Address Sanitizer on Bare-Metal Targets
Why does address sanitizer not work for bare-metal today?

- **Targets I'm considering**
  - Cortex-M
  - MPU but no MMU, fixed address space layout
  - Code and RO data in ROM
  - Memory-mapped devices, control registers

- **Instrumentation**
  - Assumes contiguous 1/8th of address space reserved
    - Only practical with MMU
    - Must all be RAM
  - All allocation sizes increased to add redzone

- **Library (compiler-rt/lib/asan)**
  - Assumes "full" operating system
    - Filesystem, mmap, spawning processes, access to own image
  - Linux kernel doesn't use this, provides own runtime library
#define SHADOW_OFFSET 0x1c000000

```c
uint64_t load_64(uint64_t *a) {
    char *shadow = (a >> 3) + SHADOW_OFFSET;
    if (*shadow)
        __asan_report_load8(a);
    return *a;
}
```
Typical Cortex-M memory map (simplified)
Shadow memory – range checks

```c
#define SHADOW_OFFSET 0x1c000000
#define MAPPING_MIN 0x20000000
#define MAPPING_MAX 0x40000000

uint32_t load_64(uint32_t *a) {
    char *shadow = (a >> 3) + SHADOW_OFFSET;
    if (shadow >= MAPPING_MIN && shadow < MAPPING_MAX)
        if (*shadow)
            __asan_report_load8(a);
    return *a;
}
```
#define SHADOW_OFFSET 0x0
#define MAPPING_MIN 0x20000000
#define MAPPING_MAX 0x40000000

uint64_t load_64(uint64_t *a) {
    char *shadow = (a & 0xffffffff) | \n        ((a & 0xffffffff) >> 3);
    if (shadow < MAPPING_MAX) {
        if (*shadow)
            __asan_report_load8(a);
    }
    return *a;
}
Shadow memory in ROM

- Prototype implemented in armlink
- Compiler generates shadow sections, with SHF_LINK_ORDER pointing to global
- User places these in correct memory region with scatter file
- Armlink fixes up addresses to match shadow mapping function
Demo – heap buffer overflow

+ runner.pl --cpu=8-M.main --image=demo_asan.axf --args 1
Running demo_heap_overflow:
p: 20700178
p[2]: 0x2a2a
asan failure, load of 2 bytes at 0x2070017e, PC=0x132a1: heap buffer overflow
Stack trace:
  pc 0x13c06 sp 0x20ffffffb8
  pc 0x13a97 sp 0x20ffffffd8
  pc 0x140e1 sp 0x20fffffff0
  pc 0x132a1 sp 0x20fffffff0
  pc 0x1423f sp 0x20fffffff8
0x206fff80: cf df df df df df df df df df df df df df df
0x20700000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20700080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20701000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20700180: f0 f0 fd fd fd fd fd fd fd fd fd fd fd fd fd
0x20700200: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd
0x20700280: fd fd fd fd fd fd fd fd fd fd fd fd fd fd fd

SIGABRT: Abnormal termination
Demo – heap use after free

+ runner.pl --cpu=8-M.main --image=demo_asan.axf --args 0
Running demo_use_after_free:
p: 20700178
*p: 42
asan failure, load of 4 bytes at 0x20700178, PC=0x135bb: heap use after free
Stack trace:
  pc 0x13c06  sp 0x20ffffffb8
  pc 0x13a97  sp 0x20ffffffd8
  pc 0x140f5  sp 0x20ffffffe0
  pc 0x135bb  sp 0x20fffffff0
  pc 0x1419f  sp 0x20fffffff8
0x206fff80:  cf  df  df  df  df  df  df  cf  cf  df  df  df  df  df  cf
0x20700000:  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
0x20700080:  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
0x20700100:  00  00  00  f0  00  04  00  00  00  00  00  00  00  00  00  f0 [fd]
0x20700180:  f0  f0  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd
0x20700200:  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd
0x20700280:  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd  fd
SIGABRT: Abnormal termination
Demo – stack buffer overflow

