Understanding the BusFault Exception with LR Value 0x8022e17

The issue you're experiencing with the Cortex-M3 BusFault exception and the seemingly impossible Link Register (LR) value can be explained by understanding how ARM Cortex-M exception handling works, particularly regarding instruction pipeline behavior and exception entry mechanics.

The Root Cause: Exception Entry During Instruction Pipeline

The key to understanding this behavior lies in how the Cortex-M3 processor handles exceptions during instruction execution pipelines.[1] [2]

When a BusFault exception occurs, the processor doesn't necessarily stop execution at the exact instruction that caused the fault. Instead, due to the **pipelined nature** of the Cortex-M3 processor, several instructions may be in various stages of execution simultaneously.[3] [4]

Why LR Points Between Instructions

The LR value of **0x8022e17** falls between two instructions:

- `8022e16: e5c3 b.n 80229a0 <main+0x8b8>`
- `8022e18: f3bf 8f4f dsb sy`

This is actually **normal behavior** for several reasons:[5] [2] [1]

1. Instruction Pipeline Effects

The Cortex-M3 implements instruction prefetching and pipeline processing. When a BusFault occurs during memory access (like your `*ptr = i` assignment), the processor may have already prefetched subsequent instructions. The exception can be triggered while these prefetched instructions are still in the pipeline.[3]

2. Precise vs Imprecise Bus Faults

According to ARM documentation, there are two types of bus faults:[2] [1] [5]

- **Precise faults**: The PC value stacked for exception return points to the instruction that caused the fault
- **Imprecise faults**: The return address is not directly related to the instruction that caused the error due to write buffer delays

Your fault appears to be hitting during the transition between instructions, which suggests an **imprecise bus fault** scenario.

3. Exception Entry Timing

The ARM Cortex-M3 exception entry process involves:[4] [6]

- Automatic stacking of registers (R0-R3, R12, LR, PC, xPSR)
- Setting LR to an EXC_RETURN value (not the actual return address)
- The stacked PC represents where execution should resume

Debugging Analysis

Bus Fault Status Register (BFSR) Examination

To determine the exact cause, check the **BFSR** register at `0xE000ED29`:[5]

```
uint8_t bfsr = *(uint8_t*)0xE000ED29;
if (bfsr & 0x80) {
    // BFARVALID - Bus Fault Address Register is valid
    uint32_t fault_address = *(uint32_t*)0xE000ED38; // BFAR
    // This contains the address that caused the fault
}
if (bfsr & 0x02) {
    // PRECISERR - Precise data bus error
}
if (bfsr & 0x04) {
    // IMPRECISERR - Imprecise data bus error
}
```

Understanding the Assignment Fault

Your problematic code:

```
*ptr = i;  // Line causing the BusFault
```

This suggests that `ptr` points to an **invalid memory address** or a **memory region with access restrictions**. The bus fault occurs when the processor attempts to write to this location.[7] [1]

Why LR = 0x8022e17 is Possible

The LR value **0x8022e17** is actually a **valid scenario** because:[8] [1] [2]

1. **Exception Pipeline Interaction**: The exception can be triggered during instruction fetch or execution, not necessarily at instruction boundaries
2. **Instruction Alignment**: ARM Thumb instructions can be 16-bit or 32-bit, and the processor may be in the middle of fetching/decoding when the exception occurs[9] [10]

3. **Write Buffer Effects**: If write buffering is enabled, the bus fault may be detected after the processor has moved past the faulting instruction[11]

Recommended Debugging Steps

1. **Check BFSR and BFAR registers** to identify the exact fault type and address
2. **Examine the `ptr` variable value** - it likely contains an invalid address
3. **Verify memory mapping** - ensure the target address is within valid RAM/peripheral regions
4. **Disable write buffer temporarily** using the DISDEFWBUF bit in the Auxiliary Control register to make faults more precise[11]
5. **Implement a comprehensive fault handler** that captures all relevant registers and fault status information[5]

Conclusion

The LR value pointing between instructions is **not impossible** but rather a consequence of the Cortex-M3's pipelined architecture and exception handling timing. The real issue is likely that your `ptr` variable contains an invalid memory address, causing the bus fault during the write operation. Focus your debugging efforts on validating the `ptr` value and ensuring it points to accessible memory regions.

