## Intel-PT Tracing Evaluation

### Test Case t0 – Control

**This test case represents the minimum control-flow complexity within the main execution function of a service**. It simply enters and returns with no branching in between the start and termination program commands.

There are only two control-flow state transitions to report here:

1. Execution enters the trace context at the start of main.
2. Execution leaves the trace context via a return statement.



Figure 1: Test case t0 trace output

### Test Case t1 - Single Conditional Branch

**Test case t1 has a very simple branching structure: a single if-statement.** The condition is set such that it always evaluates to “false” and the conditional branch skipping the contents of the statement is taken.

The instantiation of the variable a is not registered, as it produces linear control-flow and is, thus, ignored in this context.



Figure 2: Test case t1 trace output

### Test Case t2 – Loop

This test case adds a loop in the underlying codebase. **It is a very simple while-statement that runs twice and contains no additional branching structures.** It is important to note here that the loop starts with an unconditional jump instruction to the set of instructions that evaluate the loop condition and jump to the top of the loop’s contents based on that condition. The initial, unconditional jump is ignored by the decoder as it is predictable with static analysis and is therefore unnecessary to profile the runtime control-flow of the program.

The loop can be seen as two identical jump events. In the first case, a = 0, so the condition a < 2 evaluates to true and the conditional jump branches to the top of the loop. The second time, a = 1, and the loop repeats. The third time, however, the condition a < 2 evaluates to “false” and the jump is not taken.

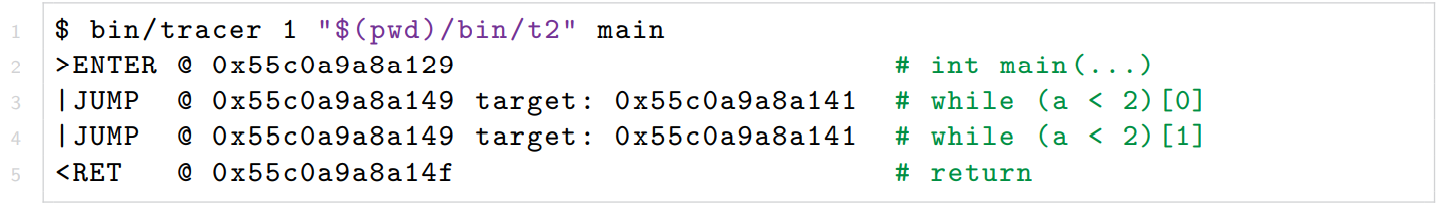


Figure 3: Test case t2 trace output

### Test Case t3 - Function Call

**Test case t3 adds a function call to the control-flow structure of the program. It is a simple integer function that always returns 0.** Note that the added function lies outside the tracing context.

There are effectively two entry - and exit points in the main function.

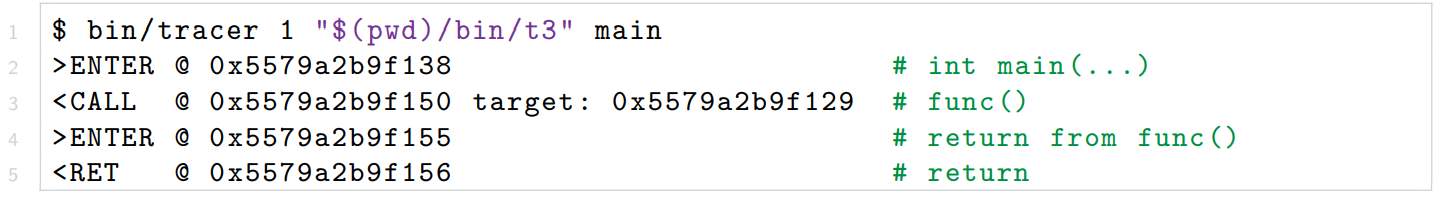


Figure 4:Test case t3 trace output

### Test Case t4 - Conditional Branch Within Loop

**This test case combines t1 and t2 with a single if-statement inside a loop.**

The output clearly indicates the combination of these control-flow structures.

