Department of Computer Science
National Tsing Hua University

CS 5244: Introduction to Cyber Physical Systems

Unit 7: Interrupts (Ch. 9)

Instructor: Cheng-Hsin Hsu

Acknowledgement: The instructor thanks Profs. Edward A. Lee & Sanjit A. Seshia at UC Berkeley for sharing their course materials

Input Mechanisms in Software

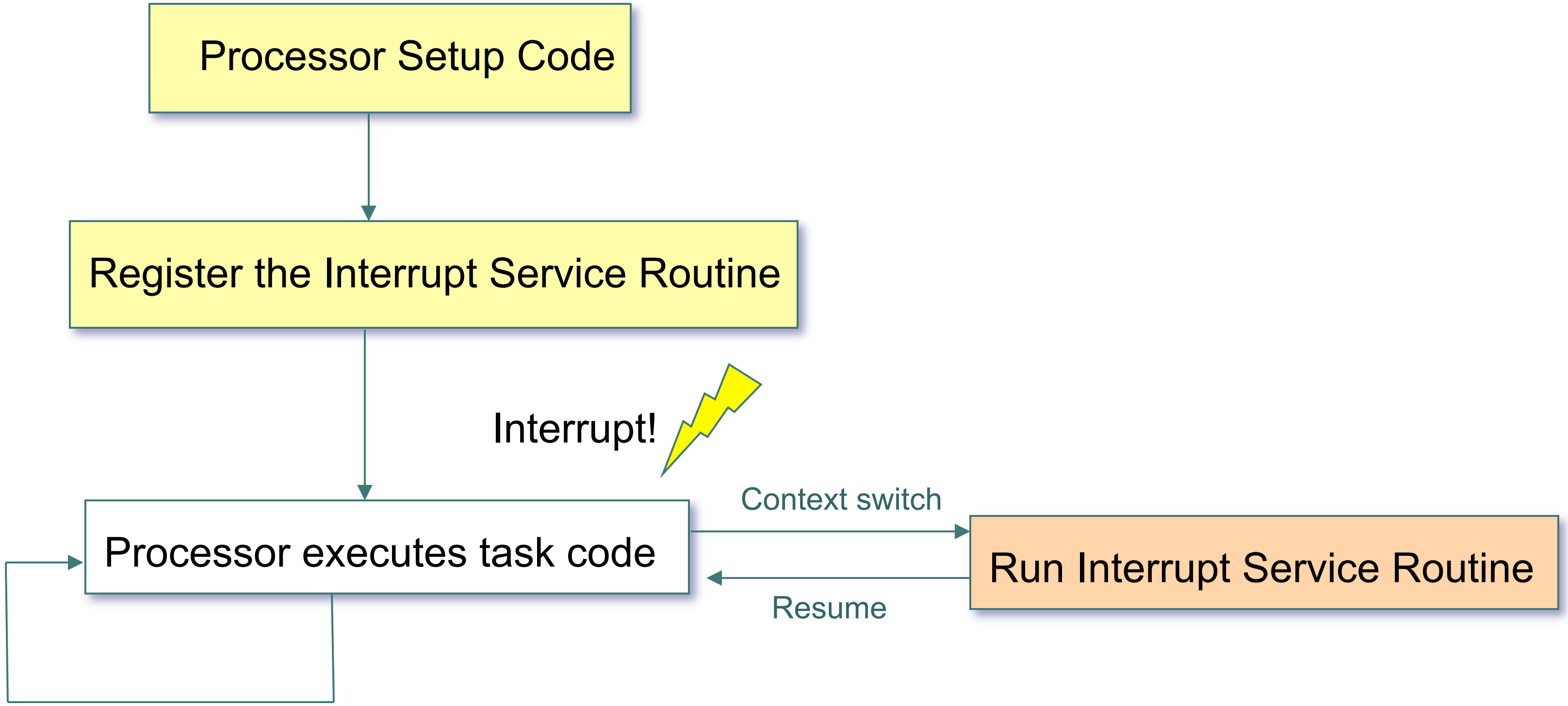
- Polling
  - Main loop checks each I/O device periodically.
  - If input is ready, processor initiates communication.

