Microprocessors & Interfacing

Interrupt (I)

Lecturer : Annie Guo

Lecture Overview

- Introduction to Interrupt
- Interrupts in AVR
  - Interrupt vector table
  - Interrupt service routine
  - System reset
  - External interrupt

CPU Interaction with I/O

Two typical approaches:

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

Interrupt System

- An interrupt system implements interrupt services
- It basically performs three tasks:
  - Detecting interrupt event
  - Responding to interrupt
  - Resuming normal programmed task

Detect Interrupt Event

- Interrupt event
  - Associated with interrupt signal
    - In different forms, including signal levels and edges.
  - 连环的，多项的 同时进行的
  - Can be multiple and simultaneous
    - There may be many sources to generate an interrupt;
    - A number of interrupts can be generated at the same time.
- Approaches are required to
  - Identify an interrupt event among multiple sources
  - Determine which interrupt to serve if there are multiple simultaneous interrupts

Respond to Interrupt

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

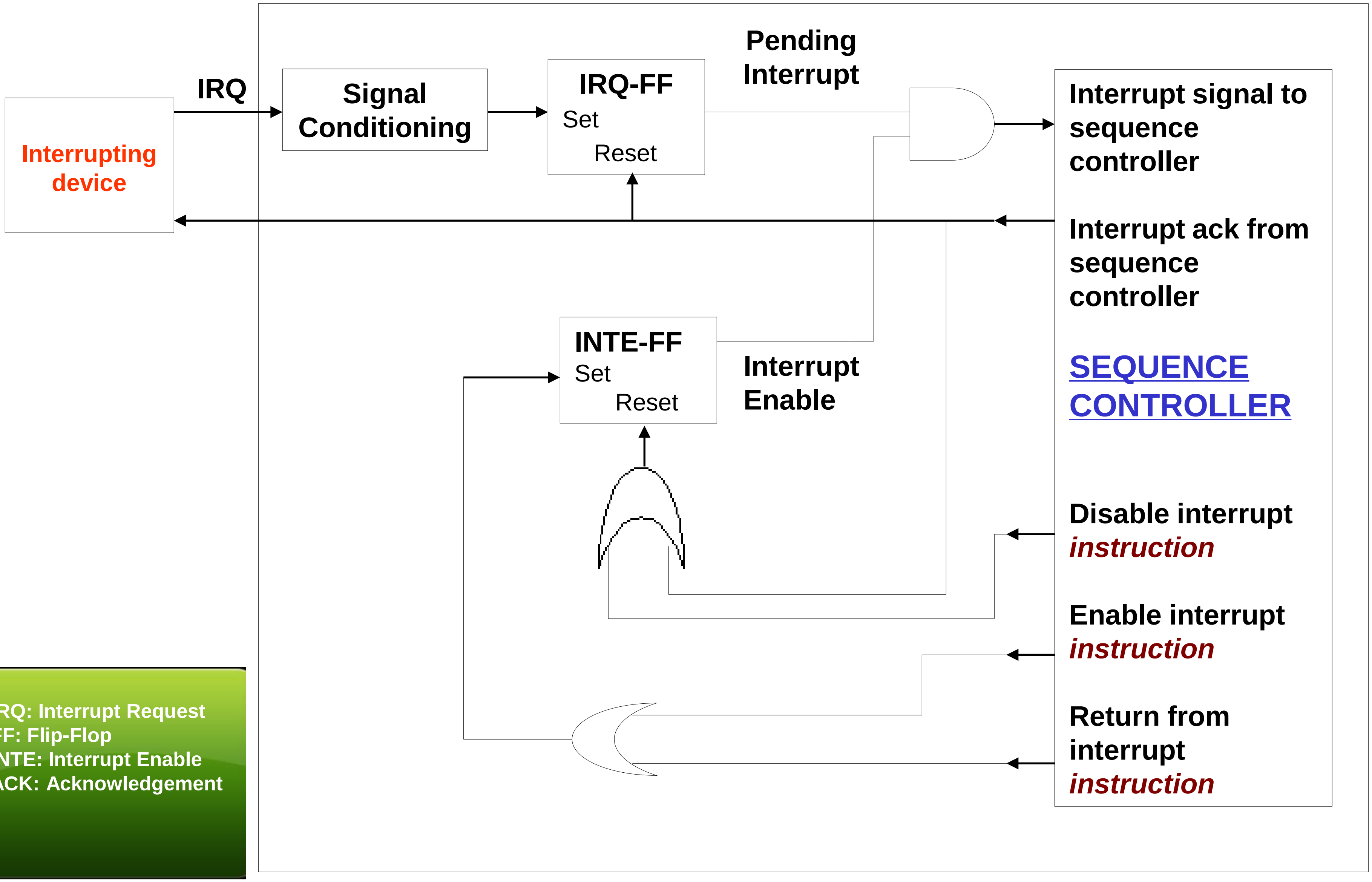
Resume Normal Task

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

Interrupt Process Control

- Interrupts can be enabled or disabled
  - Special cases exist, for example *reset*
    - Will be covered later
- Can be controlled in two ways:
  - Software control
    - Allows code to enable and disable selected or all interrupts.
  - Hardware control
    - E.g. disable further interrupts while an interrupt is being serviced

Interrupt Detection and Acknowledgement Hardware*


Interrupting device
IRQ
Signal Conditioning
IRQ-FF
Set
Reset
Pending Interrupt
Interrupt signal to sequence controller
Interrupt ack from sequence controller
SEQUENCE CONTROLLER
INTE-FF
Set
Reset
Interrupt Enable
Disable interrupt *instruction*
Enable interrupt *instruction*
Return from interrupt *instruction*

- •IRQ: Interrupt Request
- •FF: Flip-Flop
- •INTE: Interrupt Enable
- •ACK: Acknowledgement

Interrupt Detection and Acknowledgement*

- An interrupt request (IRQ) may occur at any time.
  - It can be represented by signal's rising or falling edges, or high or low levels.
- Signal Conditioning circuit detects the request.
- Interrupt Request Flip-Flop (IRQ-FF) holds 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 the controller acknowledges the interrupt with INTA signal.

Interrupt Detection and Acknowledgement (cont.)*

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

Interrupt Detection and Acknowledgement (cont.)*

- An interrupt acknowledge signal is generated by the controller when execution of the current instruction finished, and the controller 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 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 an existing interrupt.

Multiple Sources of Interrupt

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

Multiple Interrupt Masking

- Masking enables some interrupts and disables others
- Individual disable/enable bit is assigned to each interrupting source.

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 register. It is the programmer’s responsibility to save program status register and conflict registers.
- The delay from the time the pending IRQ is generated to the time the Interrupt Service Routine (ISR) starts to execute is called *interrupt latency*. 等待时间，潜伏期

Interrupt Service Routine

- A section of code to be executed when the corresponding interrupt is responded by CPU.
- Interrupt service routine is a special function, therefore can be constructed with three parts:
  - Prologue:
    - Code mainly for saving conflict registers
  - Body:
    - Code for doing the required task.
  - Epilogue:
    - Code for restoring conflict registers
    - 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.
  - E.g. SWI in ARM.
- AVR does NOT provide a software interrupt instruction.
  - Programmers can use External Interrupts to implement software interrupts.

Reset

- Reset is a special interrupt available in most processors (including AVR).
- It is not maskable.
- Its service function mainly sets the processor system to the initial state (hence called reset interrupt).
  - No need to deal with conflict registers.

AVR Interrupts

- Interrupts in AVR basically can be divided into
  - internal interrupts
  - external interrupts
- Each interrupt has a dedicated interrupt vector
  - To be discussed
- Hardware is used to detect interrupt

AVR Interrupts (cont.)

- To enable an interrupt, two control bits must be set
  - the Global Interrupt Enable bit (I bit) in the Status Register, SREG
    - Using sei instruction
  - 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
  - To be discussed

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
```

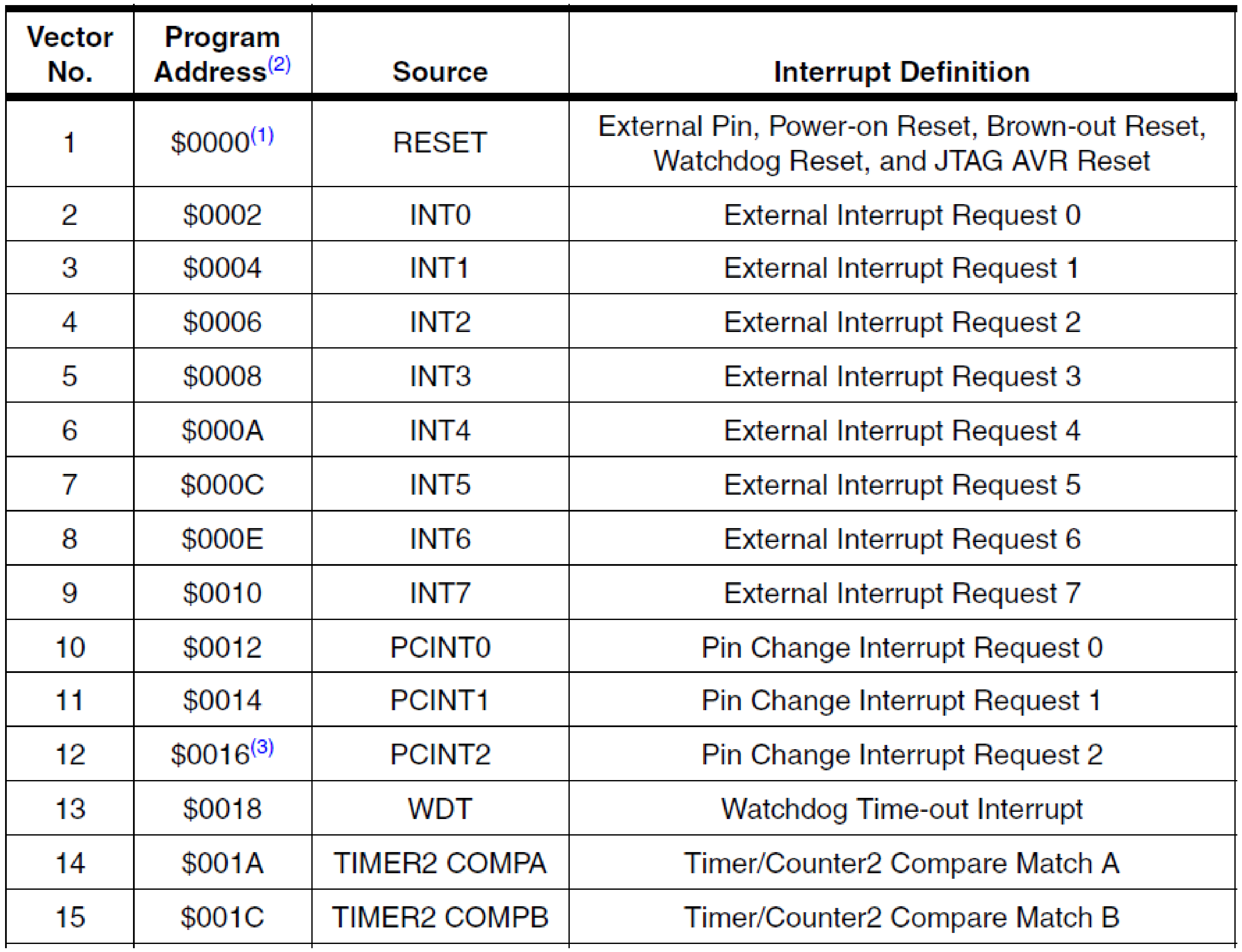
Interrupt Response Time

- Basically 4 clock cycles minimum
  - For saving the Program Counter (2 clock cycles)
  - For jumping to the interrupt routine (2 clock cycles)

Interrupt Vectors

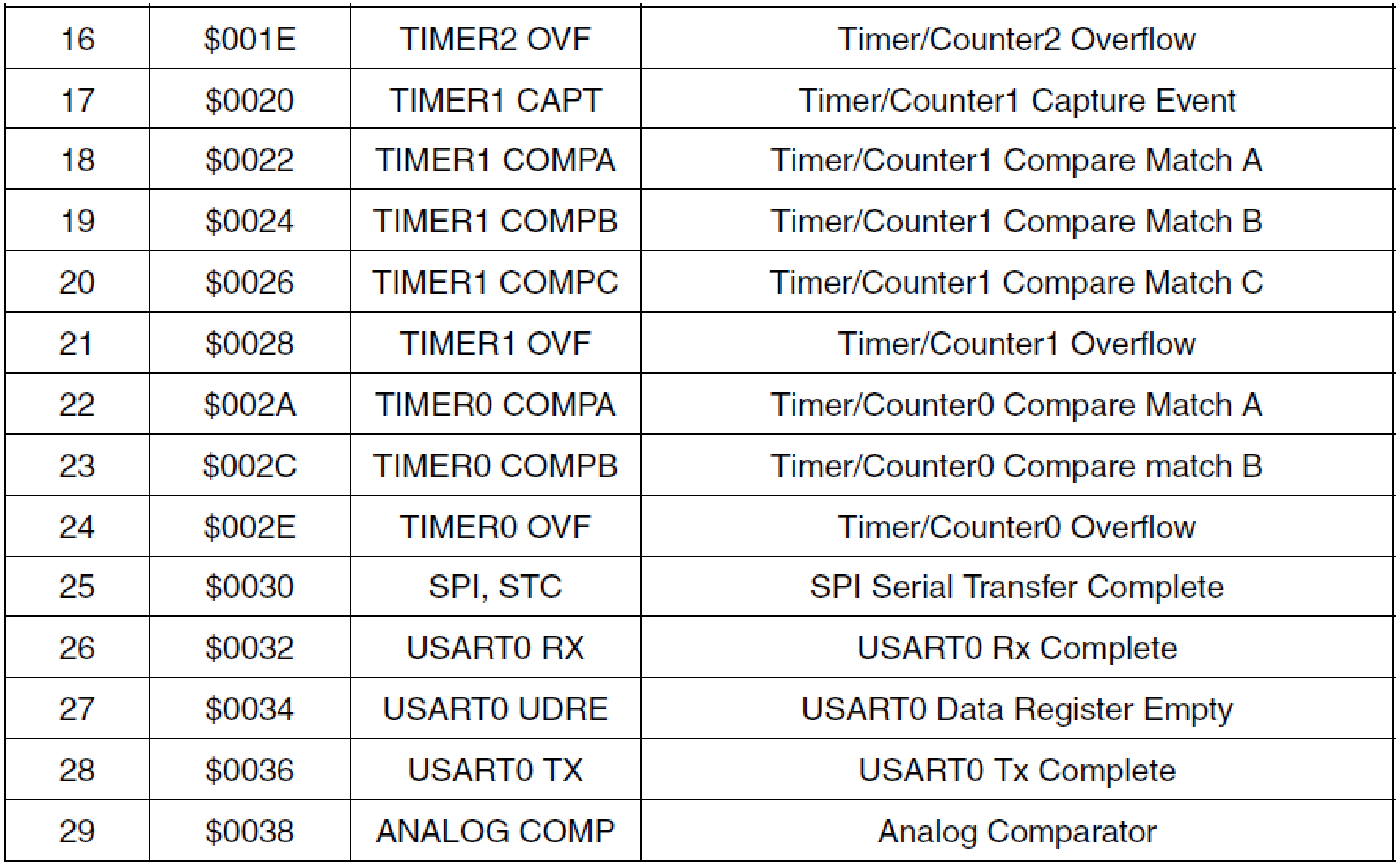
- Each interrupt has a 4-byte (2-word) interrupt vector, containing an instruction to be executed after CPU has accepted the interrupt.
