1. (a) **Value:**

<table>
<thead>
<tr>
<th>Address</th>
<th>Line Number (Method)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8020</td>
<td>(line 3) (value)</td>
</tr>
<tr>
<td>0x8028</td>
<td>(line 4) (set)</td>
</tr>
</tbody>
</table>

**MulOp:**

<table>
<thead>
<tr>
<th>Address</th>
<th>Line Number (Method)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8030</td>
<td>(line 3) (value)</td>
</tr>
<tr>
<td>0x8038</td>
<td>(line 4) (set)</td>
</tr>
<tr>
<td>0x8040</td>
<td>(line 8) (operate)</td>
</tr>
</tbody>
</table>

**AddOp:**

<table>
<thead>
<tr>
<th>Address</th>
<th>Line Number (Method)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048</td>
<td>(line 3) (value)</td>
</tr>
<tr>
<td>0x8050</td>
<td>(line 4) (set)</td>
</tr>
<tr>
<td>0x8058</td>
<td>(line 11) (operate)</td>
</tr>
</tbody>
</table>
(b) Heap layout:

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>⟨object Main⟩ + 0x0000</td>
<td>5</td>
<td>(class tag)</td>
</tr>
<tr>
<td>⟨object Main⟩ + 0x0008</td>
<td>4</td>
<td>(object size)</td>
</tr>
<tr>
<td>⟨object Main⟩ + 0x0010</td>
<td>0x8800</td>
<td>(dispatch ptr)</td>
</tr>
<tr>
<td>⟨object Main⟩ + 0x0018</td>
<td>⟨object Stack⟩</td>
<td>(stack)</td>
</tr>
<tr>
<td>⟨object Stack⟩ + 0x0000</td>
<td>4</td>
<td>(class tag)</td>
</tr>
<tr>
<td>⟨object Stack⟩ + 0x0008</td>
<td>5</td>
<td>(object size)</td>
</tr>
<tr>
<td>⟨object Stack⟩ + 0x0010</td>
<td>0x8000</td>
<td>(dispatch ptr)</td>
</tr>
<tr>
<td>⟨object Stack⟩ + 0x0018</td>
<td>⟨object Value⟩</td>
<td>(head)</td>
</tr>
<tr>
<td>⟨object Stack⟩ + 0x0020</td>
<td>void</td>
<td>(tail)</td>
</tr>
<tr>
<td>⟨object Value⟩ + 0x0000</td>
<td>1</td>
<td>(class tag)</td>
</tr>
<tr>
<td>⟨object Value⟩ + 0x0008</td>
<td>4</td>
<td>(object size)</td>
</tr>
<tr>
<td>⟨object Value⟩ + 0x0010</td>
<td>0x8020</td>
<td>(dispatch ptr)</td>
</tr>
<tr>
<td>⟨object Value⟩ + 0x0018</td>
<td>2</td>
<td>(value)</td>
</tr>
</tbody>
</table>

**IO** may be included as well, without penalty. (Note that we don’t really know the layout of IO, as it is not specified, so it was not intended to be included.)
(c) Stack layout:

