(1) How can “race condition” happen? Show “how” using an example.

Race conditions happen when machine codes (or processor instructions) for updating the contents of a shared resources, shared by more than one process, are interrupted (or context-switched) in an unpredictable timing as shown below:

![Race Condition Diagram]

**Description:**

When \( P_1 \) is preempted (interrupted) right after it executes the second instruction (ADD), \( P_2 \) may start execution. When \( P_1 \) resumes execution of its 3rd instruction, after \( P_2 \) executes its three instructions, the content of ‘A’ will be “5”, which is a wrong result. This is a result of the race condition.

**Note:** your example must use processor instructions (assembly instructions) to show how race condition can happen. I do not think any example that does not use processor instructions can explain how race conditions can happen.
(2) What is “mutual exclusion”?

Mutual exclusion is a solution for a race condition, which makes sure only one process (can execute instructions) in a critical section at a time.

(3) What does “atomic” in “atomic operations” mean?

The term, “atomic” in atomic operations” mean that once an operation (or an activity) starts execution, its execution will never be interrupted (or “never be preempted”) until that operation is completed.

(4) What “wait” system call to a semaphore exactly performs?

The “wait” system call on a semaphore performs the following activities (as an atomic operation):

- If a semaphore is ‘1’, the OS decreases the content of the semaphore to ‘0’. Then, the OS lets the process (the process that calls “wait”) to proceed into the critical section.
- If a semaphore is ‘0’, the OS brings the calling process to wait (i.e., to the “blocked” state in the short-term scheduler).

Note: both of the two descriptions (one for S = ‘1’ and the other for S = ‘0’) are necessary for full credit.

\[
\text{Wait} \\
\text{• If } S > 0, \text{ do } S = S - 1 \text{ then proceed} \\
\text{• If } S = 0, \text{ wait on this semaphore}
\]

(5) Operating systems use “queue (FIFO data structure)” for managing processes blocked on a semaphore. Why is FIFO-queue used (the best reason for using FIFO structure)?

If waiting (blocked) processes are not started in the FIFO order, a semaphore can cause starvation to the waiting processes. To prevent process starvation from happening, operating systems use a FIFO queue for managing blocked processes.