The exam is open book/notes/laptop. We do not guarantee power or Internet access, however.

Answer all 5 questions on the exam paper itself.

Write your name here: ______________________________________________________

I acknowledge and accept the honor code.

(signed) ______________________________________

<table>
<thead>
<tr>
<th>Question</th>
<th>Max</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Question 1 (25 Points)
As you may have noticed in the Joeq homework assignment, the control flow graph represented in Joeq does not include edges for exceptions. In fact, Joeq uses a factored control flow graph, where instructions that can throw exceptions do not necessarily end a basic block. These basic blocks, where control flow can only enter at the start but can exit in the middle, are a form of extended basic blocks. Each extended basic block includes one or more exception edges, which point to any exception handlers that may catch an exception thrown during that basic block. An exception edge may also point to the exit node if a thrown exception may not be caught.

Using a factored control flow graph can be more efficient because it reduces the number of nodes and edges in the control flow graph, and it can improve the effectiveness of some local analyses that operate within a basic block. However, it complicates global data flow analysis. The next few questions explore different alternatives of dealing with global data flow analysis on a factored control flow graph.

Technique 1: Modify the GEN and KILL set of each extended basic block to take into account exceptions.

![Figure 1: Factored Control Flow Graph versus Normal Control Flow Graph](image)

Consider the factored control flow graph in Figure 1. The method call instructions \(d_3\) and \(d_5\) can potentially throw exceptions. Write GEN and KILL sets for this extended basic block that will lead to a correct (but not necessarily equivalent to the meet-over-paths) solution to the reaching definitions problem.

We are merging multiple paths into a single transfer function, so to get a correct answer we need to union the GEN set for each path and intersect the KILL sets.

**GEN:** \(d_1, d_2, d_3, d_4, d_5\)

**KILL:** \(\{\}\) (plus any other defs of \(x,y,z\))
Will using these GEN and KILL sets lead to the same result as the meet-over-paths solution on the complete control flow graph, where all possible exceptions are explicitly represented in the graph? (Ignore the additional temporary variables present in the complete control flow graph.)

YES  NO

Technique 2: Define two GEN and KILL sets for each extended basic block (normal and exceptional).

Another technique is to use two different GEN and KILL sets for each extended basic block - one for normal control flow and one for any exceptional control flow. On the normal control flow edges, we use the “normal” GEN and KILL functions. On exceptional control flow edges, we use the “exception” GEN\(_e\) and KILL\(_e\) functions. We then compute the minimum fixed point solution.

![Figure 2: Extended basic block](image)

Consider the extended basic block in Figure 2. Define the “normal” GEN and KILL functions and “exception” GEN and KILL functions that will lead to a correct (but not necessarily ideal) solution to the reaching definitions problem.

For the exception exit, we need to combine the transfer functions for both possible exception exits (at d3 and d5).

Normal exit:
- GEN: \(d3, d4, d5\)
- KILL: \(d1, d2\) (plus any other defs of x,y,z)

Exception exit:
- GEN\(_e\): \(d1, d2, d3, d4\)
- KILL\(_e\): \(d5\) (plus any other defs of x,y,z)

For a general dataflow problem, when will using two separate GEN and KILL sets per extended basic block lead to the meet-over-paths solution on the complete control flow graph, where all possible exceptions are explicitly represented in the graph?
If *sometimes*, under what conditions will this technique lead to the equivalent of the meet-over-paths solution on the complete control flow graph?

*When there is only one exception (or when the transfer function at the point of each exception are equivalent), and the dataflow problem is distributive.*

Technique 3: Define different GEN and KILL sets for each instruction that can throw an exception.

A third technique is to use multiple GEN and KILL sets per extended basic block - a GEN/KILL per instruction that can throw an exception, in addition to a GEN/KILL for normal execution that does not throw any exceptions. We then calculate the minimum fixed point solution.

Figure 3: Extended basic block

For the example in Figure 3, you would define three GEN/KILL sets - one for the normal exit, one for exiting due to an exception at \(d_3\), and one for exiting due to an exception at \(d_5\). Define these functions for the reaching definition problem on the extended basic block in Figure 3.

*Each exception exit is treated separately, so there is no added imprecision.*

Normal exit:
\[
\text{GEN: } d_3, d_4, d_5 \quad \text{KILL: } d_1, d_2 \text{ (plus any other defs of } x,y,z)\
\]

Exit due to exception thrown by the method call at \(d_3\):
\[
\begin{align*}
\text{GEN}_{d_3}: & \quad d_1, d_2 \\
\text{KILL}_{d_3}: & \quad d_3, d_5 \text{ (plus any other defs of } x,y)\
\end{align*}
\]

Exit due to exception thrown by the method call at \(d_5\):
\[
\begin{align*}
\text{GEN}_{d_5}: & \quad d_1, d_3, d_4 \\
\text{KILL}_{d_5}: & \quad d_2, d_5 \text{ (plus any other defs of } x,y,z)\
\end{align*}
\]
For a general dataflow problem, when will using separate GEN and KILL sets for each instruction that can throw an exception lead to the meet-over-paths solution on the complete control flow graph?

Always    Sometimes    Never

If sometimes, under what conditions will this technique lead to the equivalent of the meet-over-paths solution on the complete control flow graph?

When the dataflow framework is distributive.

