ปฏิบัติการทดลองที่ 3

เสนอ

ผู้ช่วยศาตราจารย์.ดร.พนัส นัถฤทธิ์

จัดทำโดย

58364876นายอาทิตย์ แซ่ว่าง

58366450นายศิวศิษฎ์ สารขาว

รายงานเล่มนี้เป็นส่วนหนึ่งของรายวิชา 305281 ไมโครโพรเซสเซอร์และภาษาแอสเซมบลี

ภาคเรียนที่ 1/ปีการศึกษา 2560

ภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สาขาวิชาวิศกรรมคอมพิวเตอร์

มหาวิทยาลัยนเรศวร

คำนำ

รายงาน ปฏิบัติการทดลองที่ 2 ซึ่งเป็นส่วนหนึ่งของรายวิชา 305381 ไมโครโพรเซสเซอร์และ ภาษาแอสเซมบลี (Microprocessor and Assembly Language) จัดทำขึ้นเพื่อศึกษาการเขียนโปรแกรมภาษาแอสแซมบลีเพื่อควบคุมการทำงานของไมโครคอนโทรลเลอร์ MCS-51 ผ่าน LED

คณะผู้จัดทำหวังว่ารายงานเล่มนี้จะสามารถเป็นประโยชน์ต่อผู้ที่ต้องการศึกษาค้นคว้าข้อมูล ได้ตรงตาม วัตถุประสงค์ หากรายงานนี้มีข้อผิดพลาดประการใด ผู้จัดทำขออภัยมา ณ ที่นี้

คณะผู้จัดทำ

นายอาทิตย์ แซ่ว่าง

นายศิวศิษฎ์ สารขาว

บทนำ

ไมโครคอนโทรลเลอร์ (Microcontroller) หมายถึง ไมโครโปรเซสเซอร์ตัวหนึ่งที่มี หน่วยความจำชั่วคราว (RAM : Random Access Memory) หน่วยความจำถาวร (ROM : Read Only Memory) หน่วยรับ/ส่งข้อมูลจากภายนอก (Input/Output Port) อยู่ภายในตัวมันเพียงตัวเดียว (Single Chip) ซึ่งไมโครโปรเซสเซอร์ต้องอาศัยหน่วยความจำและหน่วยอินพุต/เอาต์พุตภายนอก ฉะนั้นไมโครคอนโทรลเลอร์จึงมีข้อดีกว่าไมโครโปรเซสเซอร์มาก ในเรื่องของความประหยัดและความสะดวกในการใช้งาน ปัจจุบันจึงนิยมใช้ไมโครคอนโทรลเลอร์ มากกว่าโดยเฉพาะในงานควบคุมทางด้านอุตสาหกรรมและเครื่องใช้ไฟฟ้าในชีวิตประจำวัน เช่น การควบคุมอุณหภูมิ การควบคุมหุ่นยนต์ เครื่องซักผ้าแบบโปรแกรมได้ การควบคุมการปิด-เปิดไฟฟ้าในอาคาร การควบคุมไฟวิ่ง เป็นต้น

ในปัจจุบันผู้ผลิตได้พัฒนาให้ไมโครคอนโทรลเลอร์สามารถทำงานได้เร็วขึ้นโดยเพิ่มความสามารถในการรองรับคริสตอลความถี่ที่สูงขึ้น รวมไปถึงการปรับปรุงการทำงานภายในให้ไมโครคอนโทรลเลอร์ใช้จำนวนสัญญาณนาฬิกาในการสร้างแมชชีนไซเคิลน้อยลง โดยในบางรุ่น 1 แมชชีนไซเคิลใช้สัญญาณนาฬิกาเพียงแค่ 1 ลูกเท่านั้น

ซึ่งการทดลองสำหรับวิชา 305381 ไมโครโพรเซสเซอร์และภาษาแอสเซมบลี (Microprocessor and Assembly Language) จะใช้ไมโครคอนโทรลเลอร์ตระกูลMCS51 ในการทดลอง ใช้ภาษา Assembly ในการโปรแกรมควบคุมไมโครคอนโลเลอร์ และใช้งานโปรแกรม Flash Magic สำหรับดาวน์โหลดโปรแกรมลงบอร์ดไมโครคอนโทรลเลอร์ การทดลองนี้เป็นการทดลอง เกี่ยวกับการใช้งานเบื้องต้น สำหรับใช้ไมโครคอนโทรลเลอร์ตระกูล MCS51 89V51RD2 เป็นการทดลองการใช้พอร์ท P0-P3 เป็นอินพุต - เอาท์พุตพอร์ท โดยการใช้ภาษาแอสเซมบลีในการเขียนโปรแกรมคำสั่งในการทดลองต่างๆ และเรียนรู้การดาวน์โหลดโปรแกรมที่ผู้ทดลองเขียนขึ้นลงในตัวไมโครคอนโทรลเลอร์ MCS51 89V51RD2

วัตถุประสงค์ของการทดลอง

1. เพื่อเรียนรู้การทำงานของ Microcontroller MCS-51
2. เพื่อเรียนรู้วิธีการใช้งานโปรแกรม Flash Magic สำหรับดาวน์โหลดโปรแกรมลงบอร์ดไมโครคอนโทรลเลอร์
3. เพื่อศึกษาการเขียนโปรแกรม rotate
4. เพื่อศึกษาการเขียนโปรแกรม Delay

