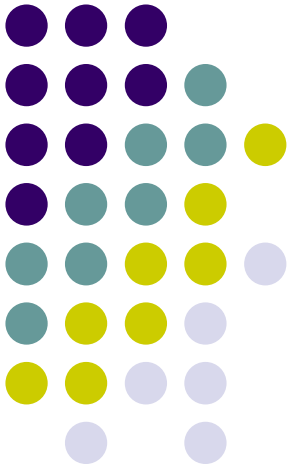
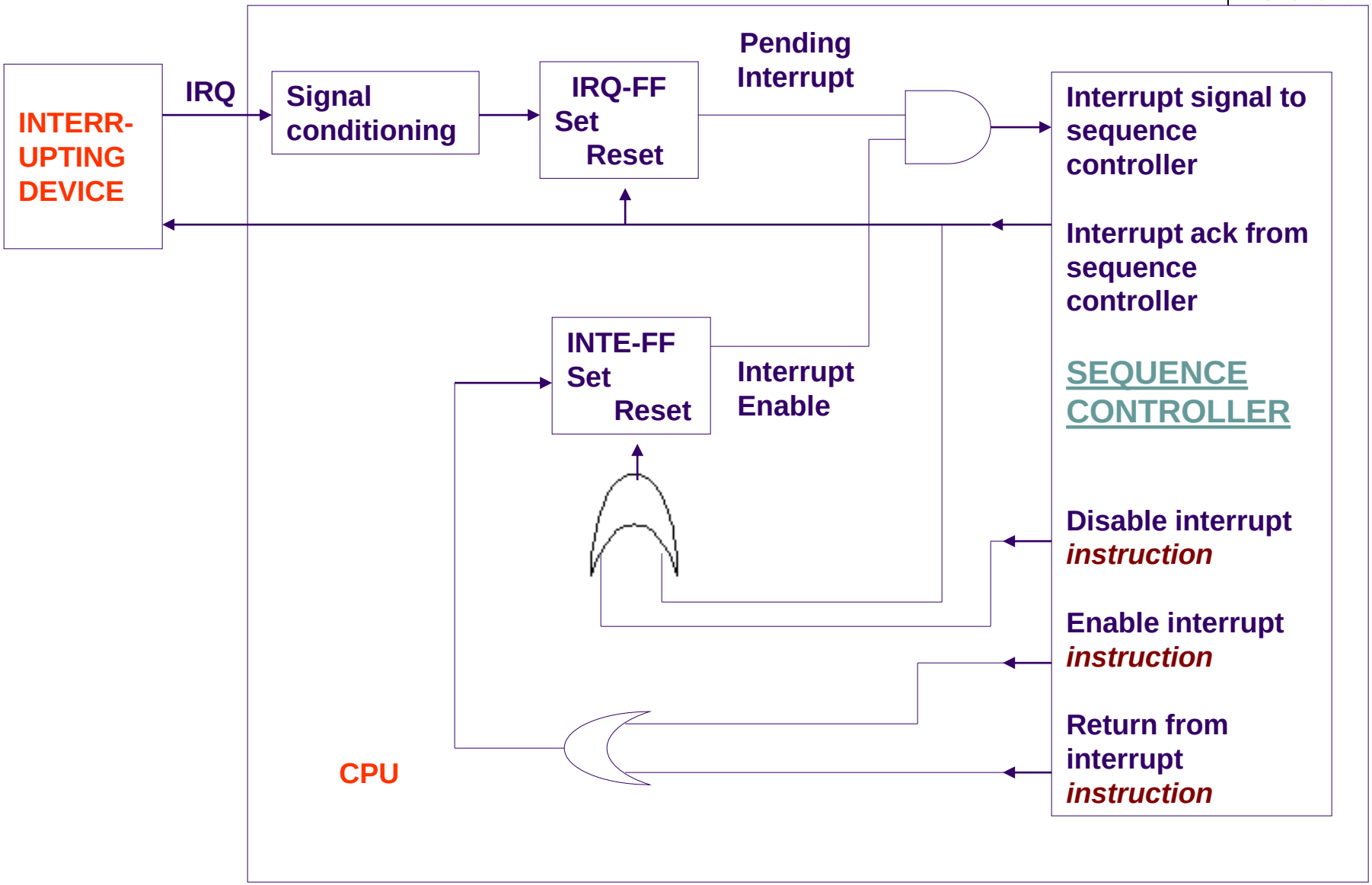
Interrupts (Filling in the rest!)

Lecturer: Sri Parameswaran

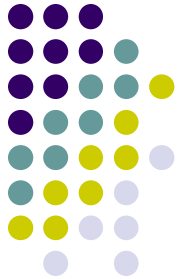
Notes by: Annie Guo



Interrupt Recognition and Acknowledgement Hardware

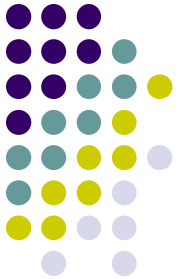


Interrupt Recognition and Ack.



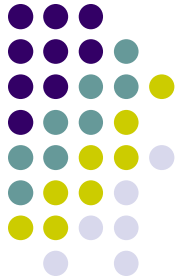
- An Interrupt Request (IRQ) may occur at any time.
  - It may have rising or falling edges or high or low levels.
  - Frequently it is an active-low signal
    - multiple devices are wire-ORed together.
      - Recall open-collector gates
- Signal Conditioning Circuit detects these different types of signals.
- Interrupt Request Flip-Flop (IRQ-FF) records the interrupt request until it is acknowledged.
  - When IRQ-FF is set, it generates a pending interrupt signal that goes towards the Sequence Controller.
  - IRQ-FF is reset when CPU acknowledges the interrupt with INTA signal.

Interrupt Recognition and Ack. (cont.)



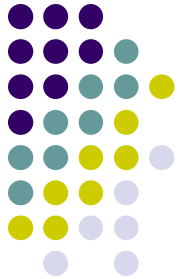
- Interrupts can be enabled and disabled by software instructions, which is supported by the hardware Interrupt Enable Flip-Flop (INTE-FF).
- When the INTE-FF is set, all interrupts are enabled and the pending interrupt is allowed through the AND gate to the sequence controller.
- The INTE-FF is reset in the following cases.
  - CPU acknowledges the interrupt.
  - CPU is reset.
  - Disable interrupt instruction is executed.

Interrupt Recognition and Ack. (cont.)



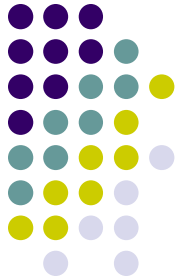
- An interrupt acknowledge signal is generated by the CPU when the current instruction has finished execution and CPU has detected the IRQ.
  - This resets the IRQ-FF and INTE-FF and signals the interrupting device that CPU is ready to execute the interrupting device routine.
- At the end of the interrupt service routine, CPU executes a return-from-interrupt instruction.
  - Part of this instruction's job is to set the INTE-FF to re-enable interrupts.
- Nested interrupts can happen If the INTE-FF is set during an interrupt service routine
  - An interrupt can therefore interrupt interrupting interrupts.

Multiple Sources of Interrupts



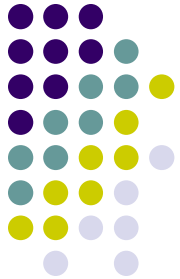
- To handle multiple sources of interrupts, the interrupt system must
  - Identify which device has generated the IRQ.
    - Using polling approach
    - Using vectoring approach
  - Resolve simultaneous interrupt requests
    - using prioritization schemes.

