การอินเตอร์รัพท์ (Interrupt) เป็นการขัดจังหวะการทำงานของโปรแกรมปกติโดยจะสั่งให้ไมโครคอนโทรลเลอร์หยุดการทำงานของโปรแกรมปกติที่กำลังทำอยู่ชั่วขณะ แล้วให้กระโดดไปทำงานที่สำคัญเร่งด่วนบางอย่างตามที่กำหนดไว้ซึ่งต้องการการตอบสนองอย่างเร่งด่วน เช่นกรณีที่เกิดการผิดพลาดจากการเขียนอ่านหน่วยความจำ และเมื่อไมโครคอนโทรลเลอร์ทำงานตอบสนองต่อการขัดจังหวะเรียบร้อยแล้ว ก็จะกลับไปทำงานปกติของโปรแกรมในตำแหน่งเดิมเพื่อประมวลผลในงานเก่าที่ทำค้างไว้ การอินเตอร์รัพท์จะแบ่งตามชนิดของการเกิดได้ดังนี้

- Hardware Interrupt เป็นการอินเตอร์รัพท์ที่เกิดจากอุปกรณ์ภายนอก แล้วส่งผ่านค่านั้นเข้ามาทำให้เกิดจากการเปลี่ยนสถานะลอจิกของพอร์ตใดพอร์ตหนึ่ง เช่นการอินเตอร์รัพท์ที่มาจากเซนเซอร์ตรวจจับไฟไหม้ (Fire Alarm) จะทำให้ไมโครคอนโทรลเลอร์สั่งงานให้ระบบสัญญาณเตือนอัคคีภัยทำงาน
- Software Interrupt จะเกิดขึ้นตามคำสั่งจากซอฟต์แวร์ ตัวอย่างเช่น การอินเตอร์รัพท์ที่เกิดตามเวลาที่ตั้งไว้ (Timer Interrupt) เพื่อให้ได้การวัดสัญญาณจากเซนเซอร์ในช่วงเวลาที่ห่างเท่าๆกัน เช่นทุกๆครั้งเมื่อครบเวลา 1 วินาที

การควบคุมการเกิดอินเตอร์รัพท์ เป็นการให้ไมโครคอนโทรลเลอร์ตอบสนองต่อการอินเตอร์รัพท์หรือไม่ แบ่งได้ดังนี้

- Enable Interrupt คือการควบคุมให้ไมโครคอนโทรลเลอร์ตอบสนองต่อการอินเตอร์รัพท์ได้ตามปกติ โดยอนุญาตให้ไมโครคอนโทรลเลอร์ไปทำตามคำสั่งที่กำหนดไว้ในอินเตอร์รัพท์
- Disable Interrupt คือการควบคุมให้ไมโครคอนโทรลเลอร์ไม่ตอบสนองกับการอินเตอร์รัพท์ เมื่อเกิดการอินเตอร์รัพท์ขึ้นไมโครคอนโทรลเลอร์จะปล่อยผ่านการอินเตอร์รัพท์ไม่ทำตามคำสั่งนั้น

การใช้งานอินเตอร์รัพท์ของ Arduino ในการทดลองส่วนแรกนี้จะเป็นการใช้งานอินเตอร์รัพท์จากภายนอก (External Interrupt) โดยจะต้องมีการกำหนดฟังก์ชั่นที่จะถูกเรียกใช้เมื่อเกิดการอินเตอร์รัพท์ แล้วจึงใช้คำสั่งที่จะกำหนดว่าให้เกิดอินเตอร์รัพท์ที่พอร์ตใด สำหรับบอร์ด Arduino จะมีพอร์ตที่สามารถใช้อินเตอร์รัพท์ได้ในแต่ละรุ่นจะแตกต่างกันดังนี้

Board	int.0	int.1	int.2	int.3	int.4	int.5
Uno, Nano, Mini, other 328-based	2	3				
Mega2560	2	3	21	20	19	18
Leonardo, Micro, other 32u4-based	3	2	0	1	7	

ฟังชั่นที่กำหนดการใช้อินเตอร์รัพท์คือ attachInterrupt(interrupt , ISR , mode) เป็นคำสั่งที่กำหนดและสร้างอินเตอร์รัพท์โดยมีรูปแบบการใช้งานของคำสั่งดังนี้