เนื้อหาและทฤษฎี

ไมโครคอนโทรลเลอร์ MCS-51

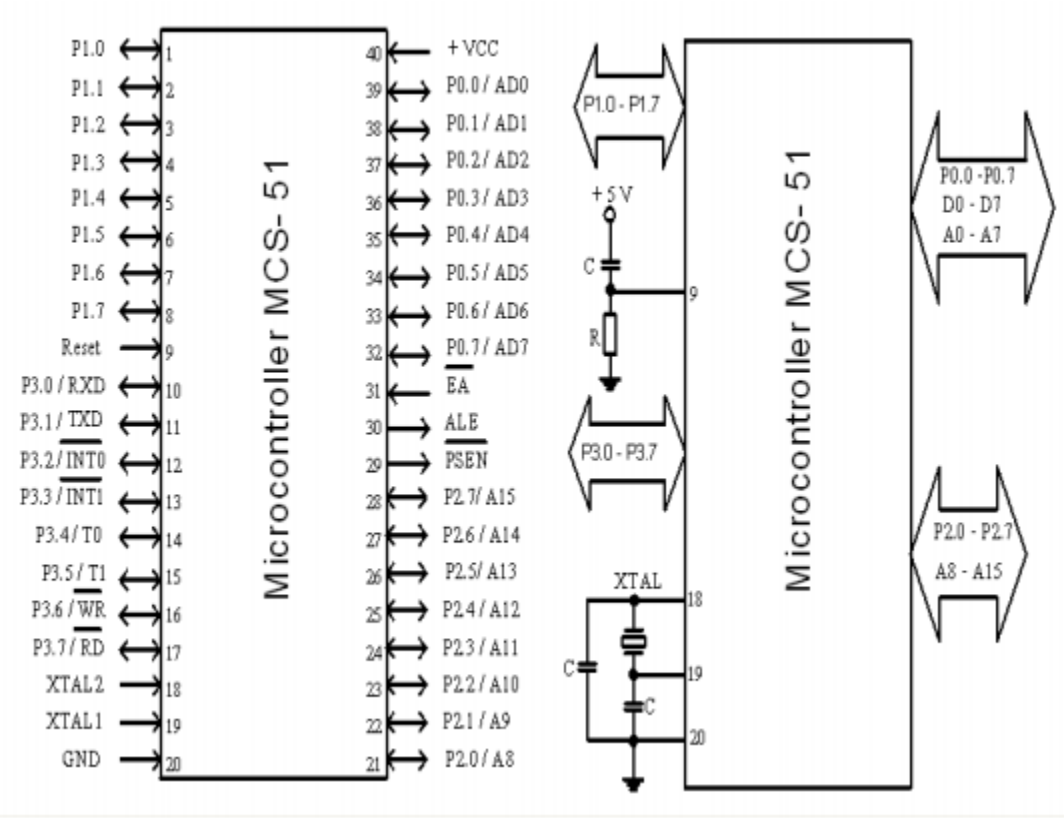
มีไทเมอร์/เคาน์เตอร์ขนาด 16 บิต 2 ตัว คือไทเมอร์ 0 และไทเมอร์ 1 ส่วนในไมโครคอนโทรลเลอร์อนุกรม 8052 ซึ่งออกมาทีหลังจะมีไทเมอร์/เคาน์เตอร์ 3 ตัว นั่นคือมีไทเมอร์ 2 เพิ่มขึ้นมา ไทเมอร์/เคาน์เตอร์แต่ละตัวสามารถเลือกใช้งานเป็นไทเมอร์ หรือเคาน์เตอร์ก็ได้ และทำงานได้อย่างเป็นอิสระต่อกัน ในการทำงานเป็นไทเมอร์นั้นจะใช้หลักการเพิ่มค่ารีจิสเตอร์ไทเมอร์ทุก ๆ Machine Cycle ซึ่งมีช่วงเท่ากับ 12 คาบสัญญาณนาฬิกาที่ถูกสร้างขึ้นจากคริสตอลที่ต่อใช้งานให้กับไมโครคอนโทรลเลอร์นั่นเอง สำหรับบทความนี้จะอธิบายรายละเอียด และตัวอย่างการใช้งานไทเมอร์ 0 และ 1 เท่านั้นนะครับ สำหรับไทเมอร์ 2 สามารถอ่านรายละเอียดได้ที่บทความการใช้พอร์ตอนุกรมของไมโครคอนโทรลเลอร์ MCS-51

โครงสร้างพื้นฐานที่สำคัญของไมโครคอนโทรลเลอร์ตระกูล 8051 ซึ่งมีรายละเอียดดังนี้

1. เป็นไมโครคอนโทรลเลอร์ที่มีหน่วยประมวลผลกลางแบบ 8 บิต
2. มีคำสั่งคำนวณทางคณิตศาสตร์ และตรรกศาสตร์ (Boolean processor)
3. มีแอดเดรสบัสขนาด 16 บิตทำให้สามารถอ้างตำแหน่งหน่วยความจำโปรแกรม และหน่วยความจำข้อมูลได้ 64 กิโลไบต์
4. มีหน่วยความจำ (RAM) ภายในขนาด 128 ไบต์ (8051/8031) หรือ 256 ไบต์ (8052/8032)
5. มีพอร์ตอนุกรมทำงานแบบดูเพล็กซ์เต็ม (Full Duplex) 1 พอร์ต
6. มีพอร์ตอินพุต/เอาต์พุตแบบขนานจำนวน 32 บิต
7. มีไทเมอร์ 2 ตัว (8051/8031) หรือ 3 ตัว (8052/8032)

8. มีวงจรควบคุมการเกิดอินเตอร์รัปต์ 5 ประเภท (8051/8031) หรือ 6 ประเภท (8052/8032)

9. มีวงจรออสซิลเลเตอร์ภายในตัว



ไมโครคอนโทรลเลอร์ MCS-51

ประกอบด้วยพอร์ตที่ใช้เป็นอินพุต/เอาต์พุตเพื่อติดต่อกับอุปกรณ์รอบนอกได้ 4 พอร์ต คือP0, P1, P2 และ P3 มีรายละเอียดแต่ละพอร์ตดังนี้

-P0 (P0.0-P0.7) เป็นอินพุตหรือเอาต์พุตพอร์ตถ้ามีการขยายหน่วยความจำภายนอกหรืออินพุต/ เอาต์พุตพอร์ตภายนอกจะใช้เป็น Data Bus (D0-D7) และ Address Bus (A0-A7)

