Recap

Interleaving behavior

Relaxed memory behaviors

X = 1;

a = Y;

Y = 1;

b = X;

X = Y = 0

a = 0, b = 0
Sequential consistency is too restrictive for this program behavior

It requires some relaxation to justify the program behavior

• Ordering of the instructions
Order relaxation

No order for Store – Load

\[
\begin{align*}
X &= Y = 0 \\
X &= 1; & Y &= 1; \\
a &= Y; & b &= X; \\
a &= 0, b &= 0
\end{align*}
\]
Order relaxation

No order for Store – Load

\[ X = Y = 0 \]

\[ X = 1; \quad Y = 1; \]
\[ a = Y; \quad b = X; \]

\[ a = 0, \quad b = 0 \]

\[ a = Y; \quad b = X; \]
\[ X = 1; \quad Y = 1; \]

\[ a = 0, \quad b = 0 \]
Order relaxation

No order for Store – Load

X = Y = 0

X = 1;

Y = 1;

a = Y;

b = X;

a = 0, b = 0

a = Y; // 0

b = X; // 0

X = 1;

Y = 1;

a = 0, b = 0

Is it allowed by any relaxed memory model?
## Total Store Order (TSO)

### Relaxation for Store – Load

<table>
<thead>
<tr>
<th>a;b</th>
<th>Store</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Load</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Another Program: Message Passing (MP)

X = Y = 0

X = 1;  
Y = 1;  

a = Y;

if(a == 1)
    b = X;

Is a=1, b=0 possible an allowed behavior?

• SC ?
• TSO ?
Program: Message Passing (MP)

X = Y = 0

X = 1;
Y = 1;
if (a == 1)
    b = X;

X = Y = 0

Y = 1;
X = 1;
a = Y;  // 1
if (a == 1)
b = X;

a = 1, b = 0

Requires Store-Store relaxation
Partial Store Order (PSO)

Relaxation for Store – Store

<table>
<thead>
<tr>
<th>a;b</th>
<th>Store</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Load</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Yet Another Program: Load Buffering (LB)

\[
\begin{align*}
X &= Y = 0 \\
\text{a} &= X; & \text{a} &= Y; \\
\text{Y} &= 1; & \text{X} &= 1;
\end{align*}
\]

Is \(a=1, b=1\) possible an allowed behavior?

- SC ?
- TSO ?
- PSO ?
Program: Message Passing (MP)

\[ X=Y=0 \]

\[ a = X; \quad b = Y; \]
\[ Y = 1; \quad X=1; \]

\[ X=Y=0 \]

\[ Y = 1; \quad a = X; \quad b = X; \]
\[ a =1, \ b = 1 \]

Requires Load-Store relaxation
## Relaxed Memory Order (RMO)

### Relaxation for Store – Store

<table>
<thead>
<tr>
<th>a;b</th>
<th>Store</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Load</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
## Summary

<table>
<thead>
<tr>
<th>ACC. Pair Relaxation</th>
<th>Memory Model</th>
<th>Store-Load</th>
<th>Store-Store</th>
<th>Load-Load</th>
<th>Load-Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>TSO</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>PSO</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>RMO</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Recap: Question

Given a program and an outcome, is it a correct outcome?

Depends on

program execution
More Specifically ...

Given a program and an outcome, is it a correct outcome?

Depends on

program execution

Depends on

Concurrency semantics/memory model
Memory Model

‘In computing, a memory model describes the interactions of threads through memory and their shared use of the data.’ ~ wikipedia

e.g.

- SC
- TSO
- PSO
- RMO
Defining a Concurrency Semantics

- Transfomational
- Operational
- Axiomatic
Transformational Semantics

Step 1: Transform the program

Step 2: Consider all interleaving executions

Reordering & interleaving explains TSO, PSO, RMO

Note: Reordering requires dependency analysis
## Dependencies

<table>
<thead>
<tr>
<th>Data-Dependence</th>
<th>Control-Dependence</th>
<th>Address-Dependence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow-Dependence</td>
<td>Output-Dependence</td>
<td>Anti-Dependence</td>
</tr>
</tbody>
</table>
## Dependencies