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7777fff8</td>
<td>0x7fffffff8</td>
<td>(saved frame pointer)</td>
</tr>
<tr>
<td>0x7777fff0</td>
<td>(object <code>Main</code>)</td>
<td>(argument 0 of <code>main</code>)</td>
</tr>
<tr>
<td>0x7777ffe8</td>
<td>0x2000</td>
<td>(return address of <code>main</code>)</td>
</tr>
<tr>
<td>0x7777ffe0</td>
<td>(object <code>IO</code>)</td>
<td>(local variable <code>io</code>)</td>
</tr>
<tr>
<td>0x7777ffd8</td>
<td>5</td>
<td>(local variable <code>num</code>)</td>
</tr>
<tr>
<td>0x7777ffd0</td>
<td>0x7777ffe8</td>
<td>(saved frame pointer)</td>
</tr>
<tr>
<td>0x7777ffc8</td>
<td>(object <code>Main</code>)</td>
<td>(argument 0 of <code>reduce</code>)</td>
</tr>
<tr>
<td>0x7777ffc0</td>
<td>(line 54)</td>
<td>(return address of <code>reduce</code>)</td>
</tr>
<tr>
<td>0x7777ffb8</td>
<td>(object <code>AddOp</code>)</td>
<td>(local variable <code>x</code>)</td>
</tr>
<tr>
<td>0x7777ffb0</td>
<td>(object <code>AddOp</code>)</td>
<td>(local variable <code>op</code>)</td>
</tr>
<tr>
<td>0x7777fa8</td>
<td>(object <code>Value</code>)</td>
<td>(local variable <code>temp</code>)</td>
</tr>
<tr>
<td>0x7777fa0</td>
<td>10</td>
<td>(local variable <code>lhs</code>)</td>
</tr>
<tr>
<td>0x7777ff98</td>
<td>1</td>
<td>(local variable <code>rhs</code>)</td>
</tr>
<tr>
<td>0x7777ff90</td>
<td>0x7777ffc0</td>
<td>(saved frame pointer)</td>
</tr>
<tr>
<td>0x7777ff88</td>
<td>1</td>
<td>(argument 2 of <code>operate</code>)</td>
</tr>
<tr>
<td>0x7777ff80</td>
<td>10</td>
<td>(argument 1 of <code>operate</code>)</td>
</tr>
<tr>
<td>0x7777ff78</td>
<td>(object <code>AddOp</code>)</td>
<td>(argument 0 of <code>operate</code>)</td>
</tr>
<tr>
<td>0x7777ff70</td>
<td>(line 35)</td>
<td>(return address of <code>operate</code>)</td>
</tr>
</tbody>
</table>

Full credit should be awarded if the frame pointer addresses are off by 8 (matching the behavior of PA4’s stack.)
2. (a)

\[
\begin{align*}
\frac{so, S_1, E \vdash e_1 \rightarrow v_1, S_2 \quad v_1 \neq \bot}{so, S_1, E \vdash \text{try } e_1 \text{ catch } e_2 \quad \text{yrt} \rightarrow v_1, S_2} \quad \text{[Try-OK]} \\
\frac{so, S_1, E \vdash e_1 \rightarrow \bot, S_2 \quad so, S_2, E \vdash e_2 \rightarrow v_2, S_3}{so, S_1, E \vdash \text{try } e_1 \text{ catch } e_2 \quad \text{yrt} \rightarrow v_2, S_3} \quad \text{[Try-Catch]} \\
\frac{so, S_1, E \vdash \text{throw} \rightarrow \bot, S_1}{so, S_1, E \vdash e_1 + e_2 \rightarrow \text{Int}(i_1 + i_2), S_3} \quad \text{[Plus-OK]} \\
\frac{so, S_1, E \vdash e_1 \rightarrow \bot, S_2}{so, S_1, E \vdash e_1 + e_2 \rightarrow \bot, S_2} \quad \text{[Plus-Fail1]} \\
\frac{so, S_1, E \vdash e_1 \rightarrow v_1, S_2 \quad so, S_2, E \vdash e_2 \rightarrow \bot, S_3 \quad v_1 \neq \bot}{so, S_1, E \vdash e_1 + e_2 \rightarrow \bot, S_3} \quad \text{[Plus-Fail2]} \\
\end{align*}
\]

Note: the rules [Plus-Fail1] and [Plus-Fail2] may be have a more general condition
than \( so, S_1, E \vdash [e_1 / e_2] \rightarrow \bot, S_2 \), such as \( so, S_1, E \vdash [e_1 / e_2] \rightarrow v, S_2 \) and \( v \neq \text{Int}(i_1) \). This handles non-integer arguments to \(+\) as errors that can be caught. Both solutions should be accepted for full credit.
(b)

$\vdash 2 \mapsto \text{Int}(2), S \quad \text{[Int]} \quad \vdash \text{throw} \mapsto \bot, S \quad \text{[Throw]}

$\vdash 2 + \text{throw} \mapsto \bot, S \quad \text{[Plus-Fail2]} \quad \vdash 3 \mapsto \text{Int}(3), S \quad \text{[Int]}

$\vdash \text{try } 2 + \text{throw catch } 3 \text{ yrt} \mapsto \text{Int}(3), S \quad \text{[Try-Catch]}$
(c) Simplicio’s scheme can cause stack misalignment when jumping to the catch handler. For example, the following code prints the wrong value:

```plaintext
1 class Main inherits IO {
2     main() : Int {
3         out_int(1 + try 2 + throw catch 3 yrt)
4     }
5 }
```

