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: $\mathcal{L}_{lf}$ (the source language for the whole compiler), $\mathcal{L}_{lf}^{mon}$, $\mathcal{C}_{lf}$, x86$_{If}$, and x86$_{If}$ (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.)
2. [15 points] Compile the following $L_\#$ program to an equivalent program in the $L_\#^{mon}$ language.

```python
a = input_int()
b = 3 + ((- a) - 7)
c = 42 if (b < 10) else (a + input_int())
print(c)
```
3. [15 points] Given the following code for the body of an x86\textsuperscript{Var} program written using symbolic variable names, write down the full assembly code for the x86\textsuperscript{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\textsuperscript{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.

```assembly
    callq _read_int
    movq %rax, x
    movq $-7, t0
    movq t0, t1
    addq x, t1
    movq t1, %rdi
    callq _print_int
```
4. [10 points] Recall that numeric comparisons on the X86 are performed by setting the condition codes (typically using a `cmpq` instruction) and then testing them using one of the `setcc` or `jcc` instructions. Our compiler finds it useful to generate both `setcc` and `jcc` instructions in different situations.

Illustrate why, by giving a short $L_{if}$ source program fragment and its translation into $x86_{if}^{Var}$, where the translated program uses both kinds of `cc-testing` instructions.
5. [15 points] For the following C\textsubscript{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 C\textsubscript{if} code.)

```c
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
```
6. [15 points] Consider the following results from liveness analysis on a x86 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?
[This page deliberately left blank.]
Concrete Syntax of Languages

\( \mathcal{L}_f \)

\[ 
\begin{align*} 
\text{cmp} & ::= \text{==} | \text{!=} | \text{<} | \text{<=} | \text{>} | \text{>=} \\
\text{exp} & ::= \text{int} | \text{bool} | \text{var} \\
& | \text{input_int()} | - \text{exp} | \text{not} \text{exp} | \text{exp} + \text{exp} | \text{exp} - \text{exp} \\
& | \text{exp and} \text{exp} | \text{exp or} \text{exp} | (\text{exp}) \\
& | \text{exp cmp exp} | \text{exp if exp else exp} \\
\text{stmt} & ::= \text{print(exp)} | \text{exp} | \text{var} = \text{exp} | \text{if exp: stmt}^+ \text{ else: stmt}^+ \\
\mathcal{L}_f & ::= \text{stmt}^* 
\end{align*} \]

\( \mathcal{L}^{\text{mon}}_f \)

\[ 
\begin{align*} 
\text{atm} & ::= \text{int} | \text{bool} | \text{var} \\
\text{cmp} & ::= \text{==} | \text{!=} | \text{<} | \text{<=} | \text{>} | \text{>=} \\
\text{exp} & ::= \text{atm} | \text{input_int()} | - \text{atm} | \text{not atm} | \text{atm + atm} | \text{atm - atm} \\
& | \text{atm cmp atm} | \text{exp if exp else exp} | \{\text{stmt}^* \text{ produce(exp)}\} \\
\text{stmt} & ::= \text{print(atm)} | \text{exp} | \text{var} = \text{exp} | \text{if exp: stmt}^+ \text{ else: stmt}^+ \\
\mathcal{L}^{\text{mon}}_f & ::= \text{stmt}^* 
\end{align*} \]

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

\( \mathcal{C}_f \)

\[ 
\begin{align*} 
\text{atm} & ::= \text{int} | \text{bool} | \text{var} \\
\text{cmp} & ::= \text{==} | \text{!=} | \text{<} | \text{<=} | \text{>} | \text{>=} \\
\text{exp} & ::= \text{atm} | \text{input_int()} | - \text{atm} | \text{not atm} | \text{atm + atm} | \text{atm - atm} \\
& | \text{atm cmp atm} \\
\text{stmt} & ::= \text{print(atm)} | \text{exp} | \text{var} = \text{exp} \\
\text{tail} & ::= \text{return exp} | \text{goto label} | \text{if atm cmp atm: goto label else: goto label} \\
\mathcal{C}_f & ::= (\text{label: stmt}^* \text{ tail}) \ldots 
\end{align*} \]
x86\Var

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

Note: this is the same as x86\If, below, except that \text{var} is allowed as an \text{arg}.

x86\If

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