รายงาน Assignment

หัวข้อ วงจรเกมเป่ายิ้งฉุบ

วิชา 240-228 DIGITAL LOGIC AND MICROCONTROLLER

จัดทำโดย

นาย ณัฐวุฒิ สมใจนึก

รหัสนักศึกษา 6610110481 Section 01

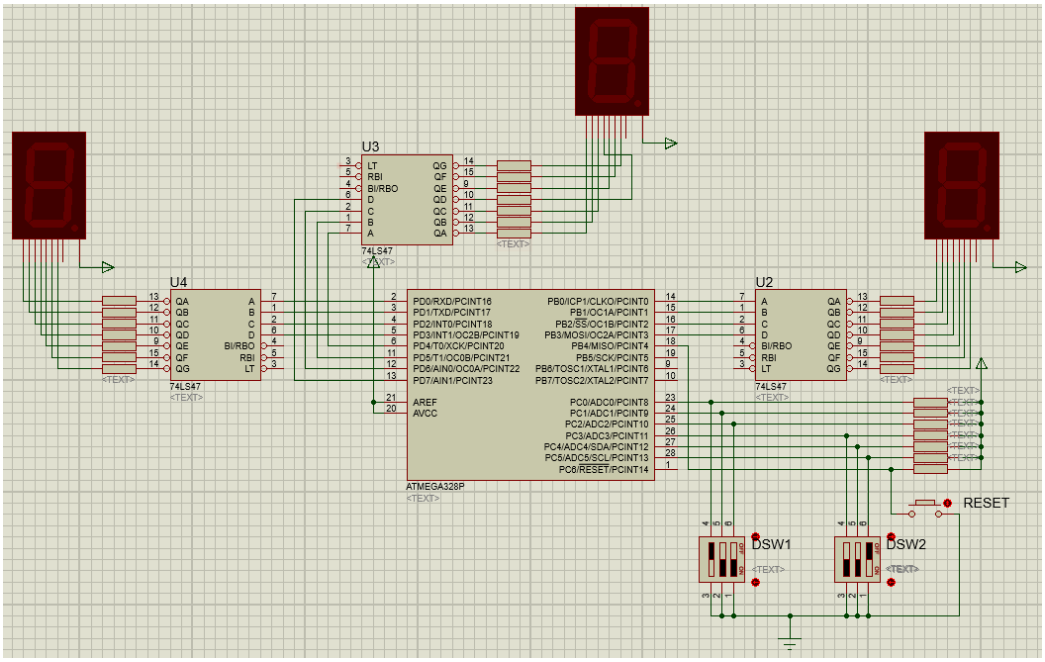
เสนอ

รองศาสตราจารย์ ดร. ปัญญยศ ไชยกาฬ

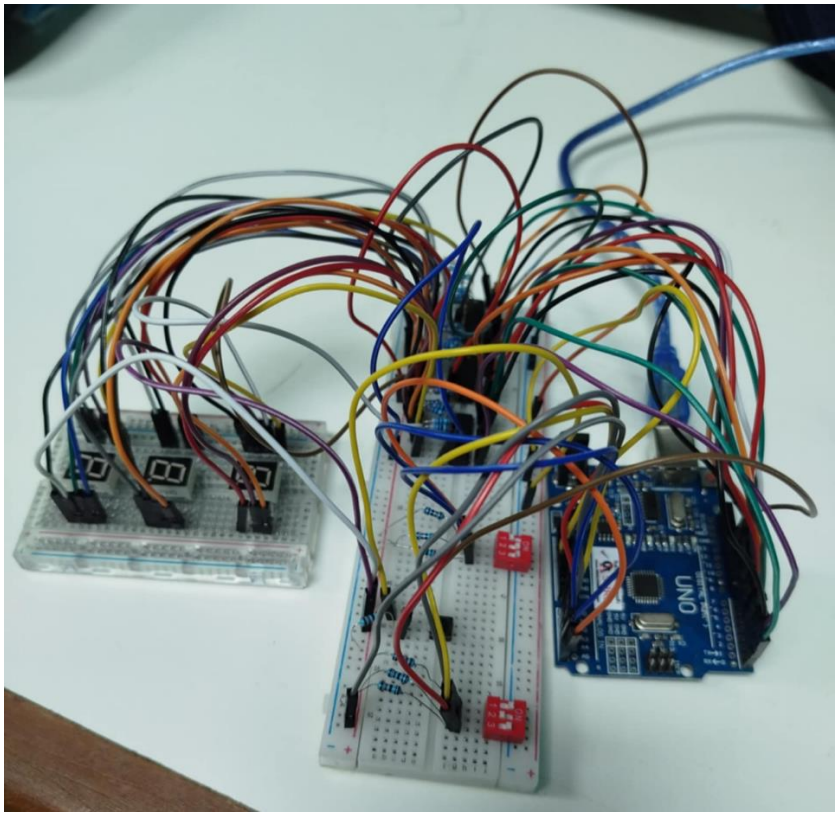
คณะวิศวกรรมศาสตร์ สาขาวิชาวิศวกรรมคอมพิวเตอร์

มหาวิทยาลัยสงขลานครินทร์

Schematic diagram ของวงจร



ภาพการต่อวงจรจริง



อธิบายการทำงานของวงจร

1. การตั้งค่าเริ่มต้น (Setup)

- PORTC เป็นอินพุตทั้งหมด ใช้สำหรับการรับค่าจาก DIP Switch (DIP SW1 = PC0-2, DIP SW2 = PC3-5)

- PORTD เป็นเอาต์พุตทั้งหมด ใช้แสดงคะแนนผู้เล่น 1 บน 7-segment

- PORTB บางส่วน (PB0-PB3) เป็นเอาต์พุต ใช้แสดงผลคะแนนผู้เล่น2 และบางส่วน (PB4-PB7) เป็นอินพุต ใช้ PB4 เป็นพินรับสัญญาณ Interrupt

- ใช้ Pin Change Interrupt ที่ PB4 เพื่อรีเซ็ตคะแนนทั้งสองฝ่ายเมื่อมีการกดปุ่ม และเปิดการรับ interrupt เป็นแบบ global

- ล้างค่าคะแนนของผู้เล่นทั้งสอง (score1, score2) และตัวแปร temp เป็น 0

2. ลูปหลัก (Loop)

- อ่านค่าตัวเลือกจาก DIP switch (ReadDIPSwitch)

หลักการทำงาน: อ่านค่าจาก พอร์ตซี แล้วแยกตัวเลือกของผู้เล่น 1 (PC0-PC2) และผู้เล่น 2 (PC3-PC5) โดยใช้การกรองบิต และเลื่อนบิตของผู้เล่น2 3ครั้งเพื่อให้ง่ายต่อการจัดการต่อ

- เช็คความถูกต้องของตัวเลือกที่ผู้เล่นเลือกโดยใช้ค่าจากการอ่าน DIP switch (ValidateInput)

หลักการทำงาน: ตรวจสอบว่าผู้เล่นเลือกตัวเลือกที่ถูกต้องหรือไม่ เช่น ต้องเลือกเพียงหนึ่งตัวเลือกจาก Rock, Paper, Scissors และห้ามไม่เลือกเลย หากไม่ถูกต้องตามเงื่อนไข จะกลับไปเริ่มลูปใหม่ ผลลัพธ์ที่เห็นคือ เวลาที่นับจะติดอยู่ที่ 0 ที่จะนับต่อเมื่อมีการใส่ค่าที่ถูกต้อง

- เปรียบเทียบตัวเลือกของผู้เล่นเพื่อหาผู้ชนะ (CompareChoices)

หลักการทำงาน: กำหนดให้ ค้อน = 1, กรรไกร = 2, กระดาษ = 4 ถ้าเสมอ (ตัวเลือกมีค่าเท่ากัน) จะกลับไปเริ่มลูปใหม่โดยไม่เพิ่มคะแนน

ตัวเลือกผู้เล่นคนที่ 1	ตัวเลือกผู้เล่นคนที่ 1	ผลการประลอง
ค้อน	กรรไกร	ผู้เล่น1 ชนะ
กรรไกร	กระดาษ	ผู้เล่น1 ชนะ
กระดาษ	ค้อน	ผู้เล่น1 ชนะ
ค้อน	กระดาษ	ผู้เล่น2 ชนะ
กรรไกร	ค้อน	ผู้เล่น2 ชนะ
กระดาษ	กรรไกร	ผู้เล่น2 ชนะ

เมื่อได้ผู้ชนะวงจรจะอัปเดตคะแนนผู้ชนะเพิ่ม 1 คะแนน (เก็บใน score1 หรือ score2) และถ้าคะแนนถึง 8 จะรีเซ็ตคะแนนทั้งหมดเป็น 0

- แสดงคะแนนของผู้เล่นทั้งสองผ่าน 7-segment (Display)

หลักการทำงาน: นำคะแนนของผู้เล่นทั้งสองแปลงเป็นเลขฐาน2 จำนวน4bit ส่งออกทางPORTD สำหรับผู้เล่น 1 และPORTB สำหรับผู้เล่น2 เพื่อให้ 7-segment decoder ทำการแปลงเลขฐาน2เป็น รูปแบบที่เหมาะสมสำหรับการแสดงบน 7-segment display

- หน่วงเวลาเป็นเวลา3 วินาทีและแสดงเวลา (Delay3Sec):

หลักการทำงาน: กำหนดค่าให้ในregister 29,30,31 เป็น 240,250,250 เพื่อลูปทำงานเป็นการหน่วงเวลาประมาณ 3 วินาที และกำหนดเงื่อนไขให้ 29 เป็นลูปชั้นนอก 30 เป็นลูปชั้นกลาง 31 เป็นลูปชั้นใน วงจรจะลูปลดค่า ลูปชั้นในจนหมดและจะไปลดค่าของลูปชั้นต่อไปลง 1 ค่า (250*250*240) รอบการทำงาน 1 วินาทีจะเป็นการลูป จนค่าr29 ลดไปประมาณ 60 ครั้ง การแสดงค่าจะตรวจสอบ r29 หาก r29 เท่ากับ 240 จะแสดงเลข 3 ซึ่งผ่านการแปลงเป็นเลขฐาน2 เพื่อส่งให้7-segment decoder และแสดงบน 7-segment display ผ่าน PORTD4-7 และจะแสดงเลขทุกครั้งที่ r29 ลดลงไป60 ค่า ผลลัพธ์ที่สังเกตุเห็นคือการนับ 3..2..1..0

3. **การจัดการการขัดจังหวะ** (Interrupt)