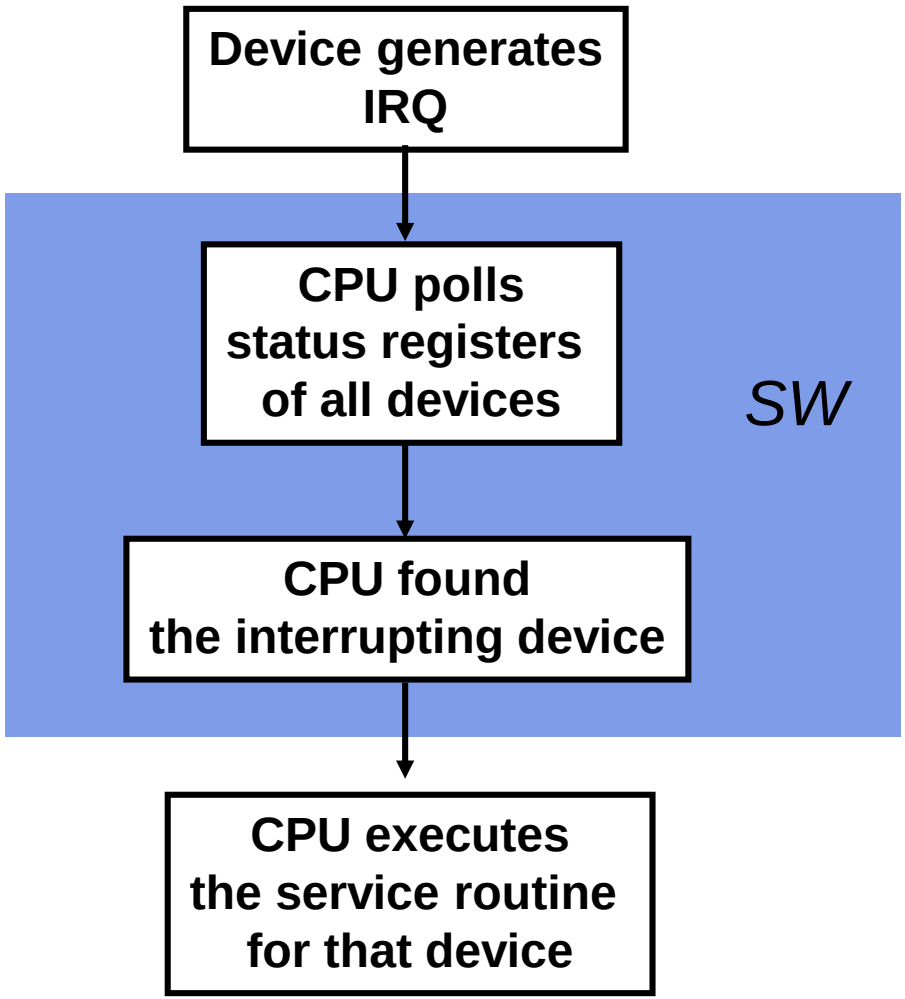
Polled Interrupts



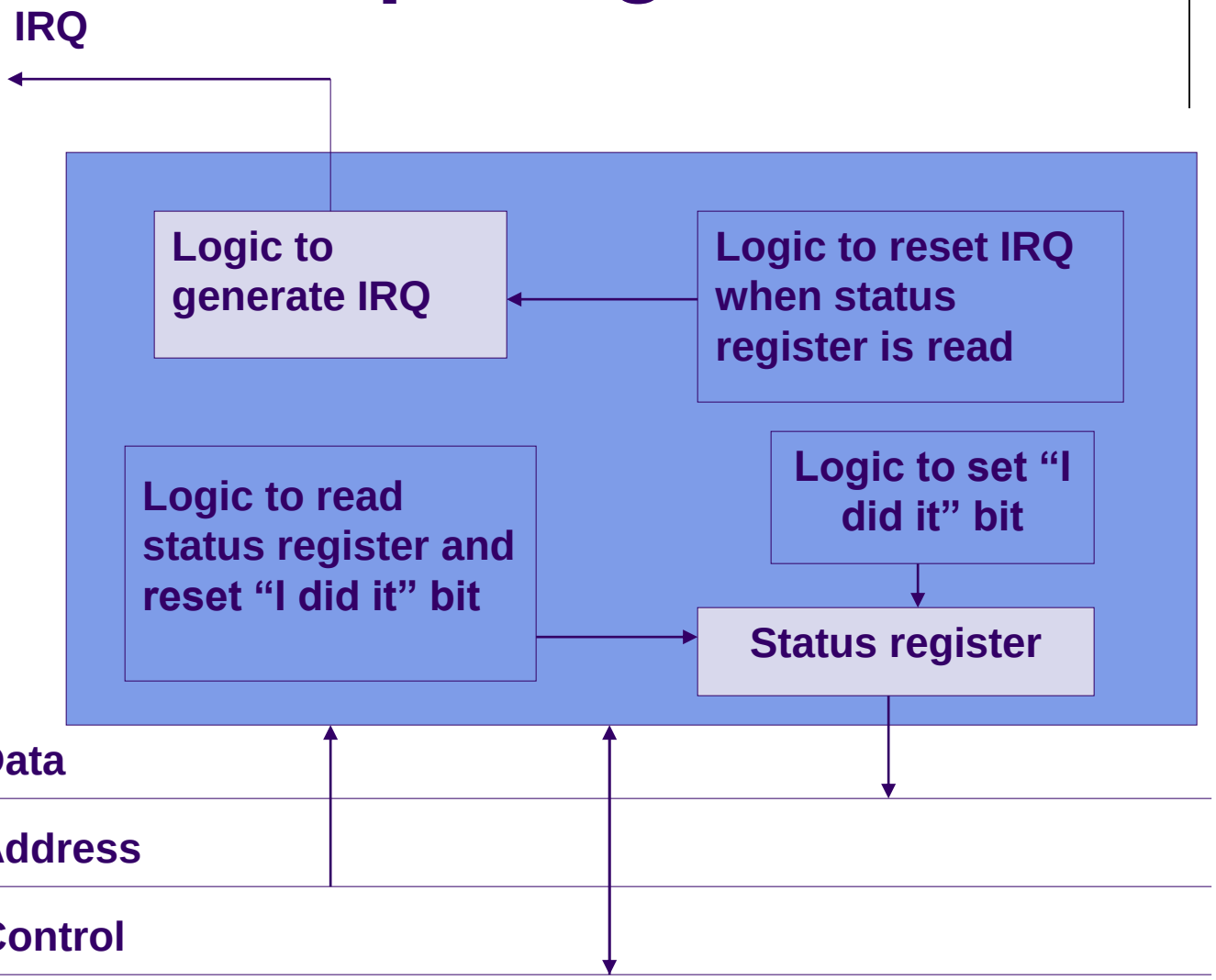
- Software, instead of hardware, is responsible for finding the interrupting source.
  - The device must have logic to generate the IRQ signal and to set an “I did it” bit in a status register that is read by CPU.
  - The bit is reset after the register has been read.

