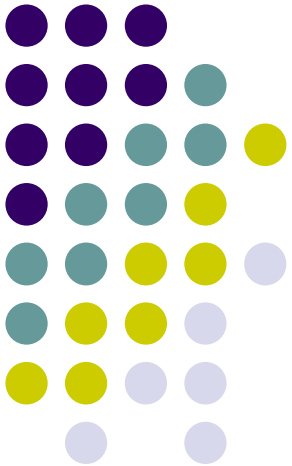
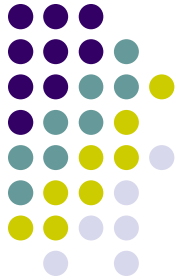
Interrupts (I)

Lecturer: Sri Parameswaran

Notes by: Annie Guo

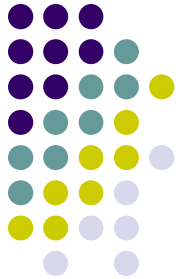


Lecture overview



- Introduction to Interrupts
  - Interrupt system specifications
  - Multiple sources of Interrupts
  - Interrupt priorities
- Interrupts in AVR
  - Interrupt Vector Table
  - Interrupt Service Routines
  - System Reset
    - Watchdog Reset

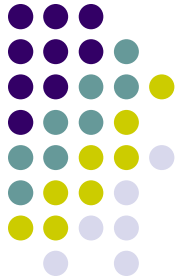
CPU Interacts with I/O



Two approaches:

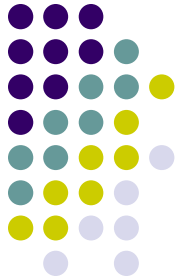
- Polling
  - Software queries I/O devices.
  - No extra hardware needed.
  - Not efficient.
    - CPU may waste processor cycles to query a device even if it does not need any service.
- Interrupts
  - I/O devices generate signals to request services from CPU .
  - Need special hardware to implement interrupt services.
  - Efficient.
    - A signal is generated only if the I/O device needs services from CPU.

Interrupt Systems



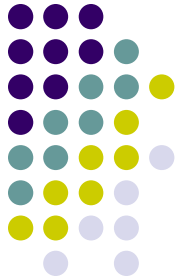
- An interrupt system implements interrupt services
- It basically performs three tasks:
  - Recognize interrupt events
  - Respond to the interrupts
  - Resume normal programmed task

Recognize Interrupt Events



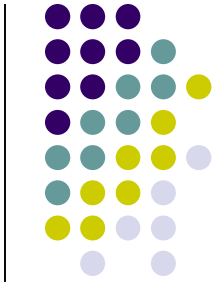
- Interrupt events
  - Associated with interrupt signals:
    - In different forms, including levels and edges.
  - Can be multiple and synchronous
    - Namely, there may be many sources to generate an interrupts; a number of interrupts can be generated at the same time.
- Approaches are required to:
  - Identify an interrupt event among multiple sources
  - Determine which interrupts to serve if there are multiple simultaneous interrupts

Respond to Interrupts



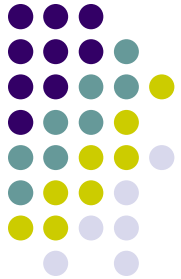
- Handling interrupt
  - Wait for the current instruction to finish.
  - Acknowledge the interrupting device.
  - Branch to the correct ***interrupt service routine*** (interrupt handler) to service interrupting device.

Resume Normal Task



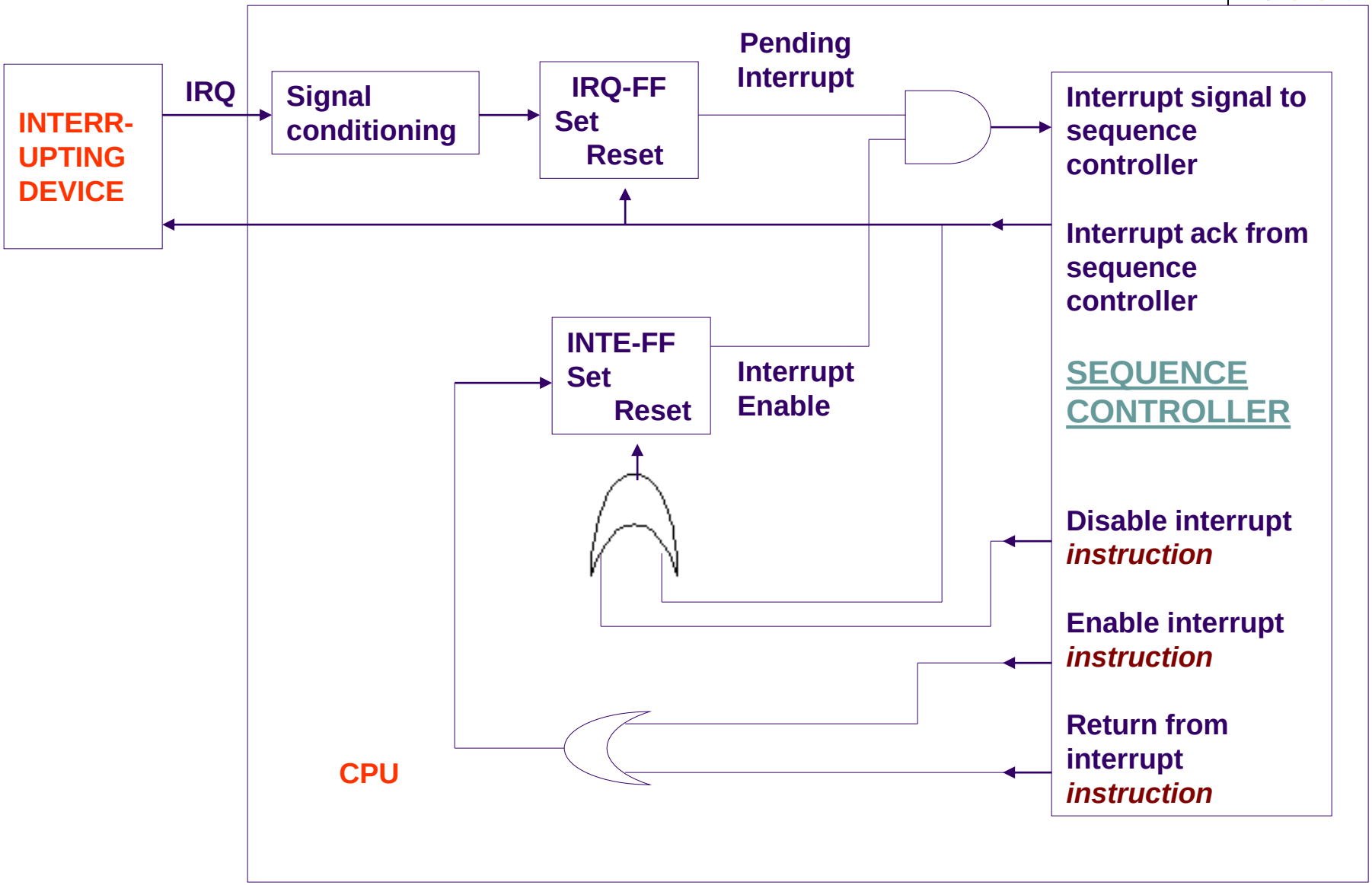
- Return to the interrupted program at the point it was interrupted.

Interrupt Process Control

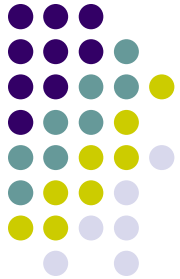


- Interrupts can be enabled or disabled
- Can be controlled in two ways:
  - Software control
    - Allow programmers to enable and disable selected/all interrupts.
  - Hardware control
    - Disable further interrupts while an interrupt is being serviced

Interrupt Recognition and Acknowledgement Hardware

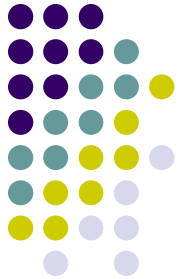


Interrupt Recognition and Ack.



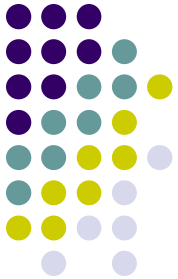
- An Interrupt Request (IRQ) may occur at any time.
  - It may have rising or falling edges or high or low levels.
  - Frequently it is an active-low signal
    - multiple devices are wire-ORed together.
      - Recall open-collector gates
- Signal Conditioning Circuit detects these different types of 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.)