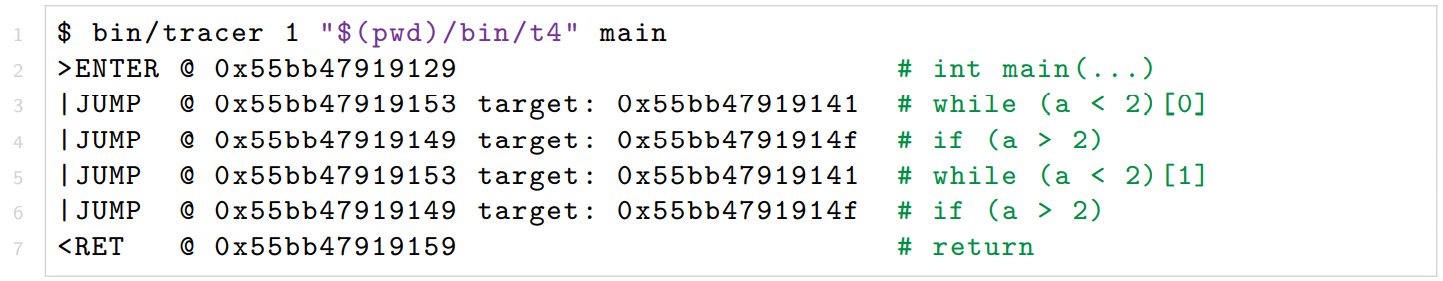


Figure 5: Test case t4 trace output

### Test Case t5 - Function Call Within Loop

**This test case combines t2 and t3 with a function call inside a loop. This should cause execution to repeatedly leave and re-enter the trace context.**

The output shows the expected combination of the control-flow structures displayed in t2 and t3.

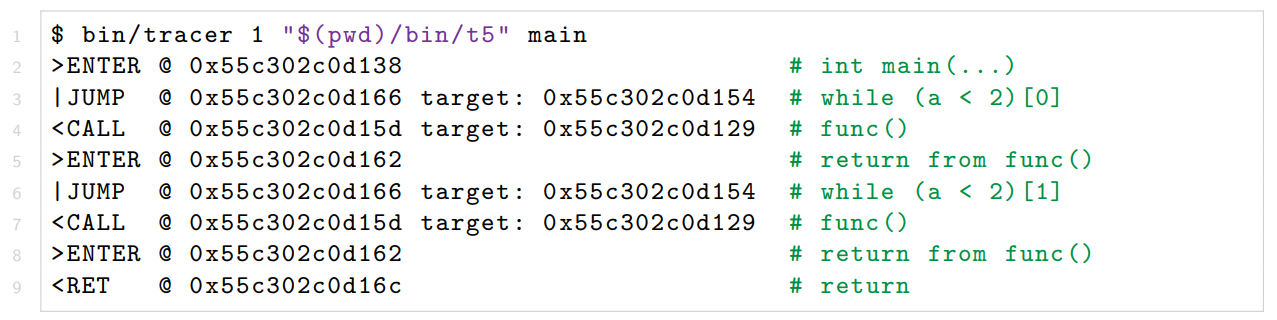


Figure 6: Test case t5 trace output

### Test Case t6 - Function Call and Conditional Branch Within Loop

**This test case does just that, combining t4 and t5 to form a simple program with a loop containing a conditional branching instruction and a function call.**

The output displays the control-flow of the program as a combination of its constituent structures.

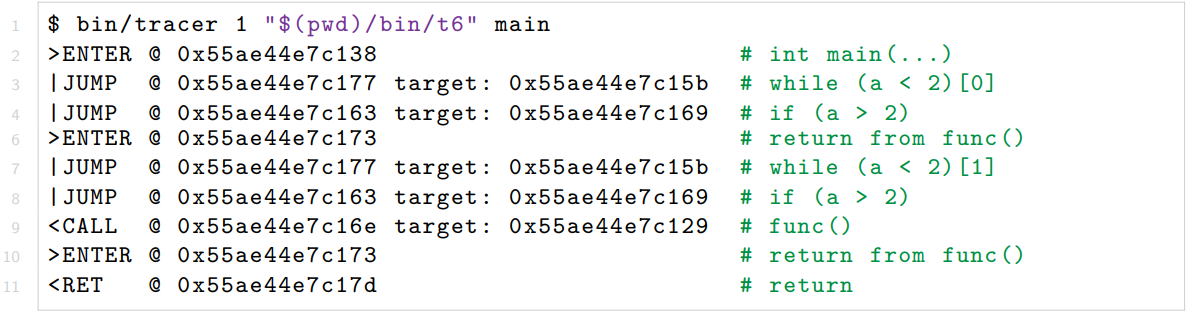


Figure 7:Test case t6 trace output

### Test Case t7 - High Complexity Scenario

The target program t7 is designed to facilitate the two scaling factors selected. **It contains two loops, one in the main function and another inside a function outside the trace context, which is called in every iteration of the main loop.**

|  |  |
| --- | --- |
| Figure 8: 1 iteration out-of-context | Figure 9: 10 iteration out-of-context |
| Figure10: 100 iteration out-of-context | Figure 11: 1,000 iterations out-of-context |

|  |
| --- |
| Figure 12: 10,000 iterations out-of-context |

### Test Case t8 - Low Complexity Scenario

The target program t8 is designed to scale by the two selected factors. It contains a single loop in the main function. As in the high-complexity testing, the primary scaling factor controls the number of iterations in the main loop.

|  |  |
| --- | --- |
| Figure 13: 1 instruction per loop | Figure 14: 10 instruction per loop |
| Figure 15: 100 instruction per loop | Figure 17: 1000 instruction per loop |

|  |
| --- |
| Figure 17: 10,000 instructions per loop |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | |