CS410P/510 Programming Language Compilation Practice Midterm

Name: ______________________________

Instructions

- This exam has 6 questions, for a total of 80 points.
- You may spend up to 1 hour, 50 minutes (110 minutes) on the exam.
- The exam is closed-book, closed-notes, except that one 8.5"x11" single-sided sheet of handwritten notes is permitted.
- No computing devices (laptops, tablets, cell phones, etc.) may be used.

Concrete syntax for all the intermediate languages mentioned in the exam can be found on the last two pages.

1. [10 points] Compiler Structure

   The compiler for the language with booleans and conditionals in Chapter 5 involves multiple intermediate and target languages, namely: $\mathcal{L}_{\mathsf{If}}$ (the source language for the whole compiler), $\mathcal{L}_{\mathsf{if}}^{mon}$, $\mathcal{C}_{\mathsf{If}}$, x86$_{\mathsf{If}}^{\mathsf{Var}}$, and x86$_{\mathsf{If}}$ (the target language for the whole compiler).

   The compiler involves multiple passes, which are listed here (with abbreviations) in no particular order:

```
prelude_and_conclusion (P&C)
shrink (S)
select_instructions (SE)
explicate_control (EC)
remove_complex_operands (RCO)
patch_instructions (PI)
assign_homes (AH)
```

   Draw a diagram that shows the order in which the passes actually execute, and indicates which language is the source and target of each pass. (Use the abbreviations to save writing.)

Solution:

$$\begin{tikzcd}
\mathcal{L}_{\mathsf{If}} \arrow[r, bend left, "\mathtt{S}"] & \mathcal{L}_{\mathsf{If}} \arrow[r, bend left, "\mathtt{RCO}"] & \mathcal{L}_{\mathsf{if}}^{mon} \arrow[dll, bend left, "\mathtt{EC}"] & \\
\mathcal{C}_{\mathsf{If}} \arrow[d, bend right, "\mathtt{SE}"] & & & \\
\mathrm{x86}_{\mathsf{If}}^{\mathsf{Var}} \arrow[r, bend right, "\mathtt{AH}"'] & \mathrm{x86}_{\mathsf{If}} \arrow[r, bend left, "\mathtt{PI}"] & \mathrm{x86}_{\mathsf{If}} \arrow[r, bend right, "\mathtt{P\&C}"'] & \mathrm{x86}_{\mathsf{If}}
\end{tikzcd}$$

2. [15 points] Compile the following $\mathcal{L}_{\mathsf{If}}$ program to an equivalent program in the $\mathcal{L}_{\mathsf{if}}^{mon}$ language.

```
a = input_int()
b = 3 + ((- a) - 7)
c = 42 if (b < 10) else (a + input_int())
print(c)
```

Solution:

```
a = input_int()
tmp.0 = -a
tmp.1 = tmp.0 - 7
b = 3 + tmp.1
c = 42 if b < 10 else
    { tmp.2 = input_int()
      produce (a + tmp.2)}
print(c)
```

3. [15 points] Given the following code for the body of an $\text{x86}_{\text{If}}^{\text{Var}}$ program written using symbolic variable names, write down the full assembly code for the x86_{If} program obtained by assigning distinct `%rbp`-relative stack locations (not registers!) to the variables `x`, `t0`, and `t1`, in the style of Chapter 2. Your answer should be in the form of a single `main` function definition, given in the syntax of x86_{If}, i.e. the usual assembler syntax of .s files. Be sure to give the *complete* function code, including entry and exit sequences, and consisting entirely of *legal* instructions.

```
callq _read_int
movq %rax, x
movq $-7, t0
movq t0, t1
addq x, t1
movq t1, %rdi
callq _print_int
```

Solution: One possible solution. The assignment of the three variables to the three stack slots is arbitrary.

```
    .globl main
main:
    pushq %rbp
    movq  %rsp, %rbp
    subq  $32, %rsp          ; rounding up to a multiple of 16
    callq _read_int
    movq %rax, -16(%rbp)     ; x : -16
    movq $-7, -8(%rbp)       ; t0 : -8
    movq -8(%rbp),  %rax
    movq %rax, -24(%rbp)     ; t1: -24
    movq -16((%rbp), %rax
    addq %rax, -24(%rbp)
    movq -24(%rbp), %rdi
    callq _print_int
    movq $0, %rax            ; not essential
    addq $32, %rsp
    popq %rbp
    retq
```

4. [10 points] Recall that numeric comparisons on the X86 are perfomed by setting the condition codes (typically using a `cmpq` instruction) and then testing them using one of the `set`*cc* or `j`*cc* instructions. Our compiler finds it useful to generate both `set`*cc* and `j`*cc* instructions in different situations.