-P1 (P1.0-P1.7) เป็นอินพุตหรือเอาต์พุตพอร์ต

-P2 (P2.0-P2.7) เป็นอินพุตหรือเอาต์พุตพอร์ต ถ้ามีการขยายหน่วยความจำภายนอกจะใช้เป็น Address Bus (A8-A15)

-P3 (P3.0-P3.7) เป็นอินพุตหรือเอาต์พุตพอร์ต ถ้าไม่ใช้เป็นอินพุต /เอาต์พุตพอร์ตก็สามารถทำหน้าที่ตามชื่อหลังได้ดังนี้

-P3.0/RxD (Receive Data) ใช้เป็นขาอินพุตสำหรับรับข้อมูลจากการสื่อสารแบบอนุกรม

-P3.1/TxD (Transmit Data) ใช้เป็นขาเอาต์พุตสำหรับส่งข้อมูลจากการสื่อสารแบบอนุกรม
-P3.2/ INT0 (Interrupt 0) รับสัญญาณขัดจังหวะจากภายนอก No. 0
-P3.3/ INT1 (Interrupt 1) รับสัญญาณขัดจังหวะจากภายนอก No. 1
-P3.4/T0 (Timer/Counter0) สามารถโปรแกรมได้ว่าจะให้เป็น Timer หรือ Counter ถ้าใช้สัญญาณ Clock จากภายนอกเข้ามาจะเป็น Counter ถ้าใช้สัญญาณ Clock จากภายในจะเป็น Timer
-P3.5/T1 (Timer/Counter1) ทำหน้าที่ทำนองเดียวกับ P3.4/T0
-P3.6/ WR (Write) ส่งสัญญาณควบคุมการเขียนข้อมูลจาก MCS-51ไปยังภายนอก
-P3.7/ RD (Read) ส่งสัญญาณควบคุมการอ่านข้อมูลจากภายนอกเข้ามายัง MCS-51 -Reset เป็นขาอินพุต

การใช้งานโปรแกรม Flash Magic สำหรับดาวน์โหลดโปรแกรมลงบอร์ดไมโครคอนโทรลเลอร์

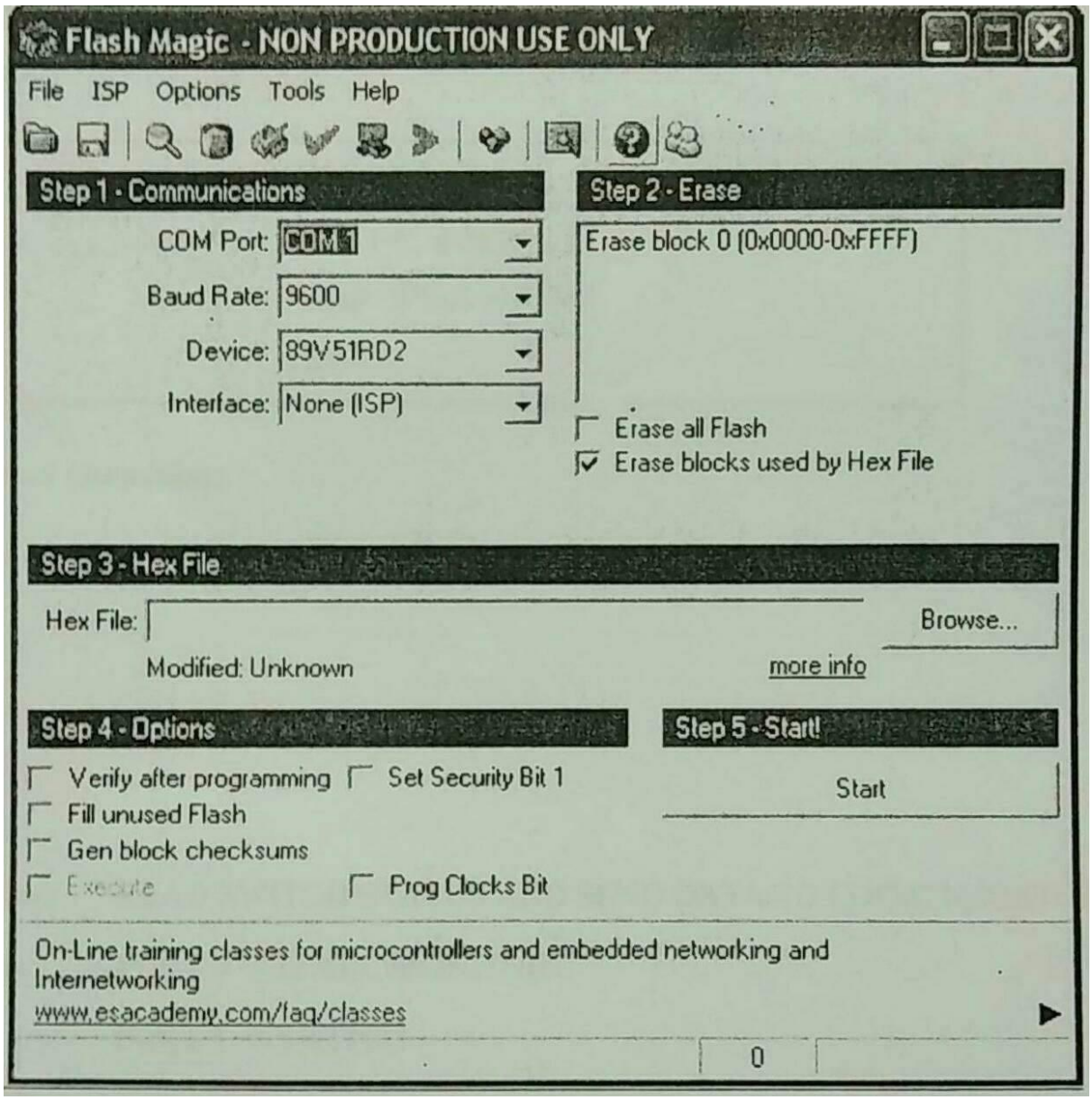
ในการเขียนโปรแกรมภาษาแอสเซมบลีเพื่อควบคุมการทำงานของไมโครคอนโทรลเลอร์นั้นจะต้องมีการคอมไฟล์โปรแกรมดังกล่าวให้เป็น Hex File แล้วจึงทำการดาวน์โหลดไฟล์นั้นลงบอร์ดไมโครคอนโทรลเลอร์ โดยในการทดลองนี้จะเลือกใช้งานโปรแกรม Flash Magic สำหรับดาวน์โหลดโปรแกรมโดยนิสิตจำเป็นต้องเลือก Option ให้สอดคล้องกับบอร์ดทดลองที่ใช้งานดังนี้

