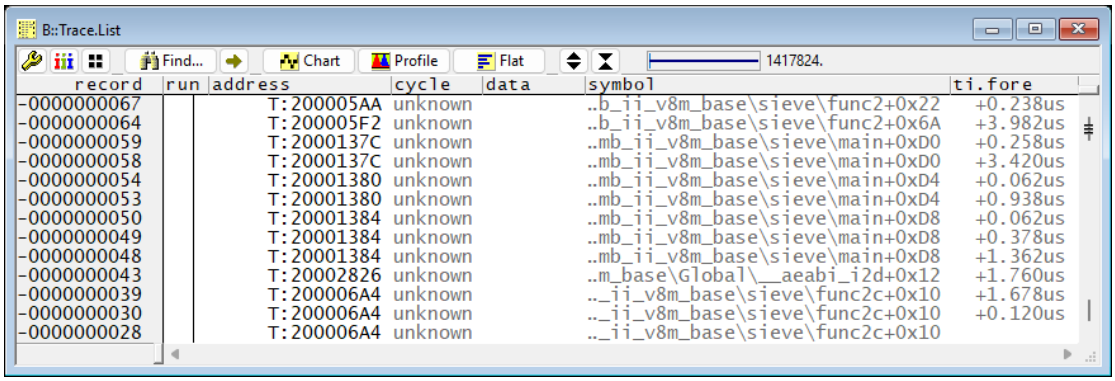
Why does the Trace.List show “unknown” instead of program code?

2026-07-28 - Comments (0) - TRACE32 PowerView

The trace hardware records only a subset of the executed instruction stream (typically branch information and other trace messages), not every executed instruction. TRACE32 reconstructs the complete execution flow by combining the recorded trace information with the program code stored in memory.

The unknown indication means that TRACE32 was able to decode the trace addresses, but could not retrieve the corresponding instructions from the program memory. Without access to the instruction bytes, the debugger cannot reconstruct the executed code. So, the “unknown” indication does not necessarily mean that trace data is missing or corrupted. It simply means that TRACE32 cannot reconstruct the instructions because the corresponding program code is unavailable.

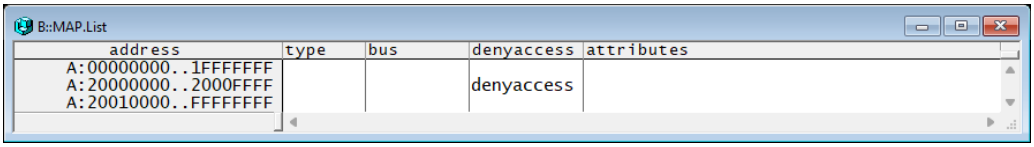


record	run	address	cycle	data	symbol	ti.fore
-0000000067		T:200005AA	unknown		..b_ii_v8m_base\sieve\func2+0x22	+0.238us
-0000000064		T:200005F2	unknown		..b_ii_v8m_base\sieve\func2+0x6A	+3.982us
-0000000059		T:2000137C	unknown		..mb_ii_v8m_base\sieve\main+0xD0	+0.258us
-0000000058		T:2000137C	unknown		..mb_ii_v8m_base\sieve\main+0xD0	+3.420us
-0000000054		T:20001380	unknown		..mb_ii_v8m_base\sieve\main+0xD4	+0.062us
-0000000053		T:20001380	unknown		..mb_ii_v8m_base\sieve\main+0xD4	+0.938us
-0000000050		T:20001384	unknown		..mb_ii_v8m_base\sieve\main+0xD8	+0.062us
-0000000049		T:20001384	unknown		..mb_ii_v8m_base\sieve\main+0xD8	+0.378us
-0000000048		T:20001384	unknown		..mb_ii_v8m_base\sieve\main+0xD8	+1.362us
-0000000043		T:20002826	unknown		..m_base\Global__aeabi_i2d+0x12	+1.760us
-0000000039		T:200006A4	unknown		.._ii_v8m_base\sieve\func2c+0x10	+1.678us
-0000000030		T:200006A4	unknown		.._ii_v8m_base\sieve\func2c+0x10	+0.120us
-0000000028		T:200006A4	unknown		.._ii_v8m_base\sieve\func2c+0x10	

Common reasons include:

- **The program memory is no longer accessible**: The debugger cannot read the target memory. Opening a `List` window at the same address typically results in a “bus error”, confirming that the memory cannot be accessed.