+ runner.pl --cpu=8-M.main --image=demo_asan.axf --args 2
Running demo_stack_overflow:
a[0]: 42
asan failure, load of 4 bytes at 0x20ffffff7c, PC=0x134d9: stack buffer overflow
Stack trace:
  pc 0x13c06 sp 0x20ffffff30
  pc 0x13a97 sp 0x20ffffff50
  pc 0x140f5 sp 0x20ffffff58
  pc 0x134d9 sp 0x20ffffff0
  pc 0x141ff sp 0x20ffffff8
0x20ffffff80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20ffffff80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20ffffff80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20ffffff80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20ffffff80: f1 f1 00 00 [04]
0x20ffffff80: f2 f2 00 04 f3 f3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Access is 0x1c bytes into the stack frame starting at 0x20ffffff60, created by the function at 0x132a0
Frame contains 2 objects:
  [0x10,0x1c): a
  [0x30,0x3c): bbb
SIGABRT: Abnormal termination
Demo – global buffer overflow

+ runner.pl --cpu=8-M.main --image=demo_asan.axf --args 3
Running demo_global_overflow:
ptr[0]: 0
ptr[1]: 1
ptr[2]: 2

asan failure, load of 4 bytes at 0x2020002c, PC=0x136c9: global buffer overflow
Stack trace:
  pc 0x13c06 sp 0x20ffffffb0
  pc 0x13a97 sp 0x20fffffff0
  pc 0x140f5 sp 0x20fffffff8
  pc 0x136c9 sp 0x20fffffff0
  pc 0x14219 sp 0x20fffffff8
0x201ffe80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x201fff00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x201ffff8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20200000: f9 f9 f9 f9 00 [04] f9 f9 cf df df df df df df cf
0x20200080: cf df df df df df cf cf df df df df df df df cf
0x20200100: cf df df df df df cf cf df df df df df df df cf
0x20200180: cf df df df df df df cf cf df df df df df df df cf

SIGABRT: Abnormal termination
Demo – global buffer overflow in ROM

```
+ runner.pl --cpu=8-M.main --image=demo_asan.axf --args 4
Running demo_global_overflow with const global:
ptr[0]: 0
ptr[1]: 1
ptr[2]: 2
asan failure, load of 4 bytes at 0x102ec, PC=0x13691: global buffer overflow
Stack trace:
  pc 0x13b9a sp 0x20fffffff0
  pc 0x13a1f sp 0x20fffffff0
  pc 0x14089 sp 0x20fffffff8
  pc 0x13691 sp 0x20fffffff0
  pc 0x141ad sp 0x20fffffff8
0x000101000: 00 00 04 f9 f9 f9 f9 00 00 00 04 f9 f9 f9 f9
0x000101800: 00 01 f9 f9 00 07 f9 f9 00 02 f9 f9 00 04 f9 f9
0x000102000: 00 03 f9 f9 00 04 f9 f9 00 05 f9 f9 00 04 f9 f9
0x000102880: 00 04 f9 f9 00 04 f9 f9 00 04 f9 f9 00 04 [04] f9 f9
0x000103000: 00 00 00 00 00 00 00 00 00 00 00 00 07 f9 f9 f9 f9
0x000103800: 00 00 00 00 00 00 00 00 00 00 00 00 00 06 f9 f9 f9
0x000104000: f9 f9 f9 f9 00 00 00 00 00 06 f9 f9 f9 f9 f9 f9 f9
SIGABRT: Abnormal termination
```
Benchmarks – code size

(Lower is better)
Benchmarks – instruction counts
(Lower is better)
What didn't I do

+ Command-line options, error checking, testing, documentation, ...
+ Heap: print nearby allocation(s) on error
+ Heap: record stack trace along-side heap allocation, print on error
+ Heap: delay re-using memory to catch more use-after-frees
+ Easy way to symbolize error reports
+ Clib: wrapper functions which check shadow memory
+ Stack use-after-free: Allocate stack frames elsewhere, poison on return
+ Libc++: Would probably need re-compiling to match user's memory layout
+ Optimisation for code-size
+ Benchmark data-size overhead
+ Investigate benchmark outliers
+ Can shadow-of-shadow be reused/moved?