1. เลือกเมนู Options >> Advanced Options … จะปรากฏหน้าต่าง Advanced Options
2. คลิกที่ TAB ชื่อ Hardware Config แล้วทำการยกเลิกเครื่องหมายถูกที่ปรากฏในช่อง Use DTR to control RST

เมื่อปรับเลือกค่า Option ให้เหมาะสมต่อการใช้งานแล้ว โปรแกรม Flash Magic จะพร้อมสำหรับการโหลดโปรแกรมลงชิปต่อไป โดยมีขั้นตอนดังนี้

- **STEP 1:** เลือกเบอร์ไมโครคอนทรลเลอร์เป็น 89V51RD2 จากนั้นให้กำหนดพอร์ตสื่อสารข้อมูล COM-x ให้ตรงกับพอร์ตสื่อสารของคอมพิวเตอร์ที่ใช้งานอยู่ จากนั้นให้กำหนดค่าความเร็วที่ใช้สื่อสารข้อมูล (ในที่นี้กำหนดให้เป็น 9600 bps ดังแสดงในรูปที่ 1)

- **STEP 2:** เลือกรูปแบบของการลบข้อมูลภายในหน่อยความจำโปรแกรม (Flash Memory) ก่อนที่จะดำเนินการโปรแกรมข้อมูลใหม่ลงไป โดยกำหนดเป็น Erase block used Hex File ซึ่งเป็นการลบข้อมูลเฉพาะ block ที่ต้องการสำหรับการเขียนโปรแกรมข้อมูลใหม่เท่านั้น(โดยจะส่งผลให้การทำงานเร็วกว่าการลบข้อมูลทั้งหมดด้วยคำสั่ง Erase all Flash)
- **STEP 3:** คลิกปุ่ม browse ... เพื่อเปิดหน้าต่าง Select Hex File และเลือกไฟล์ที่ต้องการโปรแกรมลงสู่ไมโครคอนโทรลเลอร์



รูปที่ 1: หน้าต่างของโปรแกรม Flash Magic

- **STEP 4:** เลือก Options การทำงานเพิ่มเติมตามต้องการ
- **STEP 5:** กดปุ่ม Start เพื่อเริ่มขั้นตอนการโปรแกรมลงชิปไมโครคอนโทรลเลอร์

เมื่อปรากฏหน้าต่าง Reset Device ขึ้นมาแล้ว ให้กดปุ่ม RESET บนบอร์ดไมโครคอนโทรลเลอร์ ซึ่งจะเป็นการเริ่มต้นการดาวน์โหลดโปรแกรมลงสู่ชิปทันที โดยจะสามารถสังเกตขั้นตอนการทำงานได้จาก Status bar ที่ขอบด้านล่างของโปรแกรมและเมื่อขั้นตอนการโปรแกรมเสร็จสมบูรณ์ (Finished) ก็ให้กดปุ่ม RESET บนบอร์ดไมโครคอนโทรลเลอร์อีกครั้ง ไมโครคอนโทรลเลอร์จะเริ่มต้นทำงานตามโปรแกรมที่ได้ดาวน์โหลดลงไปใหม่ทันที

การทดลองในรายวิชานี้มีวัตถุประสงค์เพื่อเรียนรู้วิธีการเขียนโปรแกรมภาษาแอสแซมบลีสำหรับใช้ควบคุมการทำงานของไมโครคอนโทรเลอร์ตระกูล MCS-51 โดยเนื้อหาในการทดลองจะประกอบด้วยการทดลองจำนวน 5 การทดลอง ดังนี้

รวมคำสั่ง JUMP แบบมีเงื่อนไขจาก CMP

1. JE : กระโดดงถ้าค่าเท่ากัน
2. JZ : กระโดดถ้าค่าเป็นศูนย์
3. JNE : กระโดดถ้าค่าไม่เท่ากัน
4. JNZ : กระโดดถ้าค่าไม่เป็นศูนย์
5. JA : กระโดดถ้าค่าเหนือกว่า
6. JNBE : กระโดดถ้าค่าไม่ต่ำกว่าหรือเท่ากัน
7. JAE : กระโดดถ้าค่าเหนือกว่า หรือเท่ากัน
8. JNB : กระโดดถ้าค่าไม่ต่ำกว่า
9. JB : กระโดดถ้าค่าต่ำกว่า
10. JNAE : กระโดดถ้าค่าไม่เหนือกว่า หรือเท่ากัน
11. JBE : กระโดดถ้าค่าต่ำกว่าหรือเท่ากัน
12. JNA : กระโดดถ้าค่าไม่เหนือกว่า
13. JG : กระโดดถ้าค่ามากกว่า

14. JNLE : กระโดดถ้าค่าไม่น้อยกว่าหรือเท่ากับ
15. JGE : กระโดดถ้าค่ามากกว่าหรือเท่ากัน
16. JNL : กระโดดถ้าค่าไม่น้อยกว่า
17. JL : กระโดดถ้าค่าน้อยกว่า
18. JNGE : กระโดดถ้าค่าไม่มากกว่า หรือเท่ากัน
19. JLE : กระโดดถ้าค่าน้อยกว่า หรือเท่ากัน
20. JNG : กระโดดถ้าค่าไม่มากกว่า

