L17 and L18 worksheet

L17 Review- Rule scheduling
Conflict Matrix (CM)

BSV compiler generates the pairwise conflict information

<table>
<thead>
<tr>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
</tr>
</thead>
</table>
| rule ra;
  x <= x+1;
endrule
rule rb;
  y <= y+2;
endrule | rule ra;
  x <= y+1;
endrule
rule rb;
  y <= x+2;
endrule | rule ra;
  x <= y+1;
endrule
rule rb;
  y <= y+2;
endrule |
Conflict Matrix (CM)

BSV compiler generates the pairwise conflict information

Example 1

\begin{verbatim}
rule ra;
  x <= x+1;
endrule
rule rb;
  y <= y+2;
endrule
\end{verbatim}

Example 2

\begin{verbatim}
rule ra;
  x <= y+1;
endrule
rule rb;
  y <= x+2;
endrule
\end{verbatim}

Example 3

\begin{verbatim}
rule ra;
  x <= y+1;
endrule
rule rb;
  y <= y+2;
endrule
\end{verbatim}

\begin{tabular}{|c|c|c|}
  \hline
  & ra & rb \\
  \hline
  ra & C & CF \\
  rb & CF & C \\
  \hline
\end{tabular}
Conflict Matrix (CM)

BSV compiler generates the pairwise conflict information

**Example 1**

```
rule ra;
  x <= x+1;
endrule
rule rb;
  y <= y+2;
endrule
```

**Example 2**

```
rule ra;
  x <= y+1;
endrule
rule rb;
  y <= x+2;
endrule
```

**Example 3**

```
rule ra;
  x <= y+1;
endrule
rule rb;
  y <= y+2;
endrule
```

<table>
<thead>
<tr>
<th></th>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
<td>CF</td>
</tr>
<tr>
<td>rb</td>
<td>CF</td>
<td>C</td>
</tr>
</tbody>
</table>

- **C**: rules can’t be executed concurrently
**Conflict Matrix (CM)**

BSV compiler generates the pairwise conflict information

---

**Example 1**

```plaintext
rule ra;
  x <= x+1;
endrule
rule rb;
  y <= y+2;
endrule
```

---

**Example 2**

```plaintext
rule ra;
  x <= y+1;
endrule
rule rb;
  y <= x+2;
endrule
```

---

**Example 3**

```plaintext
rule ra;
  x <= y+1;
endrule
rule rb;
  y <= y+2;
endrule
```

<table>
<thead>
<tr>
<th></th>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
<td>CF</td>
</tr>
<tr>
<td>rb</td>
<td>CF</td>
<td>C</td>
</tr>
</tbody>
</table>

- **C**: rules can’t be executed concurrently
- **CF**: rules can be executed concurrently; the net effect is the same as if ra executed before rb (ra<rb) or (rb<ra)

November 9, 2018
Conflict Matrix (CM)

BSV compiler generates the pairwise conflict information

Example 1

```plaintext
rule ra;
  x <= x+1;
endrule
rule rb;
  y <= y+2;
endrule
```

<table>
<thead>
<tr>
<th></th>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
<td>CF</td>
</tr>
<tr>
<td>rb</td>
<td>CF</td>
<td>C</td>
</tr>
</tbody>
</table>

Example 2

```plaintext
rule ra;
  x <= y+1;
endrule
rule rb;
  y <= x+2;
endrule
```

<table>
<thead>
<tr>
<th></th>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>

Example 3

```plaintext
rule ra;
  x <= y+1;
endrule
rule rb;
  y <= y+2;
endrule
```

<table>
<thead>
<tr>
<th></th>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>CF</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>

- C : rules can’t be executed concurrently
- CF: rules can be executed concurrently; the net effect is the same as if ra executed before rb (ra<rb) or (rb<ra)

November 9, 2018
### Conflict Matrix (CM)

BSV compiler generates the pairwise conflict information

#### Example 1

<table>
<thead>
<tr>
<th></th>
<th>rule ra;</th>
<th>x &lt;= x+1;</th>
<th>endrule</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rb</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
<td>CF</td>
</tr>
<tr>
<td>rb</td>
<td>CF</td>
<td>C</td>
</tr>
</tbody>
</table>

#### Example 2

<table>
<thead>
<tr>
<th></th>
<th>rule ra;</th>
<th>x &lt;= y+1;</th>
<th>endrule</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rb</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>

#### Example 3

<table>
<thead>
<tr>
<th></th>
<th>rule ra;</th>
<th>x &lt;= y+1;</th>
<th>endrule</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rb</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
<td>&lt;</td>
</tr>
<tr>
<td>rb</td>
<td>&gt;</td>
<td>C</td>
</tr>
</tbody>
</table>

- **C**: rules can’t be executed concurrently
- **CF**: rules can be executed concurrently; the net effect is the same as if ra executed before rb (ra<rb) or (rb<ra)
Conflict Matrix (CM)
BSV compiler generates the pairwise conflict information

Example 1

```plaintext
rule ra;
  x <= x+1;
endrule
rule rb;
  y <= y+2;
endrule
```

<table>
<thead>
<tr>
<th></th>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
<td>CF</td>
</tr>
<tr>
<td>rb</td>
<td>CF</td>
<td>C</td>
</tr>
</tbody>
</table>

Example 2

```plaintext
rule ra;
  x <= y+1;
endrule
rule rb;
  y <= x+2;
endrule
```

<table>
<thead>
<tr>
<th></th>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>C</td>
<td>C</td>
</tr>
</tbody>
</table>

Example 3

```plaintext
rule ra;
  x <= y+1;
endrule
rule rb;
  y <= y+2;
endrule
```

<table>
<thead>
<tr>
<th></th>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
<td>&lt;</td>
</tr>
<tr>
<td>rb</td>
<td>&gt;</td>
<td>C</td>
</tr>
</tbody>
</table>