Polled Interrupts Execution Flow

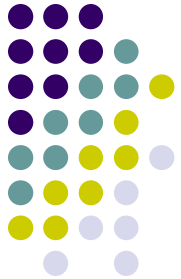




Polled Interrupt Logic

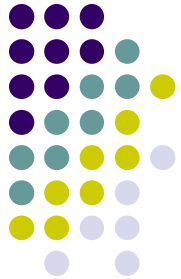


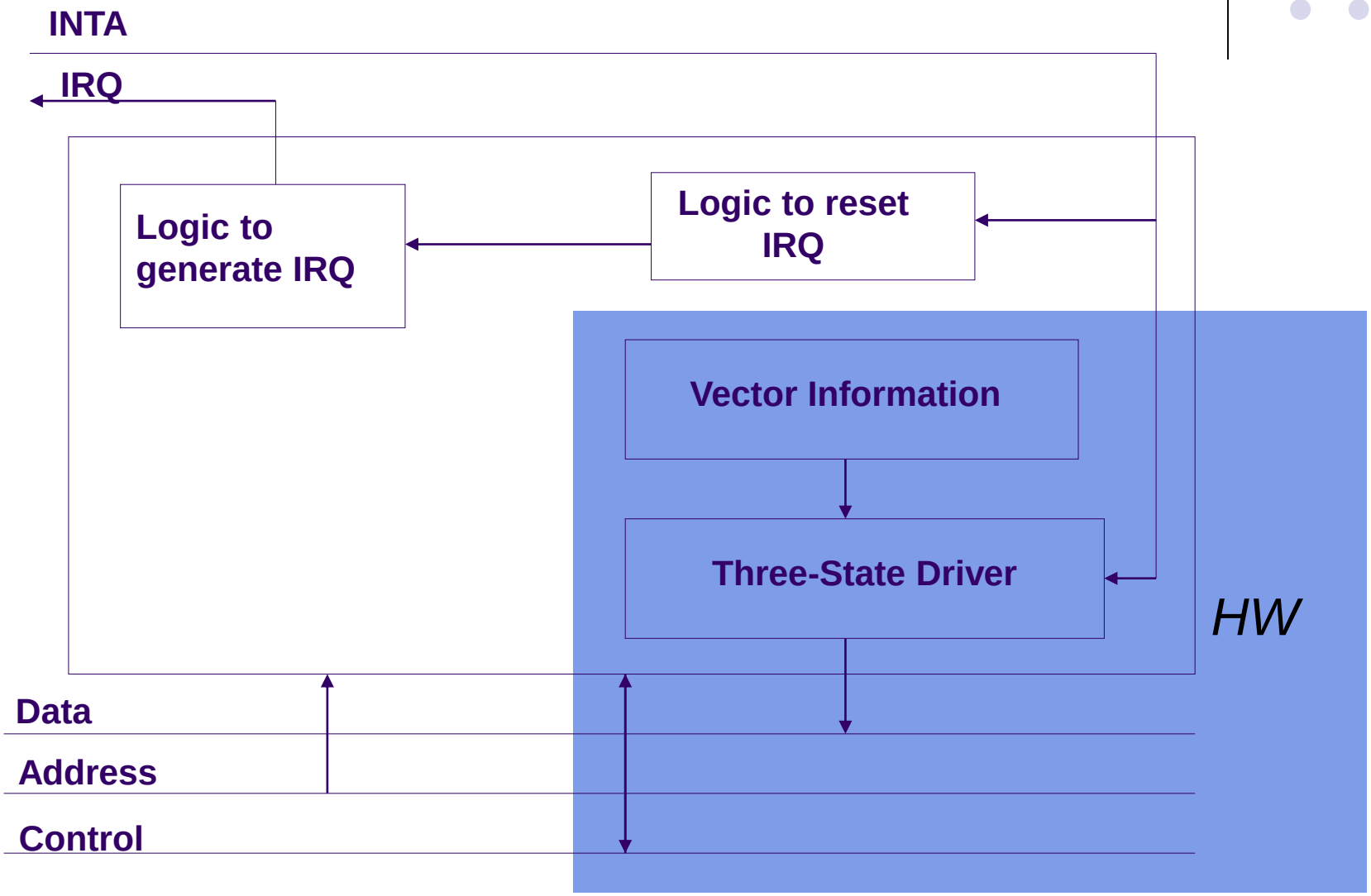
Vectored Interrupts



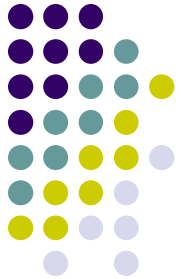
- CPU’s response to IRQ is to assert INTA.
- The interrupting device uses INTA to place information that identifies itself, called the vector, onto the data bus for CPU to read.
- CPU uses the vector to execute the interrupt service routine.

