Exercise 5 – The Stopwatch

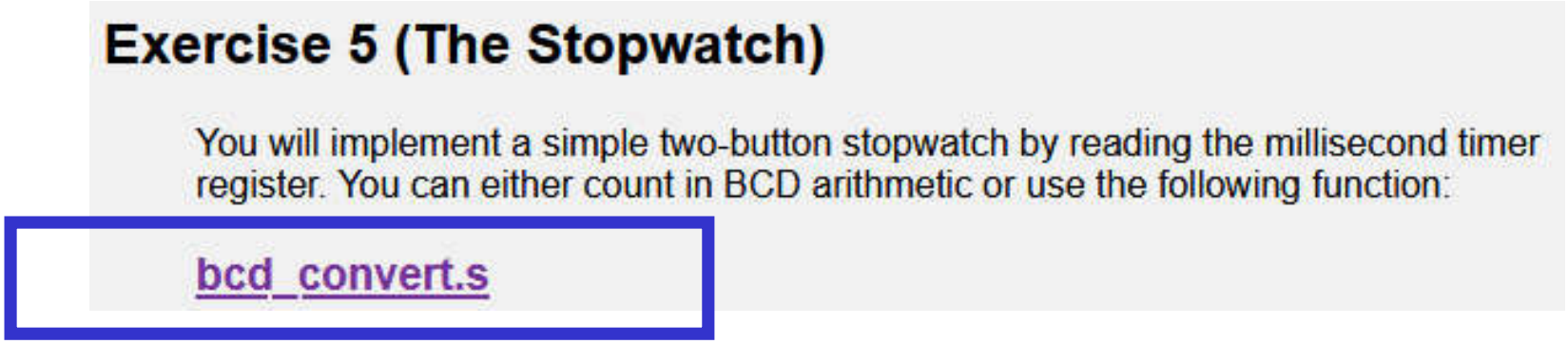
- Write a programme which increments a memory location every 100 ms

- Poll the ms timer register at address xF1001010
  Note that the time you may see is NOT the wall clock time.
  In the simulation environment, we derive the clock from CPU cycles such that time freezes in case you step through your program.
  → remember this when building a “real” system

- Build a stopwatch such that one button (e.g., A) starts the count and a second button causes it the stopwatch to pause.
- Holding the second button (stop) down for more than one second should reset the counter (stopwatch) to zero. (this is a state machine!)

Exercise 5 – The Stopwatch

- For full marks, Part 2 (the four digit decimal counter) is required
- In case you do the second part, please submit only this one for feedback



- Use an SVC system call to access the timer register
  (use different SVCs for anything that is related to I/O)
- Provide a header in your file that includes
  your name, email, date, exercise, known bugs
- As with the last exercise: submit through blackboard as a single concatenated source file (you can use the .s suffix)
  Please use a breaker to indicate the start of a file.
  Do not submit provided code (e.g. font.s) if you have not modified.

ARM – Interrupts and Exceptions

- We had SVCs and Abort: both are a result of the currently running program
  - SVC: LR contains the address following the SVC instruction
  - Abort: LR contains the address of the failing instruction
    Prefetch abort: on reading an instr. from a protected address
    Data abort: a load or store instruction touches a protected address
- An interupt is an exception that is triggered by an external event
  (CPUs have one ore more dedicated input signals to flag interupts)
  → allows hardware to call software
  - IRQs are unpredictable (they may come at any time) → may cause race conditions and other form of non-determinism in a user program
- Interrupt response times are important in many embedded systems
  →real time is not so much about being fast, but deterministic!

ARM – Interrupts and Exceptions

- ARM has two interrupt modes: IRQ and FIQ
  - Behave mostly the same, but are different modes
  - FIQ has higher priority (and may interupt an IRQ)
  - Each mode uses its own stack (you need to set that up)
  - FIQ mode has extra physical working registers R8-R12
    (can save some time when working registers are not needed to be saved on stack)