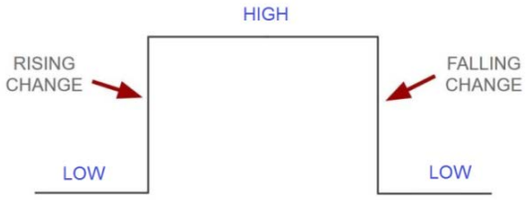
interrupt คือหมายเลขของการอินเตอร์รัพท์ แต่เนื่องจากตำแหน่งขาของหมายเลขอินเตอร์รัพท์ในบอร์ด Arduino ในแต่ละรุ่นจะไม่เหมือนกัน โดยปกติจึงควรใช้คำสั่ง digitalPinToInterrupt(pin) เพื่อเปลี่ยนจากตำแหน่งขาดิจิตอลที่กำหนดให้ทำหน้าที่รับสัญญาณเพื่อให้เกิดการอินเตอร์รัพท์ ไปเป็นหมายเลขของการอินเตอร์รัพท์ ดังนั้นจะได้เป็นคำสั่ง

attachInterrupt(digitalPinToInterrupt(pin) , ISR , mode)

ISR (Interrupt Service Routines) คือชื่อของฟังก์ชั่นที่จะถูกเรียกใช้ เมื่อเกิดการอินเตอร์รัพท์ สามารถตั้งชื่อได้ตามต้องการ ฟังก์ชั่นนี้จะต้องไม่มีพารามิเตอร์และไม่ควรส่งคืนค่าใดๆ การส่งผ่านข้อมูลกับ Main Program จึงต้องใช้ Global Variable โดยทั่วไปโปรแกรมภายในควรจะสั้นและให้ทำงานได้เร็ว ไม่สามารถใช้คำสั่ง delay() ได้

mode คือรูปแบบของสัญญาณทริกเพื่อทำให้เกิดการอินเตอร์รัพท์ มีทั้งหมด 4 รูปแบบดังนี้

- LOW จะเกิดการอินเตอร์รัพท์เมื่อสัญญาณที่ขามีสถานะเป็น LOW
- CHANGE จะเกิดการอินเตอร์รัพท์เมื่อมีการเปลี่ยนสถานะที่ขาจาก LOW ไปเป็น HIGH หรือจาก HIGH ไปเป็น LOW
- RISING จะเกิดอินเตอร์รัพท์เมื่อสถานะของสัญญาณที่ขาเปลี่ยนจาก LOW ไปเป็น HIGH
- FALLING จะเกิดอินเตอร์รัพท์เมื่อสถานะของสัญญาณที่ขาเปลี่ยนจาก HIGH เป็น LOW



1. การทดลองให้ต่อวงจรเพื่อสั่งงานให้ Arduino ส่งข้อมูลต่างๆ ไปออกที่ LED จำนวน 5 ดวง โดย LED ที่เป็นไดโอดเปล่งแสงทุกตัว จะต้องต่อขา Cathode เข้ากับขั้วไฟลบหรือลงกราวด์ และให้ขา Anode ในแต่ละตัวต่อกับขาตัวความต้านทาน 220 Ω แล้วให้นำขาตัวความต้านทานที่เหลือแต่ละตัวต่อเข้ากับขา D8, D9, D10, D11, D12 ของบอร์ด Arduino ตามลำดับ

2. ให้ทำการเขียนโปรแกรมที่ทำหน้าที่สั่งงานให้ LED แต่ละดวงที่อยู่บนบอร์ดกระพริบทุก 50 ms ตั้งแต่ดวงที่ 1 ถึง 5 เรียงตามลำดับกันไป แล้วหลังจากนั้นให้กลับมาวนซ้ำเหมือนเดิม โดยทุกรอบที่กลับมาวนซ้ำนั้นให้ลดเวลาของแต่ละดวงลง 1 ms จนกว่ารอบสุดท้ายจะเท่ากับ 1 ms ดังนี้

```
int led8 = 8;
int led9 = 9;
int led10 = 10;
int led11 = 11;
int led12 = 12;
int i,;

void setup()
{
  pinMode(led8, OUTPUT);
  pinMode(led9, OUTPUT);
  pinMode(led10, OUTPUT);
  pinMode(led11, OUTPUT);
  pinMode(led12, OUTPUT);
}

void loop()
{
   for (i=50; i>0; i--)
   {
      digitalWrite(led8, HIGH);
      delay(i);
      digitalWrite(led8, LOW);
      digitalWrite(led9, HIGH);
      delay(i);
      digitalWrite(led9, LOW);
      digitalWrite(led10, HIGH);
      delay(i);
      digitalWrite(led10, LOW);
      digitalWrite(led11, HIGH);
      delay(i);
      digitalWrite(led11, LOW);
      digitalWrite(led12, HIGH);
      delay(i);
      digitalWrite(led12, LOW);
   }
}
```

3. จากโปรแกรมในข้อ 2 จะเห็นได้ว่าเมื่อลดเวลาการทำงานของ LED ลงในรอบท้ายๆจะทำงานเร็วมากทำให้ดูไม่ทัน ดังนั้นให้แก้ไขโปรแกรมใหม่ โดยกำหนดเงื่อนไขไว้ว่าในแต่ละรอบที่ลดเวลาการทำงานของ LED ลงดวงละ 1 ms นั้น ให้เพิ่มจำนวนรอบของการทำงานทั้ง 5 ดวงขึ้นอีก 1 รอบด้วย โดยให้ใช้คำสั่ง for

