Got it — you want to **start an async** `Task`, but before continuing with other instructions, you need to ensure that the task has **at least reached a certain point in its execution** (not necessarily completed).

This is a common pattern when you want to "sync up" at a specific checkpoint inside an async method.

✅ Solution: Use `TaskCompletionSource` as a Signal

You can use a `TaskCompletionSource` to signal when the async method has reached a specific region.

Example:

```
using System;
using System.Threading.Tasks;

class Program
{
    static async Task Main()
    {
        // Create a signal object
        var checkpointReached = new TaskCompletionSource<bool>();

        // Start the async task
        var task = DoWorkAsync(checkpointReached);

        // Wait until the async method signals it reached the checkpoint
        await checkpointReached.Task;
        Console.WriteLine("Main: Checkpoint reached, proceeding with other instructions...");

        // Continue with other work
        Console.WriteLine("Main: Doing other work while task continues...");
        // Optionally wait for the task to finish
        await task;
        Console.WriteLine("Main: Task completed.");
    }

    static async Task DoWorkAsync(TaskCompletionSource<bool>
checkpoint)
    {
        Console.WriteLine("Task: Starting work...");

        // Simulate some initial work
        await Task.Delay(1000);

        // Signal that we reached the checkpoint
        checkpoint.TrySetResult(true);
        Console.WriteLine("Task: Checkpoint reached, continuing
work...");

        // Simulate more work
        await Task.Delay(2000);
        Console.WriteLine("Task: Finished work.");
    }
}
```

How It Works

- `TaskCompletionSource<bool>` acts as a manual signal.
- The async method calls `checkpoint.TrySetResult(true)` when it reaches the desired region.
- The main method awaits `checkpoint.Task` to pause until the signal is set.
- After the checkpoint, both the main method and the async method can continue independently.

🔹 Benefits

- Works without blocking threads (`await` is non-blocking).
- Allows precise synchronization at any point inside an async method.
- Avoids race conditions compared to polling or delays.

If you want, I can also show you a **thread-safe version** that supports **multiple checkpoints** so you can synchronize at multiple stages of the async task.

Do you want me to prepare that advanced version?