- C: rules can’t be executed concurrently
- CF: rules can be executed concurrently; the net effect is the same as if ra executed before rb (ra<rb) or (rb<ra)
- ra < rb: rules can be executed concurrently; the net effect is as if ra executed before rb
Concurrent rule execution

```plaintext
rule ra(p(x));
    m.f(x+1);
endrule
rule rb(q(x));
    m.g(x+2)
endrule
```
Concurrent rule execution

rule ra(p(x));
  m.f(x+1);
endrule
rule rb(q(x));
  m.g(x+2)
endrule
Concurrent rule execution

\[
\begin{align*}
\text{rule } ra(p(x)); \\
& \quad m.f(x+1); \\
\text{endrule} \\
\text{rule } rb(q(x)); \\
& \quad m.g(x+2) \\
\text{endrule}
\end{align*}
\]
Concurrent rule execution

```
rule ra(p(x));
  m.f(x+1);
endrule
rule rb(q(x));
  m.g(x+2)
endrule
```
Concurrent rule execution

\begin{verbatim}
rule ra(p(x));
  m.f(x+1);
endrule
rule rb(q(x));
  m.g(x+2)
endrule
\end{verbatim}
Concurrent rule execution

\begin{verbatim}
rule ra(p(x));
  m.f(x+1);
endrule
rule rb(q(x));
  m.g(x+2)
endrule
\end{verbatim}
Concurrent rule execution

```
rule ra(p(x));
  m.f(x+1);
endrule
rule rb(q(x));
  m.g(x+2)
endrule
```
Concurrent rule execution

```
rule ra(p(x));
m.f(x+1);
endrule
rule rb(q(x));
m.g(x+2)
endrule
```
Concurrent rule execution

\textbf{rule} \texttt{ra}(p(x));
\hspace{1em} \texttt{m.f}(x+1);
\textbf{endrule}

\textbf{rule} \texttt{rb}(q(x));
\hspace{1em} \texttt{m.g}(x+2)
\textbf{endrule}
Concurrent rule execution

- This circuit will execute rules ra and rb concurrently
Concurrent rule execution

- This circuit will execute rules ra and rb concurrently
- This circuit is correct only if rules ra and rb do not conflict (⇒ methods f and g of m do not conflict)
**Concurrent rule execution**

- This circuit will execute rules ra and rb concurrently.
- This circuit is correct only if rules ra and rb do not conflict (⇒ methods f and g of m do not conflict).
- Suppose rules ra and rb do conflict!
Concurrent rule execution

- This circuit will execute rules \texttt{ra} and \texttt{rb} concurrently.
- This circuit is correct only if rules \texttt{ra} and \texttt{rb} do not conflict (\(\Rightarrow\) methods \texttt{f} and \texttt{g} of \texttt{m} do not conflict).
- Suppose rules \texttt{ra} and \texttt{rb} do conflict!

\begin{verbatim}
rule ra(p(x));
  m.f(x+1);
endrule
rule rb(q(x));
  m.g(x+2)
endrule
\end{verbatim}
Need for a scheduler

\begin{verbatim}
rule ra (p(x));
  m.f(x+1);
endrule
rule rb (q(x));
  m.g(x+2)
endrule
\end{verbatim}
Need for a scheduler

- Guards of all rules are fed to a scheduler

```plaintext
rule ra (p(x));
  m.f(x+1);
endrule
rule rb (q(x));
  m.g(x+2)
endrule
```
Need for a scheduler

- Guards of all rules are fed to a scheduler

```plaintext
rule ra (p(x));
  m.f(x+1);
endrule
rule rb (q(x));
  m.g(x+2)
endrule
```
Need for a scheduler

- Guards of all rules are fed to a scheduler

```plaintext
rule ra (p(x));
m.f(x+1);
endrule
rule rb (q(x));
m.g(x+2)
endrule
```

Diagram showing the interaction between rules and a scheduler.
Need for a scheduler

- Guards of all rules are fed to a scheduler
- Using the CM, the scheduler lets only non-conflicting rules proceed

```plaintext
rule ra (p(x));
  m.f(x+1);
endrule
rule rb (q(x));
  m.g(x+2)
endrule
```
Need for a scheduler

- Guards of all rules are fed to a scheduler
- Using the CM, the scheduler lets only non-conflicting rules proceed

```plaintext
rule ra (p(x));
  m.f(x+1);
endrule
rule rb (q(x));
  m.g(x+2)
endrule
```
Need for a scheduler

- Guards of all rules are fed to a scheduler
- Using the CM, the scheduler lets only non-conflicting rules proceed

```
rule ra (p(x));
  m.f(x+1);
endrule
rule rb (q(x));
  m.g(x+2)
endrule
```
Need for a scheduler

- Guards of all rules are fed to a scheduler
- Using the CM, the scheduler lets only non-conflicting rules proceed
- Scheduler is a pure combinational circuit with a small number of gates

```
rule ra (p(x));
m.f(x+1);
endrule
rule rb (q(x));
m.g(x+2)
endrule
```
Need for a scheduler

- Guards of all rules are fed to a scheduler
- Using the CM, the scheduler lets only non-conflicting rules proceed
- Scheduler is a pure combinational circuit with a small number of gates
- A correct but low performance scheduler may schedule only one rule at a time

```
rule ra (p(x));
  m.f(x+1);
endrule
rule rb (q(x));
  m.g(x+2)
endrule
```
The rule scheduler
The rule scheduler

rule guards (aka can_fire signals)

Scheduler

gs1 → wfs1

gsn → wfn
The rule scheduler

rule guards
(aka can_fire signals)

Scheduler

wfs1

will_fire signals

wfn

gs1

gsn
Guards (gs1 ... gsn) of many rules may be true simultaneously, and some of them may conflict
The rule scheduler