⁂

1. https://community.st.com/ysqtg83639/attachments/ysqtg83639/mcu-touch-gfx-gui-forum/17794/1/Keil_app_note_209_fault_exceptions_Rev2.pdf
2. https://www.keil.com/appnotes/files/apnt209.pdf
3. https://s-o-c.org/cortex-m3-instruction-prefetching-and-branch-prediction-explained/
4. https://interrupt.memfault.com/blog/arm-cortex-m-exceptions-and-nvic
5. https://interrupt.memfault.com/blog/cortex-m-hardfault-debug
6. https://www.youtube.com/watch?v=oQ8Mc2TuYPU
7. https://stackoverflow.com/questions/62812939/how-to-generate-exceptions-on-cortex-m3
8. https://stackoverflow.com/questions/79622568/how-does-bx-lr-work-on-an-exception-return-in-cortex-m-processor
9. https://stackoverflow.com/questions/1554497/switching-between-arm-thumb-state
10. https://s-o-c.org/thumb-vs-arm-instructions-for-cortex-m3/
11. https://community.st.com/t5/stm32-mcus/how-to-debug-a-hardfault-on-an-arm-cortex-m-stm32/ta-p/672235
12. https://documentation-service.arm.com/static/5e8e107f88295d1e18d34714
13. https://community.st.com/t5/stm32-mcus-products/how-to-manually-set-the-values-of-sp-lr-register-of-a-stm/td-p/143562
14. https://www.ti.com/lit/pdf/spmz092
15. https://community.st.com/t5/stm32-mcus-products/stm32f7-bus-fault-with-address-in-nowhere-land/td-p/144609

16. https://stackoverflow.com/questions/40855088/how-to-return-from-a-cortex-m3-hard-usage-bus-fault
17. https://ucgosu.pl/2024/10/debugowanie-hard-faultow-na-stm32/
18. https://s-o-c.org/link-register-and-program-counter/
19. https://en.wikipedia.org/wiki/Link_register
20. https://stackoverflow.com/questions/74913735/arm-link-registerlr-usage-between-multiple-function-calls
21. https://www.reddit.com/r/embedded/comments/n11rtq/arm_lr_link_register_vs_stack_pointer_for_storing/
22. https://students.mimuw.edu.pl/~zbyszek/asm/slides/arm_inst.pdf
23. https://stackoverflow.com/questions/41790296/isb-instruction-in-arm-cortex-m
24. https://stackoverflow.com/questions/38618440/retrieving-return-address-of-an-exception-on-arm-cortex-m0
25. http://eecs.umich.edu/courses/eecs373/readings/ARM_Cortex_AppNote179.pdf
26. https://stackoverflow.com/questions/21874925/how-to-set-up-the-exception-handlers-for-cortex-m3
27. https://stackoverflow.com/questions/79005949/exception-return-issues-with-cortex-m3-on-stm32f103c8t6
28. https://www.reddit.com/r/embedded/comments/ptaxtp/some_questions_about_the_link_register_r14_on_a/
29. https://false.ekta.is/2013/05/unaligned-memory-access-fault-on-cortex-m3/
30. https://stackoverflow.com/questions/8806406/how-do-i-debug-unexpected-resets-in-a-stm32-device
31. https://stackoverflow.com/questions/33777128/find-which-instruction-caused-a-trap-on-cortex-m3
32. https://stackoverflow.com/questions/47001748/hardfault-cortex-how-do-i-debug-it
33. https://developer.arm.com/documentation/ddi0337/e/Nested-Vectored-Interrupt-Controller/NVIC-programmer-s-model/NVIC-register-descriptions