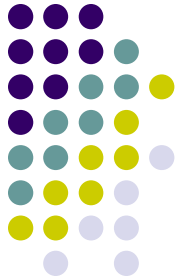
- Interrupts can be enabled and disabled by software instructions, which is supported by the hardware Interrupt Enable Flip-Flop (INTE-FF).
- When the INTE-FF is set, all interrupts are enabled and the pending interrupt is allowed through the AND gate to the sequence controller.
- The INTE-FF is reset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

Interrupt Recognition and Ack. (cont.)



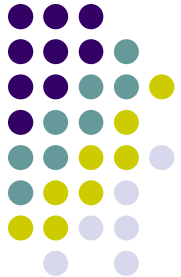
- An interrupt acknowledge signal is generated by the CPU when the current instruction has finished execution and CPU has detected the IRQ.
  - This resets the IRQ-FF and INTE-FF and signals the interrupting device that CPU is ready to execute the interrupting device routine.
- At the end of the interrupt 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



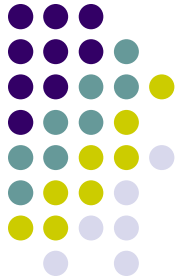
- To handle multiple sources of interrupts, the interrupt system must
  - Identify which device has generated the IRQ.
    - Using polling approach
    - Using vectoring approach
  - Resolve simultaneous interrupt requests
    - using prioritization schemes.

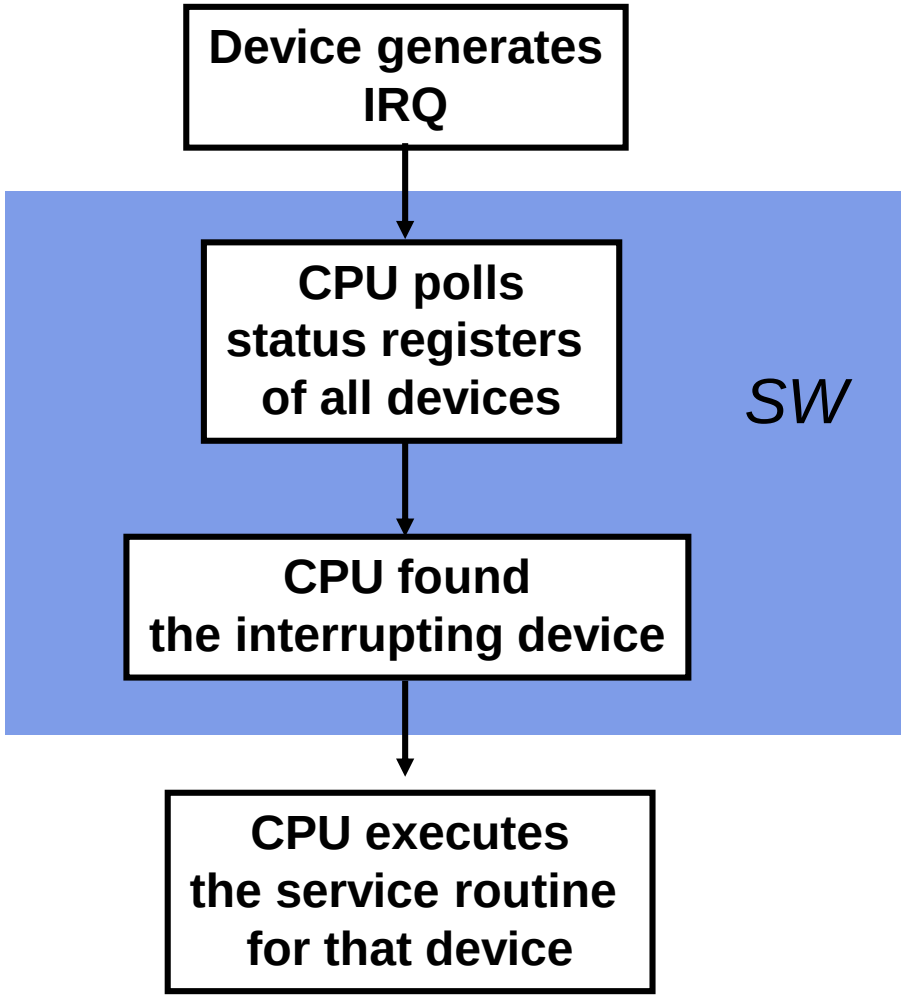
Polled Interrupts



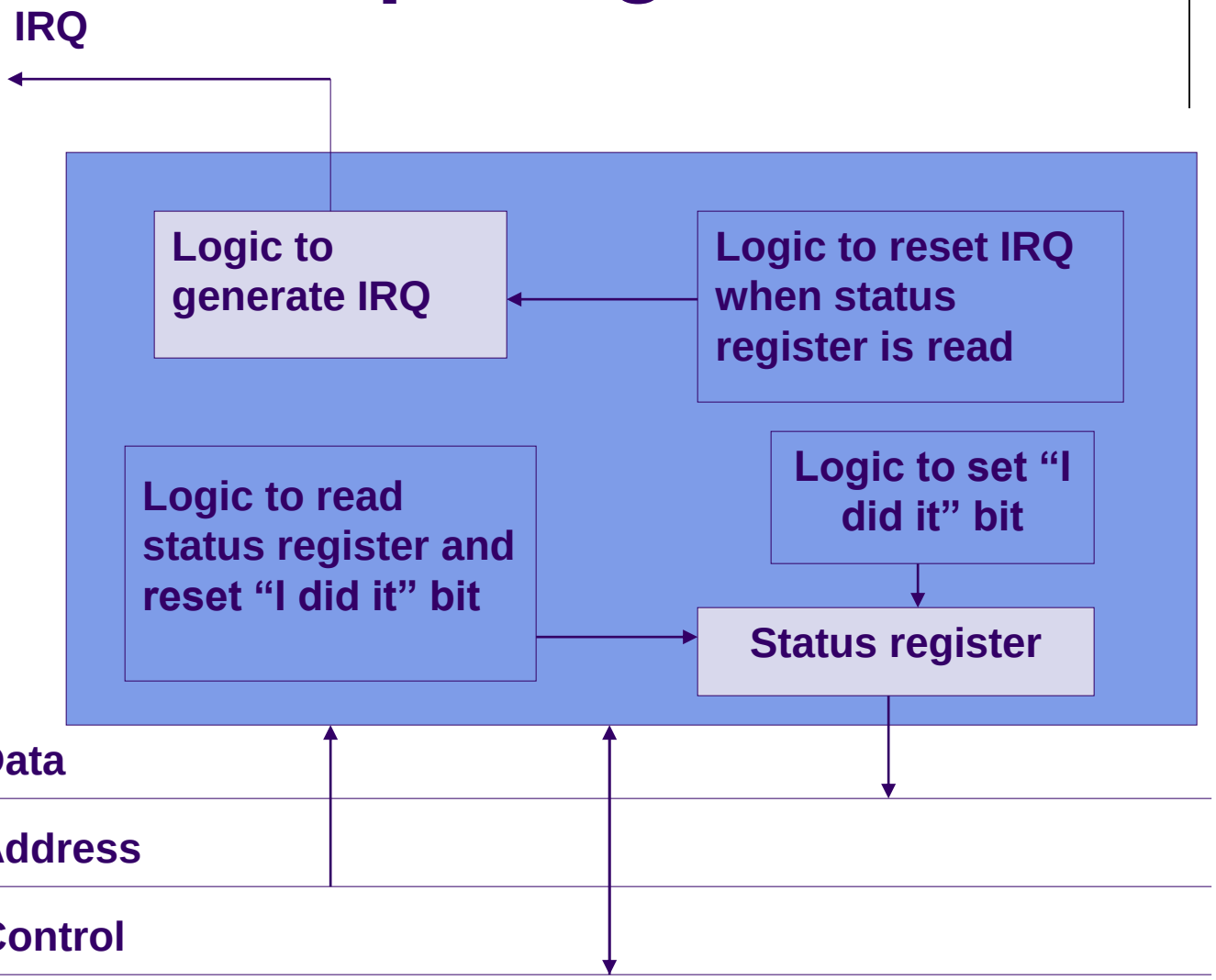
- Software, instead of hardware, is responsible for finding the interrupting source.
  - The device must have logic to generate the IRQ signal and to set an “I did it” bit in a status register that is read by CPU.
  - The bit is reset after the register has been read.

Polled Interrupts Execution Flow

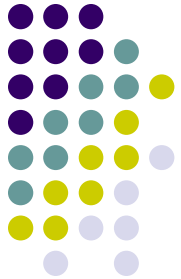




Polled Interrupt Logic

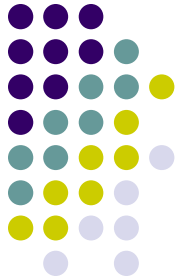


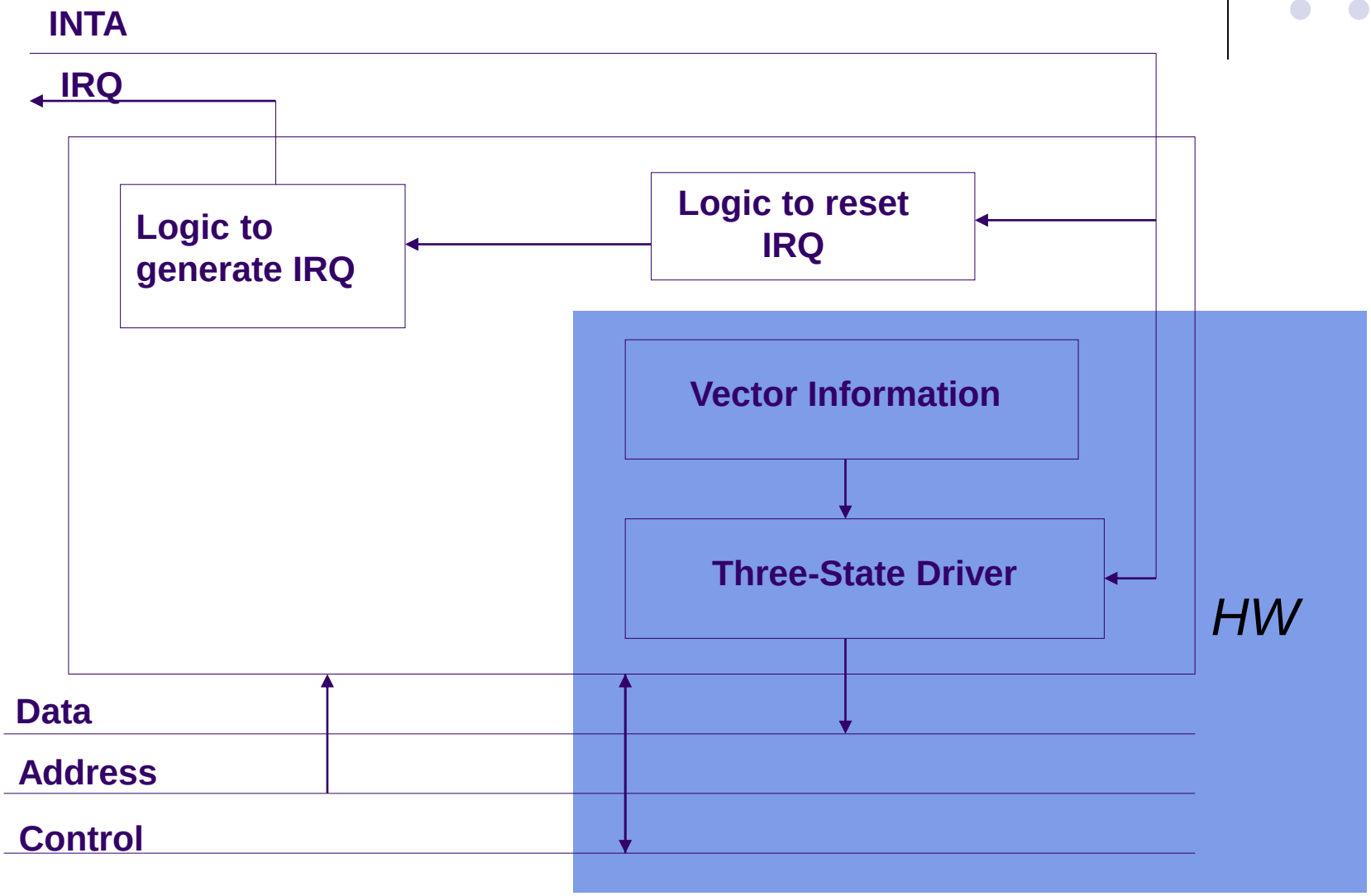
Vectored Interrupts



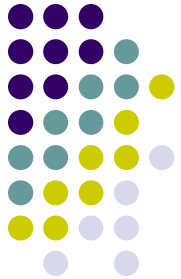
- CPU’s response to IRQ is to assert INTA.
- The interrupting device uses INTA to place information that identifies itself, called the vector, onto the data bus for CPU to read.
- CPU uses the vector to execute the interrupt service routine.

Vectored Interrupting Device Hardware



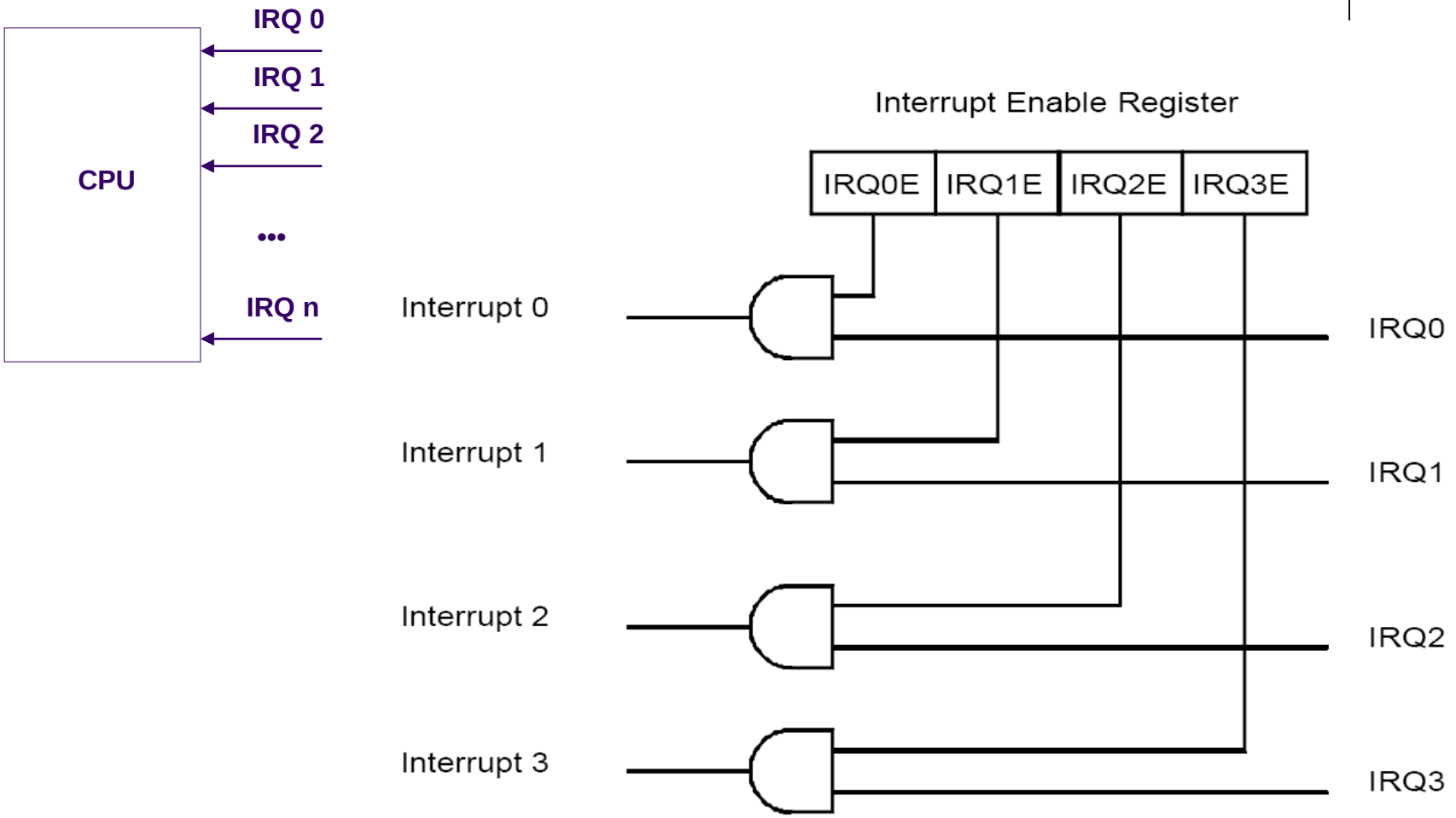


Multiple Interrupt Masking

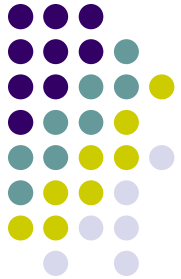


- CPU has multiple IRQ input pins.
- Masking enables some interrupts and disables other interrupts
- CPU designers reserve specific memory locations for a vector associated with each IRQ line.
- Individual disable/enable bit is assigned to each interrupting source.

Multiple Interrupt Masking Circuit

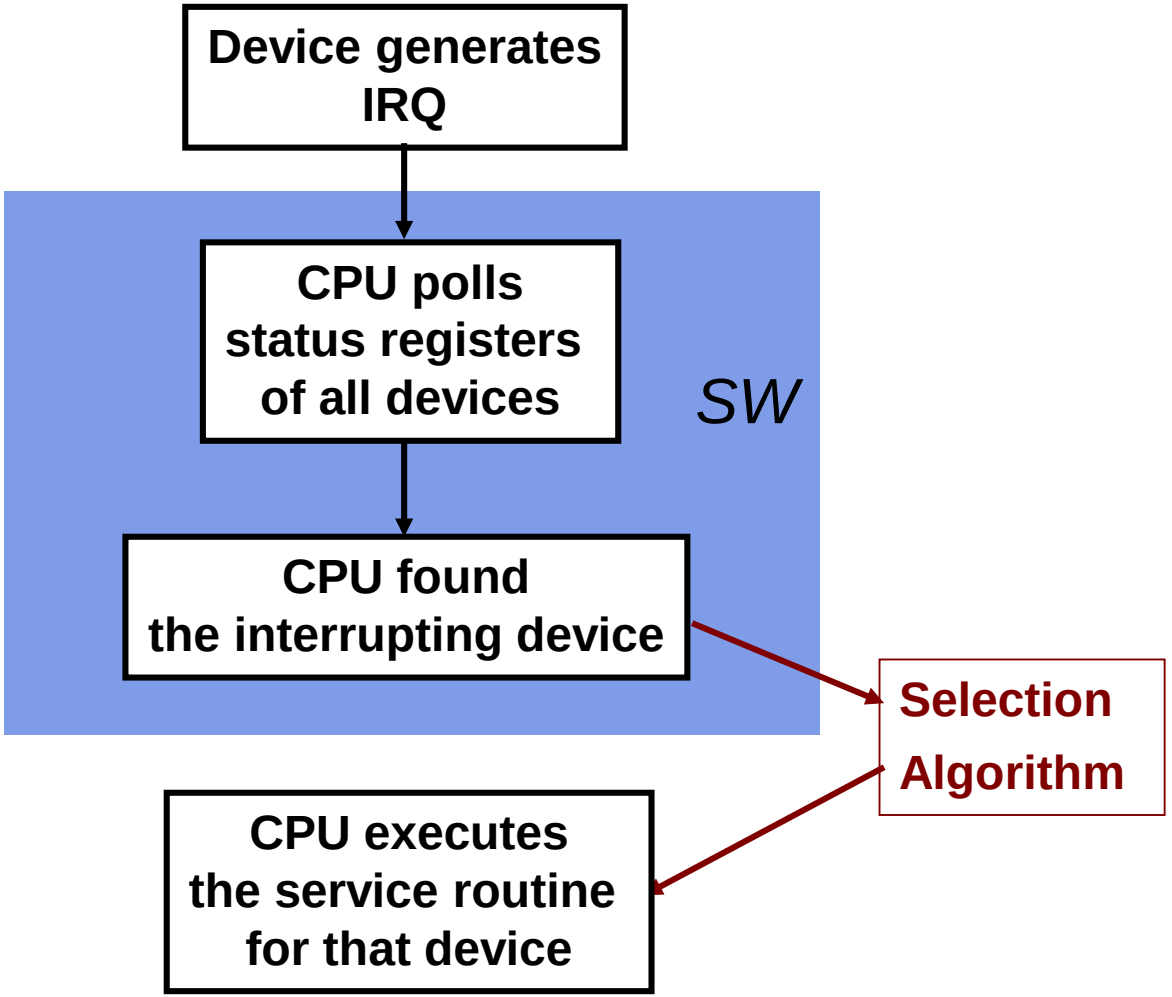


Interrupt Priorities

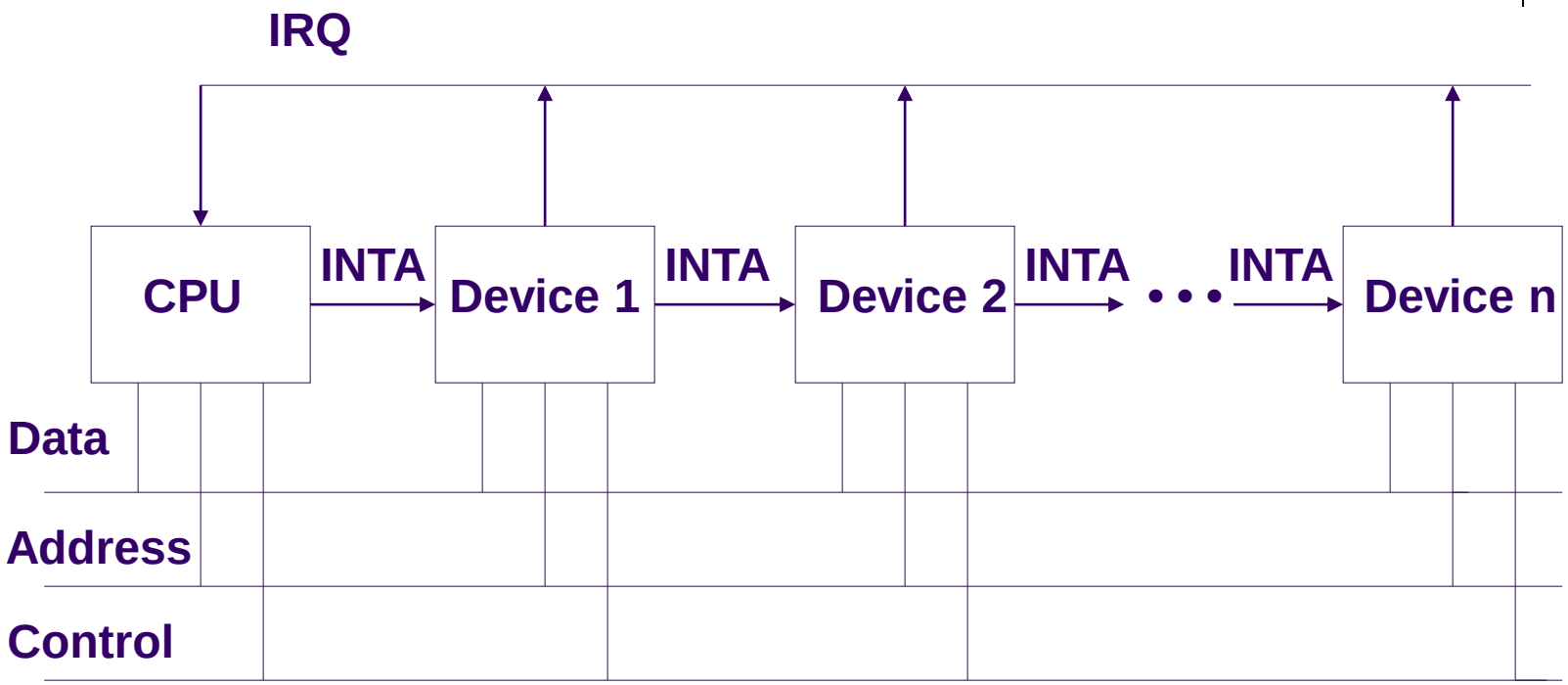


- When multiple interrupts occur at the same time, which one will be serviced first?
- Two resolution approaches:
  - Software resolution
    - Polling software determines which interrupting source is serviced first.
  - Hardware resolution
    - Daisy chain.
    - Others

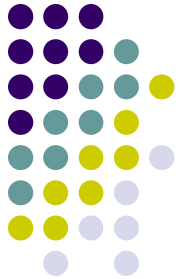
Software Resolution



Daisy Chain Priority Resolution

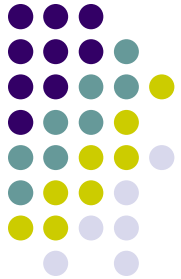


Daisy Chain Priority Resolution (cont.)



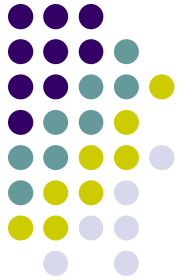
- CPU asserts INTA that is passed down the chain from device to device. The higher-priority device is closer to CPU.
- When the INTA reaches a device that generated the IRQ, that device puts its vector on the data bus and does not pass along the INTA. So lower-priority devices do NOT receive the INTA.

Other Priority Resolutions



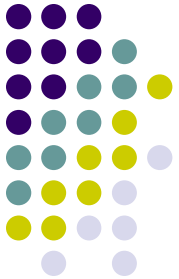
- Separate IRQ Lines.
  - Each IRQ line is assigned a fixed priority. For example, IRQ0 has higher priority than IRQ1 and IRQ1 has higher priority than IRQ2 and so on.
- Hierarchical Prioritization.
  - Higher priory interrupts are allowed while lower ones are masked.
- Nonmaskable Interrupts.
  - Cannot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

Transferring Control to Interrupt Service Routine



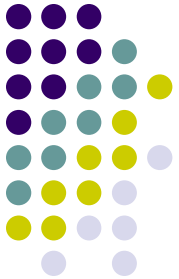
- Hardware needs to save the return address.
  - Most processors save the return address on the stack.
- Hardware may also save some registers such as program status register.
  - AVR does not save any registers. It is the programmers' responsibility to save the program status register and conflict registers.
- The delay from the time the IRQ is generated by the interrupting device to the time the Interrupt Service Routine (ISR) starts to execute is called *interrupt latency*.

Interrupt Service Routine



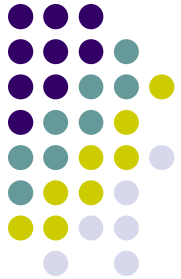
- A sequence of code to be executed when the corresponding interrupt is responded by CPU.
- Interrupt service routine is a special subroutine, therefore can be constructed with three parts:
  - Prologue:
    - Code for saving conflict registers on the stack.
  - Body:
    - Code for doing the required task.
  - Epilogue:
    - Code for restoring all saved registers from the stack.
    - The last instruction is the return-from-interrupt instruction.

Software Interrupt



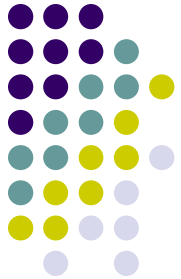
- Software interrupt is the interrupt generated by software without a hardware-generated-IRQ.
- Software interrupt is typically used to implement system calls in OS.
- Some processors have a special machine instruction to generate software interrupt.
  - SWI in ARM.
- AVR does NOT provide a software interrupt instruction.
  - Programmers can use External Interrupts to implement software interrupts.

Exceptions



- Abnormalities that occur during the normal operation of the processor.
  - Examples are internal bus error, memory access error and attempts to execute illegal instructions.
- Some processors handle exceptions in the same way as interrupts.
  - AVR does not handle exceptions.

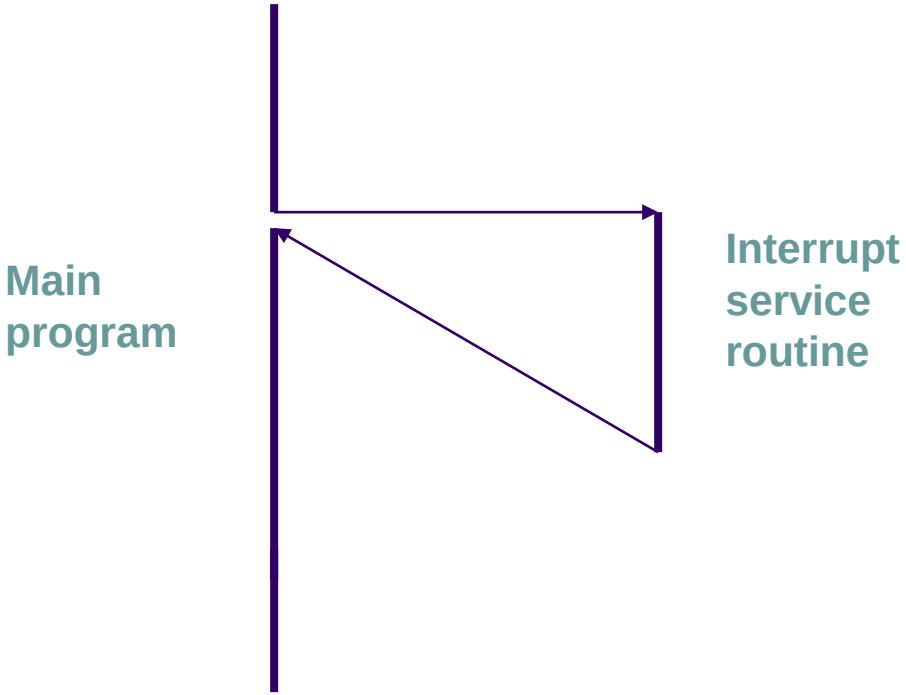
Reset



- Reset is a type of interrupt present in most processors (including AVR).
- Non-maskable.
- It does not do other interrupt processes, such as saving conflict registers. It initializes the system to some initial state.

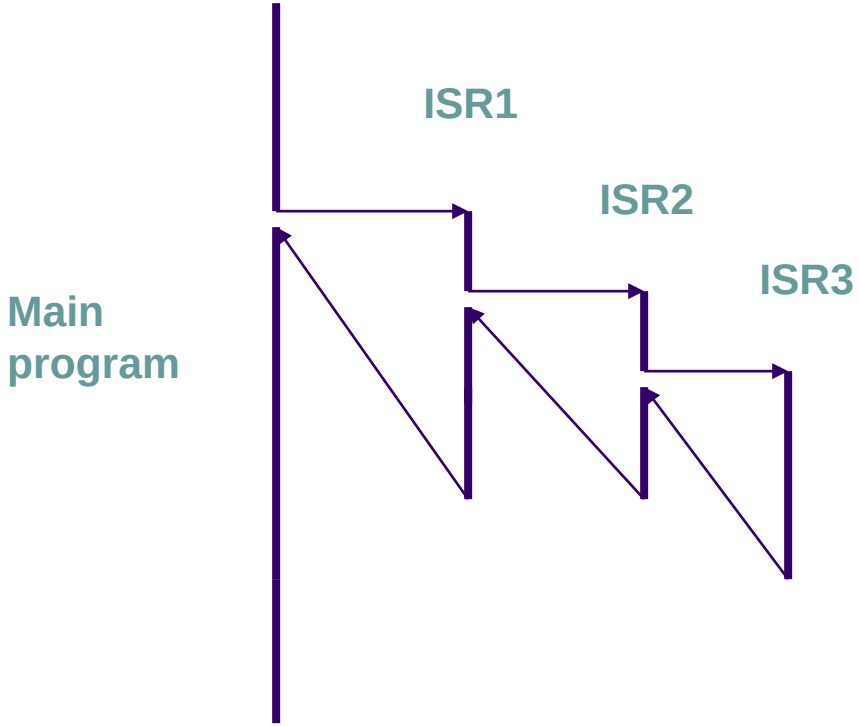
Non-Nested Interrupts

- Interrupt service routines cannot be interrupted by another interrupt.

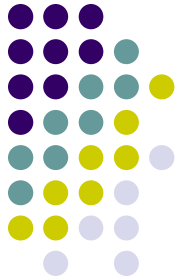


Nested Interrupts

- Interrupt service routines can be interrupted by another interrupt.

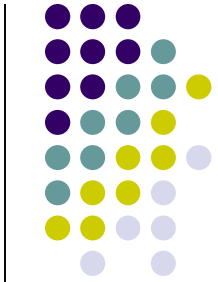


AVR Interrupts



- Basically can be divided into internal and external interrupts
- Each has a dedicated interrupt vector
- Hardware is used to recognize interrupts
- To enable an interrupt, two control bits must be set
  - the Global Interrupt Enable bit (I bit) in the Status Register
    - Using `sei`
  - the enable bit for that interrupt
- To disable all maskable interrupts, reset the I bit in SREG
  - Using `cli` instruction
- Priority of interrupts is used to handle multiple simultaneous interrupts

Set Global Interrupt Flag



- Syntax: ***sei***
- Operands: none
- Operation: I ← 1.
  - Sets the global interrupt flag (I) in SREG. The instruction following SEI will be executed before any pending interrupts.
- Words: 1
- Cycles: 1
- Example:

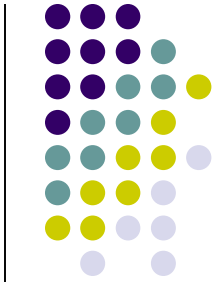
```
sei     ; set global interrupt enable
sleep   ; enter sleep state, waiting for an interrupt
```

Clear Global Interrupt Flag

- Syntax: ***cli***
- Operands: none
- Operation: I ← 0
  - Clears the Global interrupt flag in SREG. Interrupts will be immediately disabled.
- Words: 1
- Cycles: 1
- Example:

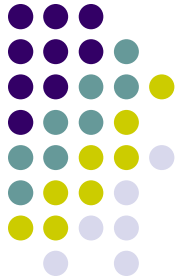
```
in r18, SREG          ; store SREG value
cli                   ; disable interrupts
; do something very important here
out SREG, r18         ; restore SREG value
```

Interrupt Response Time



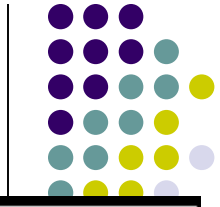
- The interrupt execution response for all the enabled AVR interrupts is basically five clock cycles minimum.
  - For saving the Program Counter (2 clock cycles)
  - For jumping to the interrupt routine (3 clock cycles)

Interrupt Vectors



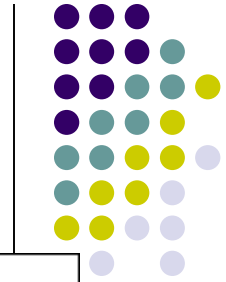
- Each interrupt has a 4-byte (2-word) interrupt vector, containing an instruction to be executed after MCU has accepted the interrupt.
- The lowest addresses in the program memory space are by default defined as the section for Interrupt Vectors.
- The priority of an interrupt is based on the position of its vector in the program memory
  - The lower the address the higher is the priority level.
- RESET has the highest priority

Interrupt Vectors in Mega2560



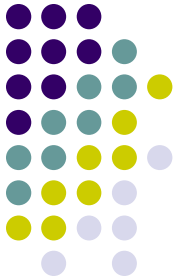
Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	PCINT0	Pin Change Interrupt Request 0
11	$0014	PCINT1	Pin Change Interrupt Request 1
12	$0016(3)	PCINT2	Pin Change Interrupt Request 2

Interrupt Vectors in Mega2560 (cont.)



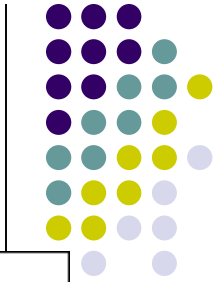
13	$0018	WDT	Watchdog Time-out Interrupt
14	$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B
16	$001E	TIMER2 OVF	Timer/Counter2 Overflow
17	$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	$002A	TIMER0 COMPA	Timer/Counter0 Compare Match A
23	$002C	TIMER0 COMPB	Timer/Counter0 Compare match B
24	$002E	TIMER0 OVF	Timer/Counter0 Overflow
25	$0030	SPI, STC	SPI Serial Transfer Complete
26	$0032	USART0 RX	USART0 Rx Complete
27	$0034	USART0 UDRE	USART0 Data Register Empty
28	$0036	USART0 TX	USART0 Tx Complete
29	$0038	ANALOG COMP	Analog Comparator

Interrupt Vectors in Mega2560 (cont.)



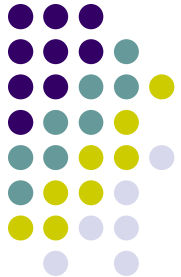
30	$003A	ADC	ADC Conversion Complete
31	$003C	EE READY	EEPROM Ready
32	$003E	TIMER3 CAPT	Timer/Counter3 Capture Event
33	$0040	TIMER3 COMPA	Timer/Counter3 Compare Match A
34	$0042	TIMER3 COMPB	Timer/Counter3 Compare Match B
35	$0044	TIMER3 COMPC	Timer/Counter3 Compare Match C
36	$0046	TIMER3 OVF	Timer/Counter3 Overflow
37	$0048	USART1 RX	USART1 Rx Complete
38	$004A	USART1 UDRE	USART1 Data Register Empty
39	$004C	USART1 TX	USART1 Tx Complete
40	$004E	TWI	2-wire Serial Interface
41	$0050	SPM READY	Store Program Memory Ready

Interrupt Vectors in Mega2560 (cont.)



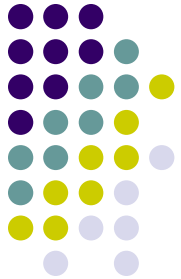
41	$0050	SPM READY	Store Program Memory Ready
42	$0052(3)	TIMER4 CAPT	Timer/Counter4 Capture Event
43	$0054	TIMER4 COMPA	Timer/Counter4 Compare Match A
44	$0056	TIMER4 COMPB	Timer/Counter4 Compare Match B
45	$0058	TIMER4 COMPC	Timer/Counter4 Compare Match C
46	$005A	TIMER4 OVF	Timer/Counter4 Overflow
47	$005C(3)	TIMER5 CAPT	Timer/Counter5 Capture Event
48	$005E	TIMER5 COMPA	Timer/Counter5 Compare Match A
49	$0060	TIMER5 COMPB	Timer/Counter5 Compare Match B
50	$0062	TIMER5 COMPC	Timer/Counter5 Compare Match C
51	$0064	TIMER5 OVF	Timer/Counter5 Overflow
52	$0066(3)	USART2 RX	USART2 Rx Complete
53	$0068(3)	USART2 UDRE	USART2 Data Register Empty
54	$006A(3)	USART2 TX	USART2 Tx Complete
55	$006C(3)	USART3 RX	USART3 Rx Complete
56	$006E(3))	USART3 UDRE	USART3 Data Register Empty
57	$0070(3)	USART3 TX	USART3 Tx Complete

Interrupt Process



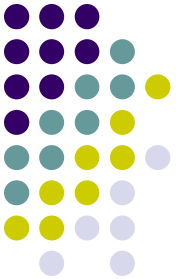
- When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled.
- The interrupt routine can set the I-bit to allow nested interrupts
- The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed.
- When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.
  - Reset interrupt is an exception

Initialization of Interrupt Vector Table in Mega2560



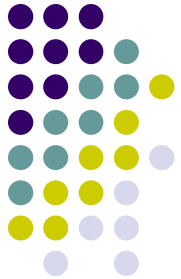
- Typically an interrupt vector contains a branch instruction (JMP or RJMP) that branches to the first instruction of the interrupt service routine.
- Or simply RETI (return-from-interrupt) if you don’t handle this interrupt.

Example of IVT Initialization in Mega2560



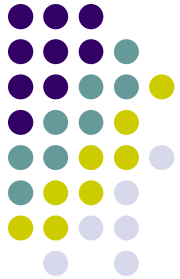
```
.include "m2560def.inc"
.cseg
.org 0x0000       ; Reset vector is at address 0x0000
rjmp RESET        ; Jump to the start of Reset interrupt service routine
                  ; Relative jump is used assuming RESET is not far
.org INT0addr     ; Addresses of vectors are defined in m2560def.inc
jmp IRQ0          ; Long jump is used assuming IRQ0 is very far away
.org INT1addr
reti              ; Return to the break point without handling the interrupt
…
RESET:            ; The interrupt service routine for RESET starts here.
…
IRQ0:             ; The interrupt service routine for IRQ0 starts here.
```

RESET in Mega2560



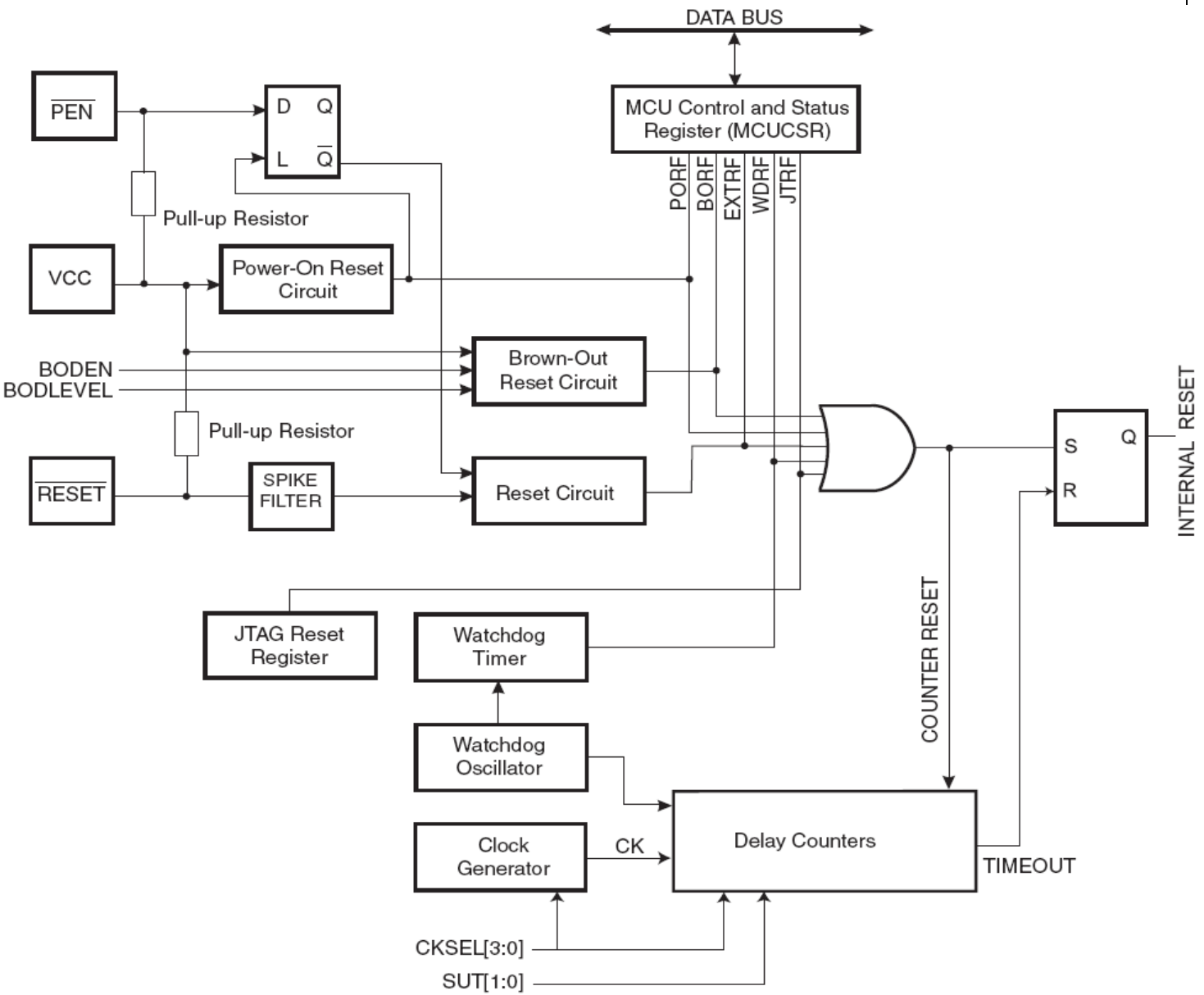
- The ATmega2560 has five sources of reset:
  - Power-on Reset.
    - The MCU is reset when the supply voltage is below the Power-on Reset threshold (VPOT).
  - External Reset.
    - The MCU is reset when a low level is present on the RESET pin for longer than the minimum pulse length.
  - Watchdog Reset.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

RESET in Mega2560 (Cont.)

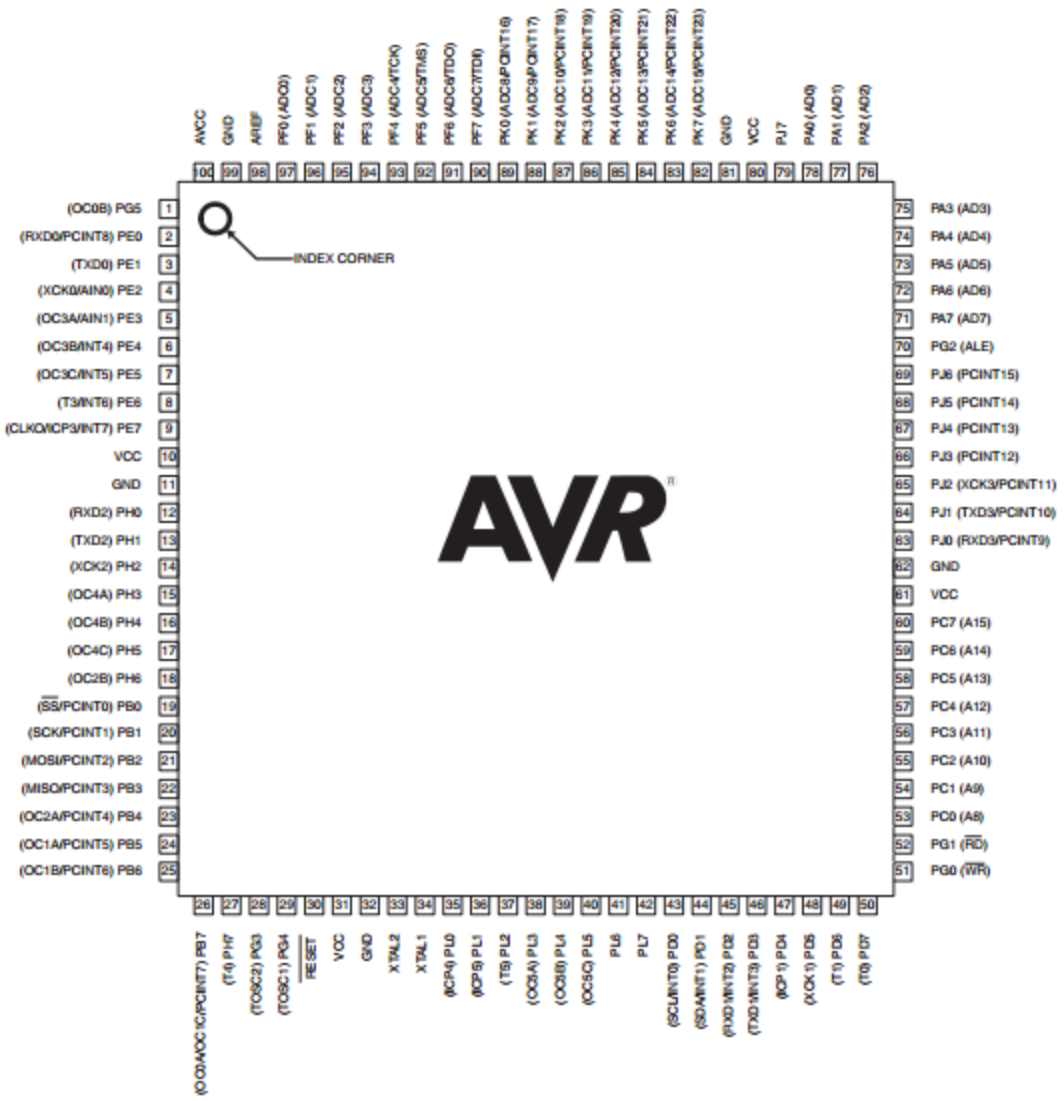


- Brown-out Reset.
  - The MCU is reset when the supply voltage VCC is below the Brown-out Reset threshold (VBOT) and the Brown-out Detector is enabled.
- JTAG AVR Reset.
  - The MCU is reset as long as there is a logic one in the Reset Register, one of the scan chains of the JTAG system.
- For each 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.

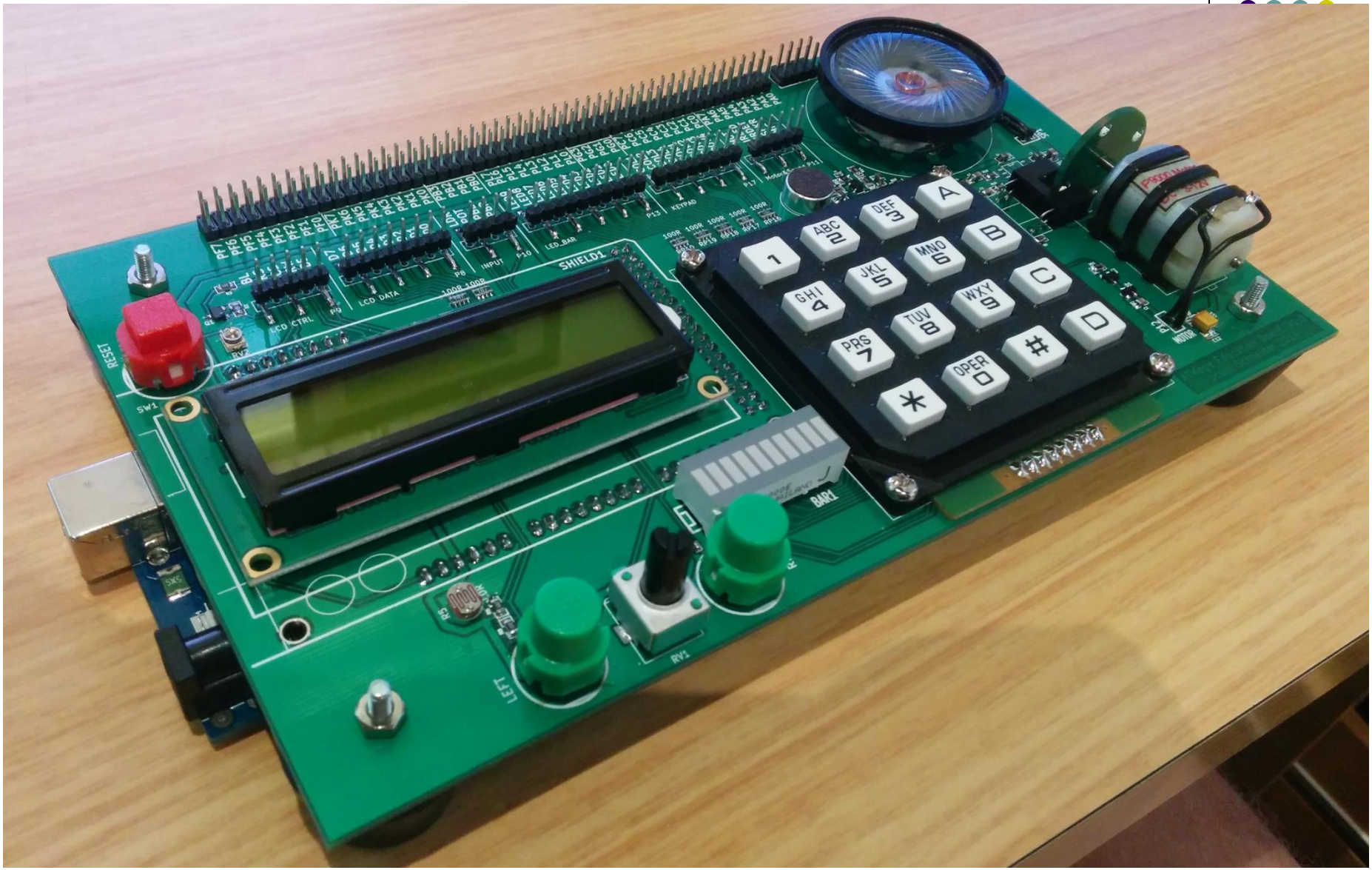
RESET Logic in Mega2560



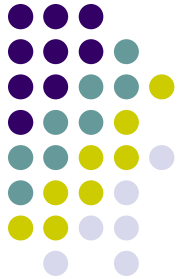
Atmega2560 Pin Configuration



Source: Atmega2560 Data Sheet

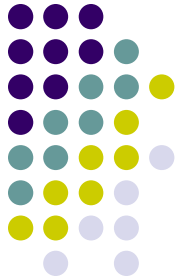

SHIELD1
LCD DATA
LCD CTRL
INPUT
LED_BAR
KEYPAD
RESET
SW1
RV2
BAR1
RV1
LEFT
1
2 ABC
3 DEF
A
4 GHI
5 JKL
6 MNO
B
7 PRS
8 TUV
9 WXY
C
*
0 OPER
#
D

Watchdog Timer



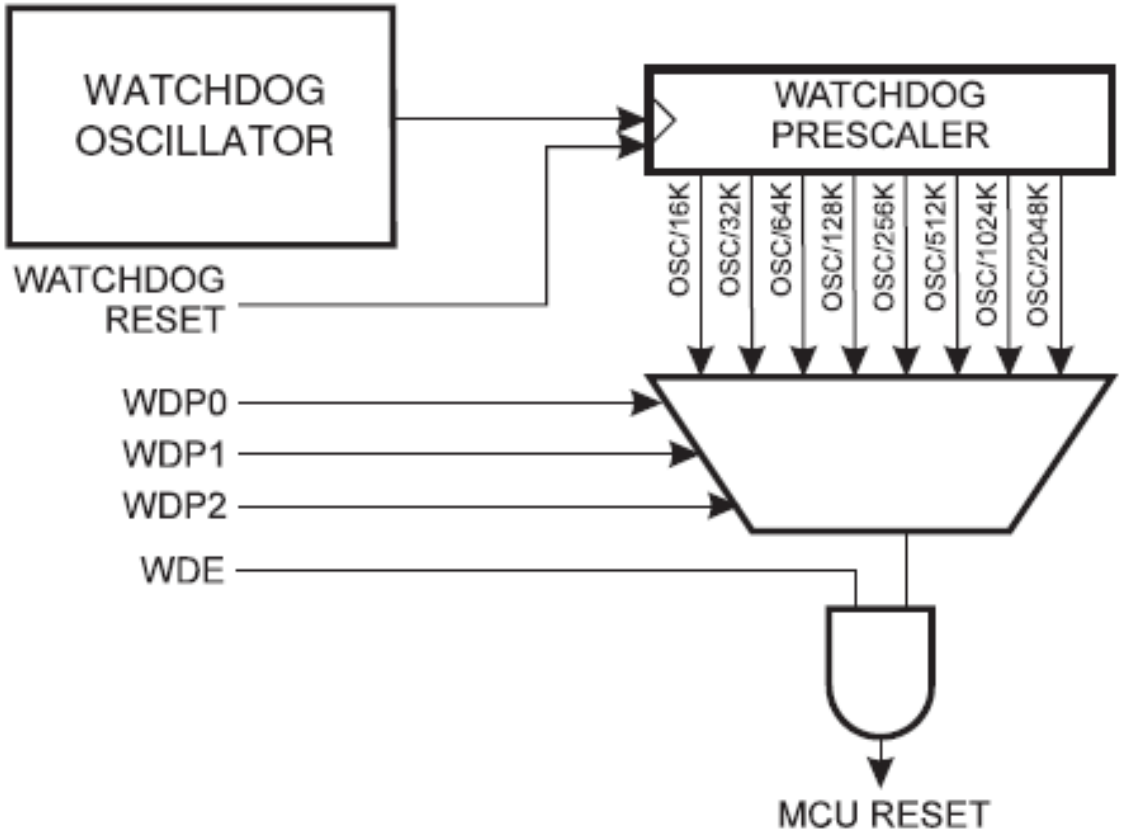
- A peripheral I/O device on the microcontroller.
- It is really a counter that is clocked from a separate On-chip Oscillator (122 kHz at Vcc=5V)
- It can be controlled to produce different time intervals
  - 8 different periods determined by WDP2, WDP1 and WDP0 bits in WDTCSR.
- Can be enabled or disabled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

Watchdog Timer (cont.)



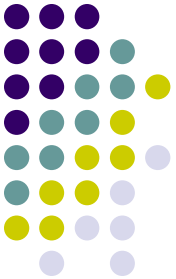
- Often used to detect software crash.
  - If enabled, it generates a Watchdog Reset interrupt when its period expires.
    - When its period expires, Watchdog Reset Flag WDRF in MCU Control Register MCUCSR is set.
      - This flag is used to determine if the watchdog timer has generated a RESET interrupt.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

Watchdog Timer Diagram



Source: Atmega2560 Data Sheet

Watchdog Timer Control Register

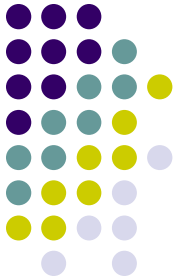


- WDTCSR is used to control the scale of the watchdog timer. It is an MM I/O register in AVR

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	

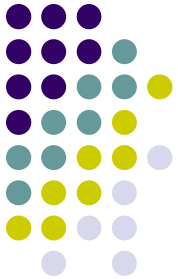
Source: Atmega2560 Data Sheet

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

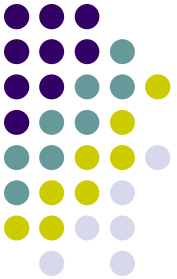
Watchdog Timer Prescale



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

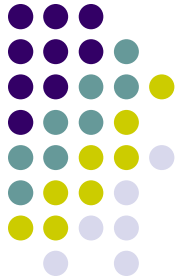
Examples



- Enable watchdog

```
; Write logical one to WDE

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

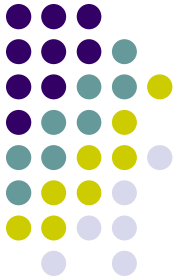
Examples



- 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
```

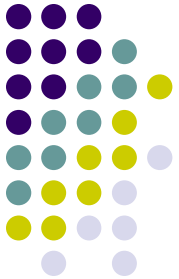
Examples



- Select a prescale
  - Refer to the data sheet

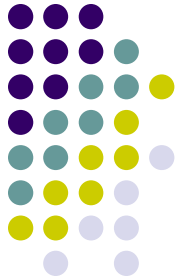
```
; Write logical one to WDCE and WDE

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

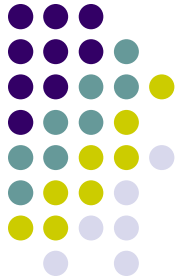
Example



- 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)?