- Guards (gs1 ... gsn) of many rules may be true simultaneously, and some of them may conflict.
- BSV compiler constructs a combinational scheduler circuit with the following property:

\begin{center}
\begin{tikzpicture}
\node (scheduler) at (0,0) [draw, rounded corners, fill=gray!30] {Scheduler};
\node (gs1) at (-2,-0.5) {gs1};
\node (gsn) at (-2,-1) {gsn};
\node (wfs1) at (2,-0.5) {wfs1};
\node (wfn) at (2,-1) {wfn};
\draw[->] (gs1) -- (scheduler);
\draw[->] (gsn) -- (scheduler);
\draw[->] (scheduler) -- (wfs1);
\draw[->] (scheduler) -- (wfn);
\end{tikzpicture}
\end{center}
The rule scheduler

- Guards \((gs_1 \ldots gsn)\) of many rules may be true simultaneously, and some of them may conflict.
- BSV compiler constructs a combinational scheduler circuit with the following property:

  for all \(i\) and \(j\), if \(wfs_i\) and \(wfs_j\) are true then the corresponding \(gs_i\) and \(gs_j\) must be true and rules \(i\) and \(j\) must not conflict with each other.
Example schedulers

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>CF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>&gt;</td>
</tr>
</tbody>
</table>

The effect ra < rb would be achieved automatically
The effect ra < rb would be achieved automatically.
Example schedulers

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>CF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>&gt;</td>
</tr>
</tbody>
</table>

The effect ra < rb would be achieved automatically.
Example schedulers

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>CF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>&gt;</td>
</tr>
</tbody>
</table>

The effect ra < rb would be achieved automatically
### Example schedulers

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>CF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>&gt;</td>
</tr>
</tbody>
</table>

The effect ra < rb would be achieved automatically.
Example schedulers

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>CF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>&gt;</td>
</tr>
</tbody>
</table>

The effect ra < rb would be achieved automatically.
Example schedulers

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>CF</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ra</th>
<th>rb</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
</tr>
<tr>
<td>rb</td>
<td>&gt;</td>
</tr>
</tbody>
</table>

The effect ra < rb would be achieved automatically.
Take-home problem

- Draw the hardware circuit for this design

```vhdl
rule stage1;
    fifo.enq(f0(inQ.first));
    inQ.deq;
endrule

rule stage2;
    outQ.enq(f1(fifo.first));
    fifo.deq;
endrule
```
Additional Problem

\[
\text{rule ra(p1);}
\begin{align*}
   & x <= y + 1; \\
\end{align*}
\text{endrule}
\text{rule rb(p2);}
\begin{align*}
   & y <= z + 2; \\
\end{align*}
\text{Endrule}
\text{rule rc(p3);}
\begin{align*}
   & z <= x + 2; \\
\end{align*}
\text{endrule}
\]

No. (ra < rb), (rb < rc) but (rc < ra) 

Yes!
Additional Problem

```
rule ra(p1);
    x <= y + 1;
endrule
rule rb(p2);
    y <= z + 2;
Endrule
rule rc(p3);
    z <= x + 2;
endrule
```

<table>
<thead>
<tr>
<th></th>
<th>ra</th>
<th>rb</th>
<th>rc</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
<td>&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>rb</td>
<td>&gt;</td>
<td>C</td>
<td>&lt;</td>
</tr>
<tr>
<td>rc</td>
<td>&lt;</td>
<td>&gt;</td>
<td>C</td>
</tr>
</tbody>
</table>

No. (ra < rb), (rb < rc) but (rc < ra)

Yes!
Can all three rules execute concurrently?

No. \((ra < rb)\), \((rb < rc)\) but \((rc < ra)\)

Yes!
Additional Problem

```plaintext
rule ra(p1);
  x <= y + 1;
endrule
rule rb(p2);
  y <= z + 2;
Endrule
rule rc(p3);
  z <= x + 2;
endrule
```

<table>
<thead>
<tr>
<th></th>
<th>ra</th>
<th>rb</th>
<th>rc</th>
</tr>
</thead>
<tbody>
<tr>
<td>ra</td>
<td>C</td>
<td>&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>rb</td>
<td>&gt;</td>
<td>C</td>
<td>&lt;</td>
</tr>
<tr>
<td>rc</td>
<td>&lt;</td>
<td>&gt;</td>
<td>C</td>
</tr>
</tbody>
</table>

- Can all three rules execute concurrently? No. (ra < rb), (rb < rc) but (rc < ra)
- Can any two rules execute concurrently? Yes!
L18- Managing Control Hazards Through Epochs
Epoch: a method to manage control hazards

Attach an epoch (color) to each instruction as it flows through the pipeline.

Change the epoch if the Execute stage detects a misprediction and set the pc to the correct pc.
Epoch: a method to manage control hazards

- Attach an epoch (color) to each instruction as it flows through the pipeline
- Change the epoch if the Execute stage detects a misprediction and set the pc to the correct pc
- Discard instructions with the old epoch, that is, if its epoch does not match the current epoch
Two-Stage Pipeline processor 
first attempt - code

```verilog
rule doFetch;
    iMem.req(MemReq{op: Ld, addr: pc, data: ?});
    let ppc = pc + 4; pc <= ppc;
    f2d.enq(F2D {pc: pc, ppc: ppc, epoch: epoch});
endrule
```
Two-Stage Pipeline processor
first attempt - code

rule doFetch;
  iMem.req(MemReq{op: Ld, addr: pc, data: ?});
  let ppc = pc + 4; pc <= ppc;
  f2d.enq(F2D {pc: pc, ppc: ppc, epoch: epoch});
endrule

rule doExecute if (state == Execute);
  let inst <- iMem.resp;
  let x = f2d.first; f2d.deq;
  let pcD = x.pc; let ppc = x.ppc; let epochD = x.epoch;
  if (epochD == epoch) begin // right-path instruction
    code to compute eInst from inst
    let mispred = eInst.nextPC != ppc;
    if (mispred) begin pc <= eInst.nextPC; epoch <= !epoch; end
    code to update the state;
    in case of a memory op, initiate memory req and
    in case of Ld go to LoadWait
  end
endrule

rule doLoadWait if (state == LoadWait); ... go to Execute ...
## mkProcTwoStageBad analysis

