsi   { rdi d rsi }
    callq collect    { d }
    jmp block2       { d }

block2:                       { d }
    movq free_ptr(%rip), %r11 { d }
    addq $16, free_ptr(%rip)  { d }
    movq $3, 0(%r11)          { r11 d }
    movq %r11, b              { b d }
    movq b, %r11              { b d }
    movq $0, 8(%r11)          { b d }
    movq $0, a                { b d }
    movq b, c                 { c d }
    cmpq c, d                 { }
    je block7                 { }
    jmp block8                { }
```



9. 12 points Given the following output of Remove Complex Operands, apply the Explicate Control pass to translate the program to $\mathcal{C}_{\mathsf{Fun}}$. You may use concrete or abstract syntax for your answer. Make sure to distinguish regular calls (concrete syntax $\mathit{fun}(\mathit{arg}_1, \ldots, \mathit{arg}_n)$) from tail calls (concrete syntax `tail` $\mathit{fun}(\mathit{arg}_1, \ldots, \mathit{arg}_n)$). A variable inside braces such as {`dub`} represents a `FunRef` AST node.

```
def dub(f:Callable[[int], int], x:int) -> int:
  tmp.0 = f(x)
  return f(tmp.0)

def inc(x:int) -> int:
  return x + 1

def main() -> int:
  fun.1 = {dub}
  fun.2 = {inc}
  tmp.3 = input_int()
  tmp.4 = fun.1(fun.2, tmp.3)
  print(tmp.4)
  return 0
```

10. 12 points Given the following $\mathcal{C}_{\mathsf{Fun}}$ program, apply the Select Instructions pass. A variable inside braces such as {id} represents a `FunRef` AST node.

```
def id(x:int) -> int:
  idstart:
    return x

def main() -> int:
  mainstart:
    fun.0 = {id}
    tmp.1 = fun.0(42)
    print(tmp.1)
    return 0
```

Recall that the following six registers are used for passing arguments to functions.

```
rdi rsi rdx rcx r8 r9
```

11. $\boxed{\text{10 points}}$ Recall that the Limit Functions pass changes all the functions in the program so that they have at most 6 parameters (the number of argument-passing registers), making it easier to implement efficient tail calls. The `limit_type` auxiliary function changes each type annotation in the program as part of the Limit Functions pass. Fill in the blanks in `limit_type`.

```
def limit_type(t):
    match t:
      case TupleType(ts):
        new_ts = [___(a)___ for t in ts]


        return ___(b)___


      case FunctionType(ps, rt):
        new_ps = [limit_type(t) for t in ps]
        new_rt = limit_type(rt)
        n = len(arg_registers)
        if len(new_ps) > n:
            front = new_ps[0 : n-1]
            back = new_ps[n-1 :]
            return ___(c)___


        else:
            return ___(d)___


      case _:
        return ___(e)___
```

12. 12 points Given the following x86 code for a function named `map_vec`, write down the code for its prelude and conclusion.

```
map_vecstart:
    movq      %rdi, -16(%rbp)
    movq      %rsi, -8(%r15)
    movq      -8(%r15), %r11
    movq      8(%r11), %rsi
    movq      %rsi, %rdi
    callq     *-16(%rbp)
    movq      %rax, %rbx
    movq      -8(%r15), %r11
    movq      16(%r11), %rsi
    movq      %rsi, %rdi
    callq     *-16(%rbp)
    movq      %rax, -16(%rbp)
    movq      free_ptr(%rip), %rsi
    movq      %rsi, %rdi
    addq      $24, %rdi
    movq      fromspace_end(%rip), %rsi
    cmpq      %rsi, %rdi
    jl block7
    movq      %r15, %rdi
    movq      $24, %rsi
    callq     collect
    jmp block6
```

```
block6:
    movq      free_ptr(%rip), %r11
    addq      $24, free_ptr(%rip)
    movq      $5, 0(%r11)
    movq      %r11, %rsi
    movq      %rsi, %r11
    movq      %rbx, 8(%r11)
    movq      $0, %rdi
    movq      %rsi, %r11
    movq      -16(%rbp), %rax
    movq      %rax, 16(%r11)
    movq      $0, %rdi
    movq      %rsi, %rax
    jmp map_vecconclusion

block7:
    movq      $0, %rsi
    jmp block6
```