Kernel Control Flow

1. Red boot hands control to int main()
2. main sets the jump instruction in the jump table in low memory.
   *(0x8) = 0x0E51FF004 ; LDR PC [PC, #-4] } put the next word into PC
   *(0x[illegible]) = & asm_SwiCallEntry
   TODO: revise this step because it does some bad things
3. Main calls InitKernel ← Public interface
4. InitKernel calls asm_InitKernelEntry ← swi Interface
5. asm_InitKernelEntry calls SWI 0 after pushing process state
6. PC = 0x8 ; SP = & asm_swiCallEntry
7. Save process state into kernel struct jump to entry point based on swi#
8. Jump table saw #0 for swi call jumps to K_InitKernel
9. schedule first task
10. asm kernel exit sets sp and LR+SPSR for scheduled proc
11. Movs PC, LR restores saved mode, 16 for user proc
12. execute first proc + switch to others via kernel api
13. last task does exit
14. In exit function, call schedule at next proc, but none left. Saved spsr, sp, Lr from redboot is scheduled
15. asm_kernel exit Movs PC, LR ← LR is instruction after InitKernel() in main
16. Main method returns control to redboot