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Multi-Cycle (Cycles)</th>
<th>Two-Stage Bad (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd_sw.S</td>
<td>3022</td>
<td>4403</td>
</tr>
<tr>
<td>no_hazard.S</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>all_control_hazard.S</td>
<td>16</td>
<td>31</td>
</tr>
</tbody>
</table>
**mkProcTwoStageBad analysis**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Multi-Cycle (Cycles)</th>
<th>Two-Stage Bad (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd_sw.S</td>
<td>3022</td>
<td>4403</td>
</tr>
<tr>
<td>no_hazard.S</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>all_control_hazard.S</td>
<td>16</td>
<td>31</td>
</tr>
</tbody>
</table>
**mkProcTwoStageBad analysis**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Multi-Cycle (Cycles)</th>
<th>Two-Stage Bad (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd_sw.S</td>
<td>3022</td>
<td>4403</td>
</tr>
<tr>
<td>no_hazard.S</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>all_control_hazard.S</td>
<td>16</td>
<td>31</td>
</tr>
</tbody>
</table>

**Why?**
**mkProcTwoStageBad analysis**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Multi-Cycle (Cycles)</th>
<th>Two-Stage Bad (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd_sw.S</td>
<td>3022</td>
<td>4403</td>
</tr>
<tr>
<td>no_hazard.S</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>all_control_hazard.S</td>
<td>16</td>
<td>31</td>
</tr>
</tbody>
</table>

**Why?**

```verilog
define rule doFetch;
    ... pc <= ppc; ...
endrule
```

```verilog
define rule doExecute if (state != LoadWait);
    ...
    let mispred = eInst.nextPC != ppc;
    if (mispred) begin pc <= eInst.nextPC; epoch <= !epoch; end
    code to initiate memory ops and go to LoadWait if necessary
endrule
```
mkProcTwoStageBad analysis

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Multi-Cycle (Cycles)</th>
<th>Two-Stage Bad (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd_sw.S</td>
<td>3022</td>
<td>4403</td>
</tr>
<tr>
<td>no_hazard.S</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>all_control_hazard.S</td>
<td>16</td>
<td>31</td>
</tr>
</tbody>
</table>

Why?

```verilog
rule doFetch;
    ... pc <= ppc; ...
endrule
rule doExecute if (state != LoadWait);
    ...
    let mispred = eInst.nextPC != ppc;
    if (mispred) begin pc <= eInst.nextPC; epoch <= !epoch; end
    code to initiate memory ops and go to LoadWait if necessary
endrule
```
**mkProcTwoStageBad analysis**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Multi-Cycle (Cycles)</th>
<th>Two-Stage Bad (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd_sw.S</td>
<td>3022</td>
<td>4403</td>
</tr>
<tr>
<td>no_hazard.S</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>all_control_hazard.S</td>
<td>16</td>
<td>31</td>
</tr>
</tbody>
</table>

**Why?**

```verbatim
rule doFetch;
  ... pc <= ppc; ...
endrule

rule doExecute if (state != LoadWait);
  ...
  let mispred = eInst.nextPC != ppc;
  if (mispred) begin pc <= eInst.nextPC; epoch <= !epoch; end
  code to initiate memory ops and go to LoadWait if necessary
endrule
```

 dofFetch and doExecute conflict! Can't execute concurrently
mkProcTwoStageBad analysis

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Multi-Cycle (Cycles)</th>
<th>Two-Stage Bad (Cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd_sw.S</td>
<td>3022</td>
<td>4403</td>
</tr>
<tr>
<td>no_hazard.S</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>all_control_hazard.S</td>
<td>16</td>
<td>31</td>
</tr>
</tbody>
</table>

Why?

rule doFetch;
  ... pc <= ppc; ...    
endrule

rule doExecute if (state != LoadWait);
  ... let mispred = eInst.nextPC != ppc;
  if (mispred) begin pc <= eInst.nextPC; epoch <= !epoch; end
  code to initiate memory ops and go to LoadWait if necessary
endrule


doFetch and doExecute conflict! Can't execute concurrently

Fix?
Rule Splitting

```plaintext
rule one;
  r1 <= e1;
endrule

rule two;
  if (p) r2 <= e2;
  else r1 <= e3;
endrule
```

rules one and two conflict and cannot execute concurrently
Rule Splitting

```plaintext
rule one;
    r1 <= e1;
endrule

rule two;
    if (p) r2 <= e2;
    else r1 <= e3;
endrule
```

- rules one and two conflict and cannot execute concurrently
- split rule two into twoT and twoF
Rule Splitting

```plaintext
rule one;
  r1 <= e1;
endrule

rule two;
  if (p) r2 <= e2;
  else r1 <= e3;
endrule

rules one and two conflict and cannot execute concurrently

split rule two into twoT and twoF
```
Rule Splitting

```
rule one;
  r1 <= e1;
endrule

rule two;
  if (p) r2 <= e2;
  else r1 <= e3;
endrule
```

Rules one and two conflict and cannot execute concurrently.

Split rule two into twoT and twoF.

```
rule one;
  r1 <= e1;
endrule

rule twoT (p);
  r2 <= e2;
endrule

rule twoF (!p);
  r1 <= e3;
endrule
```

Rules twoT and one do not conflict and can execute together.
Another way of Rule Splitting

\begin{verbatim}
rule one;
    r1 <= e1;
endrule

rule two;
    more actions;
    p = e;
    if (p) r2 <= e2;
    else r1 <= e3;
endrule
\end{verbatim}

Difficult to turn p into a guard
Another way of Rule Splitting

```
rule one;
    r1 <= e1;
endrule

rule two;
    more actions;
    p = e;
    if (p) r2 <= e2;
    else r1 <= e3;
endrule
```

- Difficult to turn p into a guard
- Create a separate rule for the conflicting action and delay its execution until the next cycle
Another way of Rule Splitting

```
rule one;
  r1 <= e1;
endrule

rule two;
  more actions;
p = e;
  if (p) r2 <= e2;
  else r1 <= e3;
endrule
```