- Interrupts
  - External hardware alerts the processor that input is ready.
  - Processor suspends what it is doing.
  - Processor invokes an interrupt service routine (ISR).
  - ISR interacts with the application concurrently.

Interrupts

- Interrupt Service Routine

  Short subroutine that handles the interrupt



Processor Setup Code

Register the Interrupt Service Routine

Interrupt!

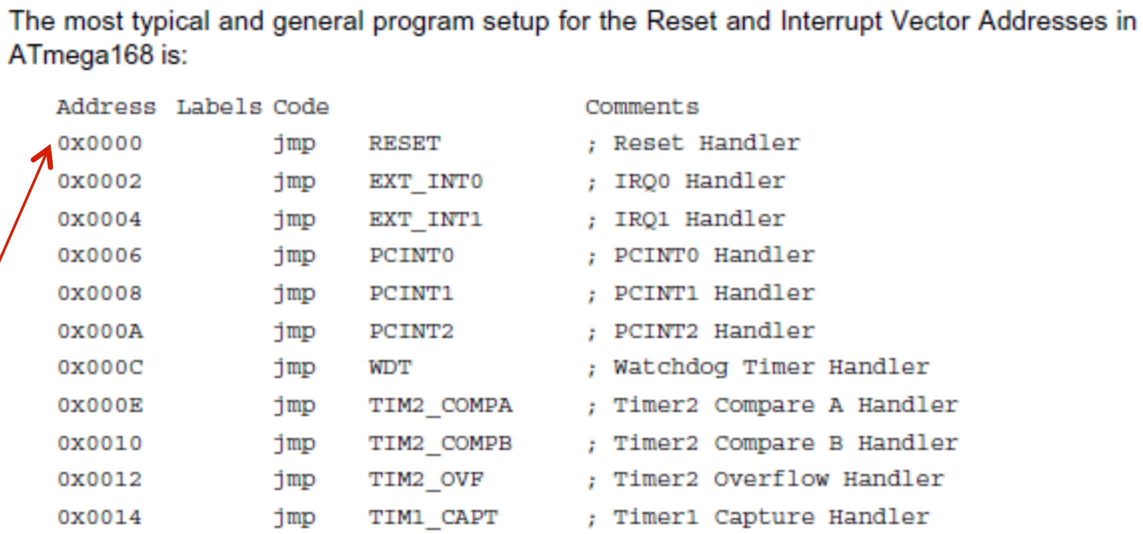
Processor executes task code

Context switch

Resume

Run Interrupt Service Routine

Interrupts: Details



The most typical and general program setup for the Reset and Interrupt Vector Addresses in ATmega168 is:

```
Address  Labels Code                  Comments
0x0000          jmp    RESET          ; Reset Handler
0x0002          jmp    EXT_INT0       ; IRQ0 Handler
0x0004          jmp    EXT_INT1       ; IRQ1 Handler
0x0006          jmp    PCINT0         ; PCINT0 Handler
0x0008          jmp    PCINT1         ; PCINT1 Handler
0x000A          jmp    PCINT2         ; PCINT2 Handler
0x000C          jmp    WDT            ; Watchdog Timer Handler
0x000E          jmp    TIM2_COMPA     ; Timer2 Compare A Handler
0x0010          jmp    TIM2_COMPB     ; Timer2 Compare B Handler
0x0012          jmp    TIM2_OVF       ; Timer2 Overflow Handler
0x0014          jmp    TIM1_CAPT      ; Timer1 Capture Handler
```

Program memory addresses, not data memory addresses.

Source: ATmega168 Reference Manual

Triggers:

- Hardware interrupt: A level change on an interrupt request pin
- Software interrupt: Special instruction or write a memory-mapped register

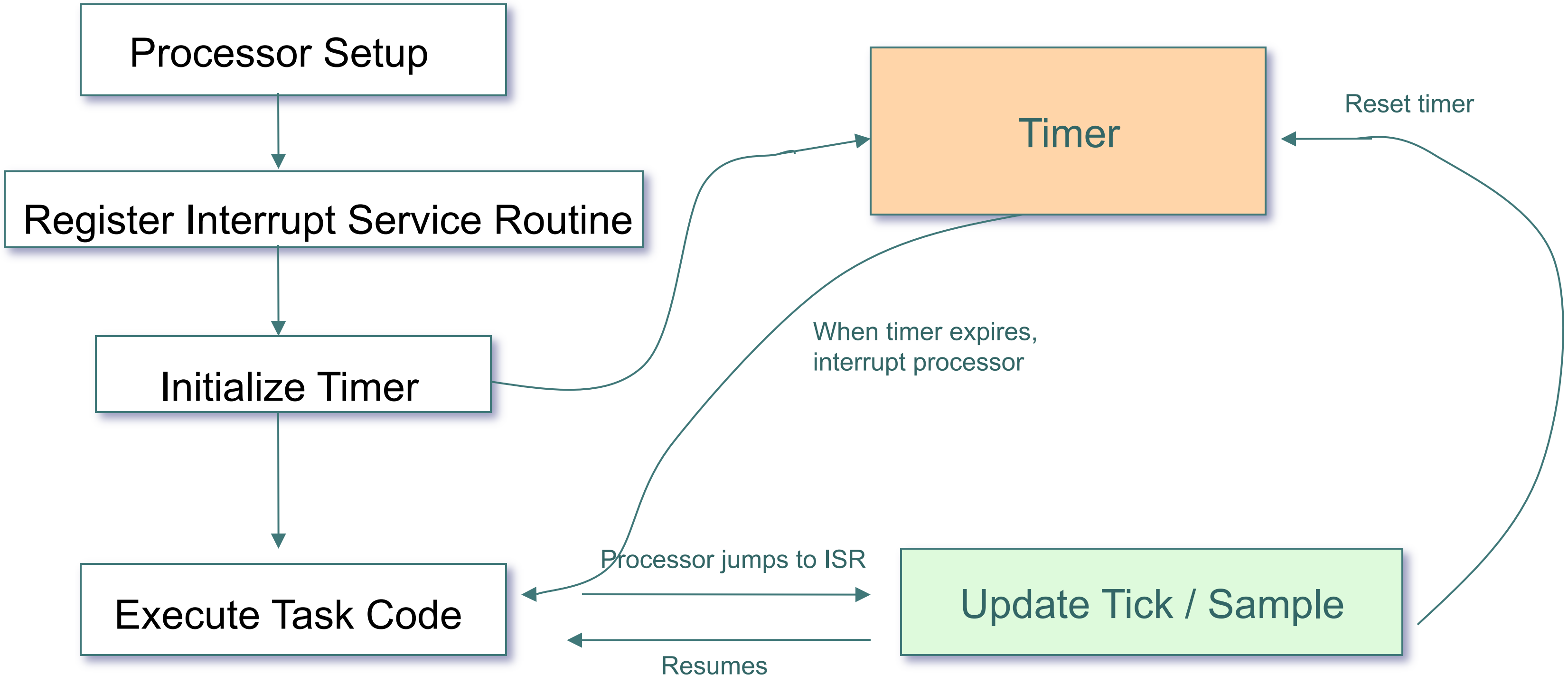
Responses:

- Disable interrupts.
- Push the current program counter onto the stack.
- Execute the instruction at a designated address in the flash memory.

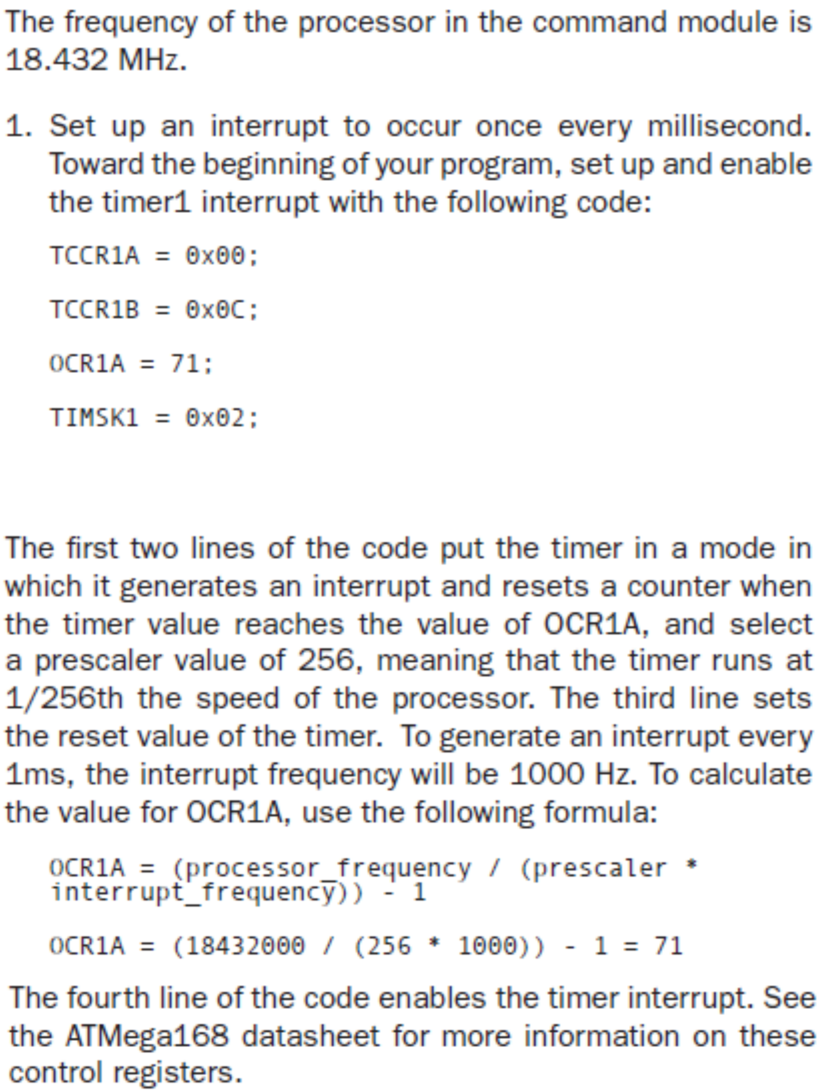
Design of interrupt service routine:

- Save and restore any registers it uses.
- Re-enable interrupts before returning from interrupt.

Timed Interrupt



Example 1: Set up a timer on an ATmega168 to trigger an interrupt every 1ms.



The frequency of the processor in the command module is 18.432 MHz.

1. Set up an interrupt to occur once every millisecond. Toward the beginning of your program, set up and enable the timer1 interrupt with the following code:

```
TCCR1A = 0x00;

TCCR1B = 0x0C;

OCR1A = 71;

TIMSK1 = 0x02;
```

The first two lines of the code put the timer in a mode in which it generates an interrupt and resets a counter when the timer value reaches the value of OCR1A, and select a prescaler value of 256, meaning that the timer runs at 1/256th the speed of the processor. The third line sets the reset value of the timer. To generate an interrupt every 1ms, the interrupt frequency will be 1000 Hz. To calculate the value for OCR1A, use the following formula:

```
OCR1A = (processor_frequency / (prescaler *
interrupt_frequency)) - 1

OCR1A = (18432000 / (256 * 1000)) - 1 = 71
```

The fourth line of the code enables the timer interrupt. See the ATMega168 datasheet for more information on these control registers.

- TCCR: Timer/Counter Control Register
- OCR: output compare register
- TIMSK: Timer Interrupt Mask

The "prescaler" value divides the system clock to drive the timer.

Setting a non-zero bit in the timer interrupt mask causes an interrupt to occur when the timer resets.

Source: iRobot Command Module Reference Manual v6

Setting up the timer interrupt hardware in C

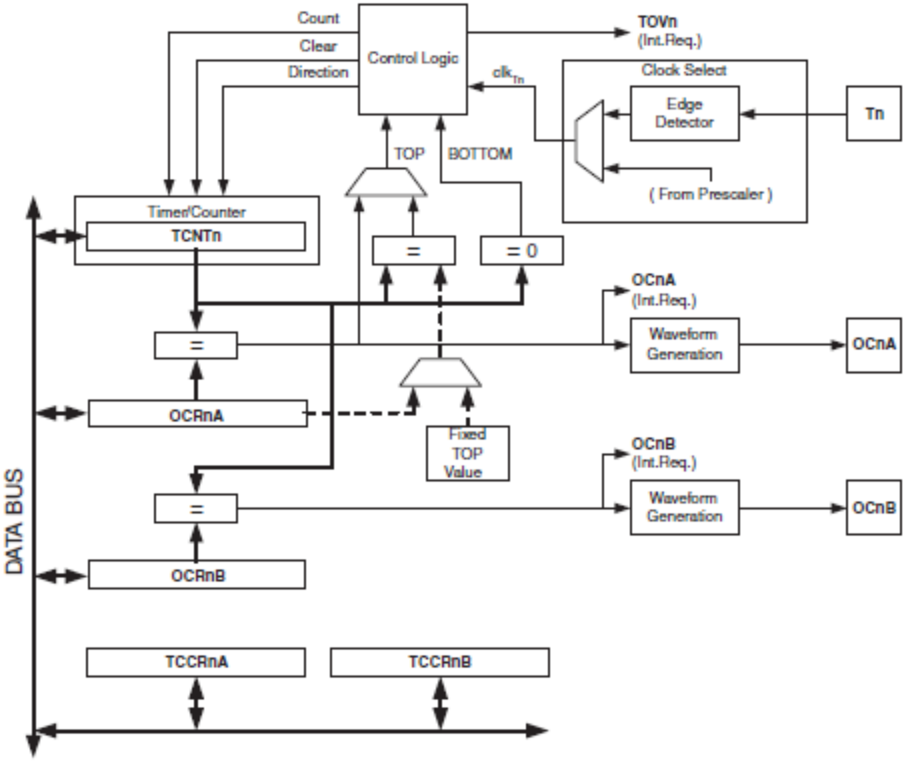
```
#include <avr/io.h>

int main (void) {
  TCCR1A = 0x00;
  TCCR1B = 0x0C;
  OCR1A = 71;
  TIMSK1 = 0x02;
  ...
}
```

memory-mapped register

This code sets the hardware up to trigger an interrupt every 1ms.

Figure 16-1. 8-bit Timer/Counter Block Diagram



Source: ATmega168 Reference Manual

Example 2: Set up a timer on the Luminary to trigger an interrupt every 1ms.

```
// Setup and enable SysTick with interrupt every 1ms
void initTimer(void) {
  SysTickPeriodSet(SysCtlClockGet() / 1000);
  SysTickEnable();
  SysTickIntEnable();
}

// Disable SysTick
void disableTimer(void) {
  SysTickIntDisable();
  SysTickDisable();
}
```

Number of cycles per sec.

Start SysTick counter

Enable SysTick timer interrupt

Source: Stellaris Peripheral Driver Library User's Guide

Example: Do something for 2 seconds then stop

```
volatile uint timer_count = 0;
void ISR(void) {
  if(timer_count != 0) {
    timer_count--;
  }
}
int main(void) {
  // initialization code
  SysTickIntRegister(&ISR);
  ... // other init
  timer_count = 2000;
  while(timer_count != 0) {
    ... code to run for 2 seconds
  }
}
```

static variable: declared outside main() puts them in statically allocated memory (not on the stack)

volatile: C keyword to tell the compiler that this variable may change at any time, not (entirely) under the control of this program.