Vectored Interrupting Device Hardware



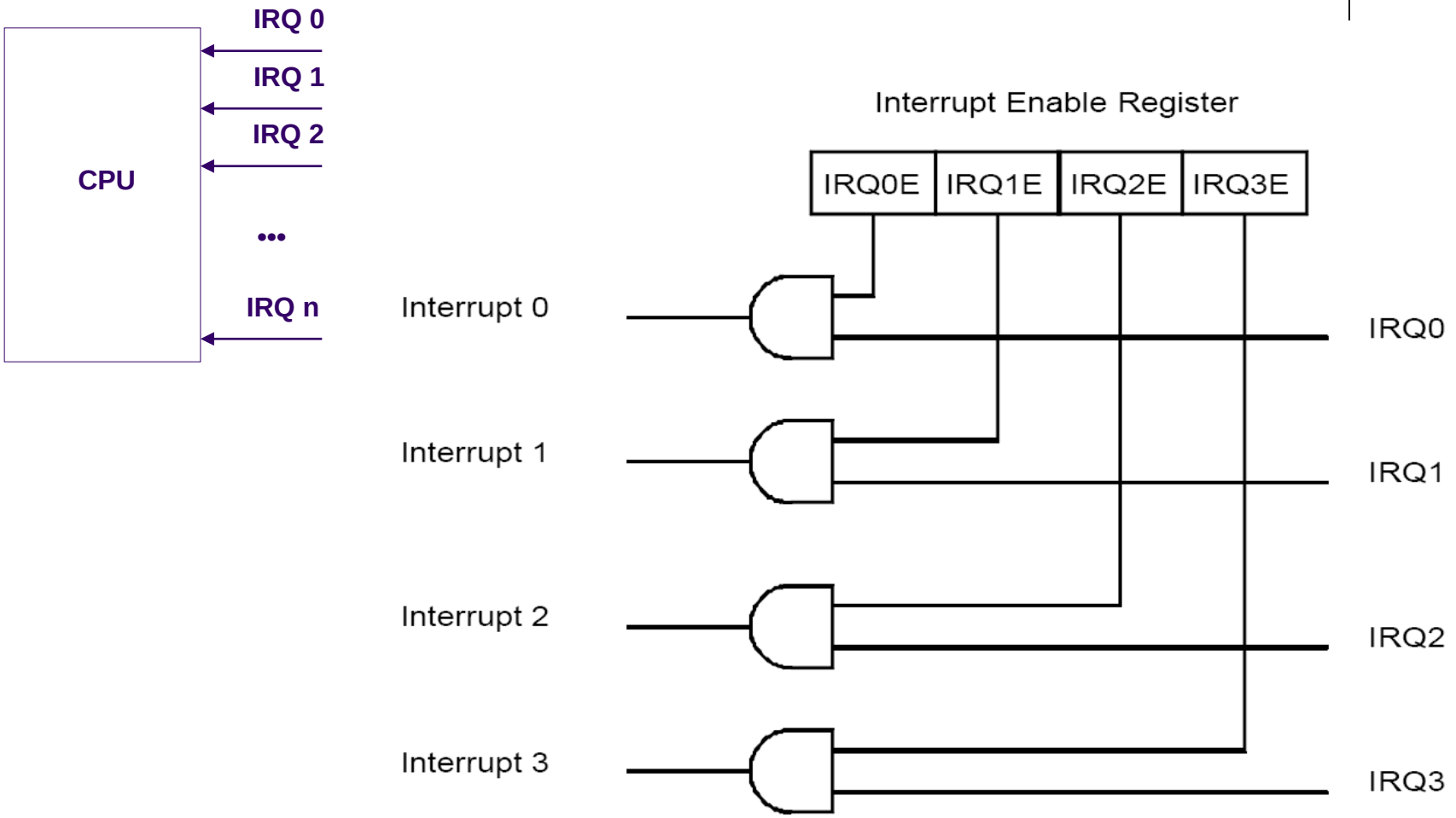


Multiple Interrupt Masking

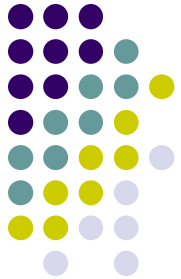


- CPU has multiple IRQ input pins.
- Masking enables some interrupts and disables other interrupts
- CPU designers reserve specific memory locations for a vector associated with each IRQ line.
- Individual disable/enable bit is assigned to each interrupting source.

Multiple Interrupt Masking Circuit

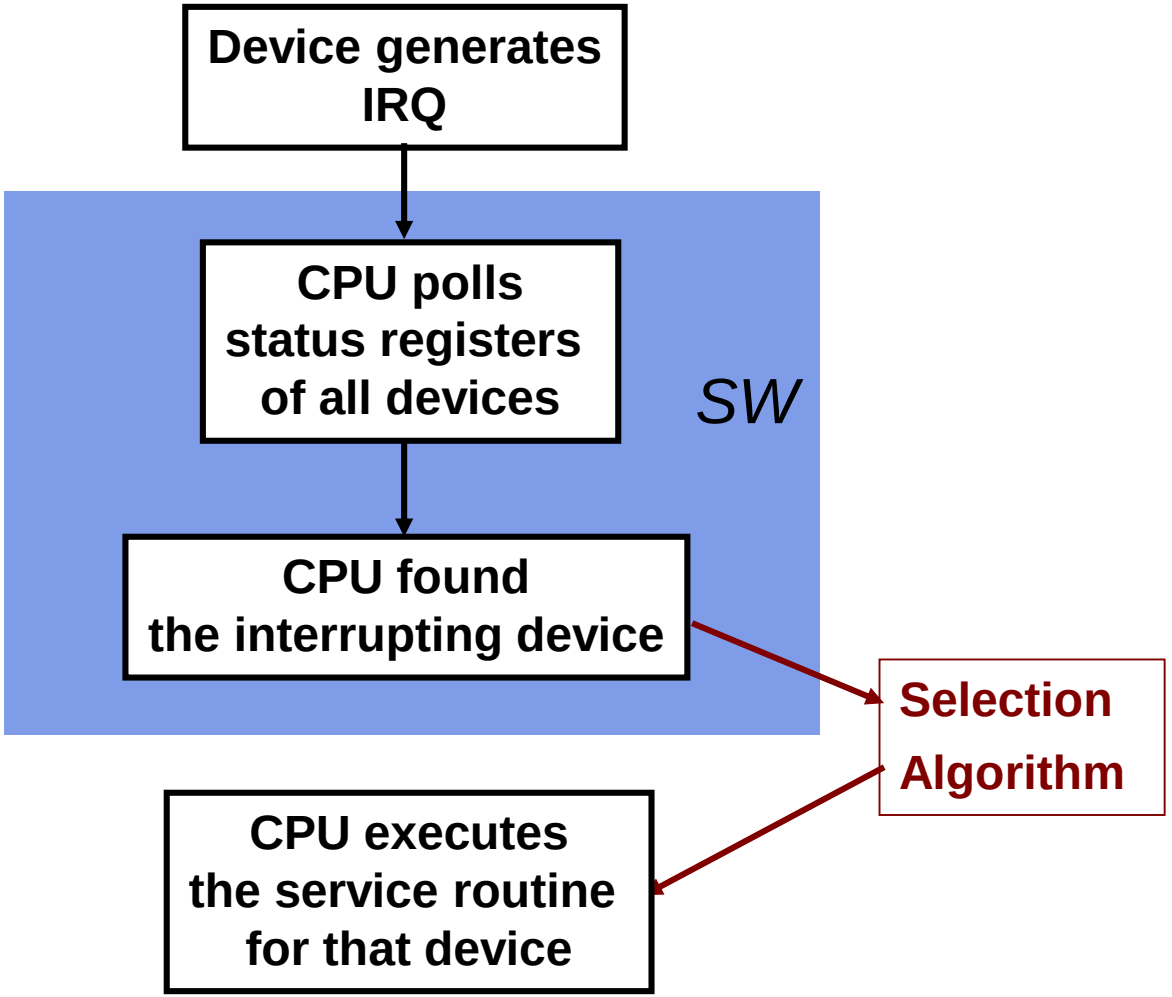


Interrupt Priorities

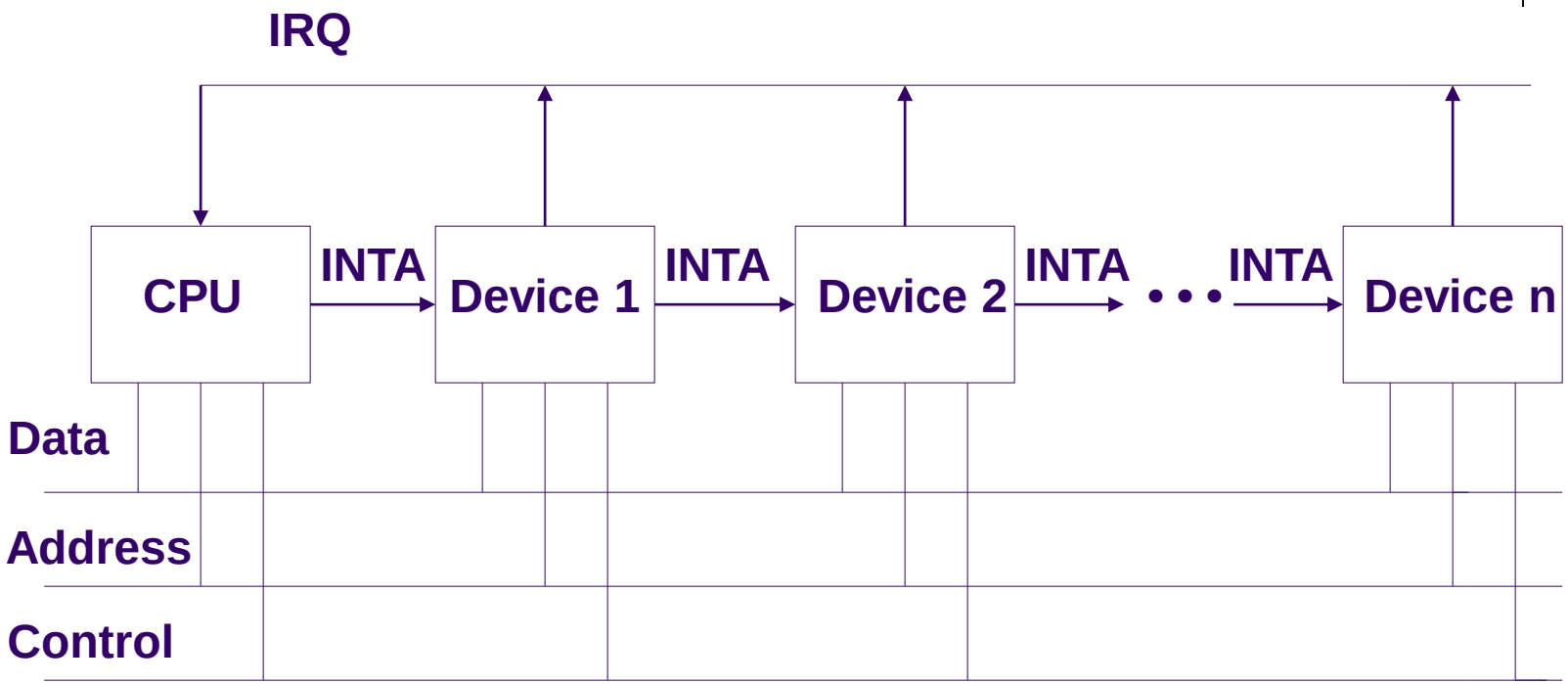


- When multiple interrupts occur at the same time, which one will be serviced first?
- Two resolution approaches:
  - Software resolution
    - Polling software determines which interrupting source is serviced first.
  - Hardware resolution
    - Daisy chain.
    - Others

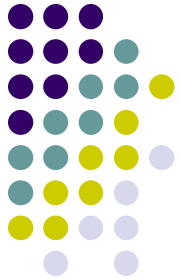
Software Resolution



Daisy Chain Priority Resolution

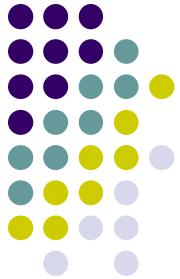


Daisy Chain Priority Resolution (cont.)



- CPU asserts INTA that is passed down the chain from device to device. The higher-priority device is closer to CPU.
- When the INTA reaches a device that generated the IRQ, that device puts its vector on the data bus and does not pass along the INTA. So lower-priority devices do NOT receive the INTA.

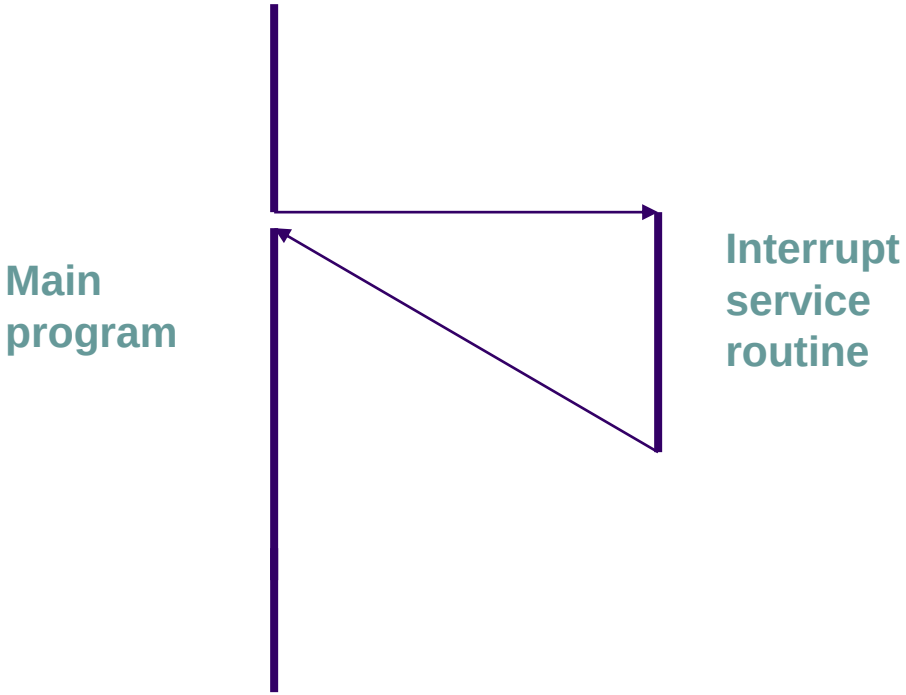
Other Priority Resolutions



- Separate IRQ Lines.
  - Each IRQ line is assigned a fixed priority. For example, IRQ0 has higher priority than IRQ1 and IRQ1 has higher priority than IRQ2 and so on.
- Hierarchical Prioritization.
  - Higher priory interrupts are allowed while lower ones are masked.
- Nonmaskable Interrupts.
  - Cannot be disabled.
  - Used for important events such as power failure.

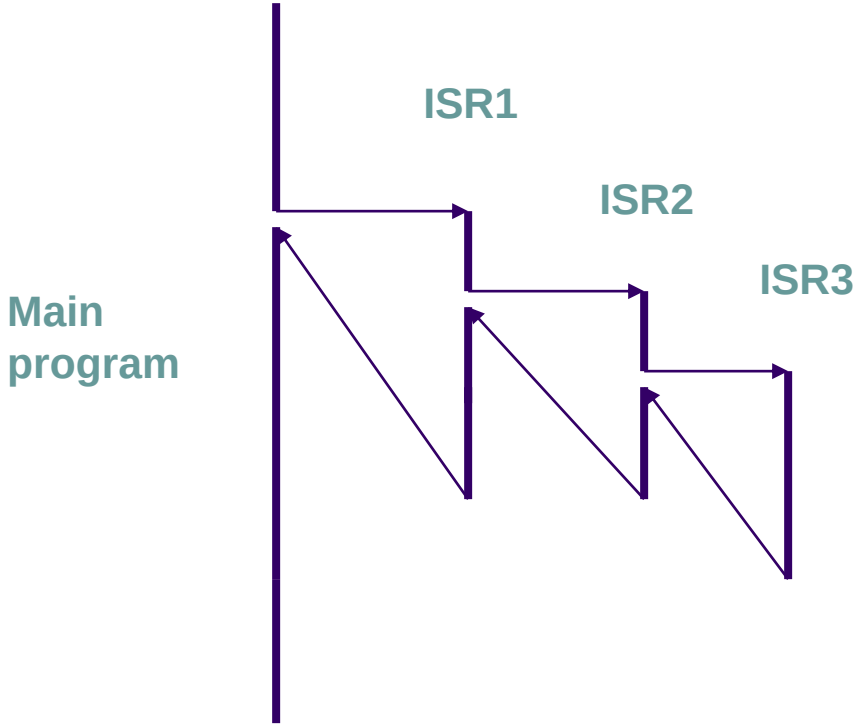
Non-Nested Interrupts

- Interrupt service routines cannot be interrupted by another interrupt.

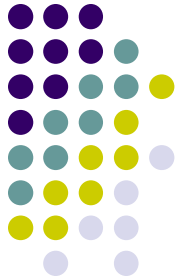


Nested Interrupts

- Interrupt service routines can be interrupted by another interrupt.

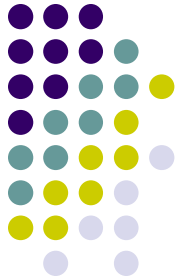


RESET in Mega2560



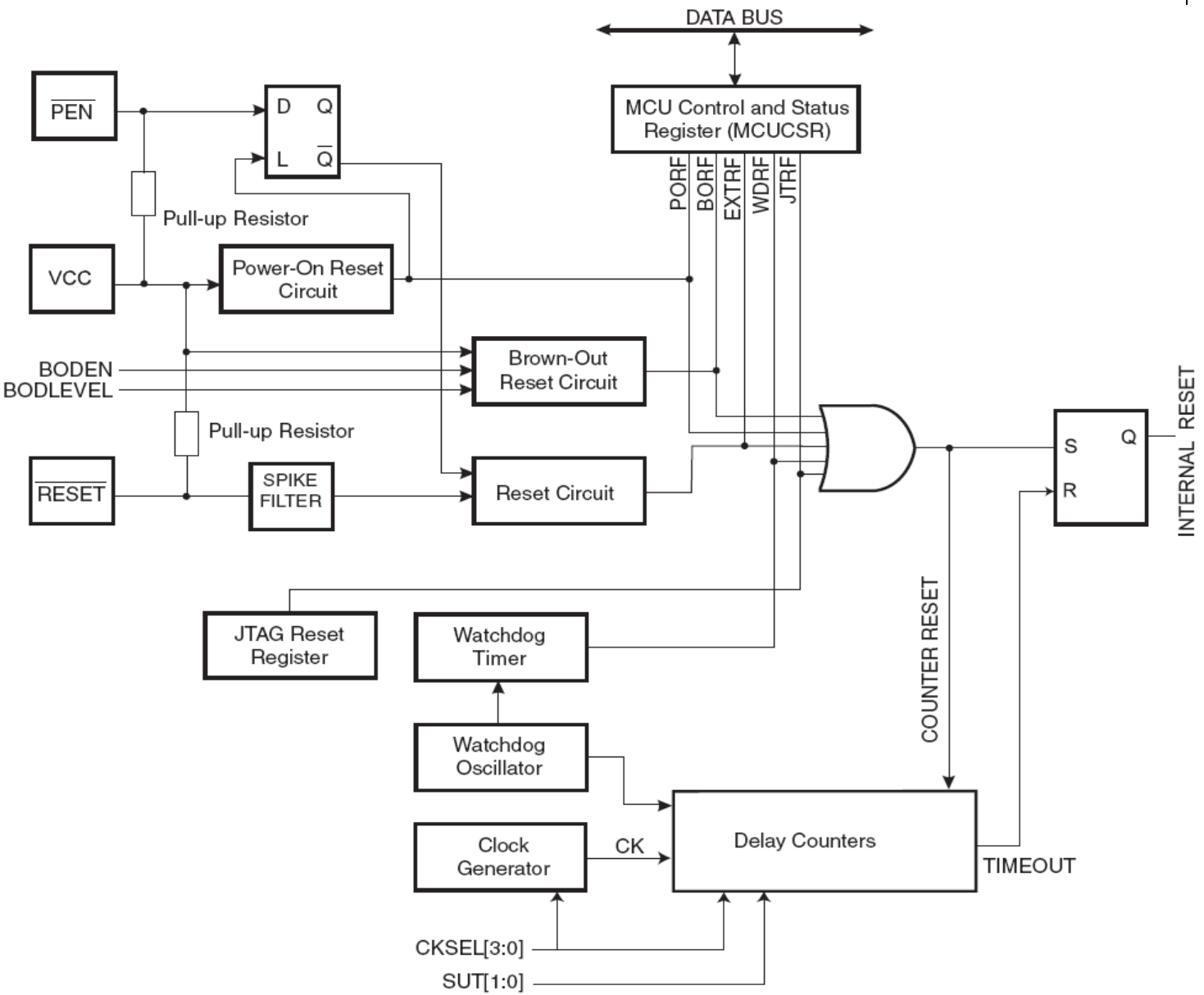
- The ATmega2560 has five sources of reset:
  - Power-on Reset.
    - The MCU is reset when the supply voltage is below the Power-on Reset threshold (VPOT).
  - External Reset.
    - The MCU is reset when a low level is present on the RESET pin for longer than the minimum pulse length.
  - Watchdog Reset.
    - The MCU is reset when the Watchdog Timer period expires and the Watchdog is enabled.

RESET in Mega2560 (Cont.)

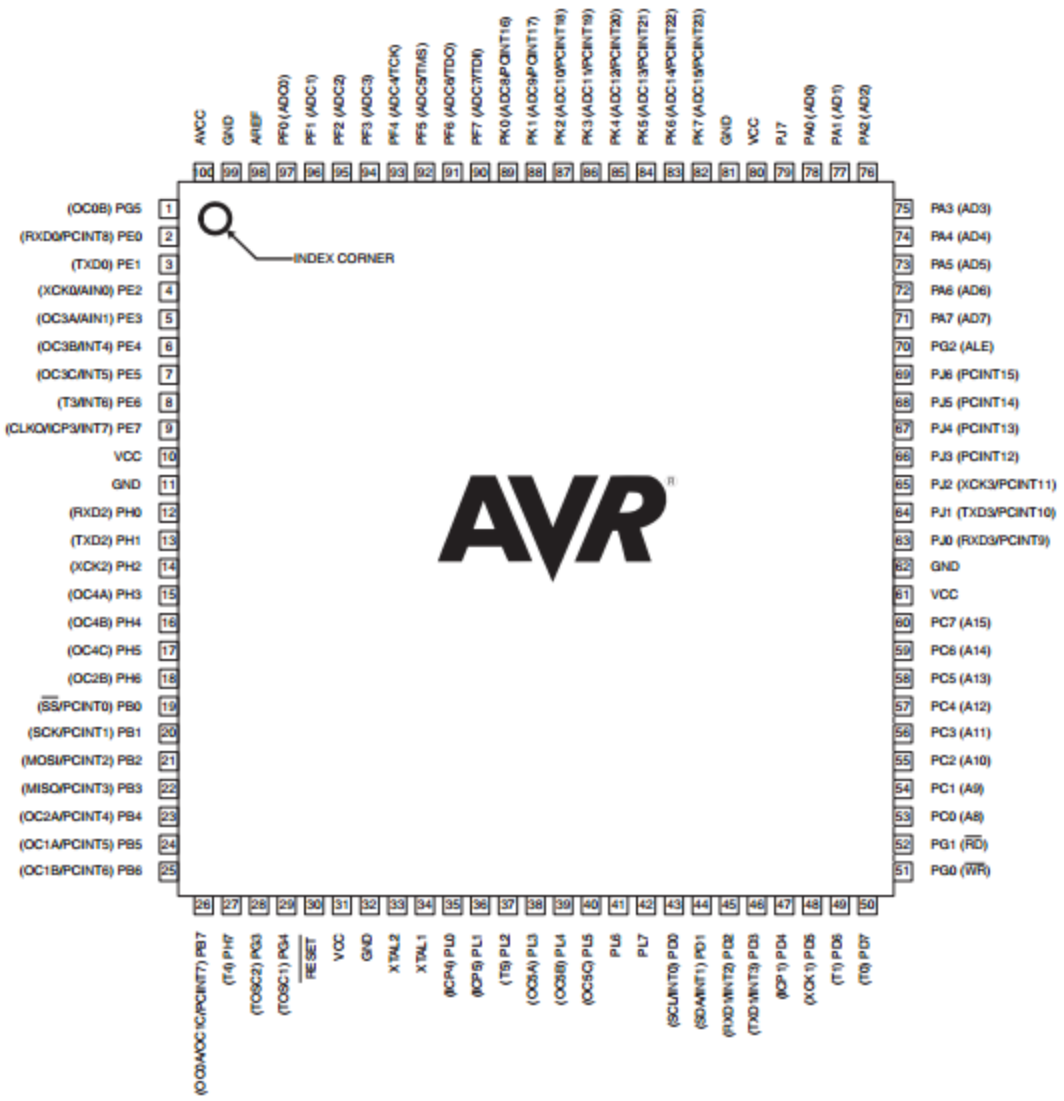


- Brown-out Reset.
  - The MCU is reset when the supply voltage VCC is below the Brown-out Reset threshold (VBOT) and the Brown-out Detector is enabled.
- JTAG AVR Reset.
  - The MCU is reset as long as there is a logic one in the Reset Register, one of the scan chains of the JTAG system.
- For each reset, there is a flag (bit) in MCU Control and State Register MCUCSR.
  - These bits are used to determine the source of the RESET interrupt.