- Difficult to turn p into a guard
- Create a separate rule for the conflicting action and delay its execution until the next cycle
- Requires a register (pReg) to remember p
Another way of Rule Splitting

```plaintext
rule one;
  r1 <= e1;
endrule

rule two;
  more actions;
  p = e;
  if (p) r2 <= e2;
  else r1 <= e3;
endrule
```

- Difficult to turn p into a guard
- Create a separate rule for the conflicting action and delay its execution until the next cycle
- Requires a register (pReg) to remember p
Fixing the two-stage pipeline

```
rule doFetch;
    ...
    pc <= ppc;
endrule

rule doExecute if (state != LoadWait);
    ...
    let mispred = eInst.nextPC != ppc;
    if (mispred) begin pc <= eInst.nextPC; epoch <= !epoch; end
code to initiate memory ops and go to LoadWait if necessary
endrule

rule doLoadWait if (state == LoadWait);
...

rule doRedirection
```
Fixing the two-stage pipeline

```verilog
rule doFetch;
  ... pc <= ppc; ... 
endrule

rule doExecute if (state != LoadWait);
  ...
  let mispred = eInst.nextPC != ppc;
  if (mispred) begin pc <= eInst.nextPC; epoch <= !epoch; end
  code to initiate memory ops and go to LoadWait if necessary
endrule
rule doLoadWait if (state == LoadWait); ...
rule doRedirection
```
Fixing the two-stage pipeline

rule doFetch;
    ...
    pc <= ppc;
endrule

rule doExecute if (state != LoadWait);
    ...
    let mispred = eInst.nextPC != ppc;
    if (mispred) begin
        pc <= eInst.nextPC;
        epoch <= !epoch;
    end
    code to initiate memory ops and go to LoadWait if necessary
endrule

rule doLoadWait if (state == LoadWait);
    ...
rule doRedirection
Fixing the two-stage pipeline

```
rule doFetch;
    ... pc <= ppc; ...
endrule

rule doExecute if (state != LoadWait);
    ...
    let mispred = eInst.nextPC != ppc;
    if (mispred) begin pc <= eInst.nextPC; epoch <= !epoch; end
    code to initiate memory ops and go to LoadWait if necessary
endrule
rule doLoadWait if (state == LoadWait); ...
rule doRedirection

◆ In doExecute remember misprediction in a register
```
Fixing the two-stage pipeline

In doExecute remember misprediction in a register
Introduce a new doRedirection rule to change epoch and pc
Fixing the two-stage pipeline

In doExecute remember misprediction in a register
Introduce a new doRedirection rule to change epoch and pc
Adjust the guards to make doExecute and doRedirection mutually exclusive
Fixing the two-stage pipeline

- In `doExecute` remember misprediction in a register
- Introduce a new `doRedirection` rule to change epoch and pc
- Adjust the guards to make `doExecute` and `doRedirection` mutually exclusive

Lab 7
Improved two-stage pipeline

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Multi-Cycle</th>
<th>Two-Stage Bad</th>
<th>Two-Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd_sw.S</td>
<td>3022</td>
<td>4403</td>
<td>4317</td>
</tr>
<tr>
<td>no_hazard.S</td>
<td>26</td>
<td>27</td>
<td>14</td>
</tr>
<tr>
<td>all_control_hazard.S</td>
<td>16</td>
<td>31</td>
<td>37</td>
</tr>
</tbody>
</table>

- With no hazard, the pipeline behaves perfectly and throughput doubles
Improved two-stage pipeline

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Multi-Cycle</th>
<th>Two-Stage Bad</th>
<th>Two-Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd_sw.S</td>
<td>3022</td>
<td>4403</td>
<td>4317</td>
</tr>
<tr>
<td>no_hazard.S</td>
<td>26</td>
<td>27</td>
<td>14</td>
</tr>
<tr>
<td>all_control_hazard.S</td>
<td>16</td>
<td>31</td>
<td>37</td>
</tr>
</tbody>
</table>

- With no hazard, the pipeline behaves perfectly and throughput doubles.
- With control hazard, we have two-cycle penalty and perform worse than Multi-cycle.
### Improved two-stage pipeline

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Multi-Cycle</th>
<th>Two-Stage Bad</th>
<th>Two-Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd_sw.S</td>
<td>3022</td>
<td>4403</td>
<td>4317</td>
</tr>
<tr>
<td>no_hazard.S</td>
<td>26</td>
<td>27</td>
<td>14</td>
</tr>
<tr>
<td>all_control_hazard.S</td>
<td>16</td>
<td>31</td>
<td>37</td>
</tr>
</tbody>
</table>

- With no hazard, the pipeline behaves perfectly and throughput doubles
- With control hazard, we have two-cycle penalty and perform worse than Multi-cycle
- Improvement 1: a cycle can be saved in case of redirection by initiating the instruction fetch from the redirection rule itself
Improved two-stage pipeline

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Multi-Cycle</th>
<th>Two-Stage Bad</th>
<th>Two-Stage</th>
<th>Two-Stage Improv 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd_sw.S</td>
<td>3022</td>
<td>4403</td>
<td>4317</td>
<td>2919</td>
</tr>
<tr>
<td>no_hazard.S</td>
<td>26</td>
<td>27</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>all_control_hazard.S</td>
<td>16</td>
<td>31</td>
<td>37</td>
<td>23</td>
</tr>
</tbody>
</table>

- With no hazard, the pipeline behaves perfectly and throughput doubles
- With control hazard, we have two-cycle penalty and perform worse than Multi-cycle
- Improvement 1: a cycle can be saved in case of redirection by initiating the instruction fetch from the redirection rule itself
### Improved two-stage pipeline

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Multi-Cycle</th>
<th>Two-Stage Bad</th>
<th>Two-Stage</th>
<th>Two-Stage Improv 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd_sw.S</td>
<td>3022</td>
<td>4403</td>
<td>4317</td>
<td>2919</td>
</tr>
<tr>
<td>no_hazard.S</td>
<td>26</td>
<td>27</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>all_control_hazard.S</td>
<td>16</td>
<td>31</td>
<td>37</td>
<td>23</td>
</tr>
</tbody>
</table>