Interrupt service routine

Registering the ISR to be invoked on every SysTick interrupt

Concurrency

```
volatile uint timer_count = 0;
void ISR(void) {
  if(timer_count != 0) {
    timer_count--;
  }
}
int main(void) {
  // initialization code
  SysTickIntRegister(&ISR);
  ... // other init
  timer_count = 2000;
  while(timer_count != 0) {
    ... code to run for 2 seconds
  }
}
```

concurrent code: logically runs at the same time. In this case, between any two **machine instructions** in main() an interrupt can occur and the upper code can execute.

Reasoning about concurrent code

```
volatile uint timer_count = 0;
void ISR(void) {
  if(timer_count != 0) {
    timer_count--;
  }
}
int main(void) {
  // initialization code
  SysTickIntRegister(&ISR);
  ... // other init
  timer_count = 2000;
  while(timer_count != 0) {
    ... code to run for 2 seconds
  }
}
```

what if the interrupt occurs right here?

Reasoning about concurrent code

```
volatile uint timer_count = 0;
void ISR(void) {
  if(timer_count != 0) {
    timer_count--;
  }
}
int main(void) {
  // initialization code
  SysTickIntRegister(&ISR);
  ... // other init
  timer_count = 2000;
  while(timer_count != 0) {
    ... code to run for 2 seconds
  }
}
```

what if the interrupt occurs right here?

Reasoning about concurrent code

```
volatile uint timer_count = 0;
void ISR(void) {
  if(timer_count != 0) {
    timer_count--;
  }
}
int main(void) {
  // initialization code
  SysTickIntRegister(&ISR);
  ... // other init
  timer_count = 2000;
  while(timer_count != 0) {
    ... code to run for 2 seconds
  }
}
```

what if the interrupt occurs right here?

Reasoning about concurrent code

```
volatile uint timer_count = 0;
void ISR(void) {
  if(timer_count != 0) {
    timer_count--;
  }
}
int main(void) {
  // initialization code
  SysTickIntRegister(&ISR);
  ... // other init
  timer_count = 2000;
  while(timer_count != 0) {
    ... code to run for 2 seconds
  }
}
```

what if the interrupt occurs twice during the execution of this code?

Reasoning about concurrent code

```
volatile uint timer_count = 0;
void ISR(void) {
  if(timer_count != 0) {
    timer_count--;
  }
}
int main(void) {
  // initialization code
  SysTickIntRegister(&ISR);
  ... // other init
  timer_count = 2000;
  while(timer_count != 0) {
    ... code to run for 2 seconds
  }
}
```

can an interrupt occur here? If it can, what happens?

Reasoning about concurrent code

```
     volatile uint timerCount = 0;
     void ISR(void) {
        … disable interrupts
D →     if(timerCount != 0) {
E →        timerCount--;
        }
        … enable interrupts
     }
     int main(void) {
        // initialization code
        SysTickIntRegister(&ISR);
        … // other init
        timerCount = 2000;
A →     while(timerCount != 0) {
B →      … code to run for 2 seconds
        }
C →  … whatever comes next
     }
```

A key question: Assuming interrupt occurs infinitely often, is position C always reached?

Reasoning about concurrent code

```
volatile uint timer_count = 0;
void ISR(void) {
  if(timer_count != 0) {
    timer_count--;
  }
}
int main(void) {
  // initialization code
  SysTickIntRegister(&ISR);
  ... // other init
  timer_count = 2000;
  while(timer_count != 0) {
    ... code to run for 2 seconds
  }
}
```

What is it about this code that makes it work?

A question:

What’s the difference between

Concurrency

and

Parallelism

Concurrency and Parallelism

A program is said to be **concurrent** if different parts of the program *conceptually execute simultaneously*.

A program is said to be **parallel** if different parts of the program *physically execute simultaneously* on distinct hardware.

A parallel program is concurrent, but a concurrent program need not be parallel.

Concurrency in Computing

- Interrupt Handling
  - Reacting to external events (interrupts)
  - Exception handling (software interrupts)
- Processes
  - Creating the illusion of simultaneously running different programs (multitasking)
- Threads
  - How is a thread different from a process?
- Multiple processors (multi-cores)

. . .

Summary

Interrupts introduce a great deal of nondeterminism into a computation. Very careful reasoning about the design is necessary.