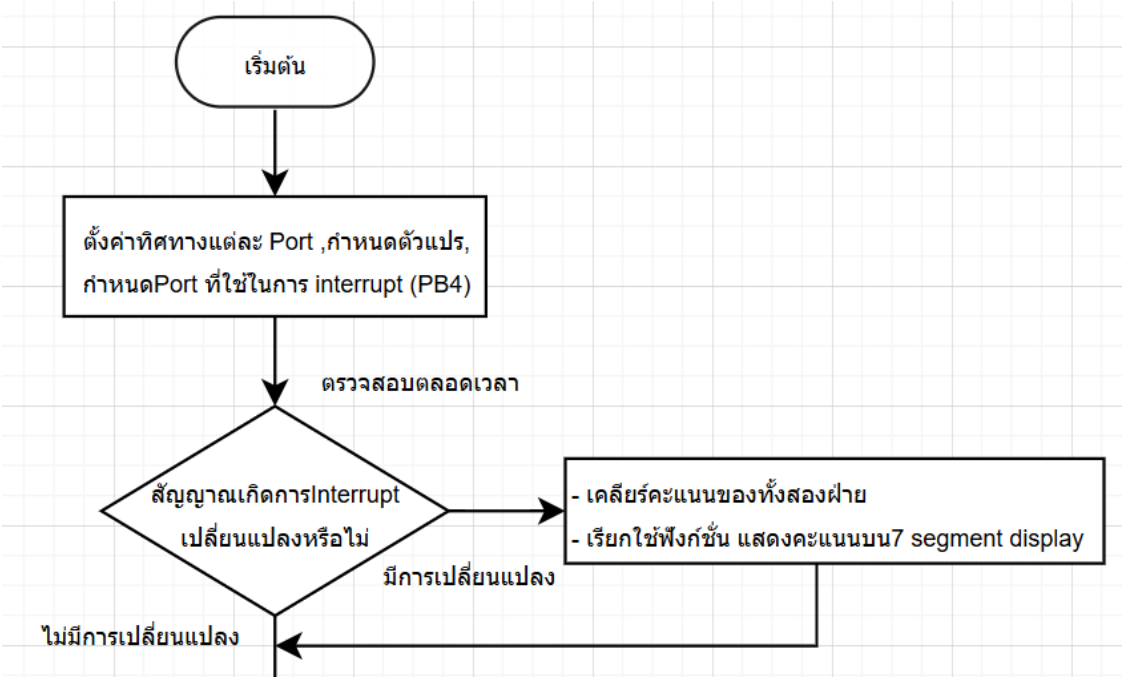
- เมื่อเกิดการขัดจังหวะ คือ เมื่อกดปุ่มที่เชื่อมกับ PB4 ระบบจะรีเซ็ตคะแนนของผู้เล่นทั้งสองเป็น 0 และอัปเดตผลลัพธ์บน 7-segment display

4. **เป้าหมายของวงจร**

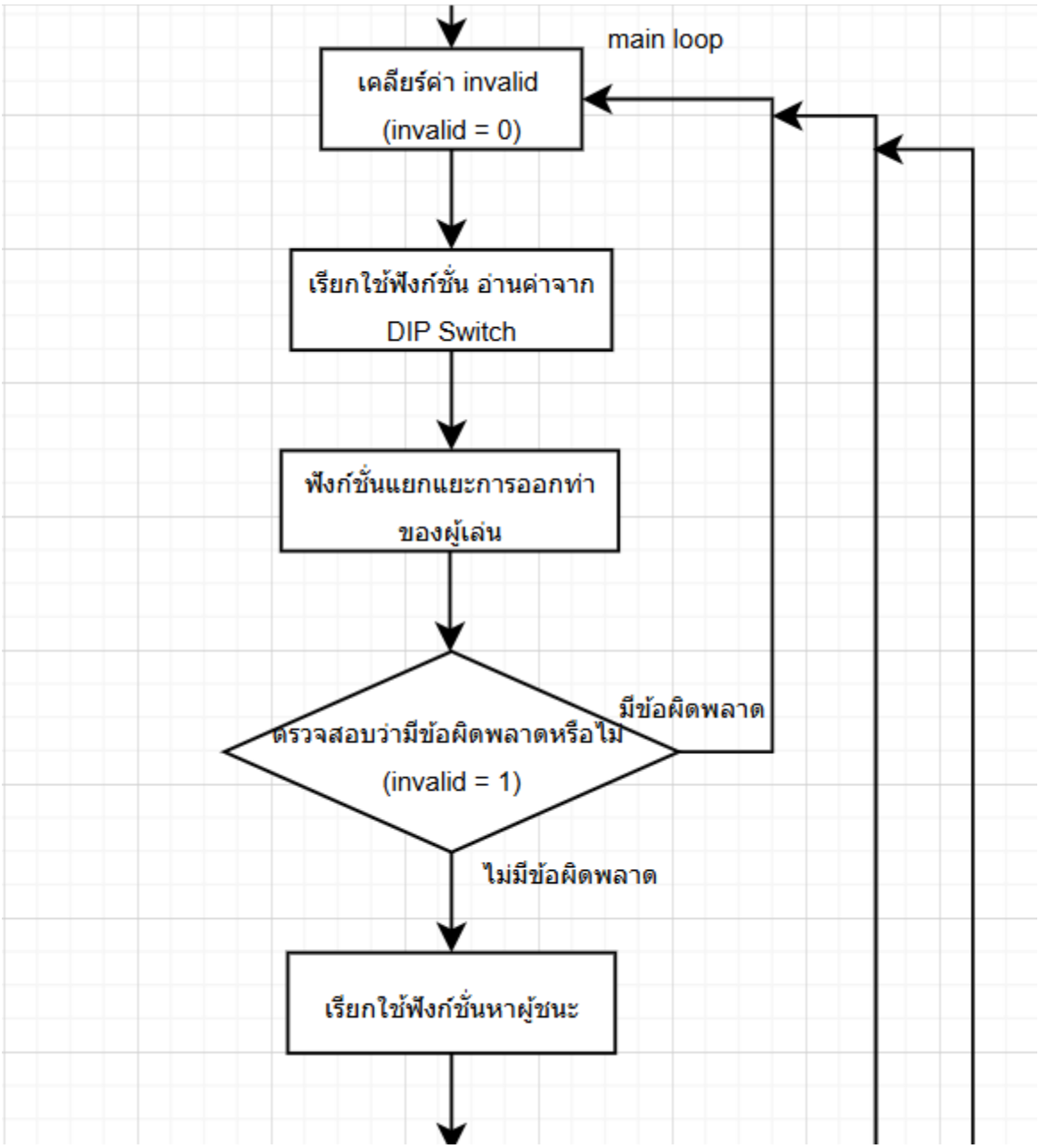
จำลองเกมเป่ายิ้งฉุบโดยให้ผู้เล่นสองคนเลือกตัวเลือกผ่าน DIP switch (PC0-PC2 สำหรับผู้เล่น 1, PC3-PC5 สำหรับผู้เล่น 2) และแสดงคะแนนสะสมผ่าน 7-segment และ ผู้เล่นสามารถเลือก ค้อน กรรไกร กระดาษ โดยเลื่อน DIP switch ให้เป็น OFF(เพราะสวิตช์ต่อแบบ pull-down) โปรแกรมตรวจสอบความถูกต้อง โดยการเปรียบเทียบ และอัปเดตคะแนนอัตโนมัติ จากนั้นแสดงผล คะแนนของผู้เล่น 1 และ 2 แสดงแยกกันบน 7-segment และ มีเวลา3วินาทีให้เลือกค้อน กรรไกร กระดาษ หรือเปลี่ยนตัวเลือก หากเลือกไม่ทันเวลา ไม่เลือก หรือเลือกมากกว่า1 วงจรจะรอให้ผู้เล่นเลือกอย่างถูกต้องแล้วจึงเริ่มการคิดคะแนนและแสดงผล

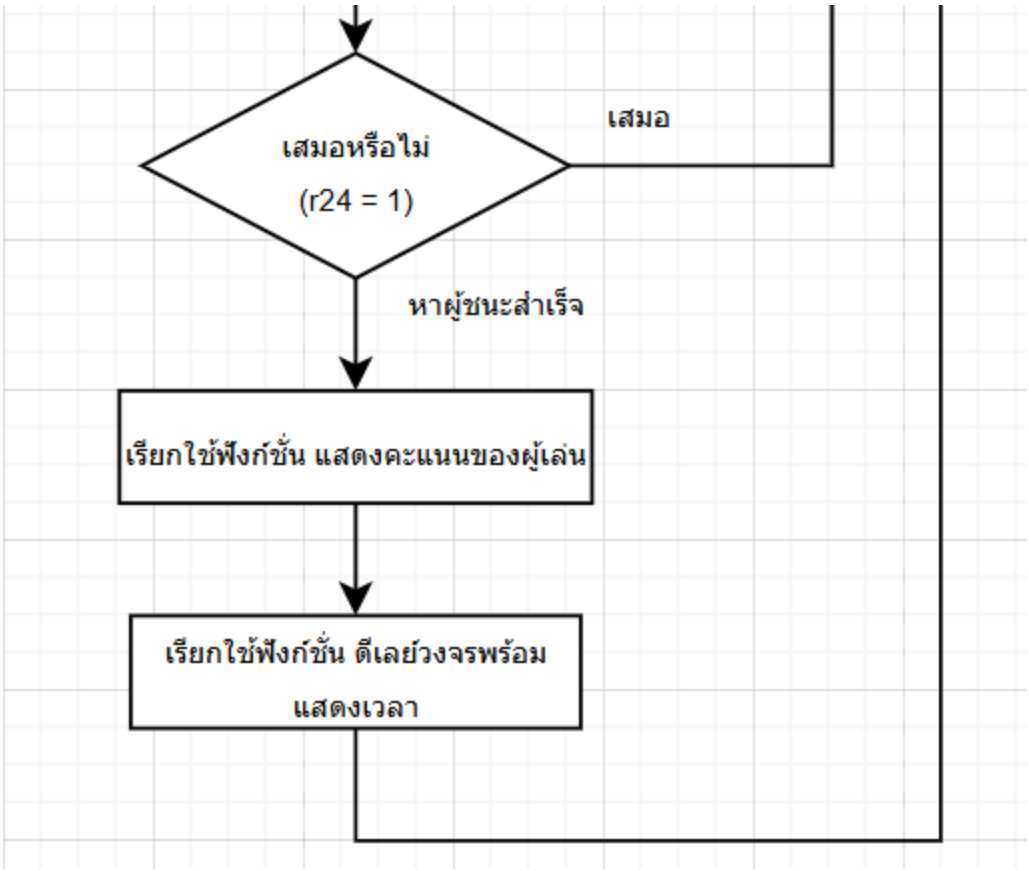
การรีเซ็ตคะแนนแบ่งเป็น 2 วิธี ได้แก่ เมื่อมีฝั่งใดฝั่ง1 มีคะแนนสะสมถึง 8 หรือเกิดการขัดจังหวะจาก PB4 และวงจรจะแสดงคะแนนสะสมของทั้งสองฝ่ายเป็น 0

