- program spends a lot of time polling

e.g. assume:

1 clock cycle per instruction
processor clock frequency $f_c = 1GHz$
UART receives bytes of data at a rate of $f_{rx} = 100kHz$

- Calculate fraction of time spent polling.

$$t_{rx} = t_{polling} + t_{mvbyte}$$

$$= 3t_c \times n + 3t_c$$

$$t_{polling} = t_{rz} - 3t_c$$

$$= \frac{1}{f_{rz}} - \frac{3}{f_c}$$

$$= 9.997 \mu s$$

n = # of loop iterations

$$\% \text{ Polling} = \frac{t_{polling}}{t_{rz}}$$

$$= \frac{9.997 \mu s}{10 \mu s \leftarrow \frac{1}{f_{rx}}}$$

$= 99.97\%$ of time is spent polling

- better way:
- enable I/O device to signal (interrupt) processor when it needs service (has data)

5) Exceptions

- condition requiring service
  - current execution is suspended
  - condition is serviced
  - original execution is resumed.
  - code is unaware that it was suspended
- Cortex-M3 exception types

reset	(power on)
SVCall	supervisor call - invokes OS
fault	execution error such as invalid instruction, unaligned memory access

interrupt request(IRQ)   request from a peripheral device for service

- exception attriputes
  - number 1-255
  - priority -3 (highest) to 31 (lowest)
  - vector exception handler address (entry point)
- exception list
  - refer to pic
- NMI = non-maskable interrupt
  - connected to reset button
    or, watchdog timer (failsafe for embedded systems)
- Vector Table
  - entry point of exception handlers
  - refer to pic-2

5.1) Operating Modes

mode	used for	can execute priviledged instr.	stack
thread	application code	no* or yes*	SP_process* or SP_Main**
handler	exception handlers	yes	SP_Main

* multi-user OS } determined by
** single-user OS } CONTROL register

- hw switches to handler mode on exception and back to thread mode on return
- priviledged instructions control system state
  e.g. CPSIE i //interrupt enable)
- separate user stacks (SP_process) and OS stack (SP_main) avoid corruption of system stack by user code

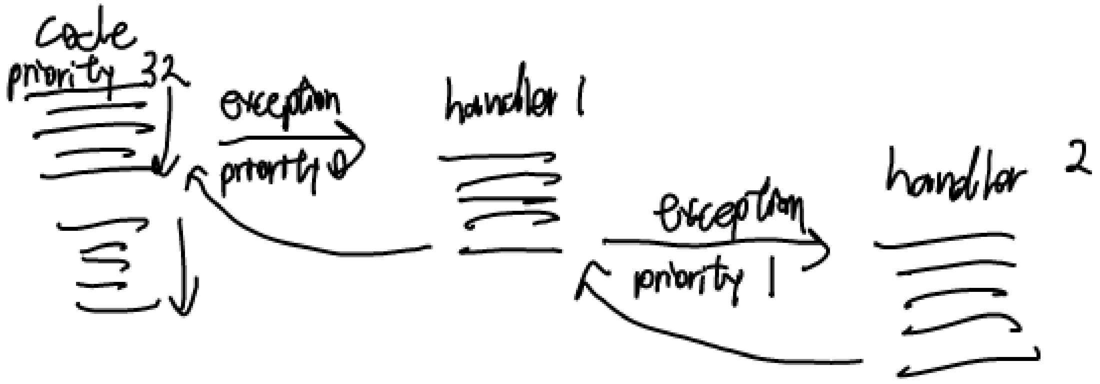
5.2) Reset

- an exception that occurs on power up or warm reset
- actions:
  - loads SP_main into r13 from vector 0
  - sets operating priority to max priority+1 =32
  - executes code in reset handler.

5.3) Exception Handling

- exceptions are handled (handler is invoked) if they have higher priority (lower #) than the operating priority
- actions
  - save current context on stack
    pushes r0-r3, r12, return address (PC), PSR (status reg), LR
  - stores EXC_Return code in LR indicating operating mode & stack in use.
  - loads entry pt of handler from vector table into PC
- exception return is initiated by loading the EXC_RETURN code into the PC
  e.g. BX lr

- actions
  - uses EXC_RETURN to set operating mode and stack pointer.
  - restors context from stack
    pops r0-r3, r12, PC, PSR, LR
- exception handlers can be suspended by higher priority exceptions (nested exceptions)


code
priority 32
exception
priority 0
handler 1
exception
priority 1
handler 2