- **Memory access is blocked by `MAP.DenyAccess`**: A `MAP.DenyAccess` setting may prevent TRACE32 from reading the corresponding code area. Verify the active memory access restrictions in the `MAP.List` window.



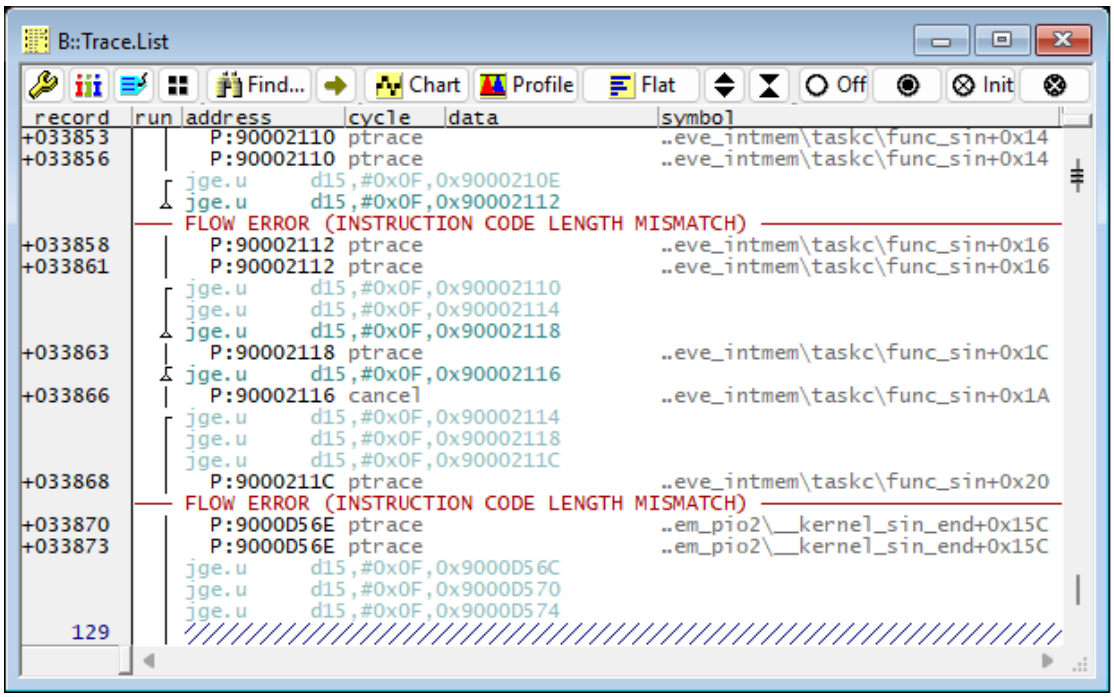
address	type	bus	denyaccess	attributes
A:00000000..1FFFFFFF				
A:20000000..2000FFFF			denyaccess	
A:20010000..FFFFFFFF				

- **User-space trace without task context**: When tracing an operating system with MMU enabled, task switch information is necessary in order to decode user-space addresses. If TRACE32 does not know which task was executing, it cannot determine the correct page table or virtual-to-physical address translation for the traced addresses.

- **The traced task no longer exists**: In operating systems with dynamic task creation and deletion, the traced task may have already terminated. Its address space is therefore no longer available, preventing TRACE32 from resolving the recorded virtual addresses.

Note

If the memory contents no longer match the code that was originally executed—for example, because boot code has been overwritten or the application has relocated itself—the disassembly displayed in the `Trace.List` window will no longer correspond to the executed instructions. As a result, TRACE32 may be unable to reconstruct the execution flow correctly, often leading to **FLOW ERROR** indications.



Workaround

If the original target memory is no longer available, you can instruct TRACE32 to use the debugger's Virtual Memory (VM) instead of reading instructions from the target.

```
Data.LOAD.Elf <file> /VM
Trace.ACCESS VM
```

Related Content:

- Why does the Trace.List show "NOACCESS"?