Flowchart **ตั้งค่าเริ่มต้น**

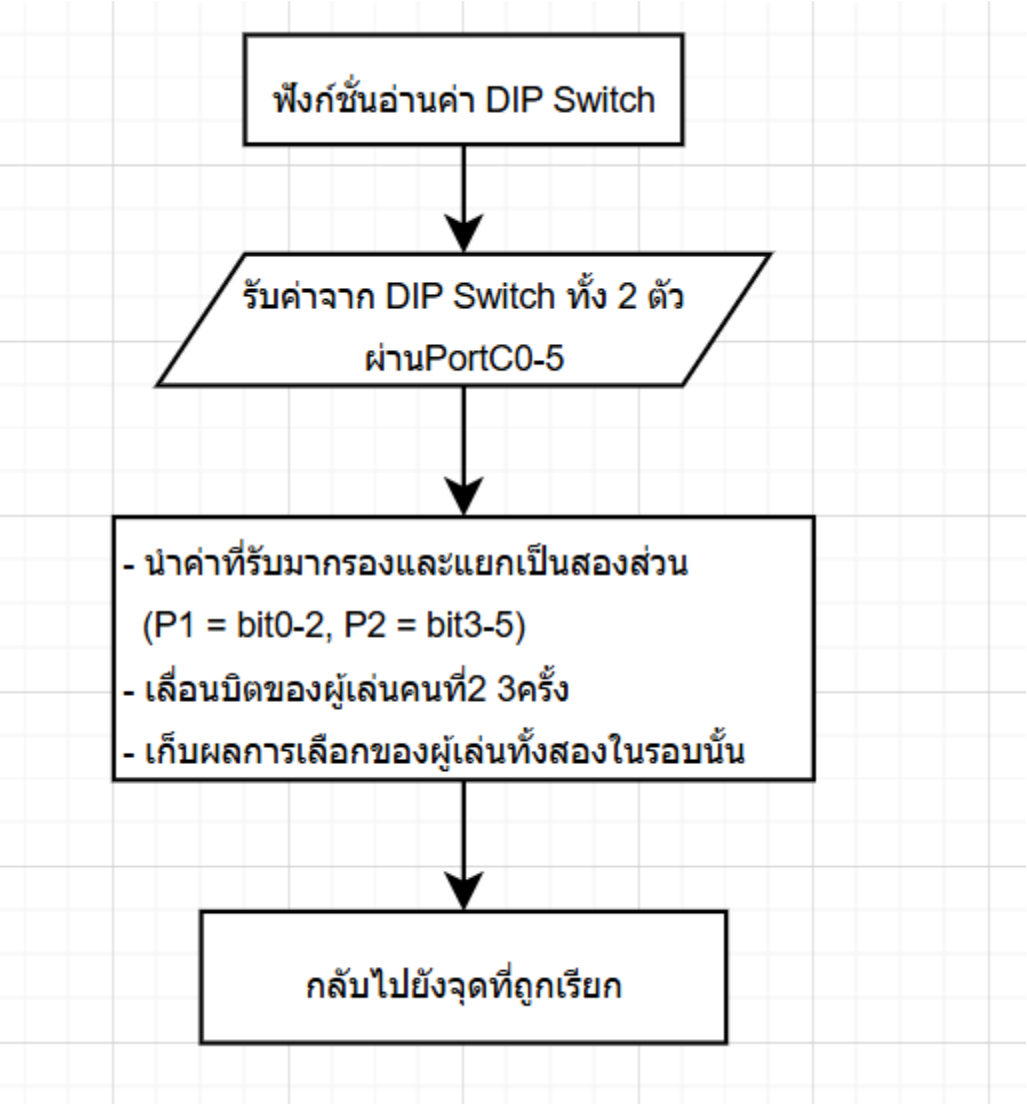


Flowchart main-loop

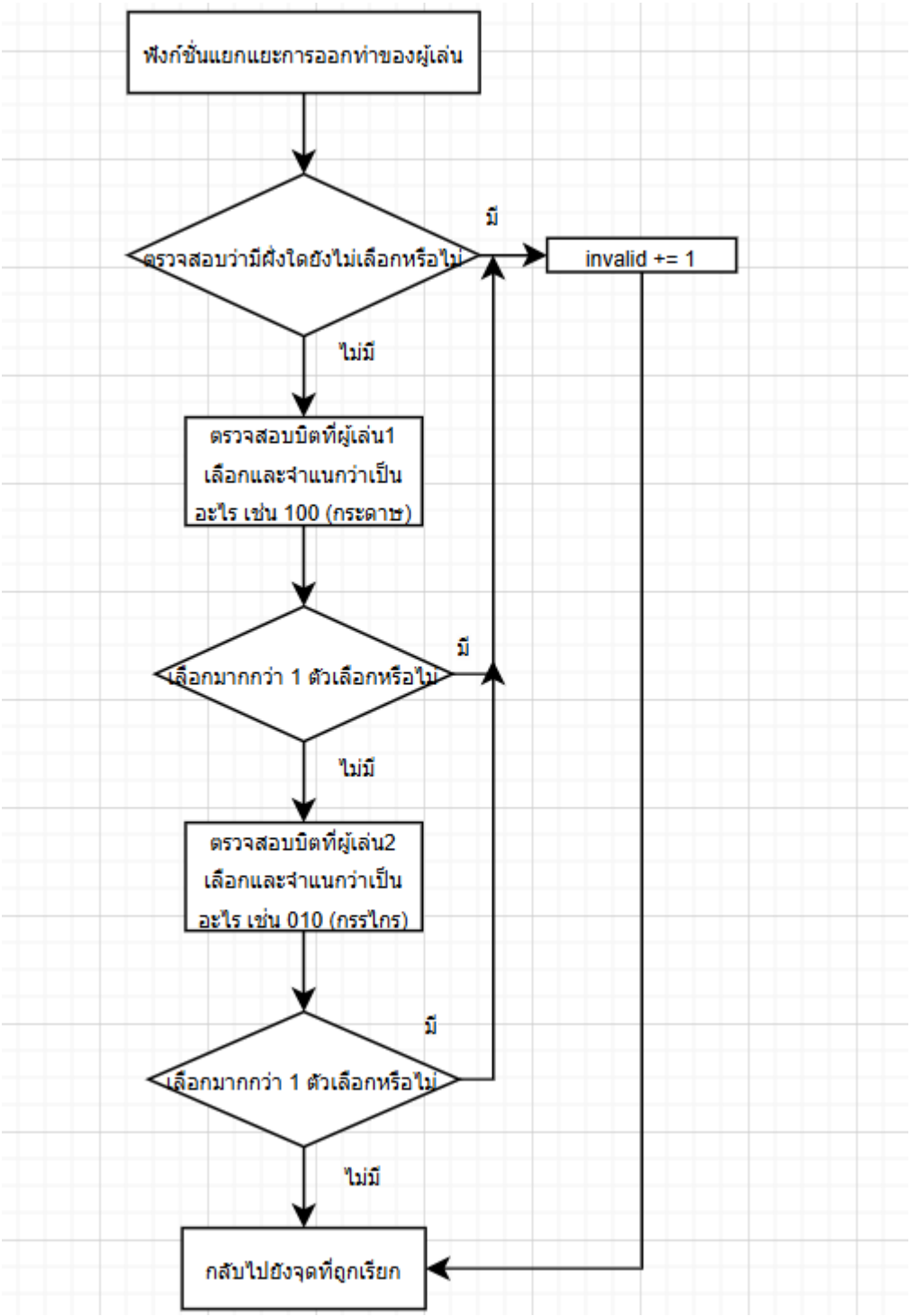



เสมอหรือไม่
(r24 = 1)
เสมอ
หาผู้ชนะสำเร็จ
เรียกใช้ฟังก์ชั่น แสดงคะแนนของผู้เล่น
เรียกใช้ฟังก์ชั่น ดีเลย์วงจรพร้อม
แสดงเวลา