4. ให้ต่อตัวความต้านทาน 10 K Ω อนุกรมกับสวิทช์ แล้วนำปลายขาข้างที่เป็นตัวต้านทานต่อกับ Vcc ของบอร์ด Arduino และปลายขาอีกข้างที่เป็นสวิทช์ต่อลงกราวด์ แล้วให้เอาขา D2 ต่อเข้ากับจุดต่อร่วมระหว่างตัวความต้านทานกับสวิทช์ โดยจะต้องมีการกำหนดค่าขาเพิ่มลงในฟังก์ชั่น setup คือ

```
pinMode(BUTTON, INPUT);
```

5. จากวงจรในข้อที่ 4 ให้เขียนโปรแกรมทดสอบการกดสวิทช์โดยแก้ไขเพิ่มจากโปรแกรมในข้อที่ 3 กำหนดเงื่อนไขไว้ว่า ถ้ามีการไม่กดสวิทช์ให้ LED ทั้งหมดตั้งแต่ดวงที่ 1 ถึง 5 ทำงานตามลำดับ แต่ถ้ากดสวิทช์ให้ LED ทั้งหมดหยุดทำงาน โดยให้ใช้คำสั่ง if

6. จากข้อที่ 5 ให้แก้ไขโปรแกรมใหม่ โดยกำหนดเงื่อนไขไว้ว่า ถ้ามีการไม่กดสวิทช์ให้ LED ตั้งแต่ดวงที่ 1 ถึง 5 ทำงานตามลำดับเหมือนในข้อที่ 5 แต่ถ้ากดสวิทช์ให้ LED ที่อยู่บนบอร์ดขา D13 ทำงานติดค้างไว้จนกว่าจะปล่อยสวิทช์ โดยให้เพิ่มคำสั่ง else

7. ขา D2 ของบอร์ด Arduino สามารถที่จะกำหนด Internal pull-up ได้ ดังนั้นจากวงจรในข้อที่ 6 ให้แก้ไขวงจร โดยเอาตัวความต้านทาน 10 K Ω ที่ต่อกับสวิทช์ออก แล้วให้แก้ไขโปรแกรมให้วงจรทำงานเหมือนเดิมโดยให้ใช้คำสั่งในการกำหนดขา Input ใหม่เป็น

```
pinMode(BUTTON, INPUT_PULLUP);
```

8. จากวงจรในข้อที่ 7 ให้สังเกตว่า เมื่อกดสวิทช์วงจรเปลี่ยนการทำงานมาทำ LED ที่ขา D13 ทันทีหรือไม่เพราะอะไร แล้วถ้าจะให้ทำทันทีโดยที่ LED ทั้งหมดตั้งแต่ดวงที่ 1 ถึง 5 ยังทำงานเรียงลำดับตามปกติจะสามารถทำได้หรือไม่อย่างไร

LED ที่ขา D13 จะไม่เปลี่ยนทันที ต้องมีการเพิ่ม interrupt เข้าไปด้วย เพื่อให้ LED ที่ D13 สามารถแทรกการทำงานของ LED 1-5

………

………

………

9. จากข้อที่ 7 ให้แก้ไขโปรแกรม โดยให้ตัดในส่วนของคำสั่ง if …… else ออกไป แล้วเปลี่ยนมาใช้คำสั่งที่ใช้ในการ Interrupt ซึ่งสามารถเขียนเป็นฟังก์ชั่นได้ดังนี้

```
void switch1()
{
  if(digitalRead(BUTTON) == LOW)
    digitalWrite(LED,HIGH);
  else
    digitalWrite(LED,LOW);
}
```

โดยที่ในส่วนของฟังก์ชั่น setup ให้เพิ่มคำสั่งกำหนดการใช้อินเตอร์รัพท์ด้วย คือ

```
attachInterrupt(digitalPinToInterrupt(BUTTON) ,switch1, CHANGE);
```

10. จากข้อที่ 9 ให้แก้ไขโปรแกรม โดยกำหนดเงื่อนไขของสวิทช์ไว้ว่า เมื่อมีการกดสวิทช์ขณะที่ LED ตั้งแต่ดวงที่ 1 ถึง 5 กำลังทำงานเรียงลำดับตามปกติ ให้ทำการจำเอาไว้ จนกว่าการทำงานจะวนมาถึงรอบสุดท้ายที่ 1 ms แล้วจึงเปลี่ยนให้มาทำงานที่ LED ขา D13 ให้สว่าง 1 วินาทีทุกครั้งที่มีการกดสวิทช์ หลังจากนั้นจึงกลับทำงานไปวนซ้ำตั้งแต่ดวงที่ 1 ถึง 5 เหมือนเดิมตามปกติ โดยให้ใช้คำสั่งเพิ่มเป็นคำสั่งที่ใช้ในการ Interrupt ด้วยการใช้คำสั่ง if ร่วมกับ Global Variable

