Assignment 5.1 Transactional Memory

Tick the correct answer.

1. A TM that uses pessimistic concurrency control provides opacity
   - true
   - false

2. Eager conflict detection implies eager version management
   - true
   - false

3. Conflict occurrence, detection, and resolution happen at once in some TM systems. These systems make use of
   - pessimistic concurrency control
   - optimistic concurrency control
   - eager conflict detection
   - validation conflict detection
   - lazy conflict detection
4. A conflict occurs between two threads A and B where B accesses memory in a non-transactional way. The transaction of A consists of two statements $s_1$, $s_2$. B has observed the state after A has executed $s_1$ but before A has executed $s_2$.

- A and B will abort.
- The TM system uses lazy version management.
- The TM does not provide single-lock atomicity.
- The TM does not provide transactional sequential consistency.

5. A zombie transaction crashes the whole program due to reading unexpected and

6. A TM implementation detects a conflict in the middle of a transaction although the actual write that created the conflict lies in the past. This kind of conflict detection is called

- eager conflict detection.
- tentative conflict detection.
- validation conflict detection.
- lazy conflict detection.

7. A TM implementation uses optimistic concurrency control. Which of the following statements must be false?

- Because a transaction is a zombie, the program crashes after a write through a pointer that is NULL.
7. A TM implementation uses optimistic concurrency control. Which of the following statements must be false?

- Because a transaction is a zombie, the program crashes after a write through a pointer that is NULL.
- The implementation checks for conflict only when committing.
- A transaction can be suspended until a conflicting transaction has finished.

Assignment 5.2 Restricted Transactional Memory

Consider the following code fragment on a machine with RTM and Caches:

```c
int data = 0;
int s = 0;

thread P1:

while (s!=1)
    if (_xbegin() == -1)
        data++;
    else
        _xend();
    data++;

thread P2:

if (_xbegin() == -1)
    data++;  // Should be _xbegin() == -1
    if (s-_xbegin() == -1)
        data++;
    _xend();
    else
        data++;
```

1. Fill in the gap with either “will”, “will not” or “may or may not”:

   After P1 and P2 both terminate, data ______ evaluate to 1.

2. Fill in the gap with either “will”, “will not” or “may or may not”:

   After P1 and P2 both terminate, data ______ evaluate to 3.

3. Consider the following interleaving of paths through the program:

   Draw a happened-before diagram for the interleaving. The initial cache states of S0 and data are S0, S0.
Draw a happened-before diagram of this interleaving. The initial cache states for \texttt{a} and \texttt{data} are \texttt{S0, S0}.

\begin{center}

\begin{tikzpicture}
  \node (P1) at (0,0) {
    \begin{verbatim}
    \end{verbatim}
  };
  \node (P2) at (0,-1) {
    \begin{verbatim}
    \end{verbatim}
  };
  \draw (P1) -- (P2);
\end{tikzpicture}
\end{center}

1. Fill in the gap with either “will”, “will not” or “may or may not”.

After \texttt{P0} and \texttt{P1} both terminate, data \underline{________} evaluate to 1.

```c
int data = 0;
int s=0;

thread P0;

\textbf{while} (s!=-1)

\textbf{if} (_xbegin()==-1){
  data++;
}
\textbf{if}((s=_xbegin())==-1){
  data++;
  \_xend();
}\textbf{else} {
  \_xend();
  data++;
}
```