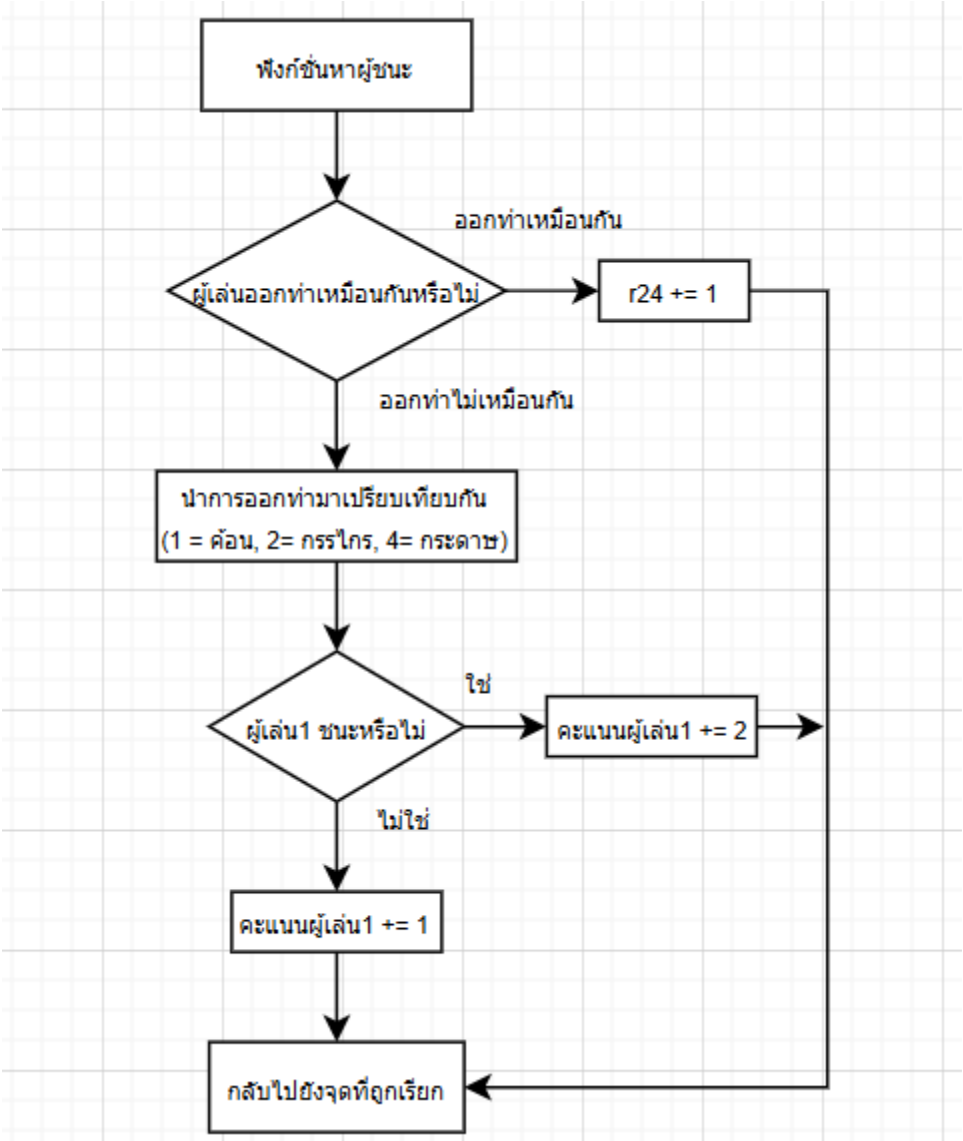
Flowchart ฟังก์ชั่นอ่านค่า DIP Switch



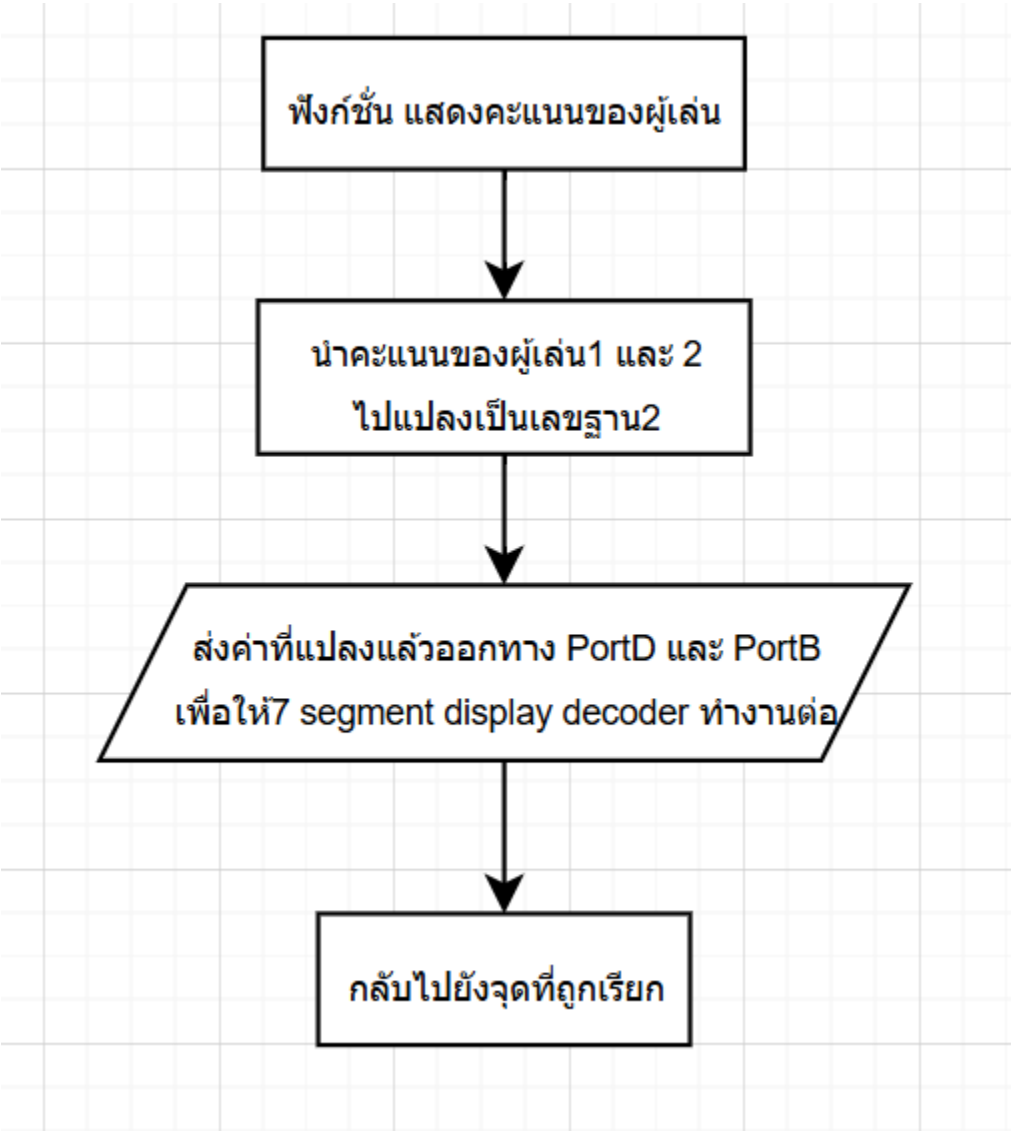
Flowchart แยกแยะการออกท่าของผู้เล่น



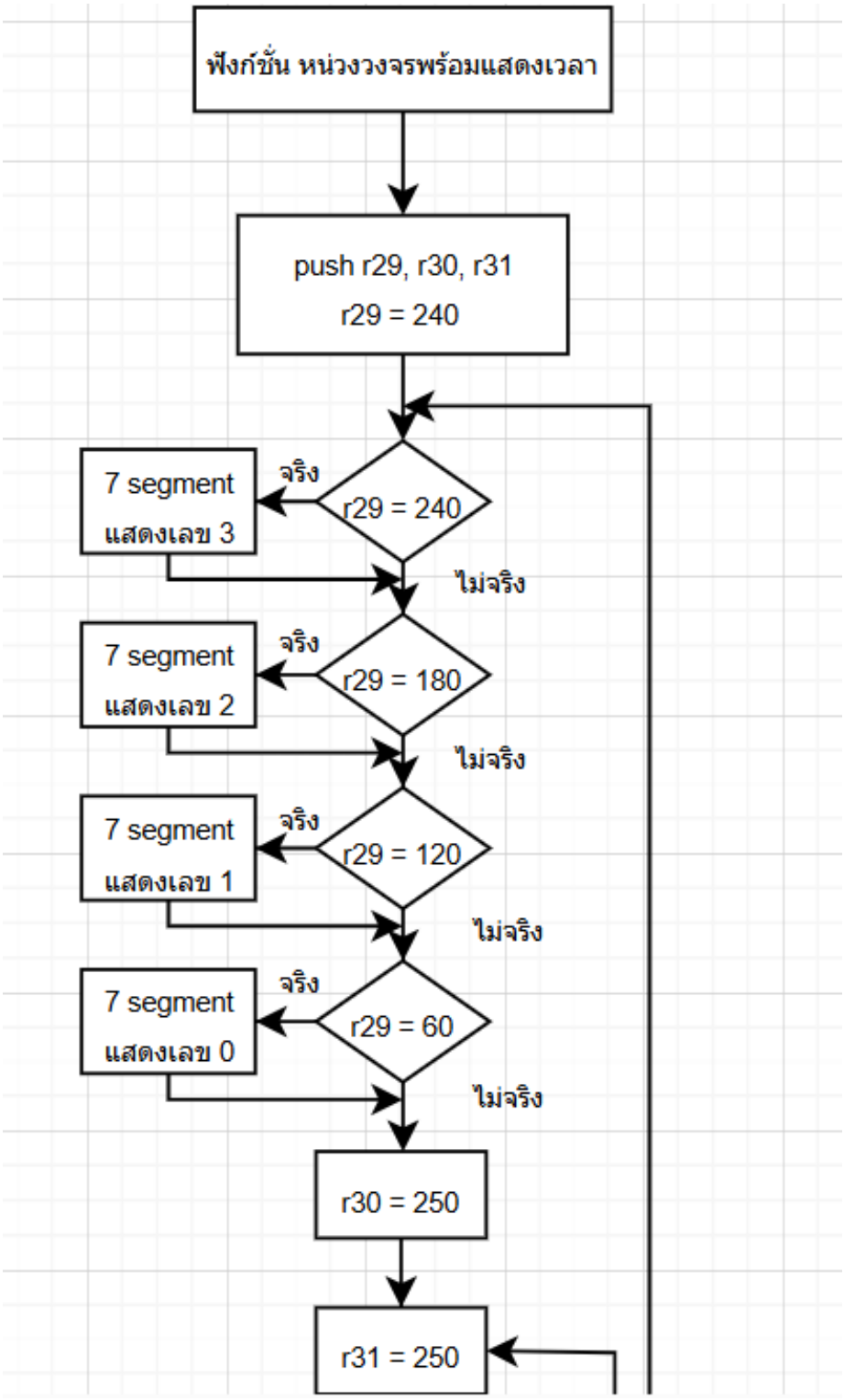
Flowchart หาผู้ชนะ

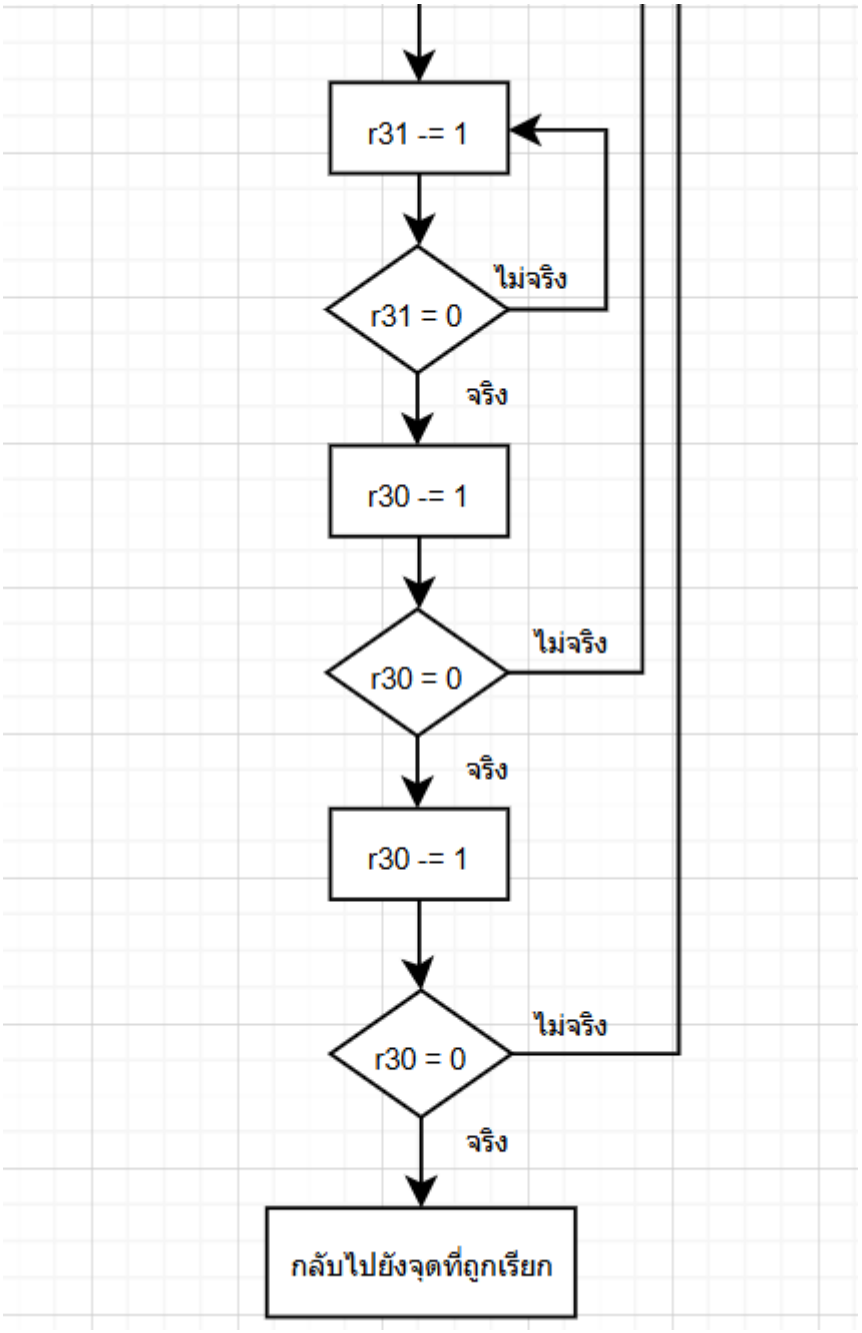


Flowchart แสดงคะแนนของผู้เล่น



Flowchart **หน่วงวงจรพร้อมแสดงเวลา**




r31 -= 1
r31 = 0
ไม่จริง
จริง
r30 -= 1
r30 = 0
ไม่จริง
จริง
r30 -= 1
r30 = 0
ไม่จริง
จริง
กลับไปยังจุดที่ถูกเรียก