การเขียนโปรแกรมโดยทั่วไปถ้าต้องการจะตั้งค่าช่วงเวลา ปกติสามารถที่จะใช้ฟังก์ชัน delay () เป็นการหยุดโปรแกรมชั่วคราวในช่วงเวลาหนึ่ง แต่ก็จะเป็นการสิ้นเปลืองเวลาโดยเปล่าประโยชน์ โดยเฉพาะอย่างยิ่งหากต้องการดำเนินการอย่างอื่นในระหว่างช่วงเวลานั้น ซึ่งเราสามารถแก้ปัญหาเหล่านี้ได้โดยการนำเอา Timer และ Interrupt เข้ามาใช้งาน

สำหรับบอร์ด Arduino จะใช้ความถี่สัญญาณนาฬิกาของไมโครคอนโทรลเลอร์คือ 16MHz ซึ่งจะได้คาบเวลาประมาณ 63ns โดยจะมีตัวจับเวลา 3 ตัว ดังนี้

- Timer0 จะเป็น timer ขนาด 8 bit ถูกใช้โดย Arduino ในฟังก์ชัน delay() , millis() และ micros()
- Timer1 จะเป็น timer ขนาด 16 bit จะถูกนำมาใช้เป็น Servo() library
- Timer2 จะเป็น timer ขนาด 8 bit จะถูกนำมาใช้เป็น Tone() library

สำหรับบอร์ดรุ่น Mega จะมีเพิ่มเป็น Timers 3, 4, 5 ซึ่งสามารถนำมาใช้แทนกันได้

Timer Interrupt มีประโยชน์ให้ไมโครคอนโทรเลอร์ทำงานตามระยะเวลาที่ตั้งไว้ ซึ่งสามารถปรับตั้งคาบเวลาในการทำงานได้ การทดลองเราจะใช้ Timer 1 ในการให้เกิด Interrupt โดยจะต้องระวังไม่ใช้ Servo Library ร่วมด้วย เพราะอาจเกิดปัญหาได้ ซึ่งตัวนับที่ใช้นี้จะเป็นตัวนับขนาด 16 bit ที่ชื่อว่า Timer1 Counter ค่าของตัวนับนี้จะเพิ่มขึ้นอีก 1 ค่าในทุกๆรอบการทำงานของไมโครคอนโทรเลอร์ และเพิ่มค่าขึ้นจนกระทั่งถึงค่าสูงสุดของตัวแปรที่ตั้งไว้ ข้อมูลที่ตั้งค่าไว้นี้จะอยู่ในรีจิสเตอร์ขนาด 16 bit จึงสามารถเก็บค่าได้ไม่เกิน 65535 แล้วก็จะทำให้เกิด Overflow ขึ้น ซึ่งจะเป็นการนำไปสู่การกระตุ้นทำให้ให้เกิด Timer Interrupt หลังจากนั้นตัวนับจะถูกตั้งค่าให้กลับไปเป็น 0 ใหม่ รอบของการนับและการเกิด Timer Interrupt นี้จะวนซ้ำไปเรื่อยๆ เราสามารถที่จะกำหนดตั้งค่าของ Timer1 Counter เพื่อให้นับค่าใดก็ได้ ซึ่งวิธีการนี้จะสามารถปรับคาบเวลาของ Timer Interrupt ได้ค่อนข้างละเอียด

โดยที่ Timer ขนาด 8 บิตจะมีคาบเวลาสูงสุดเท่ากับ 256 / 16,000,000 = 16µs และ Timer ขนาด 16 บิตจะมีคาบเวลาสูงสุดเท่ากับ 65536 / 16,000,000 = 4ms ซึ่งจะเห็นได้ว่าความถี่ของสัญญาณนาฬิกาที่ได้นั้นยังมีค่าเร็วเกินไปสำหรับการการอินเตอร์รัพท์ตามช่วงเวลาที่กำหนดไว้ หากเราต้องการอินเตอร์รัพท์ทุก ๆ วินาที ตัวจับเวลาเหล่านี้จะต้องมีการเพิ่ม Prescaler ที่เป็นค่าคงที่มีค่าตั้งแต่ 8, 64, 256 และ 1024 ซึ่งจะทำให้ลดความเร็วของสัญญาณนาฬิกาลงได้