- With no hazard, the pipeline behaves perfectly and throughput doubles.
- With control hazard, we have two-cycle penalty and perform worse than Multi-cycle.
- Improvement 1: a cycle can be saved in case of redirection by initiating the instruction fetch from the redirection rule itself.
# Improved two-stage pipeline

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Multi-Cycle</th>
<th>Two-Stage Bad</th>
<th>Two-Stage</th>
<th>Two-Stage Improv 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd_sw.S</td>
<td>3022</td>
<td>4403</td>
<td>4317</td>
<td>2919</td>
</tr>
<tr>
<td>no_hazard.S</td>
<td>26</td>
<td>27</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>all_control_hazard.S</td>
<td>16</td>
<td>31</td>
<td>37</td>
<td>23</td>
</tr>
</tbody>
</table>

- With no hazard, the pipeline behaves perfectly and throughput doubles.
- With control hazard, we have two-cycle penalty and perform worse than Multi-cycle.
- Improvement 1: a cycle can be saved in case of redirection by initiating the instruction fetch from the redirection rule itself.
- Improvement 2: Bypass pc and epoch updates from doExecute to doFetch (need EHRs - L23)
**Improved two-stage pipeline**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Multi-Cycle</th>
<th>Two-Stage</th>
<th>Two-Stage</th>
<th>Two-Stage</th>
<th>Two-Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Bad</td>
<td>Improv 1</td>
<td>Improv 2</td>
<td></td>
</tr>
<tr>
<td>gcd_sw.S</td>
<td>3022</td>
<td>4403</td>
<td>4317</td>
<td>2919</td>
<td>1530</td>
</tr>
<tr>
<td>no_hazard.S</td>
<td>26</td>
<td>27</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>all_control_hazard.S</td>
<td>16</td>
<td>31</td>
<td>37</td>
<td>23</td>
<td>9</td>
</tr>
</tbody>
</table>

- With no hazard, the pipeline behaves perfectly and throughput doubles.
- With control hazard, we have two-cycle penalty and perform worse than Multi-cycle.
- Improvement 1: a cycle can be saved in case of redirection by initiating the instruction fetch from the redirection rule itself.
- Improvement 2: Bypass pc and epoch updates from doExecute to doFetch (need EHRs - L23)
Improved two-stage pipeline

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Multi-Cycle</th>
<th>Two-Stage Bad</th>
<th>Two-Stage Improv 1</th>
<th>Two-Stage Improv 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcd_sw.S</td>
<td>3022</td>
<td>4403</td>
<td>4317</td>
<td>2919</td>
</tr>
<tr>
<td>no_hazard.S</td>
<td>26</td>
<td>27</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>all_control_hazard.S</td>
<td>16</td>
<td>31</td>
<td>37</td>
<td>23</td>
</tr>
</tbody>
</table>

- With no hazard, the pipeline behaves perfectly and throughput doubles.
- With control hazard, we have two-cycle penalty and perform worse than Multi-cycle.
- Improvement 1: a cycle can be saved in case of redirection by initiating the instruction fetch from the redirection rule itself.
- Improvement 2: Bypass pc and epoch updates from doExecute to doFetch (need EHRs - L23).
L18 Review- Managing Data Hazards through a Scoreboard
Splitting Execute into two stages

- pc
- epoch
- rf
- f2d
- iMem
- dMem

Diagram showing the flow between fetch, execute, iMem, and dMem.
Splitting Execute into two stages

- Execute step probably has the longest propagation delay (decode + register-file read + execute)
Splitting Execute into two stages

- Execute step probably has the longest propagation delay (decode + register-file read + execute)
- Separate Execute into two stages:
  - Decode and register-file-read
  - Execute – including the initiation of memory instructions
Splitting Execute into two stages

- Execute step probably has the longest propagation delay (decode + register-file read + execute)
- Separate Execute into two stages:
  - Decode and register-file-read
  - Execute – including the initiation of memory instructions
Splitting Execute into two stages

- Execute step probably has the longest propagation delay (decode + register-file read + execute)
- Separate Execute into two stages:
  - Decode and register-file-read
  - Execute – including the initiation of memory instructions
- Will require us to introduce several new concepts in pipelining
Splitting doExecute
three-stage pipeline

```plaintext
rule doFetch; ... endrule
```
Splitting doExecute
three-stage pipeline

`rule doFetch; ... endrule`

`rule doDecode;`
`  let inst <- iMem.resp;`
`  ... filter wrong-path instructions ...
  ... decode (inst) ...
  ... read rf ...`
`d2e.enq(...);`
`endrule`
Splitting doExecute
three-stage pipeline

```plaintext
rule doFetch; ... endrule

rule doDecode;
  let inst <- iMem.resp;
  ... filter wrong-path instructions ... 
  ... decode (inst) ...
  ... read rf ...
  d2e.enq(...);
endrule

rule doExecute (...);
  let dInst = d2e.first; d2e.deq;
  ... filter wrong-path instructions ...
  ... execute (dInst, rval1, rval2, pc) ...
  ... detect and handle misprediction ...
  ... launch memory instructions if needed ...
  ... save info for the LoadWait step ...
endrule
rule doLoadWait(...) ... endrule
```
Splitting doExecute
three-stage pipeline

**Danger Data Hazards:**
doDecode may read stale values (L15-L16)

```plaintext
rule doFetch; ... endrule

rule doDecode;
    let inst <- iMem.resp;
    ... filter wrong-path instructions ...
    ... decode (inst) ...
    ... read rf ...
    d2e.enq(...);
endrule

rule doExecute (...);
    let dInst = d2e.first; d2e.deq;
    ... filter wrong-path instructions ...
    ... execute (dInst, rval1,rval2, pc) ...
    ... detect and handle misprediction ...
    ... launch memory instructions if needed ...
    ... save info for the LoadWait step ...
endrule

rule doLoadWait(...) ... endrule
```
Dealing with data hazards