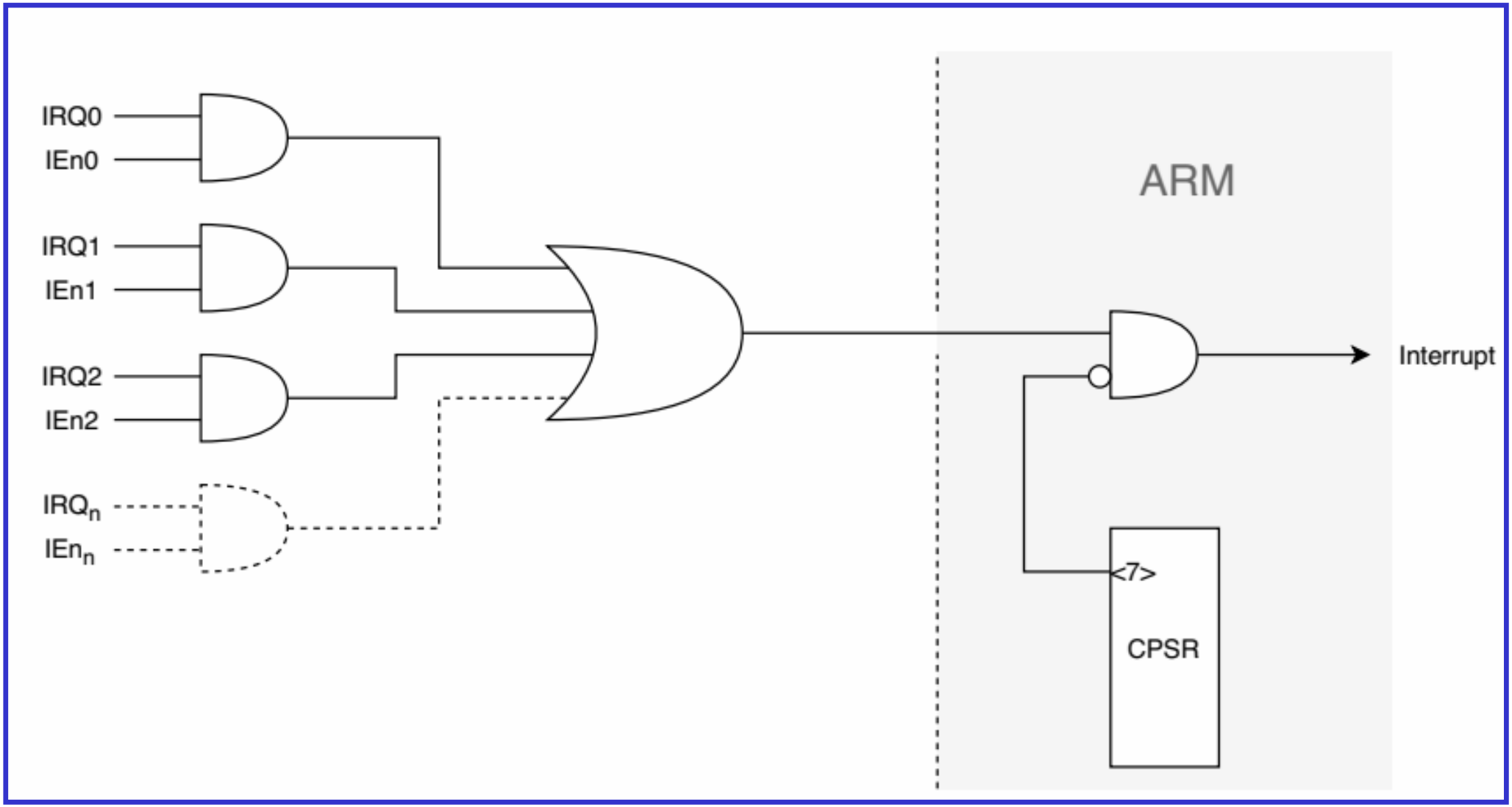
- Interrupts may be disabled to ignore (or postpone serving) them
  - A system may use this to mask problems (e.g. a defect sensor)
  - There are atomic processes running that cannot be interrupted
    (e.g. some memory alocation, semaphores etc.)
  - Be careful: this may result in missing events
    (with potentially bad side effects)

ARM – Interrupts and Exceptions

- On an interrupt, the following happens in ARM:
  - The current instruction completes (e.g., LDM)
  - R14_irq := address of the next instruction **+ 4**
    (This is because the ARM is pipelined and the PC is already incremented)
  - SPSR_irq := CPSR
  - CPSR mode is set to IRQ mode
  - IRQ is disabled in the CPSR
    (note: **in ARM a bit is set to disable the particular interrupt**)
  - The Thumb bit is cleared (if set)
  - PC := x0000 0018 ; x0000 001C for FIQ

- When writing an interrupt handler, ensure that you never corrupt the context of any user programme.
- To return from interrupt handler: **SUBS** PC, LR, **#4**
  alternative method (see the ^): LDMFD SP!, {R0-R2,PC}^

ARM – Interrupts and Exceptions



- A system commonly has multiple IRQ sources (keyboard, serial links, etc.)
- Trick: cascading through external hardware
  (sometimes using dedicated interrupt controllers)

ARM – Interrupts and Exceptions

Bit number	Function
0	**Timer compare**
1	Spartan FPGA (Not fitted here)
2	Virtex FPGA (Not fitted here)
3	Ethernet interface (Not fitted here)
4	Serial RxD ready (Not fitted here)
5	Serial TxD available (Not fitted here)
6	**Upper button**
7	**Lower button**

- The interrupts are mapped to address xF200 0000 (see bitmask in table)
- The mask is mapped to address xF200 00001 (see bitmask in table)
- you have to use byte addressing for these registers
- An IRQ is only fired if enabled in mask register and in the CPU (bit 7 in CPSR)
- Interrupts can be edge (buttons) or level triggered (timer compare)