Code โปรแกรมของวงจร

```
.include "m328Pdef.inc" ; รวมไฟล์กำหนดค่าพื้นฐานของไมโครคอนโทรลเลอร์ ATmega328P

.cseg ; ระบุว่าโค้ดต่อไปนี้อยู่ในส่วนของโค้ด (code segment)

.def temp = r16 ; กำหนดให้ r16 เป็นตัวแปรชั่วคราวชื่อ "temp" ใช้เก็บค่าต่างๆ

.def choice1 = r17 ; กำหนดให้ r17 เป็นตัวเลือกของผู้เล่น 1 (เช่น Rock, Paper, Scissors)

.def choice2 = r18 ; กำหนดให้ r18 เป็นตัวเลือกของผู้เล่น 2

.def score1 = r19 ; กำหนดให้ r19 เป็นคะแนนของผู้เล่น 1

.def score2 = r20 ; กำหนดให้ r20 เป็นคะแนนของผู้เล่น 2

.def check = r21 ; กำหนดให้ r21 เป็นตัวแปรตรวจสอบว่าตัวเลือกถูกต้องหรือไม่

.def invalid = r22 ; กำหนดให้ r22 เป็นตัวแปรบอกว่าการตรวจสอบเสร็จสิ้นหรือยัง

.org 0x0000 ; กำหนดจุดเริ่มต้นของโปรแกรมที่อยู่ 0x0000

rjmp Setup ; กระโดดไปยังส่วน Setup เพื่อเริ่มตั้งค่าโปรแกรม

.org 0x0006 ; กำหนดจุดสำหรับการขัดจังหวะ (interrupt) ที่อยู่ 0x0006

rjmp Interrupt ; กระโดดไปยังส่วน Interrupt เมื่อมีการขัดจังหวะเกิดขึ้น

Setup: ; ส่วนตั้งค่าเริ่มต้นของโปรแกรม

    ldi temp, 0x00 ; โหลดค่า 0x00 (00000000) เข้า temp เพื่อตั้งให้ PortC เป็นอินพุต

    out DDRC, temp ; ตั้งค่า PortC (PC0-PC7) ให้เป็นอินพุตทั้งหมด

    ldi temp, 0xFF ; โหลดค่า 0xFF (11111111) เข้า temp เพื่อตั้งให้ PortD เป็นเอาต์พุต

    out DDRD, temp ; ตั้งค่า PortD (pin PD0-PD7) ให้เป็นเอาต์พุตทั้งหมด

    ldi temp, 0x0F ; โหลดค่า 0x0F (00001111) เข้า temp เพื่อตั้งให้ PB0-PB3 เป็นเอาต์พุต
```

```
    out DDRB, temp ; ตั้งค่า PortB (PB0-PB3 เป็นเอาต์พุต, PB4-PB7 เป็นอินพุต)
    ldi temp, 0x01 ; โหลดค่า 0x01 เข้า temp เพื่อเปิดใช้งานการขัดจังหวะสำหรับ PortB
    sts PCICR, temp ; เก็บค่าใน PCICR เพื่อเปิดใช้งาน Pin Change Interrupt บน PortB
    ldi temp, 0x10 ; โหลดค่า 0x10 (00010000) เพื่อเลือก PB4 สำหรับการInterrupt
    sts PCMSK0, temp ; เก็บค่าใน PCMSK0 เพื่อกำหนดให้ PB4 เป็นตัว trigger Interrupt
    sei ; เปิดการใช้งานระบบขัดจังหวะทั่วทั้งชิป (global interrupt enable)
    clr score1          ; ล้างค่า score1 (คะแนนผู้เล่น 1) ให้เป็น 0
    clr score2          ; ล้างค่า score2 (คะแนนผู้เล่น 2) ให้เป็น 0
    clr temp            ; ล้างค่า temp ให้เป็น 0 เพื่อเตรียมใช้งานต่อไป
Loop: ; ลูปหลักของโปรแกรม
    clr invalid ; ล้างค่า invalid เป็น 0 (หมายถึงยังไม่มีข้อผิดพลาด)
    rcall ReadDIPSwitch ; เรียกฟังก์ชันเพื่ออ่านค่าจาก DIP switch (ตัวเลือกของผู้เล่น)
    rcall ValidateInput ; เรียกฟังก์ชันเพื่อตรวจสอบว่าตัวเลือกของผู้เล่นถูกต้องตามหลักหรือไม่
    cpi invalid, 1 ; เปรียบเทียบ invalid กับ 1 (ถ้า = 1 แปลว่ามีข้อผิดพลาด)
    breq Loop ; ถ้า invalid = 1 (มีข้อผิดพลาด) ให้กลับไปเริ่มลูปใหม่
    rcall CompareChoices ; เรียกฟังก์ชันเพื่อเปรียบเทียบตัวเลือกของผู้เล่นทั้งสอง
    cpi r24, 0x01 ; เปรียบเทียบ r24 กับ 1 (r24 = 1 ถ้าเสมอกัน)
    breq Loop ; ถ้า r24 = 1 (เสมอ) ให้กลับไปเริ่มลูปใหม่
    rcall Display ; เรียกฟังก์ชันเพื่อแสดงผลคะแนนบน 7-segment
    rcall Delay3Sec ; เรียกฟังก์ชันเพื่อหน่วงเวลา 3 วินาที
    rjmp Loop ; กระโดดกลับไปเริ่มลูปใหม่ ไม่ว่าจะเกิดอะไรขึ้น
```

```
Interrupt: ; ส่วนจัดการการขัดจังหวะ
        rcall ResetScores ; เรียกฟังก์ชันเพื่อรีเซ็ตคะแนนทั้งหมด
        reti ; ออกจากการขัดจังหวะและกลับไปทำงานปกติ
        ResetScores: ; ฟังก์ชันรีเซ็ตคะแนน
        clr score1 ; ล้างคะแนนผู้เล่น 1 ให้เป็น 0
        clr score2 ; ล้างคะแนนผู้เล่น 2 ให้เป็น 0
        rcall Display ; เรียกฟังก์ชันเพื่ออัปเดตการแสดงผลคะแนน
        ret ; กลับไปยังจุดที่เรียกฟังก์ชันนี้
ReadDIPSwitch: ; ฟังก์ชันอ่านค่าจาก DIP switch บน PortC
        in temp, PINC ; อ่านค่าจาก PortC (PINC) มาเก็บใน temp
        mov choice1, temp ; คัดลอกค่าจาก temp ไปเก็บใน choice1 (ตัวเลือกผู้เล่น 1)
        andi choice1, 0x07 ; กรองเฉพาะบิต PC0-PC2 (00000111) สำหรับตัวเลือกผู้เล่น 1 (อยู่ในรูป
        00000XXX)
        mov choice2, temp ; คัดลอกค่าจาก temp ไปเก็บใน choice2 (ตัวเลือกผู้เล่น 2)
        andi choice2, 0x38 ; กรองเฉพาะบิต PC3-PC5 (00111000) สำหรับตัวเลือกผู้เล่น 2
        lsr choice2 ; เลื่อนบิตใน choice2 ไปทางขวา 1 ตำแหน่ง
        lsr choice2 ; เลื่อนบิตใน choice2 ไปทางขวาอีก 1 ตำแหน่ง
        lsr choice2 ; เลื่อนบิตใน choice2 ไปทางขวาอีก 1 ตำแหน่ง (ทำให้อยู่ในรูป 00000XXX)
        ret ; กลับไปยังจุดที่เรียกฟังก์ชันนี้
```

```
ValidateInput: ; ฟังก์ชันตรวจสอบความถูกต้องของตัวเลือก
        cpi choice1, 0 ; เปรียบเทียบ choice1 กับ 0 (ถ้า = 0 แปลว่าไม่ได้เลือกอะไร)
        breq Nothing ; ถ้า choice1 = 0 ให้ไปที่ Nothing (ถือว่าไม่ถูกต้อง)
        cpi choice2, 0 ; เปรียบเทียบ choice2 กับ 0 (ถ้า = 0 แปลว่าไม่ได้เลือกอะไร)
        breq Nothing ; ถ้า choice2 = 0 ให้ไปที่ Nothing (ถือว่าไม่ถูกต้อง)
        ldi temp, 0x01 ; โหลดค่า 1 เข้า temp (ใช้เป็นเกณฑ์ว่าต้องเลือกแค่ 1 ตัวเลือก)
        rcall Check1 ; เรียกฟังก์ชันเพื่อตรวจสอบ choice1 เพิ่มเติม
        End_Detect: ; จุดสิ้นสุดของการตรวจสอบ
        ret ; กลับไปยังจุดที่เรียกฟังก์ชันนี้
Nothing: ; กรณีตัวเลือกไม่ถูกต้อง (เช่น ไม่ได้เลือกอะไร)
        inc invalid ; เพิ่มค่า invalid เป็น 1 (บอกว่ามีข้อผิดพลาด)
        rjmp End_Detect ; กระโดดไปยังจุดสิ้นสุดของการตรวจสอบ
Check1: ; ฟังก์ชันตรวจสอบ choice1 ว่ามีการเลือกเพียง 1 ตัวเลือกหรือไม่
        clr check ; ล้างค่า check เป็น 0 (ใช้เก็บจำนวนตัวเลือกที่ถูกเลือก)
        sbrc choice1, 0 ; ตรวจสอบบิต 0 ของ choice1 (ถ้า = 1 แปลว่าเลือกตัวเลือกนี้)
        inc check ; เพิ่มค่า check ถ้าบิต 0 เป็น 1
        sbrc choice1, 1 ; ตรวจสอบบิต 1 ของ choice1
        inc check ; เพิ่มค่า check ถ้าบิต 1 เป็น 1
        sbrc choice1, 2 ; ตรวจสอบบิต 2 ของ choice1
        inc check ; เพิ่มค่า check ถ้าบิต 2 เป็น 1
```

```
        cp temp, check ; เปรียบเทียบ temp (1) กับ check (จำนวนตัวเลือก)
        brlo Check1Fall ; ถ้า temp < check (เลือกมากกว่า 1) ให้ไปที่ Check1Fall
        rcall Check2 ; ถ้าผ่านการตรวจสอบ ให้ไปตรวจ choice2 ต่อ
        End_Check1: ; จุดสิ้นสุดของ Check1
        ret ; กลับไปยังจุดที่เรียกฟังก์ชันนี้
Check1Fall: ; กรณี choice1 ไม่ถูกต้อง (เลือกมากกว่า 1 ตัวเลือก)
        inc invalid ; เพิ่มค่า invalid เป็น 1 (บอกว่ามีข้อผิดพลาด)
        rjmp End_Check1 ; กระโดดไปยังจุดสิ้นสุดของ Check1
Check2: ; ฟังก์ชันตรวจสอบ choice2 ว่ามีการเลือกเพียง 1 ตัวเลือกหรือไม่
        clr check ; ล้างค่า check เป็น 0 (ใช้เก็บจำนวนตัวเลือกที่ถูกเลือก)
        sbrc choice2, 0 ; ตรวจสอบบิต 0 ของ choice2
        inc check ; เพิ่มค่า check ถ้าบิต 0 เป็น 1
        sbrc choice2, 1 ; ตรวจสอบบิต 1 ของ choice2
        inc check ; เพิ่มค่า check ถ้าบิต 1 เป็น 1
        sbrc choice2, 2 ; ตรวจสอบบิต 2 ของ choice2
        inc check ; เพิ่มค่า check ถ้าบิต 2 เป็น 1
        cp temp, check ; เปรียบเทียบ temp (1) กับ check (จำนวนตัวเลือก)
        brlo Check2Fall ; ถ้า temp < check (เลือกมากกว่า 1) ให้ไปที่ Check2Fall
        End_Check2: ; จุดสิ้นสุดของ Check2
        ret ; กลับไปยังจุดที่เรียกฟังก์ชันนี้
```

```
Check2Fall: ; กรณี choice2 ไม่ถูกต้อง (เลือกมากกว่า 1 ตัวเลือก)
        inc invalid ; เพิ่มค่า invalid เป็น 1 (บอกว่ามีข้อผิดพลาด)
        rjmp End_Check2 ; กระโดดไปยังจุดสิ้นสุดของ Check2
CompareChoices: ; ฟังก์ชันเปรียบเทียบตัวเลือกของผู้เล่นทั้งสอง
        clr temp ; ล้างค่า temp เป็น 0 (ใช้เป็นตัวช่วยชั่วคราว)
        ldi r24, 0x00 ; โหลดค่า 0 เข้า r24 (ใช้บอกสถานะ: 0 = ไม่เสมอ, 1 = เสมอ)
        cp choice1, choice2 ; เปรียบเทียบ choice1 กับ choice2
        breq Tie ; ถ้า choice1 = choice2 (เสมอ) ให้ไปที่ Tie
        cpi choice1, 1      ; เปรียบเทียบ choice1 กับ 1 (Rock)
        breq Select1        ; ถ้า choice1 = 1 ให้ไปที่ Select1
        cpi choice1, 2      ; เปรียบเทียบ choice1 กับ 2 (Paper)
        breq Select2        ; ถ้า choice1 = 2 ให้ไปที่ Select2
        cpi choice1, 4      ; เปรียบเทียบ choice1 กับ 4 (Scissors)
        breq Select4        ; ถ้า choice1 = 4 ให้ไปที่ Select4
        EndCompare:         ; จุดสิ้นสุดของการเปรียบเทียบ
        ret                 ; กลับไปยังจุดที่เรียกฟังก์ชันนี้
Tie: ; กรณีเสมอกัน
        inc r24 ; เพิ่มค่า r24 เป็น 1 (บอกว่าเสมอ)
        rjmp EndCompare ; กระโดดไปยังจุดสิ้นสุดของการเปรียบเทียบ
Select1: ; กรณีผู้เล่น 1 เลือก Rock (1) cpi choice2, 2 ; เปรียบเทียบ choice2 กับ 2 (Paper)
        breq P1Wins ; ถ้า choice2 = 2 (Rock vs Paper) ผู้เล่น 2 ชนะ
        cpi choice2, 4 ; เปรียบเทียบ choice2 กับ 4 (Scissors)
```

```
        breq P2Wins ; ถ้า choice2 = 4 (Rock vs Scissors) ผู้เล่น 1 ชนะ
        rjmp EndCompare ; กระโดดไปยังจุดสิ้นสุด
Select2: ; กรณีผู้เล่น 1 เลือก Paper (2)
        cpi choice2, 4 ; เปรียบเทียบ choice2 กับ 4 (Scissors)
        breq P1Wins ; ถ้า choice2 = 4 (Paper vs Scissors) ผู้เล่น 2 ชนะ
        cpi choice2, 1 ; เปรียบเทียบ choice2 กับ 1 (Rock)
        breq P2Wins ; ถ้า choice2 = 1 (Paper vs Rock) ผู้เล่น 1 ชนะ
        rjmp EndCompare ; กระโดดไปยังจุดสิ้นสุด
Select4: ; กรณีผู้เล่น 1 เลือก Scissors (4)
        cpi choice2, 1 ; เปรียบเทียบ choice2 กับ 1 (Rock)
        breq P1Wins ; ถ้า choice2 = 1 (Scissors vs Rock) ผู้เล่น 2 ชนะ
        cpi choice2, 2 ; เปรียบเทียบ choice2 กับ 2 (Paper)
        breq P2Wins ; ถ้า choice2 = 2 (Scissors vs Paper) ผู้เล่น 1 ชนะ
        rjmp EndCompare ; กระโดดไปยังจุดสิ้นสุด
P1Wins: ; กรณีผู้เล่น 1 ชนะ
        inc score1 ; เพิ่มคะแนนผู้เล่น 1 ขึ้น 1
        sbrc score1, 3 ; ตรวจสอบบิตที่ 3 ของ score1 (ถ้า = 1 แปลว่าคะแนนถึง 8)
        rcall ResetScores ; ถ้าคะแนนถึง 8 ให้รีเซ็ตคะแนนทั้งหมด
        rjmp EndCompare ; กระโดดไปยังจุดสิ้นสุด
```

```
P2Wins: ; กรณีผู้เล่น 2 ชนะ
        inc score2 ; เพิ่มคะแนนผู้เล่น 2 ขึ้น 1
        sbrc score2, 3 ; ตรวจสอบบิตที่ 3 ของ score2 (ถ้า = 1 แปลว่าคะแนนถึง 8)
        rcall ResetScores ; ถ้าคะแนนถึง 8 ให้รีเซ็ตคะแนนทั้งหมด
        rjmp EndCompare ; กระโดดไปยังจุดสิ้นสุด
Display: ; ฟังก์ชันแสดงผลคะแนนบน 7-segment
        mov r25, score2 ; คัดลอกคะแนนผู้เล่น 2 ไปยัง r25
        rcall BIN_TO_7SEG ; เรียกฟังก์ชันเพื่อแปลงคะแนนเป็นรหัส 7-segment
        out PORTB, r25 ; ส่งรหัส 7-segment ไปยัง PortB เพื่อแสดงคะแนนผู้เล่น 2
        mov r25, score1       ; คัดลอกคะแนนผู้เล่น 1 ไปยัง r25
        rcall BIN_TO_7SEG     ; เรียกฟังก์ชันเพื่อแปลงคะแนนเป็นรหัส 7-segment
        out PORTD, r25        ; ส่งรหัส 7-segment ไปยัง PortD เพื่อแสดงคะแนนผู้เล่น 1
        ret                   ; กลับไปยังจุดที่เรียกฟังก์ชันนี้
BIN_TO_7SEG: ; ฟังก์ชันแปลงตัวเลขไบนารีเป็นรหัส 7-segment
        ldi temp, 0x00 ; โหลดค่า 0 เข้า temp (ใช้เป็นตัวช่วยชั่วคราว)
        cpi r25, 0 ; เปรียบเทียบ r25 กับ 0
        breq ZERO ; ถ้า r25 = 0 ให้ไปที่ ZERO
        cpi r25, 1 ; เปรียบเทียบ r25 กับ 1
        breq ONE ; ถ้า r25 = 1 ให้ไปที่ ONE
        cpi r25, 2 ; เปรียบเทียบ r25 กับ 2
        breq TWO ; ถ้า r25 = 2 ให้ไปที่ TWO
```

```
        cpi r25, 3 ; เปรียบเทียบ r25 กับ 3
        breq THREE ; ถ้า r25 = 3 ให้ไปที่ THREE
        cpi r25, 4 ; เปรียบเทียบ r25 กับ 4
        breq FOUR ; ถ้า r25 = 4 ให้ไปที่ FOUR
        cpi r25, 5 ; เปรียบเทียบ r25 กับ 5
        breq FIVE ; ถ้า r25 = 5 ให้ไปที่ FIVE
        cpi r25, 6 ; เปรียบเทียบ r25 กับ 6
        breq SIX ; ถ้า r25 = 6 ให้ไปที่ SIX
        cpi r25, 7 ; เปรียบเทียบ r25 กับ 7
        breq SEVEN ; ถ้า r25 = 7 ให้ไปที่ SEVEN
        End_trans: ; จุดสิ้นสุดของการแปลง
        ret ; กลับไปยังจุดที่เรียกฟังก์ชันนี้
ZERO: ; รหัส 7-segment สำหรับเลข 0
        ldi r25, 0b00000000 ; โหลดรหัส 7-segment สำหรับเลข 0 เข้า r25
        rjmp End_trans ; กระโดดไปยังจุดสิ้นสุด
ONE: ; รหัส 7-segment สำหรับเลข 1
        ldi r25, 0b00000001 ; โหลดรหัส 7-segment สำหรับเลข 1 เข้า r25
        rjmp End_trans ; กระโดดไปยังจุดสิ้นสุด
TWO: ; รหัส 7-segment สำหรับเลข 2
        ldi r25, 0b00000010 ; โหลดรหัส 7-segment สำหรับเลข 2 เข้า r25
```

```
        rjmp End_trans ; กระโดดไปยังจุดสิ้นสุด

THREE: ; รหัส 7-segment สำหรับเลข 3

        ldi r25, 0b00000011 ; โหลดรหัส 7-segment สำหรับเลข 3 เข้า r25

        rjmp End_trans ; กระโดดไปยังจุดสิ้นสุด

FOUR: ; รหัส 7-segment สำหรับเลข 4

        ldi r25, 0b00000100 ; โหลดรหัส 7-segment สำหรับเลข 4 เข้า r25

        rjmp End_trans ; กระโดดไปยังจุดสิ้นสุด

FIVE: ; รหัส 7-segment สำหรับเลข 5

        ldi r25, 0b00000101 ; โหลดรหัส 7-segment สำหรับเลข 5 เข้า r25

        rjmp End_trans ; กระโดดไปยังจุดสิ้นสุด

SIX: ; รหัส 7-segment สำหรับเลข 6

        ldi r25, 0b00000110 ; โหลดรหัส 7-segment สำหรับเลข 6 เข้า r25

        rjmp End_trans ; กระโดดไปยังจุดสิ้นสุด

SEVEN: ; รหัส 7-segment สำหรับเลข 7

        ldi r25, 0b00000111 ; โหลดรหัส 7-segment สำหรับเลข 7 เข้า r25

        rjmp End_trans ; กระโดดไปยังจุดสิ้นสุด

Delay3Sec: ; ฟังก์ชันหน่วงเวลา 3 วินาที

        push r29 ; บันทึกค่าใน r29 ลง stack เพื่อป้องกันการเปลี่ยนแปลง

        push r30 ; บันทึกค่าใน r30 ลง stack

        push r31 ; บันทึกค่าใน r31 ลง stack
```