- Detect data hazard (aka read-after-write (RAW) hazard)
  - Compare sources in the decoded inst with the destinations of older instructions in the pipeline
Dealing with data hazards

- Detect data hazard (aka read-after-write (RAW) hazard)
  - Compare sources in the decoded instruction with the destinations of older instructions in the pipeline
  - Requires a *Scoreboard* -- a data structure to keep track of the destinations of the instructions in the pipeline beyond the Decode stage
Dealing with data hazards

Detect data hazard (aka read-after-write (RAW) hazard)

- Compare sources in the decoded inst with the destinations of older instructions in the pipeline
- Requires a Scoreboard -- a data structure to keep track of the destinations of the instructions in the pipeline beyond the Decode stage

Stall the Decode from dispatching the instruction as long as RAW hazard prevails
Dealing with data hazards

- Detect data hazard (aka read-after-write (RAW) hazard)
  - Compare sources in the decoded inst with the destinations of older instructions in the pipeline
  - Requires a Scoreboard -- a data structure to keep track of the destinations of the instructions in the pipeline beyond the Decode stage

- Stall the Decode from dispatching the instruction as long as RAW hazard prevails
  - RAW hazard will disappear as the pipeline drains
Scoreboard

search1  search2  insert  remove
scoreboard
**Scoreboard**

- **method insert(dst):** inserts the destination of an instruction or Invalid in the scoreboard when the instruction is decoded
Scoreboard

- **method insert(dst)**: inserts the destination of an instruction or Invalid in the scoreboard when the instruction is decoded
- **method search1(src)**: searches the scoreboard for a data hazard, i.e., a dst that matches src
- **method search2(src)**: same as search1
Scoreboard

- **method insert(dst):** inserts the destination of an instruction or Invalid in the scoreboard when the instruction is decoded
- **method search1(src):** searches the scoreboard for a data hazard, i.e., a dst that matches src
- **method search2(src):** same as search1
- **method remove:** deletes the oldest entry when an instruction commits
method `insert(dst)`: inserts the destination of an instruction or Invalid in the scoreboard when the instruction is decoded

method `search1(src)`: searches the scoreboard for a data hazard, i.e., a dst that matches src

method `search2(src)`: same as `search1`

method `remove`: deletes the oldest entry when an instruction commits

we will provide you a scoreboard implementation
doDecode rule

```haskell
rule doDecode;
  let inst <- iMem.resp;
  let x = f2d.first; f2d.deq;
  if (x.epoch == epoch) begin
    let dInst = decode2(inst); // src1 and src2 are Maybe types;
  end
endrule
```

decode2 is the modified decoder
doDecode rule

rule doDecode;
  let inst <- iMem.resp;
  let x = f2d.first; f2d.deq;
  if (x.epoch == epoch) begin
    let dInst = decode2(inst); // src1 and src2 are Maybe types;
    // check for data hazard
    if (!(sb.search1(dInst.src1)||sb.search2(dInst.src2))) begin
  end
endrule

decode2 is the modified decoder
doDecode rule

rule doDecode;
let inst <- iMem.resp;
let x = f2d.first; f2d.deq;
if (x.epoch == epoch) begin
  let dInst = decode2(inst); // src1 and src2 are Maybe types;
  // check for data hazard
  if (!(sb.search1(dInst.src1)||sb.search2(dInst.src2))) begin
    read rVal1 and read rVal2 from rf
  end
endrule

decode2 is the modified decoder
doDecode rule

rule doDecode;
    let inst <- iMem.resp;
    let x = f2d.first; f2d.deq;
    if (x.epoch == epoch) begin
        let dInst = decode2(inst); // src1 and src2 are Maybe types;
        // check for data hazard
        if (!(sb.search1(dInst.src1)||sb.search2(dInst.src2))) begin
            read rVal1 and read rVal2 from rf
            sb.insert(dInst.dst); // to stall future inst for data hazard
        end
    end
endrule

decode2 is the modified decoder
doDecode rule

```verilog
rule doDecode;

  let inst <- iMem.resp;
  let x = f2d.first; f2d.deq;
  if (x.epoch == epoch) begin
    let dInst = decode2(inst); // src1 and src2 are Maybe types;
    // check for data hazard
    if (!(sb.search1(dInst.src1)||sb.search2(dInst.src2))) begin
      read rVal1 and read rVal2 from rf
      sb.insert(dInst.dst); //to stall future inst for data hazard
      enqueue into e2d fifo: pc, ppc, epoch, rVal1, rVal2, dInst
    end
  end
endrule
```

decode2 is the modified decoder
doDecode rule

rule doDecode;
  let inst <- iMem.resp;
  let x = f2d.first; f2d.deq;
  if (x.epoch == epoch) begin
    let dInst = decode2(inst); // src1 and src2 are Maybe types;
    // check for data hazard
    if (!(sb.search1(dInst.src1)||sb.search2(dInst.src2))) begin
      read rVal1 and read rVal2 from rf
      sb.insert(dInst.dst); //to stall future inst for data hazard
      enqueue into e2d fifo: pc, ppc, epoch, rVal1, rVal2, dInst
    end
  end
endrule

decode2 is the modified decoder

Still not quite correct. Why?
doDecode rule

rule doDecode;
let inst <- iMem.resp;
let x = f2d.first; f2d.deq;
if (x.epoch == epoch) begin
  let dInst = decode2(inst); // src1 and src2 are Maybe types;
  // check for data hazard
  if (!(sb.search1(dInst.src1)||sb.search2(dInst.src2))) begin
    read rVal1 and read rVal2 from rf
    sb.insert(dInst.dst); // to stall future inst for data hazard
    enqueue into e2d fifo: pc, ppc, epoch, rVal1, rVal2, dInst
  end
end
endrule

Still not quite correct. Why?
We need to keep the fetched instruction while stalling!
doDecode rule

rule doDecode;
let inst <- iMem.resp;
let x = f2d.first; f2d.deq;
if (x.epoch == epoch) begin
  let dInst = decode2(inst); // src1 and src2 are Maybe types;
  // check for data hazard
  if (!(sb.search1(dInst.src1)||sb.search2(dInst.src2))) begin
    read rVal1 and read rVal2 from rf
    sb.insert(dInst.dst); //to stall future inst for data hazard
    enqueue into e2d fifo: pc, ppc, epoch, rVal1, rVal2, dInst
  end
end endrule

decode2 is the modified decoder

Still not quite correct. Why?
We need to keep the fetched instruction while stalling!
Need a register to hold the fetched instruction while stalling
Fixing the doDecode rule

```plaintext
rule doDecode;
    let inst <- iMem.resp;
    let x = f2d.first; f2d.deq;
    if (x.epoch == epoch) begin
        let dInst = decode2(inst); // src1 and src2 are Maybe types;
        if (!(sb.search1(dInst.src1)||sb.search2(dInst.src2))) begin
            read rVal1 and read rVal2 from rf
            sb.insert(dInst.dst); // to stall future inst for data hazard
            enqueue into e2d fifo: pc, ppc, epoch, rVal1, rVal2, dInst
        end else begin
        end
    end else begin
        f2d.deq; ...
    end
endrule
```
Fixing the doDecode rule

```verilog
rule doDecode;
    let inst <- iMem.resp;
    let x = f2d.first; f2d.deq;
    if (x.epoch == epoch) begin
        let dInst = decode2(inst); // src1 and src2 are Maybe types;
        if (!((sb.search1(dInst.src1)||sb.search2(dInst.src2)))) begin
            read rVal1 and read rVal2 from rf
            sb.insert(dInst.dst); //to stall future inst for data hazard
            enqueue into e2d fifo. pc, ppc, epoch, rVal1, rVal2, dInst
            f2d.deq;
        end else begin
            end
        end else begin
            f2d.deq; ...
        end
endrule
```
Fixing the doDecode rule

```
rule doDecode;
let inst <- iMem.resp;
let x = f2d.first; f2d.deq;
if (x.epoch == epoch) begin
    let dInst = decode2(inst); // src1 and src2 are Maybe types;
    if (!(sb.search1(dInst.src1)||sb.search2(dInst.src2))) begin
        read rVal1 and read rVal2 from rf
        sb.insert(dInst.dst); //to stall future inst for data hazard
        enqueue into e2d fifo: pc, ppc, epoch, rVal1, rVal2, dInst
        f2d.deq;
    end else begin
    end else begin
    f2d.deq; ... end
endrule
```

stalled instruction must be saved
Fixing the doDecode rule

```
rule doDecode;
    let inst <- iMem.resp;
    let x = f2d.first; f2d.deq;
    if (x.epoch == epoch) begin
        let dInst = decode2(inst); // src1 and src2 are Maybe types;
        if (!(sb.search1(dInst.src1)||sb.search2(dInst.src2))) begin
            read rVal1 and read rVal2 from rf
            sb.insert(dInst.dst); //to stall future inst for data hazard
            enqueue into e2d fifo. pc, ppc, epoch, rVal1, rVal2, dInst
            f2d.deq;
        end else begin
            fetchedInst <= inst; ...
        end
    end else begin
        fetchedInst <= inst; ...
    end
endrule
```

stalled instruction must be saved
Fixing the doDecode rule

```verilog
rule doDecode;
    let inst <- iMem.resp;
    let x = f2d.first; f2d.deq;
    if (x.epoch == epoch) begin
        let dInst = decode2(inst); // src1 and src2 are Maybe types;
        if (!(sb.search1(dInst.src1)||sb.search2(dInst.src2))) begin
            read rVal1 and read rVal2 from rf
            sb.insert(dInst.dst); //to stall future inst for data hazard
            enqueue into e2d fifo. pc, ppc, epoch, rVal1, rVal2, dInst
            f2d.deq;
        end else begin
            fetchedInst <= inst; ...
        end
    end else begin
        f2d.deq; ...
    end
endrule
```

stalled instruction must be saved
Fixing the doDecode rule

rule doDecode;

let inst <- iMem.resp;

let x = f2d.first; f2d.deq;

if (x.epoch == epoch) begin

let dInst = decode2(inst); // src1 and src2 are Maybe types;

if (!(sb.search1(dInst.src1)||sb.search2(dInst.src2))) begin
read rVal1 and read rVal2 from rf
sb.insert(dInst.dst); //to stall future inst for data hazard
enqueue into e2d fifo. pc, ppc, epoch, rVal1, rVal2, dInst
f2d.deq;
end else begin

dInst = decode2(inst); // src1 and src2 are Maybe types;

end else begin
fetchedInst <= inst; ...
end

end else begin f2d.deq; ... end

endrule

stalled instruction must be saved

No new instruction should be fetched in the stalled state

Lab 7
Execute rule

rule doExecute (...);
    ... filter wrong-path instructions ...
    ... execute (dInst, rval1,rval2, pc) ...
    ... detect and handle misprediction ...
    ... launch memory instructions if needed ...
    ... save info for the LoadWait step ...
endrule

rule doLoadWait (...);
    ...
endrule
Execute rule

```plaintext
rule doExecute (...);
  ... filter wrong-path instructions ...
  ... execute (dInst, rval1,rval2, pc) ...
  ... detect and handle misprediction ...
  ... launch memory instructions if needed ...
  ... save info for the LoadWait step ...
endrule

rule doLoadWait (...);
  ...
endrule
```

`sbc.remove` has to be inserted whenever an instruction completes execution.
Execute rule

rule doExecute (...);
  ... filter wrong-path instructions ...
  ... execute (dInst, rval1, rval2, pc) ... 
  ... detect and handle misprediction ... 
  ... launch memory instructions if needed ... 
  ... save info for the LoadWait step ... 
endrule

rule doLoadWait (...);
  ...
endrule

sb.remove has to be inserted whenever an instruction completes execution