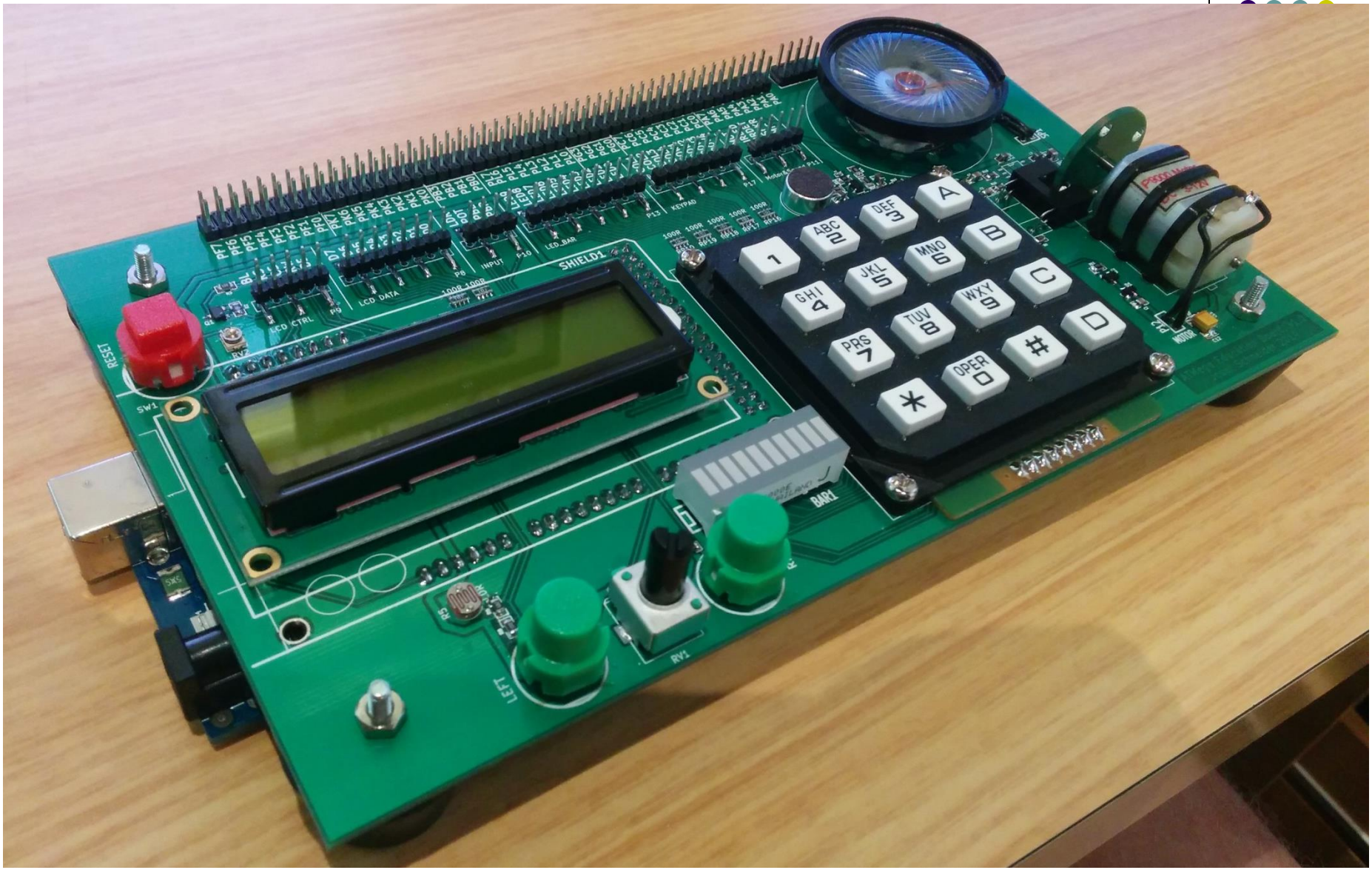
RESET Logic in Mega2560



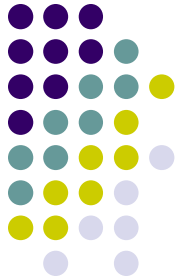
Atmega2560 Pin Configuration



Source: Atmega2560 Data Sheet

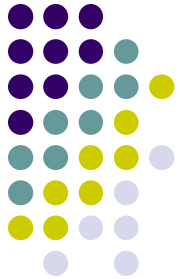

SHIELD1
RESET
SW1
LCD DATA
LCD CTRL
INPUT
LED_BAR
KEYPAD
BAR1
RV1
LEFT
1
2 ABC
3 DEF
A
4 GHI
5 JKL
6 MNO
B
7 PRS
8 TUV
9 WXY
C
*
0 OPER
#
D

Watchdog Timer



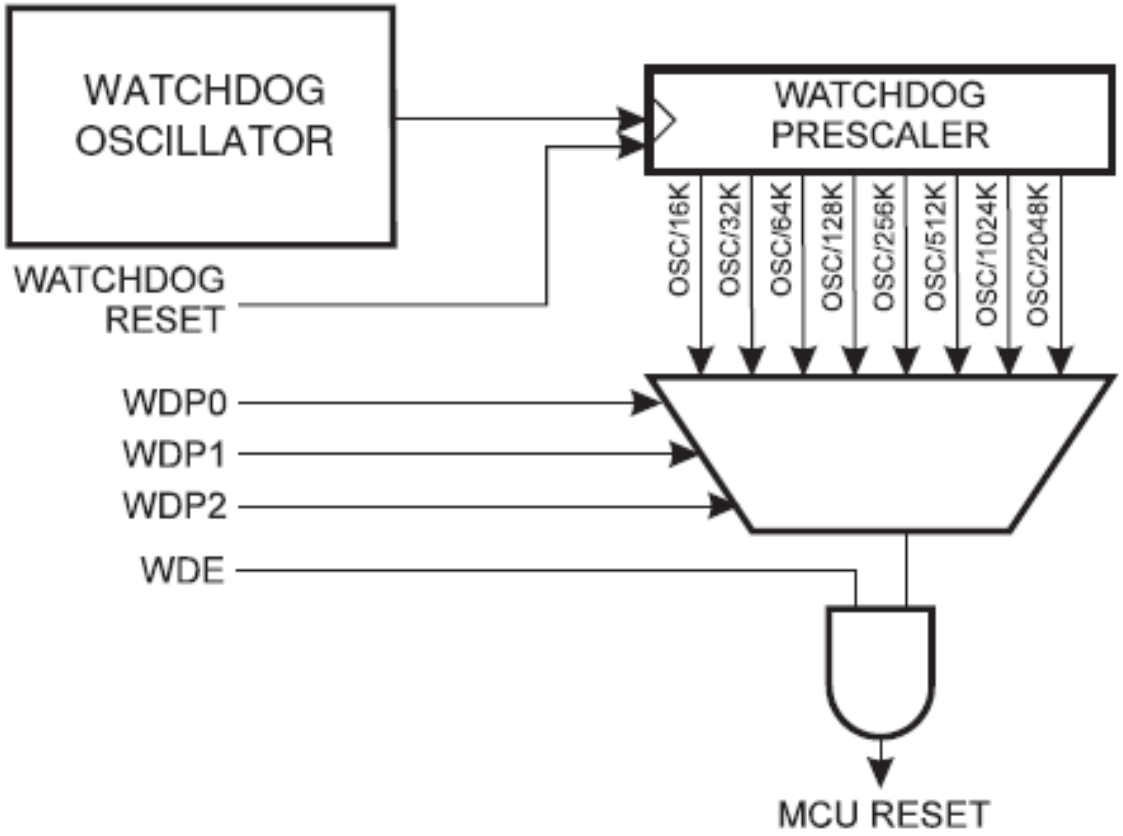
- A peripheral I/O device on the microcontroller.
- It is really a counter that is clocked from a separate On-chip Oscillator (122 kHz at Vcc=5V)
- It can be controlled to produce different time intervals
  - 8 different periods determined by WDP2, WDP1 and WDP0 bits in WDTCSR.
- Can be enabled or disabled by properly updating WDCE bit and WDE bit in Watchdog Timer Control Register WDTCSR.