- The lowest address space in the program memory is, by default, defined as the section for Interrupt Vectors.
- The priority of an interrupt is based on the position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

Interrupt Vectors in Mega2560

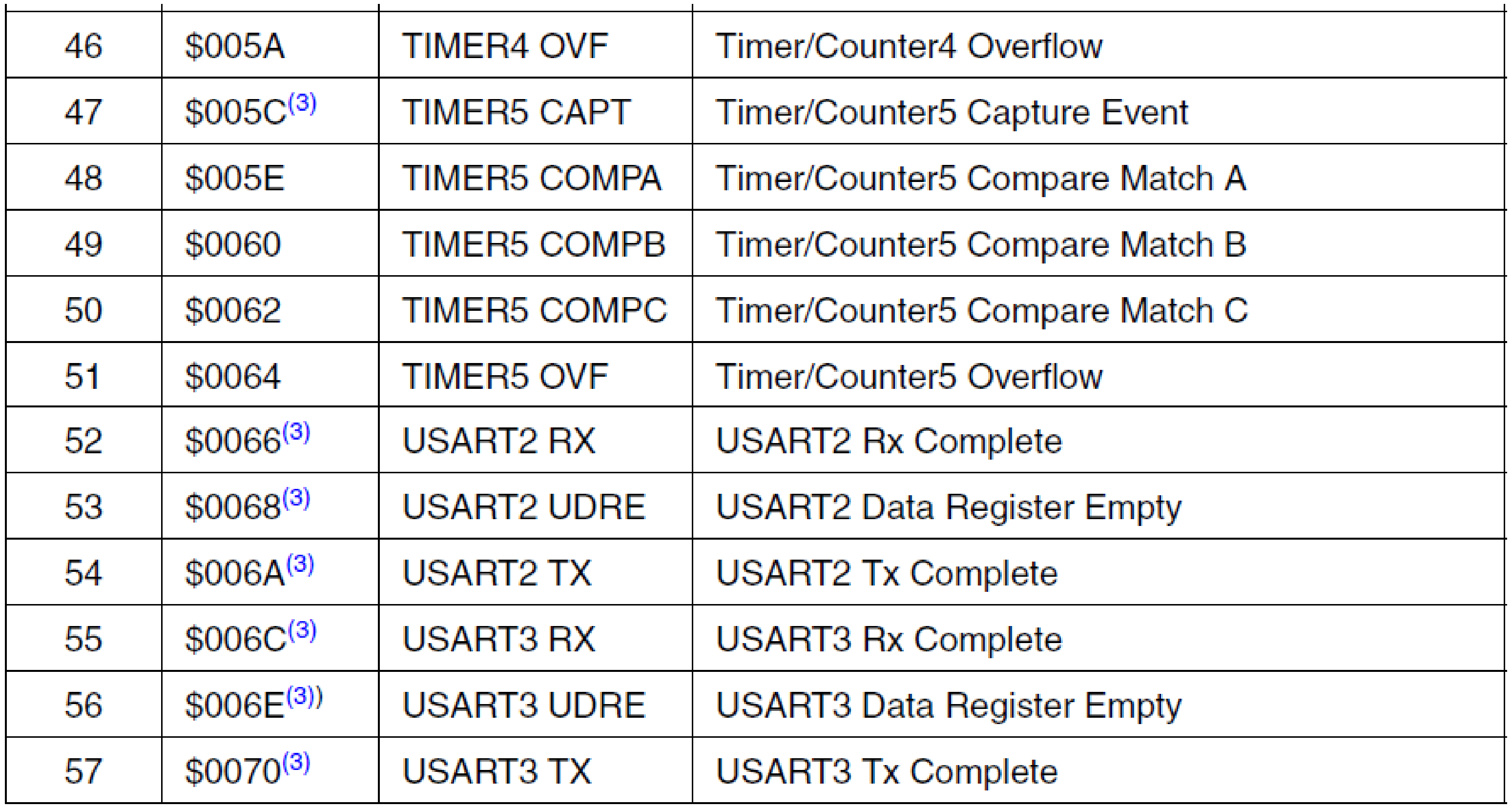


16	$001E	TIMER2 OVF	Timer/Counter2 Overflow
17	$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	$002A	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

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
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

Interrupt Vectors in Mega2560



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 service starts,
  - the Global Interrupt Enable I-bit is cleared (0) and
  - all interrupts are disabled.
    - The user software can set the I-bit to allow nested interrupts
- When the AVR exits from an interrupt,
  - the Global Interrupt Enable I-bit is set (1)
    - Via execution of the *reti* instruction
  - the execution returns to the main program and executes one more instruction before any pending interrupt is served.
    - The Reset interrupt is an exception

Initialization of Interrupt Vector Table (IVT) 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 need to handle this interrupt.

Example of IVT Initialization in Mega2560