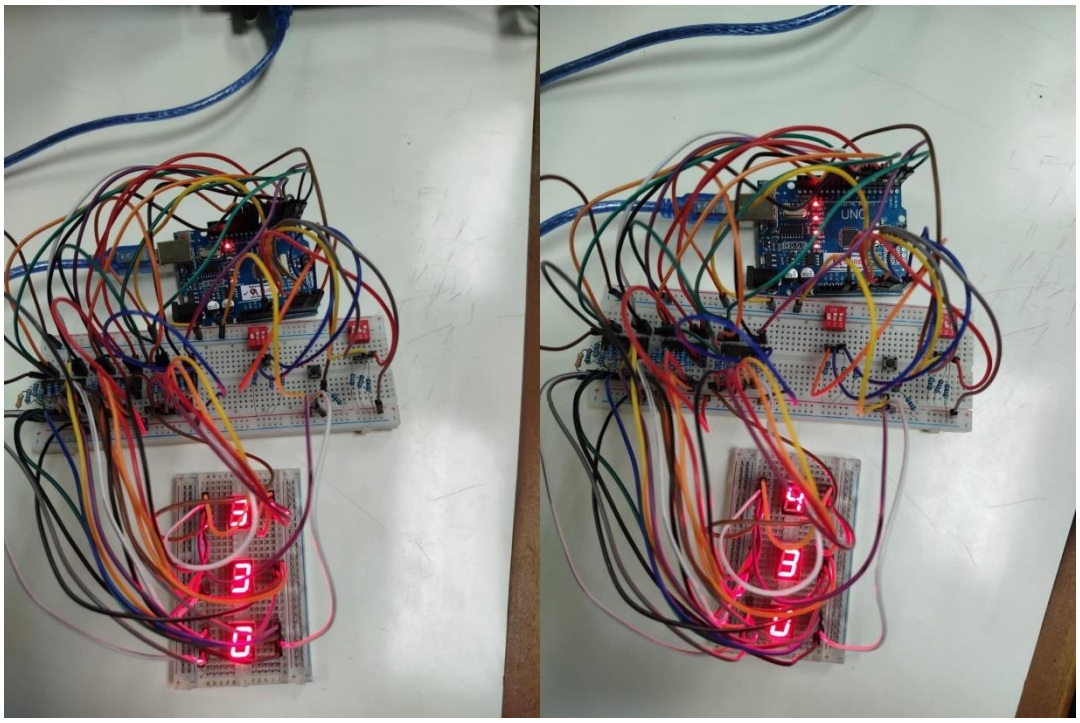
```
ldi r29, 240 ; โหลดค่า 240 เข้า r29(ใช้เป็นตัวนับรอบนอก) ## จำลองใน proteus แนะนำ 16
Checktime:          ; ลูปนอก
cpi r29, 240  ; เปรียบเทียบ r29 กับ 240 (เหลือ 3 วินาที) ## หากจำลองใน proteus แนะนำ 16
breq ShowTime3      ; ถ้า r29 = 240 ให้ไปแสดงผล 3 วินาที
cpi r29, 180   ; เปรียบเทียบ r29 กับ 180 (เหลือ 2 วินาที) ## หากจำลองใน proteus แนะนำ 12
breq ShowTime2      ; ถ้า r29 = 180 ให้ไปแสดงผล 2 วินาที
cpi r29, 120    ; เปรียบเทียบ r29 กับ 120 (เหลือ 1 วินาที) ## หากจำลองใน proteus แนะนำ 8
breq ShowTime1      ; ถ้า r29 = 120 ให้ไปแสดงผล 1 วินาที
cpi r29, 60     ; เปรียบเทียบ r29 กับ 60 (เหลือ 0 วินาที) ## หากจำลองใน proteus แนะนำ 4
breq ShowTime0      ; ถ้า r29 = 60 ให้ไปแสดงผล 0 วินาที
OuterLoop:          ; จุดต่อไปหลังจากแสดงผล
ldi r30, 250        ; โหลดค่า 250 เข้า r30 (ใช้เป็นตัวนับลูปกลาง)
MiddleLoop:         ; ลูปกลาง
ldi r31, 250        ; โหลดค่า 250 เข้า r31 (ใช้เป็นตัวนับลูปใน)
InnerLoop:          ; ลูปใน
dec r31             ; ลดค่า r31 ลง 1
brne InnerLoop      ; ถ้า r31 ยังไม่เท่ากับ 0 ให้วนลูปในต่อ

dec r30             ; ลดค่า r30 ลง 1
brne MiddleLoop     ; ถ้า r30 ยังไม่เท่ากับ 0 ให้วนลูปกลางต่อ

dec r29             ; ลดค่า r29 ลง 1
brne Checktime      ; ถ้า r29 ยังไม่เท่ากับ 0 ให้วนลูปนอกต่อ

pop r31             ; ดึงค่า r31 กลับจาก stack
pop r30             ; ดึงค่า r30 กลับจาก stack
```