ค่า Prescale จะใช้เป็นตัวคูณรอบการทำงานของไมโครคอนโทรลเลอร์ เพื่อใช้เป็นตัวปรับการทำงานให้ช้าลง เช่น ถ้า Prescale มีค่าเป็น 2 จะทำให้ตัว Timer1 counter มีค่าเพิ่มขึ้นเป็นทุก 2 รอบการทำงานของไมโครคอนโทรลเลอร์ ดังนั้น Prescale จึงเป็นตัวคูณเพื่อยืดเวลาการเกิด Timer Interrupt ออกไป จากหลักการข้างต้นคาบเวลาของการเกิด Timer Interrupt นั้นสามารถคำนวณได้ตามสมการดังนี้

$$\text{Timer Interrupt} = \frac{\text{Prescaler} \times \text{จำนวนครั้งการนับจนเกิด Overflow}}{\text{Clock}} \quad [S]$$

Clear Timer on Compare หรือ CTC Mode

CTC Mode จะเป็น Timer Interrupts การทำงานจะเกิดการทริกเมื่อตัวจับเวลานับถึงค่าที่กำหนดไว้ล่วงหน้า ค่านี้จะถูกเก็บไว้ใน Compare Match Register ในเวลาเดียวกันตัวจับเวลาจะถูกเคลียร์และตั้งค่าให้กลับไปเป็นศูนย์เพื่อจะเริ่มการนับใหม่ โดยการทำงานจะต้องใช้ Timer Registers ดังต่อไปนี้

Timer/Counter Control Register (TCCRx) เป็นรีจิสเตอร์ที่ทำหน้าที่ควบคุมตัวนับและจับเวลา มีสองส่วนคือ TCCR1A และ TCCR1B ขึ้นอยู่กับว่าบิตใดถูกตั้งค่าบนรีจิสเตอร์นี้ ซึ่งสามารถกำหนดค่าการใช้งานในแต่ละบิตได้ดังนี้

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	-	-	WGM11	WGM10	**TCCR1A**
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	-	MGM13	MGM12	CS12	CS11	CS10	**TCCR1B**
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

โดยสามารถตั้งค่า Timer 1 ให้ทำงานได้แตกต่างกันดังนี้

- โหมดเปรียบเทียบเอาต์พุต (COM) เช่น ถ้า COM1A = 3 เมื่อ TCNT1 มากกว่า OCR1A จะทำให้ขา OC1A มีค่าเป็น HIGH
- โหมดการสร้างคลื่น (WGM) ที่ใช้ในการเลือกการสร้างสัญญาณต่างๆ คือ Non-PWM , PWM , Fast-PWM
- โหมดเลือกสัญญาณนาฬิกา (CS) ที่ใช้ในการกำหนดค่าของตัวแปร Prescaler

Timer/Counter Register (TCNTx) เป็นรีจิสเตอร์ที่ไว้เก็บค่าการการนับ ณ เวลาปัจจุบัน ค่านี้จะเพิ่มขึ้นโดยอัตโนมัติตามจำนวนการนับของสัญญาณนาฬิกาที่เข้ามา

Output Compare Register (OCRx) เป็นรีจิสเตอร์ที่ไว้เก็บค่าเปรียบเทียบ (Compare Match Register) กับค่าผลการนับที่เกิดขึ้นของรีจิสเตอร์ TCNTx