MICROCONTROLLER MCS-51 - LAB 3

Summary: CONDITIONAL JUMP INSTRUCTION TO BRANCHING PROGRAM

LAB 3-1

Description: SW1 CONTROL BLINK LED BIT 0 OF LOGIC MONITOR

Hardware: PORT 0 -> LOGIC MONITOR

PORT 1 -> SWITCH

ASM Code:

```
        ORG     0000H
        LOGIC_DATA EQU P0
        SWITCH  EQU P1
        SW1     EQU P1.7
START:      MOV     LOGIC_DATA, #00H
        MOV     SWITCH, #0FFH
CHK:        JNB SW1, CHK
        MOV     LOGIC_DATA, #01H
        LCALL   DELAY
        MOV     LOGIC_DATA, #00H
        LCALL   DELAY
        SJMP    CHK

DELAY:      MOV     R5, #5
DELAY0: MOV     R6, #255
DELAY1: MOV     R6, #255
DELAY2: DJNZ    R7, DELAY2
        DJNZ    R6, DELAY1
        DJNZ    R5, DELAY0
        RET
        END
```

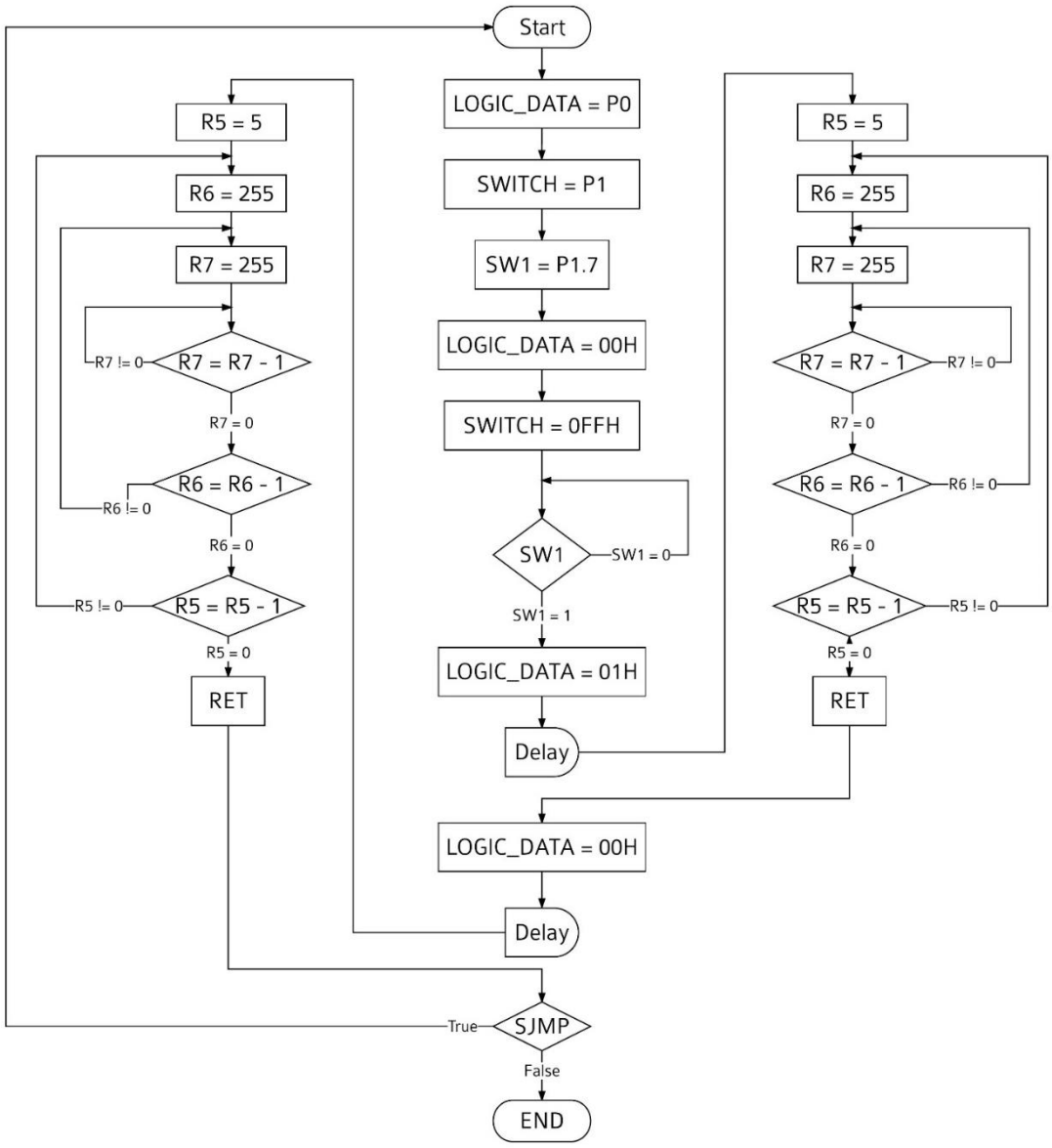
วิธีการทดลอง

1. เขียนโค้ดที่โจทย์กำหนดให้ จากนั้นทำการแปลงไฟล์ให้ได้ไฟล์นามสกุล .hex

2. ทำการรัน ASM Code จากโปรแกรม Flash Magic เพื่อดาวน์โหลดโค้ดลงบอร์ดไมโครคอนโทรลเลอร์ ผ่านสายสื่อข้อมูล

3. สังเกตผลการทดลองและบันทึกผลการทดลอง

Flow Chart:


Start
LOGIC_DATA = P0
SWITCH = P1
SW1 = P1.7
LOGIC_DATA = 00H
SWITCH = 0FFH
SW1
SW1 = 0
SW1 = 1
LOGIC_DATA = 01H
Delay
LOGIC_DATA = 00H
Delay
SJMP
True
False
END
R5 = 5
R6 = 255
R7 = 255
R7 = R7 - 1
R7 != 0
R7 = 0
R6 = R6 - 1
R6 != 0
R6 = 0
R5 = R5 - 1
R5 != 0
R5 = 0
RET
R5 = 5
R6 = 255
R7 = 255
R7 = R7 - 1
R7 != 0
R7 = 0
R6 = R6 - 1
R6 != 0
R6 = 0
R5 = R5 - 1
R5 != 0
R5 = 0
RET

การทำงานของโปรแกรม :

ORG 0000H	เริ่มต้นการทำงานที่ตำแหน่ง 0000H
LOGIC_DATA EQU P0	ประกาศตัวแปร LOGIC_DATA สำหรับเรียกแทนพอร์ต 0
SWITCH EQU P1	ประกาศตัวแปร SWITCH สำหรับเรียกแทนพอร์ต 1
SW1 EQU P1.7	ประกาศตัวแปร SW1 สำหรับเรียกแทนพอร์ต 1 บิตที่ 7
START: MOV LOGIC_DATA, #00H	เริ่มโปรแกรมย่อย ชื่อ start กำหนดให้ LOGIC_DATA = 0 เพื่อใช้เป็นตัวชี้ข้อมูล
MOV SWITCH, #0FFH	กำหนดให้ SWITCH = FFH
CHK: JNB SW1, CHK	เริ่มโปรแกรมย่อย ชื่อ CHK ถ้า SW1 ไม่ถูกกด ให้กลับไปที่ CHK
MOV LOGIC_DATA, #01H	กำหนดให้ LOGIC_DATA = 1 เพื่อใช้เป็นตัวชี้ข้อมูล
LCALL DELAY	เรียกใช้โปรแกรมย่อยเพื่อหน่วงเวลา
MOV LOGIC_DATA, #00H	กำหนดให้ LOGIC_DATA = 0 เพื่อใช้เป็นตัวชี้ข้อมูล
LCALL DELAY	เรียกใช้โปรแกรมย่อยเพื่อหน่วงเวลา
SJMP CHK	กระโดดไปที่คำสั่ง CHK
DELAY: MOV R5, #5	โปรแกรมย่อยเพื่อหน่วงเวลา กำหนดให้ R5 = 5
DELAY0: MOV R6, #255	โปรแกรมย่อยเพื่อหน่วงเวลา 0 กำหนดให้ R6 = 255
DELAY1: MOV R7, #255	โปรแกรมย่อยเพื่อหน่วงเวลา 1 กำหนดให้ R7 = 255
DELAY2: DJNZ R7, DELAY2	โปรแกรมย่อยเพื่อหน่วงเวลา 2 ถ้า R7 ไม่เท่ากับ 0 ให้ไปที่ DELAY2
DJNZ R6, DELAY1	ถ้า R6 ไม่เท่ากับ 0 ให้ไปที่ DELAY1

DJNZ R5, DELAY0	ถ้า R5 ไม่เท่ากับ 0 ให้ไปที่ DELAY0
RET	สิ้นสุดโปรแกรมหน่วงเวลา
END	จบการทำงาน

Results and discussion:

ถ้ายังไม่ได้สับ SW1 หรือมีลอจิกเป็น 0 LED ทั้ง 8 ดวงจะติดเป็นสีเขียว

แต่เมื่อสับ SW1 ให้มีลอจิกเป็น 1 LED0 จะติด-ดับ สลับเป็นสีเขียวและสีแดงไปเรื่อย ๆ

ส่วน LED1 - LED7 จะติดเป็นสีเขียว

LAB 3-2

Description: MULTI-PATTERN MOVING DISPLAY CONTROLLED VIA 2 BIT SWITCH

Hardware: PORT 0 -> LOGIC MONITOR

PORT 1 -> SWITCH

ASM Code:

```
            ORG     0000H
            LOGIC_DATA  EQU P0
            SWITCH  EQU P1
            SW1     EQU P1.7
            SW2     EQU P1.6
START:      MOV     SWITCH, #0FFH
CHK_SW:     JB      SW1, CHK_SW2
            JB      SW2, TURN_LEFT
            MOV     LOGIC_DATA, #00H
            SJMP    CHK_SW

CHK_SW2:    JB      SW2, BLINK_DISP
TURN_RIGHT: LCALL   DISP_L2R
            SJMP    CHK_SW

TURN_LEFT:  LCALL   DISP_R2L
            SJMP    CHK_SW

BLINK_DISP: MOV     LOGIC_DATA, #55H
            LCALL   DELAY
            MOV     LOGIC_DATA, #55H
            LCALL   DELAY
            SJMP    CHK_SW

DISP_L2R:   MOV     R0, #08H
            MOV     A, #80H
LOOP_L2R:   MOV     LOGIC_DATA, A
            RR  A
            LCALL   DELAY
            DJNZ    R0, LOOP_L2R
            RET
DISP_R2L:   MOV     R0, #08H
            MOV     A, #01H
LOOP_R2L:   MOV     LOGIC_DATA, A
```

```
DELAY:          MOV       R6, #255
DELAY1:  MOV       R7, #255
DELAY2:  DJNZ      R7, DELAY2
         DJNZ      R6, DELAY1
         RET
         END
```