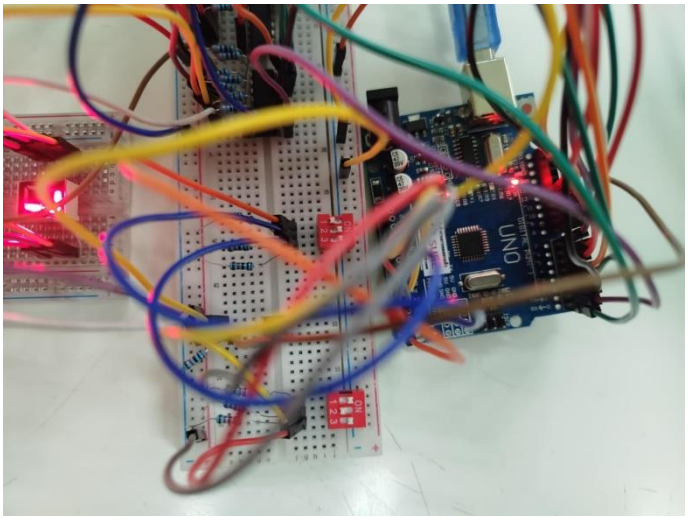
```
        pop r29             ; ดึงค่า r29 กลับจาก stack
        ret                 ; กลับไปยังจุดที่เรียกฟังก์ชันนี้
ShowTime3: ; แสดงผลเมื่อเหลือ 3 วินาที
        ldi r26, 0x30 ; โหลดค่า 0x30 (00110000) เข้า r26 (อาจเป็นรหัสไฟกระพริบ)
        or r26, r25 ; รวมค่า r26 กับ r25 (คะแนนเดิม) เพื่อแสดงผล
        out PORTD, r26 ; ส่งผลลัพธ์ไปยัง PortD
        rjmp OuterLoop ; กระโดดไปยังจุดต่อไป
ShowTime2: ; แสดงผลเมื่อเหลือ 2 วินาที
        ldi r26, 0x20 ; โหลดค่า 0x20 (00100000) เข้า r26
        or r26, r25 ; รวมค่า r26 กับ r25 เพื่อแสดงผล
        out PORTD, r26 ; ส่งผลลัพธ์ไปยัง PortD
        rjmp OuterLoop ; กระโดดไปยังจุดต่อไป
ShowTime1: ; แสดงผลเมื่อเหลือ 1 วินาที
        ldi r26, 0x10 ; โหลดค่า 0x10 (00010000) เข้า r26
        or r26, r25 ; รวมค่า r26 กับ r25 เพื่อแสดงผล
        out PORTD, r26 ; ส่งผลลัพธ์ไปยัง PortD
        rjmp OuterLoop ; กระโดดไปยังจุดต่อไป
ShowTime0: ; แสดงผลเมื่อหมดเวลา
        out PORTD, r25 ; ส่งค่า r25 (คะแนนเดิม) ไปยัง PortD โดยไม่เปลี่ยนแปลง
        rjmp OuterLoop ; กระโดดไปยังจุดต่อไป
```

รูปการทำงานของวงจรจริง

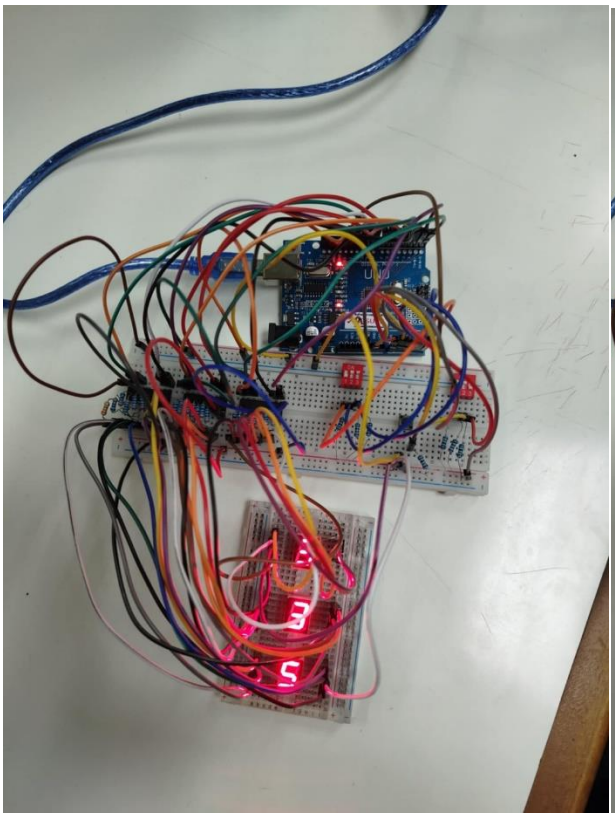
หลักการใช้งาน: 7-segment display ตัวบน = scoreboard player1, ตัวกลาง = ตัวหน่วงเวลาต่อรอบ, ตัวล่าง = scoreboard player2

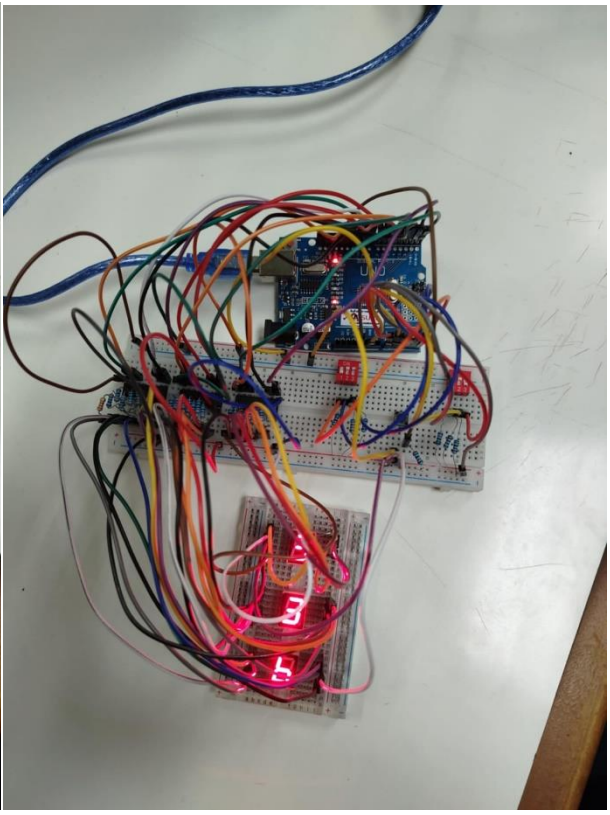
DIP Switch ตัวล่าง = ตัวเลือกของผู้เล่น1, DIP Switch ตัวบน = ตัวเลือกของผู้เล่น2, เลขแทนการบนDIP ได้แก่ 3 = ค้อน, 2 = กรรไกร, 1 = กระดาษ

