4. Interrupts

COMP2121 • KC Notes

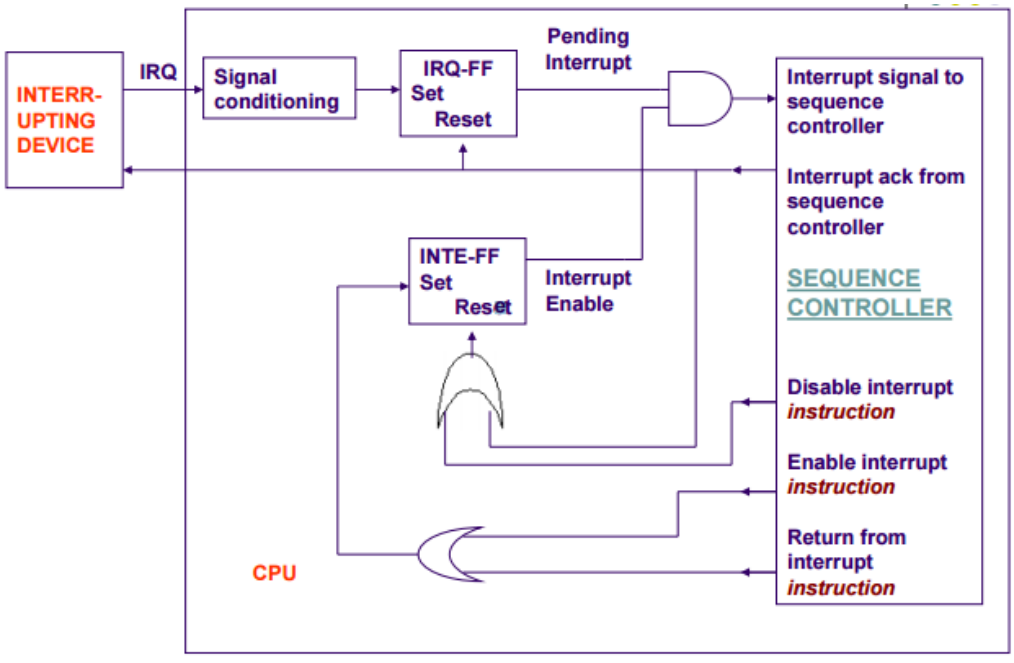
4.1 Introduction to Interrupts

- **Interrupt**: **asynchronous** events that needs to be recognised and dealt with by the CPU
- To recognise an asynchronous event:
  - **Polling**: a software query to the I/O device
    - But, not efficient, reduces performance, wastes cycles to query a device
  - **Interrupts**: I/O device generates signals to request services
    - Needs special hardware to implement interrupt services
    - Signal is only generated if it needs the CPU – efficient

4.2 Interrupt Process Control

1. **Recognise interrupt events**: identify an interrupt
   - Identify one among multiple sources
   - Determine which interrupt to serve if there are many simultaneous interrupts
2. **Respond to interrupt**
   - Wait for the current instruction to finish
   - Acknowledge the interrupting device
   - Branch to the **interrupt service routine (ISR)/interrupt handler**
3. **Resume normal task**: return to the point where it was interrupted

- Controlled through **software** (enable certain/all interrupts) **or hardware** (disable further interrupts while an interrupt is being serviced)



1. **Interrupt Request (IRQ)**: many devices are wire-ORed together (active-low signal)
    - **Signal Conditioning Circuit** detects different types of signals
    - **Interrupt Request Flip-Flop** (IRQ-FF): catches and records the interrupt request until CPU acknowledges and services it
    - When IRQ-FF is set, a pending interrupt signal is sent towards the **sequence controller**
2. IRQ-FF is reset when it receives a **INTA (interrupt acknowledge) signal** from the CPU
    - INTA is only generated when:
        - Current instruction is finished
        - CPU has detected the IRQ
    - **Interrupt Enable Flip-Flop** (INTE-FF): allows the interrupt to be enabled or disabled by software instructions
    - Setting the INTE-FF enables interrupts, and the **pending interrupt is allowed through to the sequence controller**
3. When the interrupt service routine is complete, the CPU executes a **return-from-interrupt instruction**

- **Nested interrupts** can happen if **INTE-FF is set during an ISR**

4.3 Multiple Interrupt Sources

- **Determine which device generated** the IRQ to execute the correct ISR
    - **Interrupt polling**: software, where CPU **reads each device's status register** and finds the device that generated the IRQ
    - **Vectored interrupts**: When responding to the CPU's INTA signal, the device places a **vector (address of ISR)** onto the data bus for the CPU to read – this vector **identifies the device**
    - **Multiple Interrupt Masking**: the CPU has multiple IRQ input pins
        - Use a **mask register** to enable a specific IRQ by enabling/disabling an individual bit that is assigned to an interrupting source
- **Prioritise simultaneous** IRQs
    - **Software resolution**: **polling order** determines which source is serviced first
    - **Hardware resolution**: needed for vectored interrupts
        - **Daisy chain priority resolution**: CPU **asserts INTA that is passed down the chain** from device to device, with the higher priority device closer to CPU
        - When INT reaches device that generated IRQ, device puts vector onto data bus and **does not pass the INTA** – lower-priority devices does not get the INTA
        - **Separate IRQ Lines**: each line with a fixed priority, IRQ0 > IRQ1 > IRQ2
        - **Hierarchical prioritisation**: lower interrupts are masked
        - **Non-maskable interrupts**: cannot be disabled, reserved for critical events like power failure

4.4 Transferring Control to ISR

- Hardware **saves the return address and may also save registers** like program status register
    - AVR: **must be done in software** by saving program status register and conflict registers
- **Interrupt latency**: the time from IRQ generated to ISR being executed
- ISR is a **special subroutine**:
    - Prologue: save conflict registers on the stack
    - Body: code
    - Epilogue: Restore saved registers, and **return-from-interrupt** instruction

4.5 Software Interrupts and Exceptions

- **Software interrupt**: generated by software without a hardware-generated IRQ
    - Implement system calls in OS, like CTRL+V
    - AVR: use external interrupts to implement software interrupts
- **Reset**: non-maskable, only initialises system to some initial state
- **Exceptions**: abnormalities during the normal operation of the processor
    - E.g. internal bus error, memory access error, illegal instructions
    - AVR: does not handle exceptions