```
.include "m2560def.inc"
.cseg
.org 0x0000
; first vector -----
      rjmp RESET                ; Jump to the start of Reset interrupt service routine
                                 ; Relative jump is used if RESET is not far
      nop                        ; to make the vector 4 bytes.
;second vector ----
      jmp IRQ0                  ; Long jump is used assuming IRQ0 is very far away
; third vector -----
      reti                      ; Return to the interrupted point (no handling for this interrupt).
; … some code here
RESET:                          ; The interrupt service routine for RESET starts here.
; … some code here
IRQ0:                           ; The interrupt service routine for IRQ0 starts here.
```

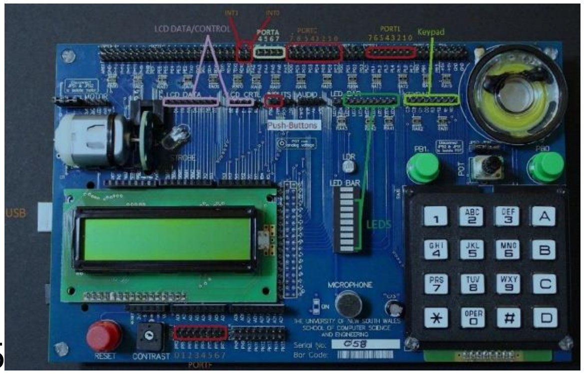
RESET in Mega2560

- 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.
  - Etc.

External Interrupts

触发

- The external interrupts are triggered through the INT7:0 pins.
  - If enabled, the interrupts can be triggered even if the INT7:0 pins are configured as outputs
    - This feature provides a way of generating a software interrupt.
  - Can be triggered by a falling or rising edge or a logic level
    - Specified in External Interrupt Control Register
      - EICRA (for INT3:0)
      - EICRB (for INT7:4)

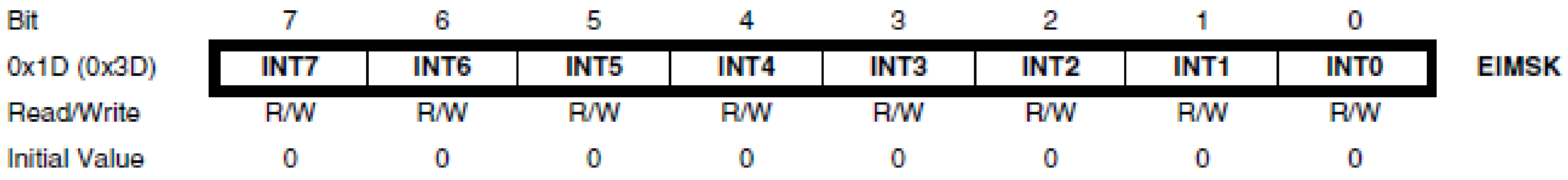


External Interrupts (cont.)

- To enable an external interrupt, two bits must be set
  - I bit in SREG
  - INTx bit in EIMSK
- To generate an external interrupt, the following must be met:
  - The interrupt must be enabled
  - The associated external pin must have a designed signal produced.

EIMSK

- External Interrupt Mask Register
  - A bit is set to enable the related interrupt



Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	EIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

NOTE: All tables in the notes are copied from the ATmega2560 data sheet

EICRA

- External Interrupt Control Register *A*
  - For INT0-3
  - Defines the type of signal that activates the external interrupt
    - on the rising or falling edge or level sensed

Bit	7	6	5	4	3	2	1	0
(0x69)	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

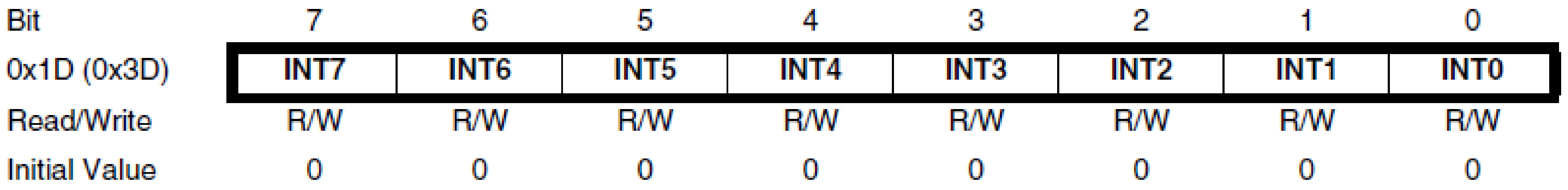
ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any edge of INTn generates asynchronously an interrupt request
1	0	The falling edge of INTn generates asynchronously an interrupt request
1	1	The rising edge of INTn generates asynchronously an interrupt request

ISC01 ISC00:
INT0为低电平时产生中断请求
INT0引脚上任意的逻辑电平变化都将引发中断
INT0的下降沿产生异步中断请求
INT0的上升沿产生异步中断请求

异步的

EIFR

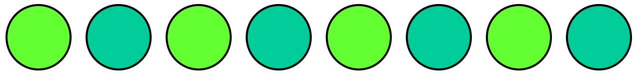
- Interrupt flag register
  – A bit in the register is set when an edge-triggered interrupt is enabled and an event on the related INT pin happens.



Bit	7	6	5	4	3	2	1	0
0x1D (0x3D)	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Example 1

- Design a system, where the state of LEDs toggles under the control of the user, and the number of toggles is counted.



Example 1 (solution)

- Use an external interrupt
  - Connect the external interrupt pin to a push button
    - INT0 to be used. The related pin is Port D Bit 0
  - When the button is pressed, an interrupt is generated
- In the assembly code
  - Set up the interrupt
    - Create the interrupt vector
    - Enable the interrupt
  - Write a service routine for this interrupt
    - Change the display pattern
    - Write the pattern to the port connected to LEDs
    - Increase the toggle count

Code for Example 1

