breakpoints has the disadvantage of severely limiting the number of memory addresses that can be concurrently watched for. For example, the INTEL x86 architecture provides only four debug registers (i.e., DR0 to DR3) for storing memory addresses to watch for. Use of hardware breakpoints also has the disadvantage of inherently binding a debugger implementation to specifically-supported hardware (e.g., by relying on the x86 debug registers).

[0005] Another mechanism that has been explored is to use software breakpoints, in which each traced operation that accesses memory (e.g., a read or a write to memory storing data, or a read/fetch for execution from memory storing program code) is instrumented with additional code instructions that check whether or not the access is to a memory address of interest. However, instrumentation has the severe disadvantage of causing a check for the occurrence of a breakpoint to be performed for every memory access, regardless of whether or not that memory access would actually correspond to a breakpoint. In many environments, execution of the instrumented code instructions results in the use of tens to hundreds of additional processor cycles for each memory access, which has a significant detrimental effect on replay performance that is perceivable by a human user.

[0006] Previous solutions by the inventor to address the foregoing challenges through use of caches (e.g., a cache of a hardware or emulated processor, or some other cache memory structure) as part of breakpoint detection. These solutions greatly limit the number of processor cycles needed to perform a breakpoint check as compared to instrumentation, and are described in U.S. Patent Application Ser. No 15/405,849, filed January 13, 2017, and titled“EFFICIENT BREAKPOINT DETECTIONS VIA CACHES,.” the entire contents of which are incorporated by reference in their entirety.

[0007] These solutions use a cache to limit breakpoint checks to only memory accesses that cause cache misses to occur, rather than performing a breakpoint check on every memory access. These solutions can offer great performance improvements over non-cache-based breakpoint detection solutions, while providing a virtually limitless number of breakpoints. They involve evicting a cache line if it overlaps with a memory address that corresponds to a breakpoint, and then“single stepping” the processor on each cache miss in order to determine if a breakpoint was encountered. While being an advancement over instrumentation for breakpoint detection, these solutions severely impact execution performance when cache misses occur (even if no breakpoint was hit in connection with the cache miss), and also cause cache misses to occur more frequently than they might absent use of the cache for breakpoint detection (i.e., because they proactively evict cache lines that overlap with a watched-for memory address).