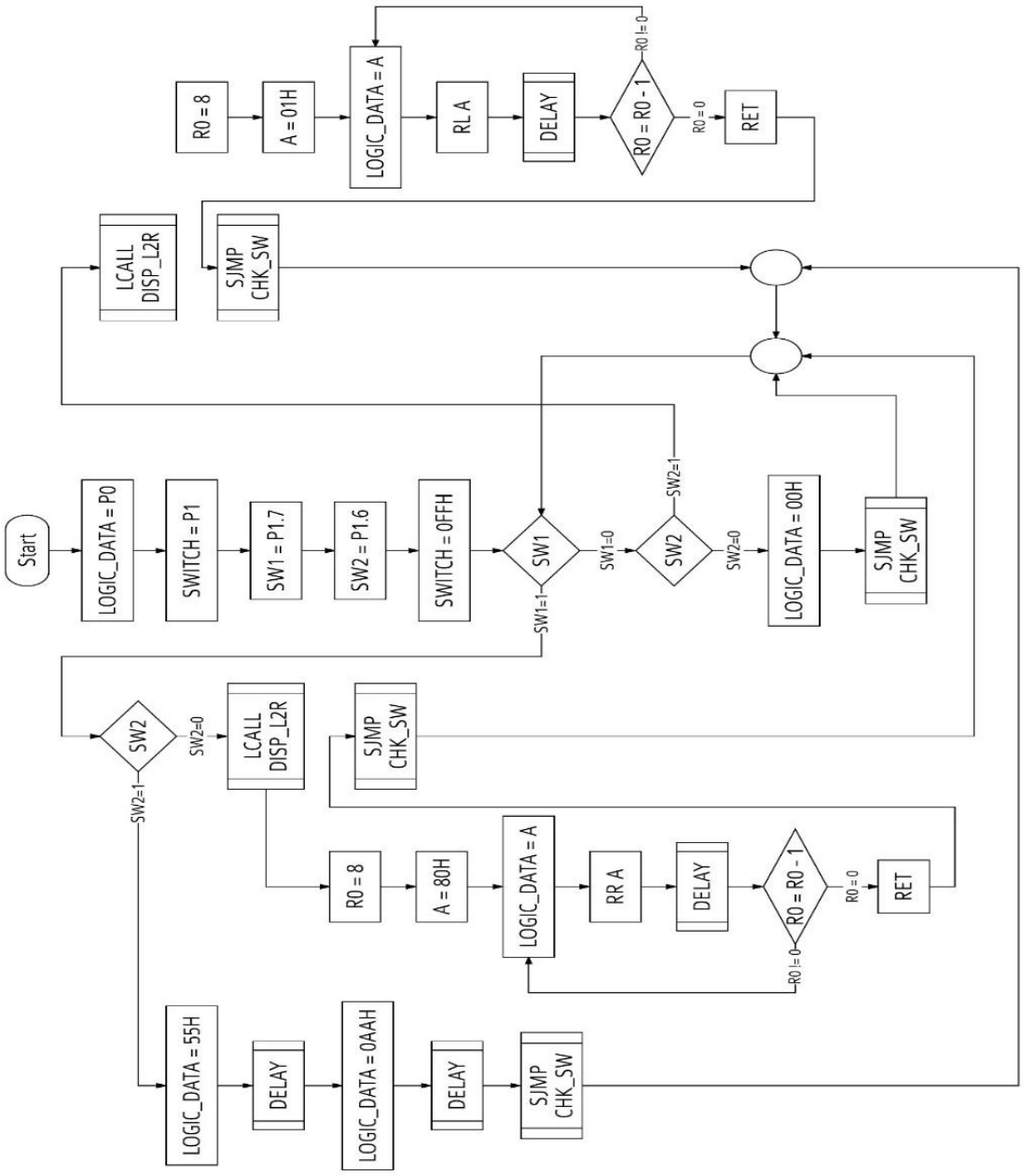
วิธีการทดลอง

1. เขียนโค้ดที่โจทย์กำหนดให้ จากนั้นทำการแปลงไฟล์ให้ได้ไฟล์นามสกุล .hex

2. ทำการรัน ASM Code จากโปรแกรม Flash Magic เพื่อดาวน์โหลดโค้ดลงบอร์ดไมโครคอนโทรลเลอร์ ผ่านสายสื่อข้อมูล

3. สังเกตผลการทดลองและบันทึกผลการทดลอง

Flow Chart:


Start
LOGIC_DATA = P0
SWITCH = P1
SW1 = P1.7
SW2 = P1.6
SWITCH = 0FFH
SW1
SW1=1
SW1=0
SW2
SW2=1
SW2=0
LOGIC_DATA = 00H
SJMP CHK_SW
SW2
SW2=0
SW2=1
LCALL DISP_L2R
SJMP CHK_SW
R0 = 8
A = 80H
LOGIC_DATA = A
RR A
DELAY
R0 = R0 - 1
R0 != 0
R0 = 0
RET
LOGIC_DATA = 55H
DELAY
LOGIC_DATA = 0AAH
DELAY
SJMP CHK_SW
LCALL DISP_L2R
SJMP CHK_SW
R0 = 8
A = 01H
LOGIC_DATA = A
RL A
DELAY
R0 = R0 - 1
R0 != 0
R0 = 0
RET

การทำงานของโปรแกรม :