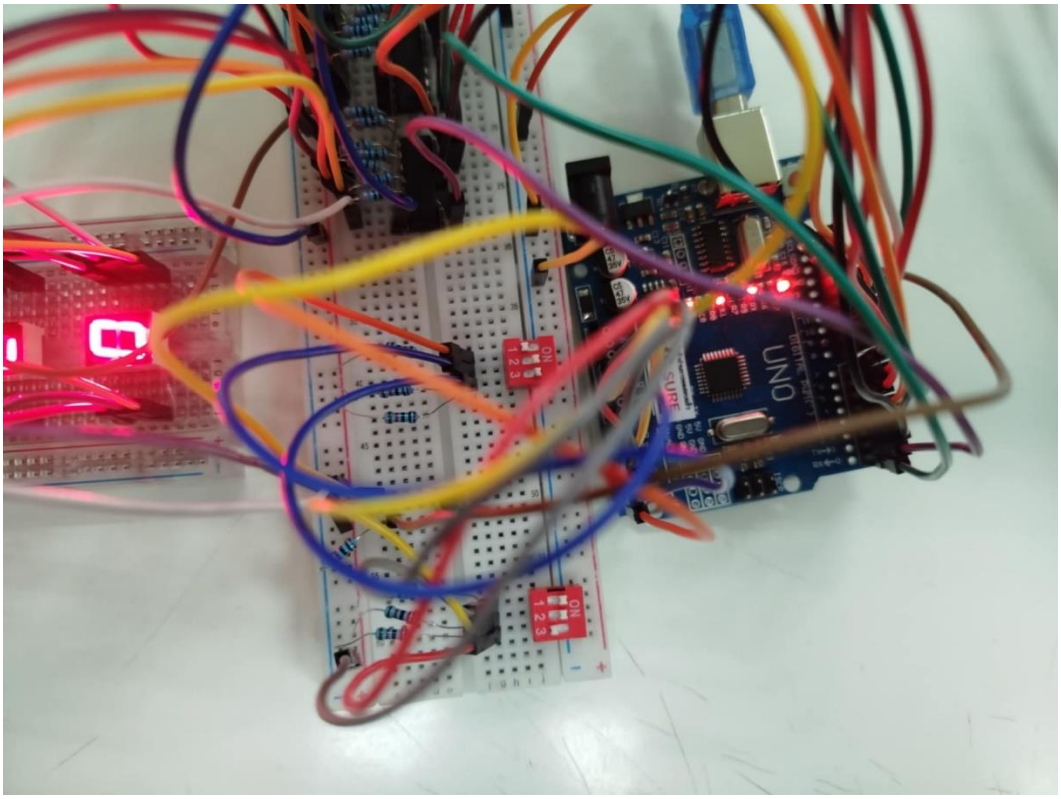




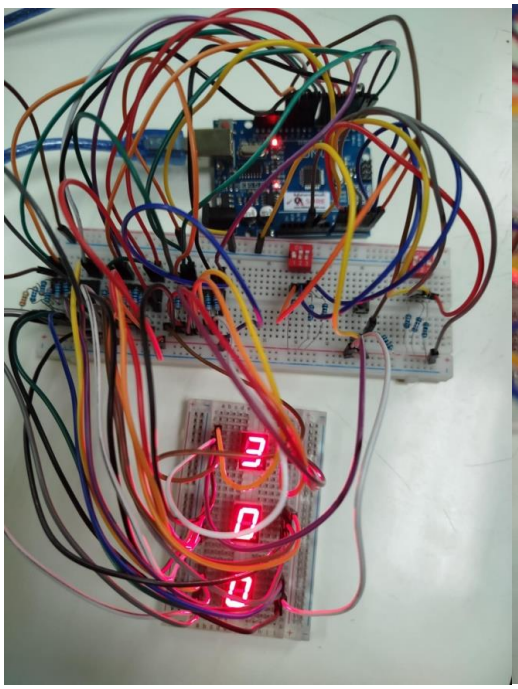
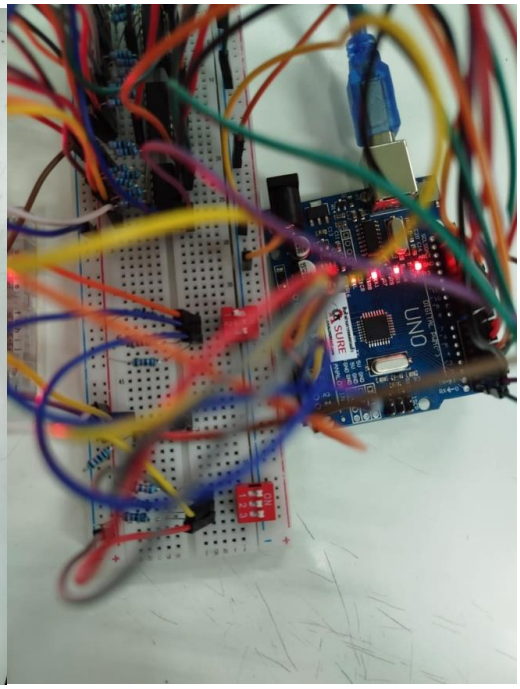
กรณี กรรไกร vs กระดาษ ผู้เล่นคนแรกชนะ 7-segment ตัวบนเพิ่มขึ้น



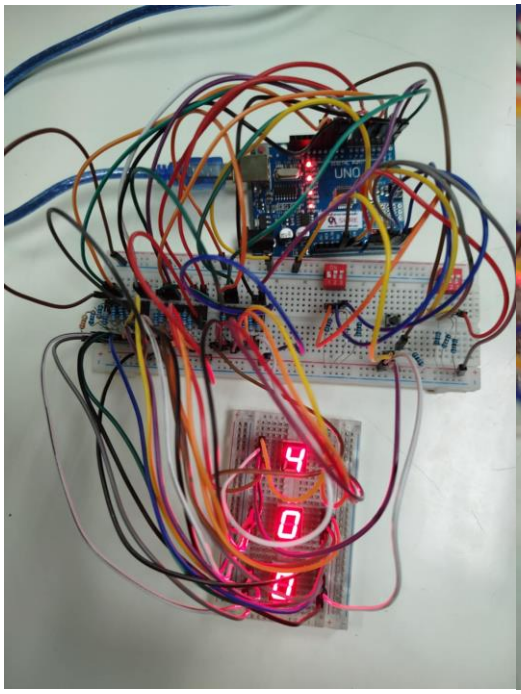
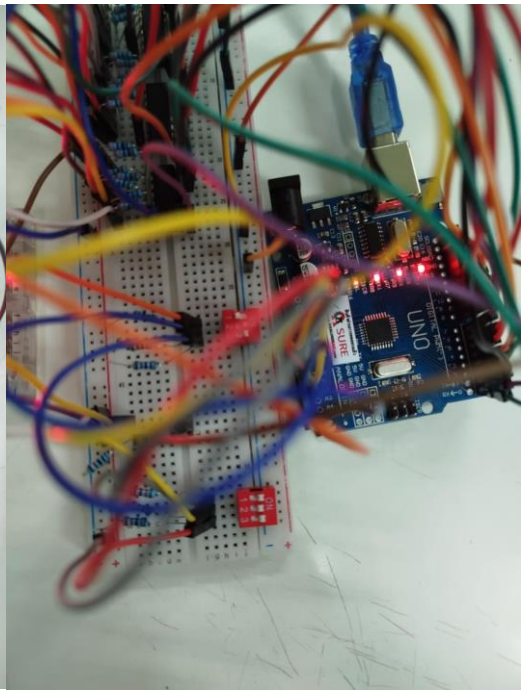




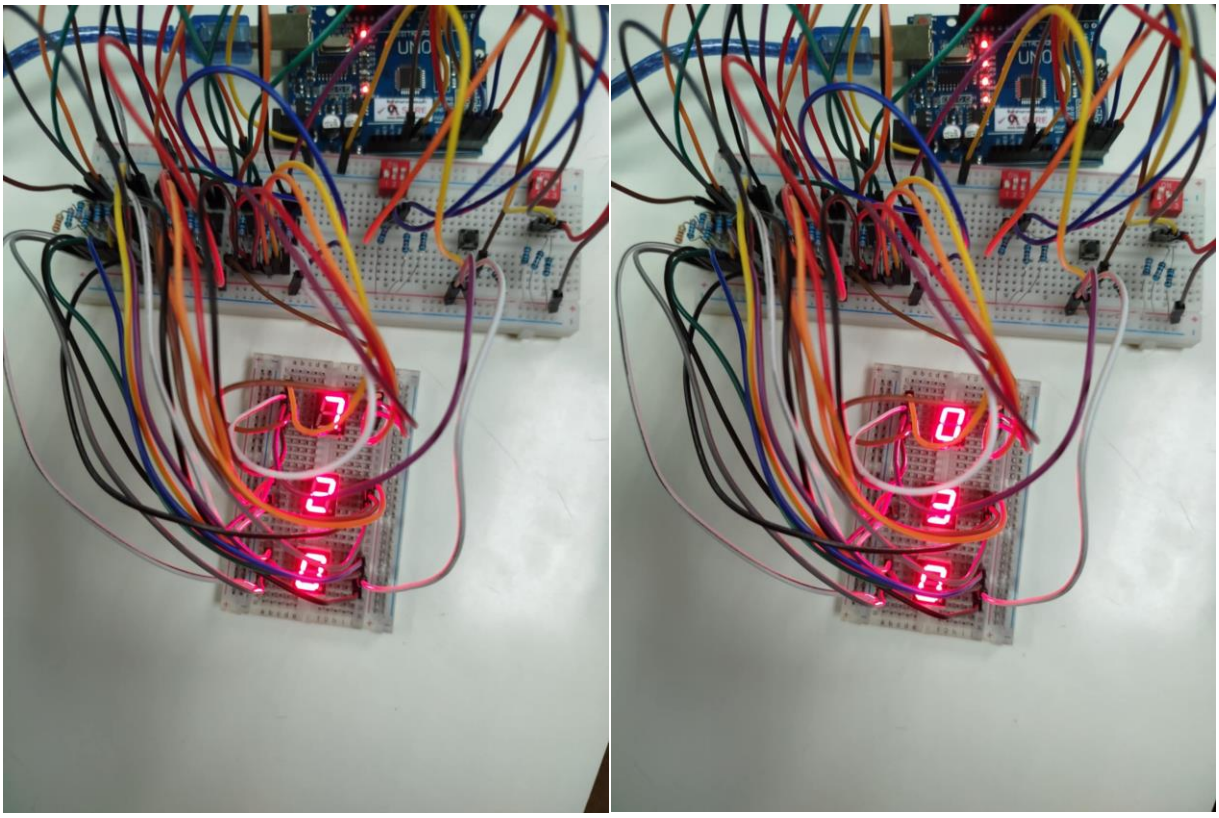
กรณี ค้อน vs กระดาษ ผู้เล่นคนสองชนะ 7-segment ตัวล่างเพิ่มขึ้น

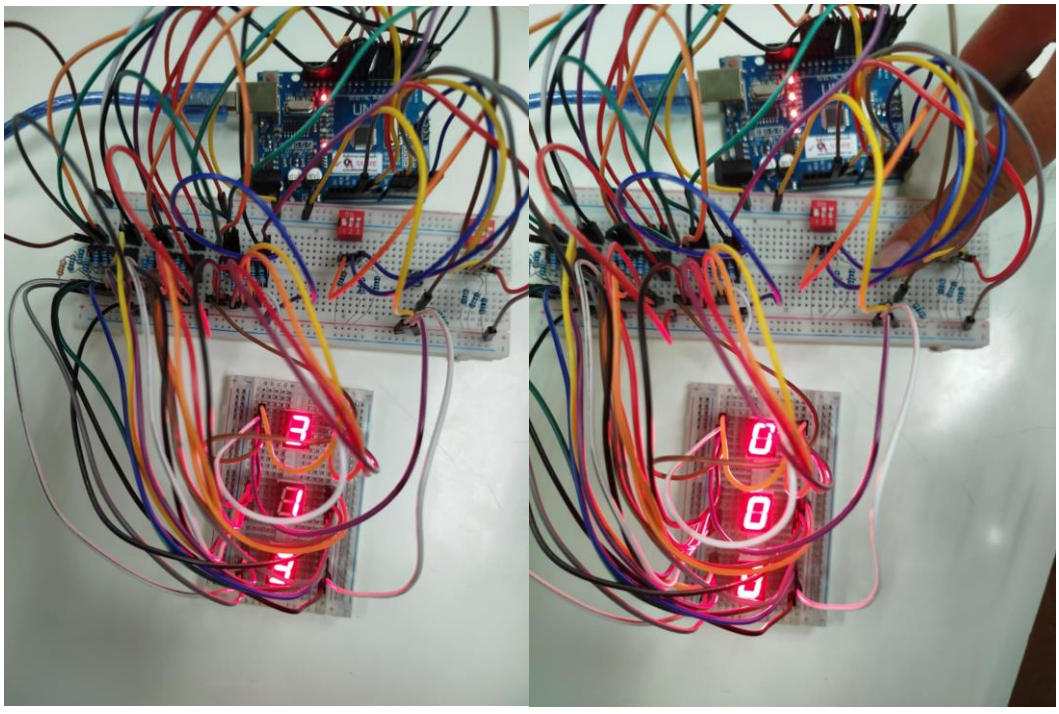
กรณีผู้เล่นเลือกมากกว่า1ตัวเลือก 7 segment ตัวกลาง จะนับถึง 0 และหยุดรอให้ผู้เล่นเลือกตัวเลือกให้ถูกต้อง

กรณีมีผู้เล่นไม่เลือกตัวเลือก 7 segment ตัวกลาง จะนับถึง 0 และหยุดรอให้ผู้เล่นเลือกตัวเลือกให้ถูกต้อง



7-segment display ที่แสดงคะแนนของผู้เล่นทั้งสอง จะรีเซ็ตเป็น 0



เมื่อกดปุ่ม Interrupt (PB4) คะแนนที่แสดงบน 7-segment display ทั้งสองฝั่งจะถูกรีเซ็ต เป็น0