This generates the following pseudo-assembly:

```plaintext
1 ; prolog...
2 push 1
3 push 2
4 jmp catch
5 pop r1
6 pop r2
7 add r1, r2
8 push r1
9 jmp tryend
10 catch:
11   push 3
12 tryend:
13   pop r1
14   pop r2
15   add r1, r2
16   push r1
17   pop r1
18 ; call out_int(r1)
19 ; epilog...
```

The catch handler only expects a stack depth of 1, since we are evaluating $1 + \ldots$, but when evaluating the `throw` expression, we have pushed another value (2) onto the stack, since we were evaluating $2 + \ldots$. Then the code following the try expression will pop the wrong value (2, instead of 1) and will wrongly produce $2 + 3 = 5$ instead of the correct result $1 + 3 = 4$.

Also, code that does not use a frame pointer can cause more serious issues, as this causes the entire activation record to be misaligned. For example, in the above code, there is an extra value (1) on the stack, which displaces the return value and likely causes the program to crash. This is avoided when there is a frame pointer, as resetting the stack pointer to the frame pointer resolves this misalignment.

(Solutions that identify this problem without stating the additional assumption that a frame pointer is not used will receive partial credit.)

Solutions that correctly identify any mutated global state that affects execution of the program should receive full credit, provided they state the conditions that would lead to such state being modified and state why this needs to be maintained by the compiler. Solutions that do not state one or more of these should receive partial credit.
Note that in the absence of any global state, Simplicio’s scheme is actually completely valid! This is similar to `setjmp/longjmp`, which is a common error handling scheme in C. It also resembles stack unwinding, but the unwind target can be determined statically as this is intraprocedural.
3. (a) Optimized CFG:
(b) Optimized CFG:

```
Entry(a, b)

x := a
y := a * 2
z := a - a
w := a + z
v := b
if a > b goto B2
else goto B3

x := a + 2
h := x
goto B4

B2

h := a
goto B4

B3

u := x + b
h := x
v := v + 2
if v < 10 goto B4
else goto B5

B4

f := x * 32
g := v - b
return (f, g)

B5

Exit
```
4. (a) Live variables:

ENTRY$(a, b)$

- $x := a$ Live: $a, b$
- $y := x \cdot 2$ Live: $a, b, x$
- $z := x - a$ Live: $b, x, z$
- $w := x + z$ Live: $b, x$
- $v := b$ Live: $b, v, x$
- if $x > v$ goto B2
- else goto B3 Live: $b, v, x$

B2

- $x := x + 2$ Live: $b, v, x$
- $h := x$ Live: $b, h, v, x$
- goto B4 Live: $b, h, v, x$

B3

- $h := x$ Live: $b, h, v, x$
- goto B4 Live: $b, h, v, x$

B4

- $u := h + b$ Live: $b, h, v, x$
- $h := x$ Live: $b, v, x$
- $v := v + 2$ Live: $b, h, v, x$
- if $v < 10$ goto B4
- else goto Live: $b, h, v, x$

B5

- $f := h \cdot 32$ Live: $b, h, v$
- $g := v - b$ Live: $f, g$
- return $(f, g)$ Live: 

Exit
(b) Interference graph:

```
(a - b;
a - x;
b - h;
b - v;
b - z;
b - x;
b - f;
f - g;
f - v;
h - v;
h - x;
v - x;
x - z;
```

extra:
```
a - y;
b - u;
b - w;
b - y;
u - v;
u - x;
w - x;
x - y;
```

The dashed edges arise from extending the live range of variables to include the point immediately after they are written. This is not necessary for full credit. To see why this is necessary, consider the following code:
This arises by assigning registers $x \mapsto r1, y \mapsto r2, z \mapsto r1$. Note that this is not a valid assignment, as the assignment to $r1$ overwrites the existing value of $x$, which is still live, even though $z$ is not live. Dead code elimination can often eliminate such variables, but it is not valid to do so when instructions have side effects. For example, if `something` is a memory-mapped IO address, such as receiving data from an external device, then it would not be valid to remove the load.
Any valid coloring is accepted for full credit.