   Illustrate why, by giving a *short* $\mathcal{L}_{\mathsf{If}}$ source program fragment and its translation into $\text{x86}_{\mathsf{If}}^{\mathsf{Var}}$, where the translated program uses both kinds of *cc*-testing instructions.

Solution: We use `set`*cc* to store the result of a comparison into a boolean variable, and `j`*cc* to implement the conditional jump associated with an `if` statement or expression.

Here's one simple example.

```
x = a < 2
if a > 3:
  b = 42
else:
  b = 99
```

Resulting $\text{x86}_{\mathsf{If}}^{\mathsf{Var}}$ code:

```
    cmpq $2,a
    setl %al
    movzbq %al,x
    cmpq $3,a
    jg L2
    movq $42,b
    jmp L3
L2: movq $99,b
L3:
```

5. [15 points] For the following $\mathcal{C}_{\mathsf{If}}$ program, fill in the live-after and live-before sets at each specified point in the program. (Note: although in our compiler we compute liveness information for X86 code, exactly the same ideas can be used to compute liveness for $\mathcal{C}_{\mathsf{If}}$ code.)

```
                             live-before =
start:
  a = 1
                             live-after =
  b = 2
                             live-after =
  t3 = input_int()
                             live-after =
  if t3 == 0: goto block1
  else: goto block2

                             live-before =
block1:
  t2 = a
                             live-after =
  t3 = -t2
                             live-after =
  goto block3

                             live-before =
block2:
  t3 = b
                             live-after =
  t4 = 20
                             live-after =

  goto block3

                             live-before =
block3:
  x = t3
                             live-after =
  t5 = x + 10
                             live-after =
  print(t5)
                             live-after =
  return 0
```

Solution:

```
                              live-before = {}
start:
  a = 1
                              live-after = {a}
  b = 2
                              live-after = {a,b}
  t3 = input_int()
                              live-after = {a,b,t3}
  if t3 == 0: goto block1
  else: goto block2

                              live-before = {a}
block1:
  t2 = a
                              live-after = {t2}
  t3 = -t2
                              live-after = {t3}
  goto block3

                              live-before = {b}
block2:
  t3 = b
                              live-after = {t3}
  t4 = 20
                              live-after = {t3}

  goto block3

                              live-before = {t3}
block3:
  x = t3
                              live-after = {x}
  t5 = x + 10
                              live-after = {t5}
  print(t5)
                              live-after = {}
  return 0
```

6. [15 points] Consider the following results from liveness analysis on a x86$_{\mathsf{Var}}$ program using symbolic variable names, where the live-after set is listed next to each instruction.

```
start:
        callq   read_int    ; %rax
        movq    %rax, x     ; x
        movq    $1, y       ; x,y
        movq    $2, z       ; x,y,z
        movq    y, w        ; x,w,z
        addq    $2, w       ; x,w,z
        movq    z, t        ; x,w,t
        addq    w, t        ; t,x
        movq    t, %rax     ; %rax,x
        addq    x, %rax     ; %rax
        jmp     conclusion
```

(a) Draw the interference graph for the variables `x,y,z,w,t`. (You can ignore `%rax`.)

(b) What is the minimum number of locations (registers or stack slots) needed to hold the five variables in this code?

Solution:

(a) [12 pts] Here are the edge adjacency lists for the graph:

```
t : w x
w : t x z
x : t w y z
y : x z
z : w x y
```

(b) [3 pts] Three locations are necessary and sufficient. Sufficiency is shown by by this assignment: `loc1:t,z; loc2:w,y; loc3:x`. Necessity follows because the graph contains several fully connected subgraphs (*cliques*), namely `t-w-x`, `x-w-z` and `x-z-y`. (A less precise argument is that it follows because there are multiple program points where three variables are simultaneously live. But some of these could conceivably share the same slot if they also share the same value, is in the `movq` special case we studied.)

[This page deliberately left blank.]

Concrete Syntax of Languages

$\mathcal{L}_{\mathsf{If}}$

$$\begin{array}{rcl}
\mathit{cmp} & ::= & \texttt{==} \mid \texttt{!=} \mid \texttt{<} \mid \texttt{<=} \mid \texttt{>} \mid \texttt{>=} \\
\mathit{exp} & ::= & \mathit{int} \mid \mathit{bool} \mid \mathit{var} \\
& \mid & \texttt{input\_int()} \mid \texttt{-}\ \mathit{exp} \mid \texttt{not}\ \mathit{exp} \mid \mathit{exp}\ \texttt{+}\ \mathit{exp} \mid \mathit{exp}\ \texttt{-}\ \mathit{exp} \\
& \mid & \mathit{exp}\ \texttt{and}\ \mathit{exp} \mid \mathit{exp}\ \texttt{or}\ \mathit{exp} \mid \texttt{(}\mathit{exp}\texttt{)} \\
& \mid & \mathit{exp}\ \mathit{cmp}\ \mathit{exp} \mid \mathit{exp}\ \texttt{if}\ \mathit{exp}\ \texttt{else}\ \mathit{exp} \\
\mathit{stmt} & ::= & \texttt{print(}\mathit{exp}\texttt{)} \mid \mathit{exp} \mid \mathit{var}\,\texttt{=}\,\mathit{exp} \mid \texttt{if}\ \mathit{exp}\texttt{:}\ \mathit{stmt}^{+}\ \texttt{else:}\ \mathit{stmt}^{+} \\
\mathcal{L}_{\mathsf{If}} & ::= & \mathit{stmt}^{*}
\end{array}$$