```
.include "m2560def.inc“

.def        temp = r16
.def        output = r17
.def        count = r18                          ; count number of interrupts
.equ        PATTERN = 0b01010101

                                                 ; set up interrupt vectors
            jmp RESET
.org        INT0addr                             ; defined in m2560def.inc
            jmp EXT_INT0

RESET:
            ser temp                             ; set Port C as output
            out DDRC, temp
            out PORTC, temp
            ldi output, PATTERN
```

ISC01 ISC00:
INT0为低电平时产生中断请求
INT0引脚上任意的逻辑电平变化都将引发中断
INT0的下降沿产生异步中断请求
INT0的上升沿产生异步中断请求

Code for Example 1 (cont.)

```
            ldi temp, (2 << ISC00)     ISC01 = 1, ISC00 = 0    ; set INT0 as falling edge triggered interrupt
            sts EICRA, temp            sts: Store Direct to SRAM

        External Interrupt Control Registers – EICRA (INT3:0) and EICRB (INT7:4)

            in temp, EIMSK                                     ; enable INT0        EIMSK – External Interrupt Mask Register
            ori temp, (1<<INT0)        ori: Logical OR Register and Constant
            out EIMSK, temp

            sei                                                ; enable Global Interrupt
            jmp main

EXT_INT0:
            push temp                                          ; save register
            in temp, SREG                                      ; save SREG
            push temp

            com output                                         ; flip the pattern   One's Complement,Rd ← 0xFF – Rd
            out PORTC, output
            inc count

            pop temp                                           ; restore SREG
            out SREG, temp
            pop temp                                           ; restore register
            reti
```

Code for Example 1 (cont.)

```
main:
            clr count
            clr temp
loop:
            inc temp                ; a dummy task in main

            cpi temp, 0x1F          ; the following section in red
            breq reset_temp         ; shows the need to save SREG
            rjmp loop               ; in the interrupt service routine
reset_temp:
            clr temp
            rjmp loop
```

Example 2

- Based on Example 1, implement a software interrupt
  - When the counter that counts LED toggles reaches to the maximum value 0xFF, all LEDs are turned on.

Example 2 (solution)

- Use another external interrupt as software interrupt
  - INT1 is used (Port D bit 1)
  - Software generates the external interrupt request
- In the main program, test if counter=0xFF
  - If yes, write a value (based on the interrupt type chosen) to the pin to invoke the interrupt.

Code for Example 2

```
.include "m2560def.inc"
.include  "my_macros.inc"                              ; macros for oneSecondDelay

.def        temp =r16
.def        output = r17
.def        count = r18
.equ        PATTERN = 0b01010101
.equ        MAX = 0b11111111

                                                       ; set up interrupt vectors
            rjmp RESET
.org        INT0addr
            rjmp EXT_INT0
.org        INT1addr
            jmp EXT_INT1

RESET:
                                                                 ; continued
```

Code for Example 2 (cont.)

```
; continued
            ser temp                                ; set Port C as output
            out DDRC, temp
            ldi output, PATTERN
            out PORTC, temp
            ldi temp, 0b00000010
            out DDRD, temp                          ; set Port D bit 1 as output
            out PORTD, temp


            ldi temp, (2 << ISC00) | (2 << ISC10)   ; set INT0 and INT1 as
            sts EICRA, temp                         ; falling edge sensed interrupts

            in temp, EIMSK                          ; enable INT0 and INT1
            ori temp, (1<<INT0) | (1<<INT1)
            out EIMSK, temp

            sei                                     ; enable Global interrupt
            jmp main
                                                            ; continued
```

Code for Example 2 (cont.)

```
; continued
EXT_INT0:
            push temp                       ; save register
            in temp, SREG                   ; save SREG
            push temp

            com output                      ; flip the pattern
            out PORTC, output
            inc count

            pop temp                        ; restore SREG
            out SREG, temp
            pop temp                        ; restore register
            reti

                                            ; continued
```

Code for Example 2 (cont.)

```
; continued
EXT_INT1:
            push temp
            in temp, SREG
            push temp

            ldi output, MAX
            out PORTC, output
            oneSecondDelay                  ; macro for one second delay
                                            ; stored in “my_macro.inc”

            ldi output, PATTERN             ; set pattern for normal LED display
            sbi PORTD, 1                    ; set bit for INT1
            pop temp
            out SREG, temp
            pop temp
            reti
                                                        ; continued
```

Code for Example 2 (cont.)

```
; continued

                                         ; main - does nothing but increment a counter
main:
            clr count
            clr temp
loop:
            inc temp
            cpi count, 0xFF
            breq max_value               ; if reaches max value
            rjmp loop
max_value:
            cbi PORTD, 1                 ; generate an INT1 request
            clr count                    ; prepare for the next sw interrupt
            rjmp loop
```

Reading Material

- Chapter 10: Interrupts and Real-Time Events. Microcontrollers and Microcomputers by Fredrick M. Cady.
- Mega2560 Data Sheet.
  - System Control and Reset.
  - Interrupts.

Homework

1. Based your code for Task 3 in Lab 2, modify it by using an external interrupt to start and stop the LEDs' display.