Note: Because the question asked specifically about “meet-over-paths” solution as opposed to MFP, the answer is “Sometimes” because MOP=MFP only if the dataflow framework is distributive. Very few students picked up on this. If you didn’t notice the MOP vs MFP distinction in the question and answered “Always”, we only deducted one point.

Question 2 (15 Points)
Here is a sequence of three-address statements. You may think of a through h as variables or pseudo-registers. Assume that only e is live on exit from the block, and only b and c are defined before the block.

\[
\begin{align*}
    a &= b + c \\
    d &= a + b \\
    e &= c + d \\
    f &= d + e \\
    g &= d + f \\
    h &= f + g
\end{align*}
\]

Here are the nodes of the interference graph for this block. Draw the edges below.

\[
\begin{align*}
    \text{a} &\longrightarrow \text{b} \\
    \text{c} &\longrightarrow \text{d} \\
    \text{e} &\longrightarrow \text{f} \\
    \text{g} &\longrightarrow \text{h}
\end{align*}
\]
Question 3 (20 Points)
For the flow graph below, with entry node 1:

a) What is the dominance relation? For each of the five nodes, list all the nodes that node dominates:

Node 1 dominates: __1, 2, 3, 4, 5____________________
Node 2 dominates: __2________________________________
Node 3 dominates: __3________________________________
Node 4 dominates: __4, 5_______________________________
Node 5 dominates: __5_______________________________

Codes used in grading:
M1: You didn’t include each edge in its own set of dominated nodes (ie 1 dominates 1, 2 dominates 2, etc)

b) Which are the back edges? (use x → y to represent the edge from node x to node y)

_____3 → 1_______________________________________

c) Draw, in the space below, a depth-first spanning tree for this flow graph, choosing the lowest-numbered unvisited node whenever there is a choice of which node to visit next. For each edge that is NOT a tree edge, indicate (by labeling the edge) whether it is forward, retreating, or cross.
The values in this framework are sets of variables of the flow graph. That is, IN[B] and OUT[B] for each block is a set of variables. The goal is to make the value at any point p be the set of variables x such that along any path from the entry to p we encounter at least one certain definition of x (i.e., a definition is "certain" if it definitely assigns a value to x, e.g. x = y+z).

a) What direction do you need for this framework? ____Forward____  (2 points)

b) What is the meet operator? ____Intersection____  (2 points)

c) The transfer functions use "Gen" and "Kill" sets for each block B. Some possible definitions for these sets are:
1. The empty set.
2. The set of variables that are certainly defined in B.
3. The set of variables that may be defined in B (a variable x "may be defined" in B if there is a statement in B that either certainly defines it or might define it depending on run-time values, e.g., an assignment *p = y, where pointer p might point to x).
4. The set of variables that are certainly NOT defined in B.
5. The set of variables that may NOT be defined in B.

Choose one of these definitions for Gen and one for Kill. Write the numbers of your choices here: Gen __2__ Kill __1__ (2 points)

d) In terms of your Gen and Kill from part (c), write the transfer function for a block B:
\[ \text{OUT}[B] = \text{IN}[B] \cup \text{Gen}[B] \] (2 points)

e) Suppose block B consists of only the statement a=x. Consider the situations where x is or is not in IN[B] and situations where x is or is not in OUT[B]. In what situations should you conclude that the programmer may have made an error and report "variable x may be undefined at statement a=x"?

If x is not IN[B] (4 points)

Under what circumstances can you conclude that the programmer definitely has made an error and report "you have an undefined variable x at statement a=x"?

Never (2 points)

Question 5 (25 Points)
Assume the following machine model:
- The instruction format is of the form, Opcode Dest, Src1, Src2, ...
- ST = 2 cycles. Other instructions = 1 cycle.
- The destination register is updated in the edge between any two clock cycles

Machine can execute one LD or ST and one arithmetic operation in a clock cycle.

```
1: ADDi r2, r1, 4
2: ADDi sp, sp, 12
3: ST a, r0
4: LD r3, -4(sp)
5: LD r4, -8(sp)
6: ADDi sp, sp, 8
```
7: ST 0(sp), r2
8: LD r5, a
9: ADDi r4, r4, 1

(a) Draw, in the space below, the data dependence graph for the code above. Label each edge with its minimum delay.

<table>
<thead>
<tr>
<th>Edge</th>
<th>Delay</th>
<th>Edge</th>
<th>Delay</th>
<th>Edge</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-7</td>
<td>1</td>
<td>3-8</td>
<td>2</td>
<td>3-4</td>
<td>2</td>
</tr>
<tr>
<td>2-4</td>
<td>1</td>
<td>4-6</td>
<td>0 or 1</td>
<td>3-5</td>
<td>2</td>
</tr>
<tr>
<td>2-5</td>
<td>1</td>
<td>5-6</td>
<td>0 or 1</td>
<td>3-7</td>
<td>2</td>
</tr>
<tr>
<td>2-6</td>
<td>1</td>
<td>5-9</td>
<td>1</td>
<td>4-7</td>
<td>1</td>
</tr>
<tr>
<td>2-7</td>
<td>1</td>
<td>6-7</td>
<td>1</td>
<td>5-7</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7-8</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) Perform list scheduling and derive the schedule. Show the final scheduled code below.

One of the possible scheduling

<table>
<thead>
<tr>
<th>ALU</th>
<th>LD/ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

(c) How many cycles does your schedule take? _______ 6 _______