<table>
<thead>
<tr>
<th>Data-Dependence</th>
<th>Control-Dependence</th>
<th>Address-Dependence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow-Dependence</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output-Dependence</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anti-Dependence</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
a = X;
Y = a;
```
## Dependencies

<table>
<thead>
<tr>
<th>Data-Dependence</th>
<th>Control-Dependence</th>
<th>Address-Dependence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow-Dependence</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a = X;</td>
<td>X = 1;</td>
<td></td>
</tr>
<tr>
<td>Y = a;</td>
<td>X = 2;</td>
<td></td>
</tr>
</tbody>
</table>
## Dependencies

<table>
<thead>
<tr>
<th>Data-Dependence</th>
<th>Control-Dependence</th>
<th>Address-Dependence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow-Dependence</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output-Dependence</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anti-Dependence</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **a = X;**
- **Y = a;**
- **X = 1;**
- **X = 2;**
- **a = X;**
- **X = 1;**
# Dependencies

<table>
<thead>
<tr>
<th>Data-Dependence</th>
<th>Control-Dependence</th>
<th>Address-Dependence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow-Dependence</td>
<td>Output-Dependence</td>
<td>Anti-Dependence</td>
</tr>
<tr>
<td>a = X; Y = a;</td>
<td>X = 1; X = 2;</td>
<td>a = X; Y = 1;</td>
</tr>
<tr>
<td>a = X; Y = a;</td>
<td>X = 1; X = 2;</td>
<td>a = X; if(a==1) Y = 1;</td>
</tr>
</tbody>
</table>
## Dependencies

<table>
<thead>
<tr>
<th>Data-Dependence</th>
<th>Control-Dependence</th>
<th>Address-Dependence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow-Dependence</td>
<td>Output-Dependence</td>
<td>Anti-Dependence</td>
</tr>
<tr>
<td>a = X; Y = a;</td>
<td>X = 1; X = 2;</td>
<td>a = X; Y = 1;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>if(a==1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y = 1;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>a = X; b = Y[a];</td>
</tr>
</tbody>
</table>
True & False Dependence

True dependence: Cannot be eliminated by any transformation

\[ a = X; \]
\[ Y = a \times 5; \]

False dependence: Can be eliminated, depends upon transformations

\[ a = X; \]
\[ Y = a \times 0; \]
\[ \Rightarrow a = X; \]
\[ Y = 0; \]
Limitations of Transformational Approach

1. Dependency analysis is undecidable in general

   * cannot always differentiate true & false dependencies

   ```
   func(int m){
   :
   a = X;
   Y = a*m;    // is m = 0 here?
   }
   ```
Limitations of Transformational Approach

1. Dependency analysis is undecidable in general

   * cannot always differentiate true & false dependencies

2. Does not explain all behaviors of modern architectures
Limitations of Transformational Approach

Example:

\[
\begin{align*}
X[0] &= Y[1] = 0; \\
X[0] &= 1; & a &= X[0]; & \quad /\!/ 1 & \quad c &= Y[0]; & \quad /\!/ 1 \\
b &= Y[a*0]; & \quad /\!/ 0 & \quad d &= X[a*0]; & \quad /\!/ 0
\end{align*}
\]

Allowed in PowerPC and ARMv7

* even without removing false dependencies
Limitations of Transformational Approach

X[0] = Y[1] = 0;

\[
\begin{align*}
X[0] &= 1; \\
\text{a} &= X[0]; \quad \text{// 1} \\
\text{b} &= Y[\text{a} \times 0]; \quad \text{// 0} \\
\text{c} &= Y[0]; \quad \text{// 1} \\
\text{d} &= X[\text{a} \times 0]; \quad \text{// 0} \\
\text{Y}[1] &= 1;
\end{align*}
\]

Sequentialization:
\[
C_0 \ || \ C_1 \rightarrow C_0;C_1
\]

\[
\begin{align*}
X[0] &= 1; \\
\text{a} &= X[0]; \\
\text{b} &= Y[\text{a} \times 0]; \quad \text{// 0} \\
\text{c} &= Y[0]; \\
\text{d} &= X[\text{a} \times 0]; \quad \text{// 0} \\
\text{Y}[1] &= 1;
\end{align*}
\]
Limitations of Transformational Approach

\[
\begin{align*}
X[0] &= 1; \\
a &= X[0]; \\
b &= Y[a*0]; & // 0
\end{align*}
\]

\[
\begin{align*}
Y[1] &= 1; \\
c &= Y[0]; \\
d &= X[a*0]; & // 0
\end{align*}
\]

\[
\begin{align*}
X[0] &= 1; \\
a &= 1; \\
b &= Y[a*0]; & // 0
\end{align*}
\]

\[
\begin{align*}
Y[1] &= 1; \\
c &= 1 \\
d &= X[a*0]; & // 0
\end{align*}
\]

constant propagation
Limitations of Transformational Approach

Another Example: Allowed by ARMv8.

```
X = Y = 0;

a = X; // 1
X = 1;

Y = X;
X = Y;
```
Limitations of Transformational Approach

1. Dependency analysis is undecidable in general
   * cannot always differentiate true & false dependencies

2. Does not explain all behaviors of modern architectures

3. Assumes correctness of transformations
   * instead of proving them correct!