**I/O Systems**

* Interconnection network
  + Circuit that transfers information b/t processor, memory unit, and I/O devices
  + E.g. system bus
  + Processor communicates over interconnection via shared address space
    - I/O and memory devices share the same address space
    - Can use memory instructions to access I/O devices
    - E.g. LDR R2, DATAIN
      * Where DATAIN is a memory-mapped address of a register for an input device
* I/O device interface
  + Circuit between a device and the interconnection network
  + Includes DATA, STATUS, and CONTROL registers for the device accessible with LDR/STR
  + Registers can be seen as locations in memory (i.e. memory-mapped)
* **Program-controlled I/O**
  + E.g. read keyboard characters → store in memory → display characters on screen
  + Signalling protocol
    - Status register of device contains a status flag bit – the “ready” signal
    - Polling:
      * E.g. keyboard → KBD\_STATUS register → KIN flag at b1
        + KIN = 1 means character is ready to be sent
        + READWAIT:

LDRB R4, KBD\_STATUS ;load status byte

ANDS R4, R4, #2 ;check second bit

CBZ R4, READWAIT ;if KIN = 0, loop back

LDRB R5, KBD\_DATA ;if KIN = 1, load data byte

* + - * + KIN is cleared automatically when KBD\_DATA is read
      * E.g. display → DSP\_STATUS register → DOUT flag at b2
        + DOUT = 1 means character is ready to be received
        + WRITEWAIT:

LDRB R4, DSP\_STATUS ;load status byte

ANDS R4, R4, #4 ;check third bit

CBZ R4, WRITEWAIT ;if DOUT = 0, loop back

STRB R5, DSP\_DATA ;if DOUT = 1, save data byte

* + - Problems with polling
      * Continuous involvement of processor – can’t get anything else done
* **Interrupt-driven I/O**
  + An interrupt causes processor to deviate from its normal instruction sequence and respond to a high priority event
  + Interrupt service routine – works like a subroutine that is called to service an I/O device
    - Different from subroutine:
      * Function performed by ISR may not be related to program being executed
      * ISR must not affect the state of current program
    - Processor finishes execution of current instruction before servicing interrupts
    - Processor saves the state of PC and the processor status register (PS)
    - PC ← address of first instruction in ISR
    - Execute ISR
      * ISR must handle saving & restoring registers that will be used
    - PC ← address of next instruction in main program
  + Interrupt latency – delay caused by state saving and calling the ISR
  + Interrupts must be disabled when an ISR is in execution – otherwise can cause recursive calls, infinite loops etc.
    - PS register → interrupt enable (IE) bit
  + **Handling single-device interrupt:**
    - Device sends interrupt request signal
    - Processor interrupts program and saves PC & PS
    - IE bit set to 0 → disables (ignores) interrupts
    - ISR called → processor sends interrupt acknowledge signal → device deactivates request signal
    - ISR finishes → processor restores saved state
    - IE bit set to 1 → enables interrupts
    - Resume execution of interrupted program
  + **Multiple-device interrupt:**
    - Processor needs to know which device raised the request signal
      * IQR bit in the device’s status register indicates it is requesting an interrupt
      * Can poll every device to see which device has IRQ = 1
      * Or, use vectored interrupts
    - Interrupt vector table – a fixed portion of memory that holds addresses to ISRs
      * I/O device provides an interrupt code along with request signal
      * Interrupt code is used as a pointer into the vector table to get the address of the corresponding ISR
    - Shared interrupt signal
      * Devices share the same wire – signal sent if any device requests to interrupt
      * Processor calls a generic ISR that polls every device
      * Or device supplies an interrupt code
    - Dedicated interrupt signal
      * Processors knows the source of request – ISR can be retrieved immediately
      * Minimizes interrupt latency but more wires are required (one per device)
    - Interrupt nesting
      * Allows higher-priority interrupt request to be processed while an ISR is in execution
      * Similar to subroutine nesting
    - Arbitration – determines priority between simultaneous interrupt requests
    - Can set IE bit for each device
      * E.g. KIE = 1 enables interrupts from keyboard
      * E.g. DIE = 0 disables interrupts from display
    - Other control registers
      * IPS – where PS is automatically saved upon interrupt request
      * IENABLE – one bit per device to recognize requests from each device
      * IPENDING – one bit per device to indicate if a request has not yet been serviced
  + **Exceptions**
    - Refers to any type of interruption of execution
    - E.g. I/O interruptions, error recovery, debugging