Watchdog Timer (cont.)



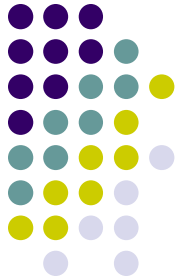
- Often used to detect software crash.
  - If enabled, it generates a Watchdog Reset interrupt when its period expires.
    - When its period expires, Watchdog Reset Flag WDRF in MCU Control Register MCUCSR is set.
      - This flag is used to determine if the watchdog timer has generated a RESET interrupt.
  - Program needs to reset it before its period expires by executing instruction *WDR*.

Watchdog Timer Diagram



Source: Atmega2560 Data Sheet

Watchdog Timer Control Register

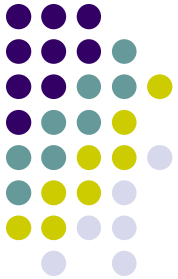


- WDTCSR is used to control the scale of the watchdog timer. It is an MM I/O register in AVR

Bit	7	6	5	4	3	2	1	0	
(0x60)	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	WDTCSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

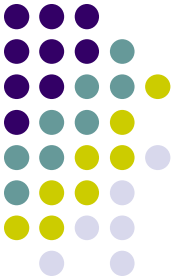
Source: Atmega2560 Data Sheet

WDTCSR Bit Definition



- Bit 7 – WDIF - Watchdog interrupt flag
- Bit 6 – WDIE - Watchdog interrupt enable
- Bit 4
  - WDCE - Watchdog change enable
    - Should be set before any changes to be made
- Bit 3
  - WDE - Watchdog enable
    - Set to enable watchdog; clear to disable the watchdog
- Bits 5,2-0
  - Prescaler
    - Named WDP3, WDP2, WDP1, WPD0
      - Determine the watchdog time reset intervals

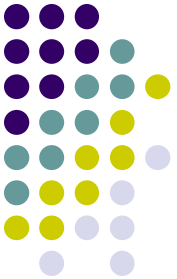
Watchdog Timer Prescale



WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 5.0V
0	0	0	0	2K (2048) cycles	16ms
0	0	0	1	4K (4096) cycles	32ms
0	0	1	0	8K (8192) cycles	64ms
0	0	1	1	16K (16384) cycles	0.125s
0	1	0	0	32K (32768) cycles	0.25s
0	1	0	1	64K (65536) cycles	0.5s
0	1	1	0	128K (131072) cycles	1.0s
0	1	1	1	256K (262144) cycles	2.0s
1	0	0	0	512K (524288) cycles	4.0s
1	0	0	1	1024K (1048576) cycles	8.0s

Source: Atmega64 Data Sheet

Examples



- Enable watchdog

```
; Write logical one to WDE

ldi r16, (1<<WDE)
sts WDTCSR, r16
```

Examples

- Disable watchdog
  - Refer to the data sheet

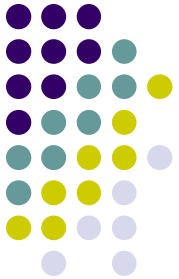
```
; Write logical one to WDCE and WDE

ldi r16, (1<<WDCE)|(1<<WDE)
sts WDTCSR, r16

; Turn off WDT
ldi r16, (0<<WDE)
sts WDTCSR, r16
```

Examples

- Select a prescale
  - Refer to the data sheet

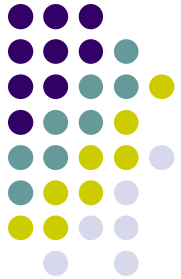
```
; Write logical one to WDCE and WDE

ldi r16, (1<<WDCE)|(1<<WDE)
sts WDTCSR, r16

; set time-out as 1 second
ldi r16, (1<<WDP2)|(1<<WDP1)
sts WDTCSR, r16
```

Watchdog Reset

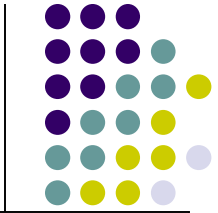


- Syntax: ***wdr***
- Operands: none
- Operation: reset the watchdog timer.
- Words: 1
- Cycles: 1

Example

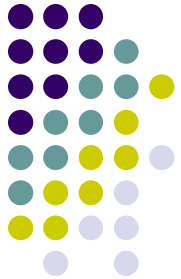


- The program in the next slide is not robust. May lead to a crash. Why? How to detect the crash?



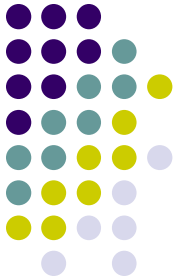
```
; The program returns the length of a string.

.include "m2560def.inc"
.def i=r15                ; store the string length when execution finishes.
.def c=r16                ; store s[i], a string character

.cseg

main:
        ldi ZL, low(s<<1)
        ldi ZH, high(s<<1)
        clr i
        lpm c, z+
loop:
        cpi c, 0
        breq endloop
        inc i
        lpm c, Z+
        rjmp loop
endloop:…

s: .db “hello, world”
```

Reading Material



- Chapter 8: Interrupts and Real-Time Events. Microcontrollers and Microcomputers by Fredrick M. Cady.
- Mega2560 Data Sheet.
  - System Control and Reset.
  - Watchdog Timer.
  - Interrupts.

Homework



1. Refer to the AVR Instruction Set manual, study the following instructions:
   - Bit operations
     - `sei`, `cli`
     - `sbi`, `cbi`
   - MCU control instructions
     - `wdr`

Homework

1. What is the function of the following code?

```
; Write logical one to WDCE and WDE
        ldi r16, (1<<WDCE)|(1<<WDE)
        sts WDTCSR, r16

; set time-out as 2.1 second
        ldi r16, (1<<WDP2)|(1<<WDP1)|(1<<WDP0)
        sts WDTCSR, r16

; enable watchdog
        ldi r16, (1<<WDE)
        sts WDTCSR, r16

loop:   oneSecondDelay          ; macro for one second delay
        wdr
        rjmp loop
```

Homework

2. How an I/O device signals the microprocessor that it needs service?

Homework

3. Why do you need software to disable interrupts (except for the non-maskable interrupts)?