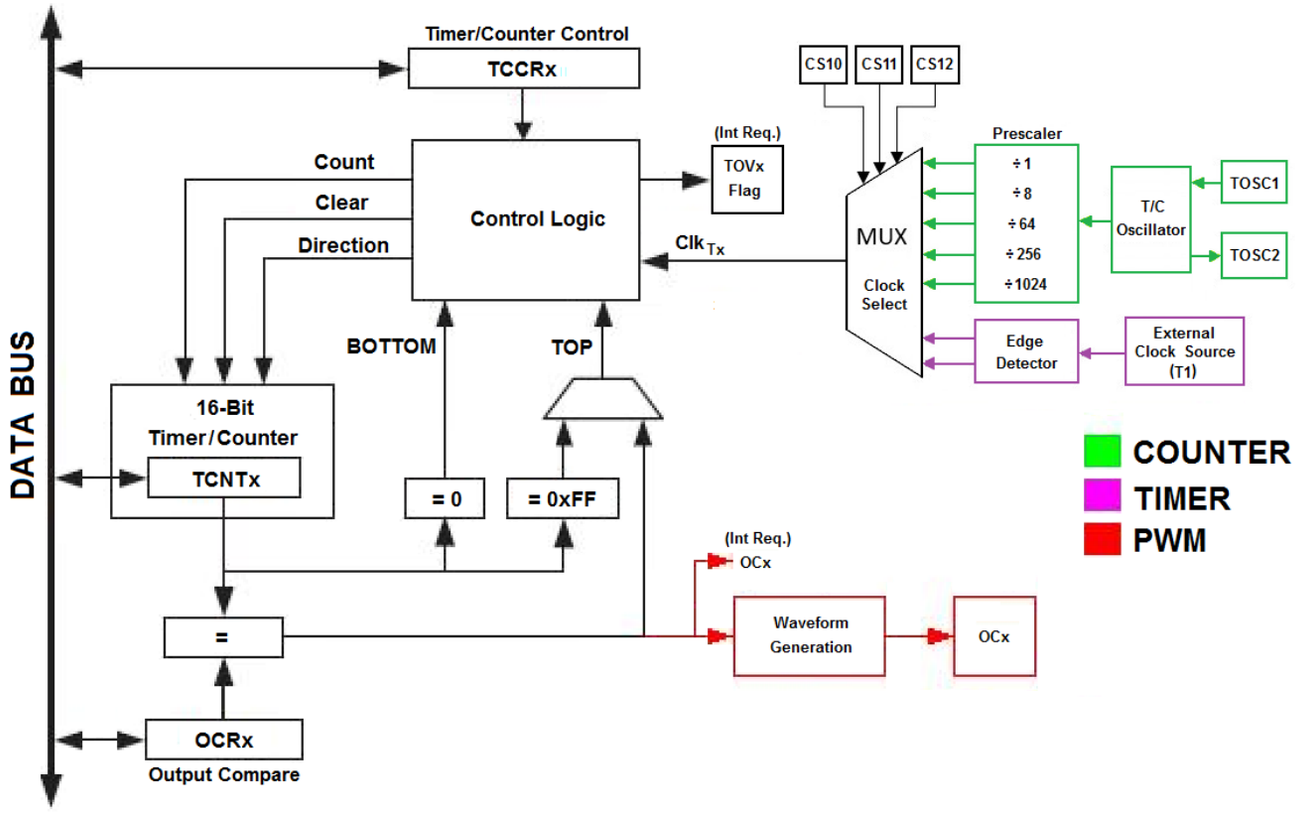
Input Capture Register (ICRx) รีจิสเตอร์เก็บค่าอินพุต (สำหรับตัวจับเวลา 16 บิตเท่านั้น)

Timer/Counter Interrupt Mask Register (TIMSKx) ทำหน้าที่ Enable/Disable ให้กับ Timer Interrupt

Timer/Counter Interrupt Flag Register (TIFRx) เป็น Flag Register ของ Timer Interrupt

ถ้าต้องการอินเตอร์รัพท์ทุกๆวินาทีจะมีค่าใน (Compare Match Register) เท่ากับ clock speed / (prescaler * interrupt frequency) -1 ที่ต้องลบ 1 เนื่องจากค่าในรีจิสเตอร์มีค่าเริ่มต้นเป็นศูนย์ สำหรับไมโครคอนโทรลเลอร์ ATmega328 จะมีค่า Prescaler สูงสุดคือค่า 1024 ค่าของรีจิสเตอร์จะได้เป็น 16MHz / (1024 * 1Hz) -1 = 15624 เมื่อกำหนดให้เกิดการอินเตอร์รัพท์ที่ความถี่ 1 Hz หรือทุก 1 วินาที สำหรับ Timer 0 และ Timer 2 จะไม่สามารถนำมาใช้งานได้ เพราะเป็นขนาด 8 บิต เนื่องจากค่าสูงสุดในการจัดเก็บมีค่าเพียง 255 เท่านั้น ดังนั้นจึงมีเพียง Timer 1 ที่สามารถนำมาใช้งานได้ เพราะสามารถเก็บค่าสูงสุดได้ถึง 65535 Prescaler จะถูกตั้งค่าจากรีจิสเตอร์ TCCRxB โดยการเลือกบิตของสัญญาณนาฬิกาที่แตกต่างกันตามค่าของ CS10, CS11, CS12 สามารถกำหนดค่าใน Timer1 ได้ดังนี้

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clkI/O/(no prescaling)
0	1	0	clkI/O/8 (from prescaler)
0	1	1	clkI/O/64 (from prescaler)
1	0	0	clkI/O/256 (from prescaler)
1	0	1	clkI/O/1024 (from prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge
1	1	1	External clock source on T1 pin. Clock on rising edge



การทดลองจะเป็นการสร้าง Timer Interrupt ซึ่งในส่วนของฟังก์ชั่นsetup จะต้องมีการกำหนดค่าต่างๆในการใช้งานอินเตอร์รัพท์ได้ดังนี้

- เริ่มจากการตั้งค่ารีจิสเตอร์ 2 ตัวคือ TCCRxA และ TCCRxB ให้เป็น 0

- ตั้งค่าเริ่มต้นของ TCNTx ที่จะใช้เป็นตัวนับค่า ให้เป็น 0

- ตั้งค่าตัวเปรียบเทียบ OCRxA ที่จะถูกใช้ในการ Compare กับตัวนับค่า TCNTx ซึ่งสามารถคำนวณได้จากสมการ

$$\text{compare match register} = \frac{16000000\ [\text{Hz}]}{\text{Interrupt frequency [Hz] x Prescaler}} - 1$$