$\mathcal{L}_{\mathsf{if}}^{mon}$

$$\begin{array}{rcl}
\mathit{atm} & ::= & \mathit{int} \mid \mathit{bool} \mid \mathit{var} \\
\mathit{cmp} & ::= & \texttt{==} \mid \texttt{!=} \mid \texttt{<} \mid \texttt{<=} \mid \texttt{>} \mid \texttt{>=} \\
\mathit{exp} & ::= & \mathit{atm} \mid \texttt{input\_int()} \mid \texttt{-}\ \mathit{atm} \mid \texttt{not}\ \mathit{atm} \mid \mathit{atm}\ \texttt{+}\ \mathit{atm} \mid \mathit{atm}\ \texttt{-}\ \mathit{atm} \\
& \mid & \mathit{atm}\ \mathit{cmp}\ \mathit{atm} \mid \mathit{exp}\ \texttt{if}\ \mathit{exp}\ \texttt{else}\ \mathit{exp} \mid \{\mathit{stmt}^{*}\ \texttt{produce(}\mathit{exp}\texttt{)}\} \\
\mathit{stmt} & ::= & \texttt{print(}\mathit{atm}\texttt{)} \mid \mathit{exp} \mid \mathit{var}\,\texttt{=}\,\mathit{exp} \mid \texttt{if}\ \mathit{exp}\texttt{:}\ \mathit{stmt}^{+}\ \texttt{else:}\ \mathit{stmt}^{+} \\
\mathcal{L}_{\mathsf{if}}^{mon} & ::= & \mathit{stmt}^{*}
\end{array}$$

Note: the concrete expression $\{\mathit{stmt}^{*}\ \texttt{produce(}\mathit{exp}\texttt{)}\}$ corresponds to the AST form $\texttt{Begin(}\mathit{stmt}^{*}\texttt{,}\ \mathit{exp}\texttt{)}$.

$\mathcal{C}_{\mathsf{If}}$

$$\begin{array}{rcl}
\mathit{atm} & ::= & \mathit{int} \mid \mathit{bool} \mid \mathit{var} \\
\mathit{cmp} & ::= & \texttt{==} \mid \texttt{!=} \mid \texttt{<} \mid \texttt{<=} \mid \texttt{>} \mid \texttt{>=} \\
\mathit{exp} & ::= & \mathit{atm} \mid \texttt{input\_int()} \mid \texttt{-}\ \mathit{atm} \mid \texttt{not}\ \mathit{atm} \mid \mathit{atm}\ \texttt{+}\ \mathit{atm} \mid \mathit{atm}\ \texttt{-}\ \mathit{atm} \\
& \mid & \mathit{atm}\ \mathit{cmp}\ \mathit{atm} \\
\mathit{stmt} & ::= & \texttt{print(}\mathit{atm}\texttt{)} \mid \mathit{exp} \mid \mathit{var}\ \texttt{=}\ \mathit{exp} \\
\mathit{tail} & ::= & \texttt{return}\ \mathit{exp} \mid \texttt{goto}\ \mathit{label} \mid \texttt{if}\ \mathit{atm}\ \mathit{cmp}\ \mathit{atm}\texttt{:}\ \texttt{goto}\ \mathit{label}\ \texttt{else:}\ \texttt{goto}\ \mathit{label} \\
\mathcal{C}_{\mathsf{If}} & ::= & (\mathit{label}\texttt{:}\ \mathit{stmt}^{*}\ \mathit{tail})\ldots
\end{array}$$

$\text{x86}^{\mathsf{Var}}_{\mathsf{If}}$

$$
\begin{array}{rcl}
\mathit{reg} & ::= & \texttt{rsp} \mid \texttt{rbp} \mid \texttt{rax} \mid \texttt{rbx} \mid \texttt{rcx} \mid \texttt{rdx} \mid \texttt{rsi} \mid \texttt{rdi} \mid \\
 & & \texttt{r8} \mid \texttt{r9} \mid \texttt{r10} \mid \texttt{r11} \mid \texttt{r12} \mid \texttt{r13} \mid \texttt{r14} \mid \texttt{r15} \\
\mathit{bytereg} & ::= & \texttt{ah} \mid \texttt{al} \mid \texttt{bh} \mid \texttt{bl} \mid \texttt{ch} \mid \texttt{cl} \mid \texttt{dh} \mid \texttt{dl} \\
\mathit{arg} & ::= & \texttt{\$}\mathit{int} \mid \texttt{\%}\mathit{reg} \mid \texttt{\%}\mathit{bytereg} \mid \mathit{int}\texttt{(\%}\mathit{reg}\texttt{)} \mid \mathit{var} \\
\mathit{cc} & ::= & \texttt{e} \mid \texttt{ne} \mid \texttt{l} \mid \texttt{le} \mid \texttt{g} \mid \texttt{ge} \\
\mathit{instr} & ::= & \texttt{addq}\ \mathit{arg}\texttt{,}\mathit{arg} \mid \texttt{subq}\ \mathit{arg}\texttt{,}\mathit{arg} \mid \texttt{negq}\ \mathit{arg} \mid \texttt{movq}\ \mathit{arg}\texttt{,}\mathit{arg} \\
 & \mid & \texttt{pushq}\ \mathit{arg} \mid \texttt{popq}\ \mathit{arg} \mid \texttt{callq}\ \mathit{label} \mid \texttt{retq} \\
 & \mid & \texttt{xorq}\ \mathit{arg}\texttt{,}\ \mathit{arg} \mid \texttt{cmpq}\ \mathit{arg}\texttt{,}\ \mathit{arg} \mid \texttt{set}\mathit{cc}\ \mathit{arg} \mid \texttt{movzbq}\ \mathit{arg}\texttt{,}\ \mathit{arg} \\
 & \mid & \texttt{jmp}\ \mathit{label} \mid \texttt{j}\mathit{cc}\ \mathit{label} \mid \mathit{label}\texttt{:}\ \mathit{instr} \\
\text{x86}^{\mathsf{Var}}_{\mathsf{If}} & ::= & \texttt{.globl main} \\
 & & \texttt{main:}\ \mathit{instr}\ldots
\end{array}
$$

Note: this is the same as x86_{If}, below, except that *var* is allowed as an *arg*.

x86_{If}

$$
\begin{array}{rcl}
\mathit{reg} & ::= & \texttt{rsp} \mid \texttt{rbp} \mid \texttt{rax} \mid \texttt{rbx} \mid \texttt{rcx} \mid \texttt{rdx} \mid \texttt{rsi} \mid \texttt{rdi} \mid \\
 & & \texttt{r8} \mid \texttt{r9} \mid \texttt{r10} \mid \texttt{r11} \mid \texttt{r12} \mid \texttt{r13} \mid \texttt{r14} \mid \texttt{r15} \\
\mathit{bytereg} & ::= & \texttt{ah} \mid \texttt{al} \mid \texttt{bh} \mid \texttt{bl} \mid \texttt{ch} \mid \texttt{cl} \mid \texttt{dh} \mid \texttt{dl} \\
\mathit{arg} & ::= & \texttt{\$}\mathit{int} \mid \texttt{\%}\mathit{reg} \mid \texttt{\%}\mathit{bytereg} \mid \mathit{int}\texttt{(\%}\mathit{reg}\texttt{)} \\
\mathit{cc} & ::= & \texttt{e} \mid \texttt{ne} \mid \texttt{l} \mid \texttt{le} \mid \texttt{g} \mid \texttt{ge} \\
\mathit{instr} & ::= & \texttt{addq}\ \mathit{arg}\texttt{,}\mathit{arg} \mid \texttt{subq}\ \mathit{arg}\texttt{,}\mathit{arg} \mid \texttt{negq}\ \mathit{arg} \mid \texttt{movq}\ \mathit{arg}\texttt{,}\mathit{arg} \\
 & \mid & \texttt{pushq}\ \mathit{arg} \mid \texttt{popq}\ \mathit{arg} \mid \texttt{callq}\ \mathit{label} \mid \texttt{retq} \\
 & \mid & \texttt{xorq}\ \mathit{arg}\texttt{,}\ \mathit{arg} \mid \texttt{cmpq}\ \mathit{arg}\texttt{,}\ \mathit{arg} \mid \texttt{set}\mathit{cc}\ \mathit{arg} \mid \texttt{movzbq}\ \mathit{arg}\texttt{,}\ \mathit{arg} \\
 & \mid & \texttt{jmp}\ \mathit{label} \mid \texttt{j}\mathit{cc}\ \mathit{label} \mid \mathit{label}\texttt{:}\ \mathit{instr} \\
\text{x86}_{\mathsf{If}} & ::= & \texttt{.globl main} \\
 & & \texttt{main:}\ \mathit{instr}\ldots
\end{array}
$$