- เปิดโหมด CTC ให้ทำงาน โดยใช้รีจิสเตอร์ TCCRxB

- ตั้งค่า Prescaler โดยใช้รีจิสเตอร์ TCCRxB ซึ่งจะต้องเลือกสัญญาณนาฬิกาที่ได้จากค่าบิตของ CS10, CS11, CS12

ตัวอย่างการตั้งค่า Prescaler ของโปรแกรม เช่น ถ้าต้องการตั้งค่าให้ Prescaler เป็น 256 จะได้

```
TCCR1B |= (1 << CS12);
```

หรือ ถ้าต้องการให้ Prescaler เป็น 1024 จะได้

```
TCCR1B |= (1 << CS12) | (1 << CS10);
```

- ให้ทำการ Enable เพื่อให้ไมโครคอนโทรลเลอร์ตอบสนองต่อ Timer Interrupt ได้ โดยใช้รีจิสเตอร์ TIMSKx

รายละเอียดการคำนวณหาค่าต่างๆ เพิ่มเติมให้เปิดดูได้จากไฟล์ Timer-Interrupts Calculator

ตัวอย่างการตั้งค่า Timer1 interrupt เมื่อกำหนดให้ทำการอินเตอร์รัพท์ทุก 1 วินาที และมีกำหนดค่า Prescaler เท่ากับ 256

```
void setup()
 {
   noInterrupts();                          // disable all interrupts
   // Clear Timer/Counter Control Register for Interrupt 1, bytes A and B (TCCR1?)
   TCCR1A = 0;                              // Clear TCCR1A/B registers
   TCCR1B = 0;
   TCNT1  = 0;                              // Initialize counter value to 0  (16-bit counter register)
   // set compare match register for TIMER1: CLOCKFREQUENCY / frequency / prescaler - 1
   OCR1A = 62499;                           // 16MHz/(1Hz*256) - 1 (must be <65536)
   // Timer/Counter Control Register for Interrupt 1 on register B
   TCCR1B |= (1 << WGM12);                  // Mode 4, turn on CTC mode
   // Clock Select Bit, Set CS12, CS11 and CS10 bits
   TCCR1B |= (1 << CS12);                    // Set CS12 bit for 256 prescaler
   TIMSK1 |= (1 << OCIE1A);    // enable timer compare interrupt , The value in OCR1A is used for compare
   interrupts();                            // enable all interrupts
 }
```

11. ให้ทำการเขียนโปรแกรมตั้งค่าเริ่มต้นของ Timer1 interrupt เพื่อกำหนดให้ทำการอินเตอร์รัพท์ทุก 1 วินาที โดยมีค่า Prescaler เท่ากับ 1024 ลงในส่วนของฟังก์ชั่น setup

```
   noInterrupts();                          // disable all interrupts
   // Clear Timer/Counter Control Register for Interrupt 1, bytes A and B (TCCR1?)
   TCCR1A = 0;                              // Clear TCCR1A/B registers
   TCCR1B = 0;
   TCNT1  = 0;                              // Initialize counter value to 0  (16-bit counter register)
   // set compare match register for TIMER1: CLOCKFREQUENCY / frequency / prescaler - 1
   OCR1A = 15624;                           // 16MHz/(1Hz*1024) – 1  (must be <65536)
   // Timer/Counter Control Register for Interrupt 1 on register B
   TCCR1B |= (1 << WGM12);                  // Mode 4, turn on CTC mode
   // Clock Select Bit, Set CS12, CS11 and CS10 bits
   TCCR1B |= (1 << CS12) | (1 << CS10);     // Set CS10 and CS12 bits for 1024 prescaler
   TIMSK1 |= (1 << OCIE1A);    // enable timer compare interrupt , The value in OCR1A is used for compare
   interrupts();                            // enable all interrupts
```

12. ให้อธิบายความแตกต่างของโปรแกรมในข้อ 11 กับโปรแกรมตามตัวอย่างด้านบน และโปรแกรมทั้งสองนี้ใช้งานแทนกันได้หรือไม่เพราะเหตุใด พร้อมทั้งให้ยกตัวอย่างการคำนวณประกอบคำอธิบาย

แตกต่างกัน ภายในส่วนของ Program สามารถใช้ prescale ต่างกัน จาก 256 → 1024
ซึ่งสามารถใช้แทนกันได้ เพราะได้ช่วงเวลาการใช้เท่ากัน

สูตรการคำนวณ $\frac{16 \times 10^6}{\text{prescale} \times Hz} - 1$ เราต้องการ 1 Hz (เรามี 1 Hz เท่ากันจึงแทนค่าได้)

จะได้ $\frac{16 \times 10^6}{256 \times 1} = 62500 - 1$; $\frac{16 \times 10^6}{1024 \times 1} = 15625 - 1$

$= 62499$; $= 15624$

หลังจากได้กำหนดค่า Timer Interrupt ในฟังก์ชั่น setup แล้วเมื่อครบตามเวลาที่กำหนดไว้จะเกิดการอินเตอร์รัพท์ไปที่ฟังก์ชั่น ISR (TIMERx_COMPA_vect) ซึ่งภายในของฟังก์ชั่นนี้จะเป็นโปรแกรมกำหนดสิ่งที่จะให้ Arduino ทำเมื่อเกิดการเรียกใช้อินเตอร์รัพท์ ฟังก์ชั่นนี้จะไม่มีพารามิเตอร์ การส่งผ่านข้อมูลกับ Main Program จึงต้องใช้ตัวแปรแบบ Global โปรแกรมภายจะต้องสั้นและทำงานได้รวดเร็ว โดยระยะเวลาการทำงานของฟังก์ชั่นนี้ทั้งหมดจะต้องมีช่วงของคาบเวลาน้อยกว่าช่วงเวลาของ Timer Interrupt ที่ตั้งไว้ให้มากที่สุด

13. จากโปรแกรม Timer1 interrupt ในข้อ 11 ที่เป็นส่วนที่กำหนดให้ทำการอินเตอร์รัพท์ทุก 1 วินาทีนั้น ให้ทำการแก้ไขโปรแกรมโดยเพิ่มส่วนที่ทำหน้าที่เป็นฟังก์ชั่น ISR (TIMERx_COMPA_vect) โดยที่โปรแกรมภายในจะคำนวณหาค่าให้กับตัวแปร วินาที, นาทีและชั่วโมง ซึ่งจะต้องกำหนดตัวแปรเหล่านี้เป็น Global Variable ด้วย

```
ISR(TIMER1_COMPA_vect)
{
  sec++;
  if (sec >= 60)
  {
    minutes++;
    sec = 0;
  }
  if (minutes >= 60)
  {
    hours++;
    minutes = 0;
  }
}
```

14. ให้อธิบายการทำงานของโปรแกรมในข้อ 13 พร้อมทั้งให้ยกตัวอย่างประกอบคำอธิบาย

ก็คือ การนับเวลาของตัว Arduino คือ ให้ วินาทีเพิ่มขึ้นเรื่อยๆ แล้วถ้าครบ 60 วิ => 1 นาที แล้วให้ วินาทีรีเซ็ตเป็น 0 ใหม่ จากนั้นถ้าครบ 60 นาที => 1 hr. แล้วให้ นาที รีเซ็ตเป็น 0 ใหม่ เพื่อรับวนรอบต่อไป

15. ให้ต่อวงจรเพื่อสั่งงานให้ Arduino ส่งข้อมูลไปออกที่ LED จำนวน 2 ดวง โดยให้ LED ทั้งสองดวงจะต้องต่อขา Cathode เข้ากับขั้วไฟลบหรือลงกราวด์ และให้ขา Anode ในแต่ละตัวต่อกับขาตัวความต้านทาน 220 Ω แล้วให้นำขาตัวความต้านทานที่เหลือแต่ละตัวต่อเข้ากับขา D8 และ D9 ของบอร์ด Arduino ตามลำดับ

16. ให้แก้ไขโปรแกรมในข้อ 13 ในส่วนของการตั้งค่าเริ่มต้น Timer interrupt และฟังก์ชั่น ISR ที่จะมีการเรียกใช้เมื่อเกิด Timer Interrupt กำหนดเงื่อนไขไว้ว่าให้ LED ที่ขา D8 กระพริบทุก 1 วินาที โดยที่การกระพริบนั้นให้ไฟติดค้างเป็นเวลา 0.5 วินาที และให้ LED ที่ขา D9 กระพริบทุก 1 นาที โดยที่การกระพริบนั้นให้ไฟติดค้างเป็นเวลา 1 วินาที

17. ให้แก้ไขโปรแกรมในข้อ 13 ทำเป็นเครื่องจับเวลา โดยมีสวิทช์ตัวที่ 1 ทำหน้าที่เป็น Start-Stop และมีสวิทช์ตัวที่ 2 ทำหน้าที่เป็น Clear แล้วให้นำค่าของตัวแปร ชั่วโมง, นาที และวินาที ไปใช้ในฟังก์ชั่นการแสดงผลเป็นตัวเลขบน Dot Matrix LCD ผ่านทาง I2C

18. จากข้อ 17 ให้นำไปประยุกต์ใช้งานร่วมกับ Stepping Motor ที่มีการทำงานร่วมกับ I2C โดยต้องกำหนด Address ของ I2C ทั้งสองตัวให้ต่างกันด้วย