ORG 0000H	เริ่มต้นการทำงานที่ตำแหน่ง 0000H
LOGIC_DATA EQU P0	ประกาศตัวแปร LOGIC_DATA สำหรับเรียกแทนพอร์ต 0
SWITCH EQU P1	ประกาศตัวแปร SWITCH สำหรับเรียกแทนพอร์ต 1
SW1 EQU P1.7	ประกาศตัวแปร SW1 สำหรับเรียกแทนพอร์ต 1 บิตที่ 7
SW2 EQU P1.6	ประกาศตัวแปร SW2 สำหรับเรียกแทนพอร์ต 1 บิตที่ 6
START: MOV LOGIC_DATA, #FFH	เริ่มโปรแกรมย่อย ชื่อ start กำหนดให้ LOGIC_DATA = FFH เพื่อใช้เป็นตัวชี้ข้อมูล
MOV SWITCH, #0FFH	กำหนดให้ SWITCH = FFH
CHK_SW: JB SW1, CHK_SW2	เริ่มโปรแกรมย่อย ชื่อ CHK_SW ถ้า SW1 ถูกกด ให้ไปที่ CHK_SW2
JB SW2, TURN_LEFT	ถ้า SW2 ถูกกด ให้ไปที่ TURN_LEFT
MOV LOGIC_DATA, #00H	กำหนดให้ LOGIC_DATA = 00H เพื่อใช้เป็นตัวชี้ข้อมูล
SJMP CHK_SW	กระโดดไปที่คำสั่ง CHK_SW
CHK_SW2: JB SW2, BLINK_DISP	เรียกใช้โปรแกรมย่อยชื่อ CHK_SW2 ถ้า SW2 ถูกกด ให้ไปที่ BLINK_DISP
TURN_RIGHT: LCALL DISP_R2L	เรียกใช้โปรแกรมย่อยชื่อ TURN_RIGHT เรียกใช้โปรแกรมย่อยชื่อ DISP_R2L
SJMP CHK_SW	กระโดดไปที่คำสั่ง CHK_SW
TURN_LEFT: LCALL DISP_R2L	เรียกใช้โปรแกรมย่อยชื่อ TURN_LEFT

	เรียกใช้โปรแกรมย่อยชื่อ DISP_L2R
SJMP CHK_SW	กระโดดไปที่คำสั่ง CHK_SW
BLINK_DISP: MOV LOGIC_DATA, #55H	เรียกใช้โปรแกรมย่อยชื่อ BLINK_DISP กำหนดให้ LOGIC_DATA = 55H เพื่อใช้เป็นตัวชี้ข้อมูล
LCALL DELAY	เรียกใช้โปรแกรมย่อยเพื่อหน่วงเวลา
MOV LOGIC_DATA, #0AAH	กำหนดให้ LOGIC_DATA = AAH เพื่อใช้เป็นตัวชี้ข้อมูล
LCALL DELAY	เรียกใช้โปรแกรมย่อยเพื่อหน่วงเวลา
SJMP CHK_SW	กระโดดไปที่คำสั่ง CHK_SW
DISP_L2R: MOV R0, #08H	เรียกใช้โปรแกรมย่อยชื่อ DISP_L2R กำหนดให้ R0 = 08H เพื่อใช้เป็นตัวชี้ข้อมูล
MOV A, #80H	กำหนดให้ A = 80H เพื่อใช้เป็นตัวชี้ข้อมูล
LOOP_L2R: MOV LOGIC_DATA, A	เรียกใช้ LOOP_L2R กำหนดให้ LOGIC_DATA = A เพื่อใช้เป็นตัวชี้ข้อมูล
RR A	เลื่อน A ไปทางขวา
LCALL DELAY	เรียกใช้โปรแกรมย่อยเพื่อหน่วงเวลา
DJNZ R0, LOOP_L2R	ถ้า R0 ไม่เท่ากับ 0 ให้ไปที่ LOOP_L2R
RET	สิ้นสุดโปรแกรมหน่วงเวลา
DISP_R2L: MOV R0, #08H	เรียกใช้โปรแกรมย่อยชื่อ DISP_R2L กำหนดให้ R0 = 08H เพื่อใช้เป็นตัวชี้ข้อมูล
MOV A, #01H	กำหนดให้ A = 01H เพื่อใช้เป็นตัวชี้ข้อมูล
LOOP_R2L: MOV LOGIC_DATA, A	เรียกใช้ LOOP_R2L กำหนดให้ LOGIC_DATA = A เพื่อใช้เป็นตัวชี้ข้อมูล

RL A	เลื่อน A ไปทางซ้าย
LCALL DELAY	เรียกใช้โปรแกรมย่อยเพื่อหน่วงเวลา
DJNZ R0, LOOP_L2R	ถ้า R0 ไม่เท่ากับ 0 ให้ไปที่ LOOP_R2L
RET	สิ้นสุดโปรแกรมหน่วงเวลา
DELAY0: MOV R6, #255	โปรแกรมย่อยเพื่อหน่วงเวลา 0 กำหนดให้ R6 = 255
DELAY1: MOV R7, #255	โปรแกรมย่อยเพื่อหน่วงเวลา 1 กำหนดให้ R7 = 555
DELAY2: DJNZ R7, DELAY2	โปรแกรมย่อยเพื่อหน่วงเวลา 2
DJNZ R6, DELAY1	
RET	สิ้นสุดโปรแกรมหน่วงเวลา
END	จบการทำงาน

Results and discussion:

ถ้า SW1 และ SW2 มีลอจิกเป็น 0 LED0 - LED7 จะติดเป็นสีเขียว

ถ้า SW1 มีลอจิกเป็น 1 และ SW2 มีลอจิกเป็น 0 LED จะติดทีละดวงเป็นสีแดงวิ่งจาก LED7 ไป LED0 ไปเรื่อย ๆ

ถ้า SW1 มีลอจิกเป็น 0 และ SW2 มีลอจิกเป็น 1 LED จะติดทีละดวงเป็นสีแดงวิ่งจาก LED0 ไป LED7 ไปเรื่อย ๆ

ถ้า SW1 และ SW2 มีลอจิกเป็น 0 LED จะติดทีละดวงเป็นสีแดงวิ่งจาก LED0 ไป LED7 และ ติดทีละดวงเป็นสีแดงวิ่งจาก LED7 ไป LED0 เป็นแบบนี้ไปเรื่อย ๆ

LAB 3-3

Description: LOOKUP TABLE TECHNIQUE TO DECODE BINARY DATA INTO 7-SEGMENT DISPLAY

Hardware: PORT 0 -> 7-SEGMENT (COMMON CARTODE)

PORT 1 -> SWITCH

ASM Code:

```
            ORG     0000H
            SEG_DATA    EQU P0
START:          MOV     R4, #00H
            MOV     R5, #0AH
GET_CODE:   MOV     DPTR,
#CODE_DATA
            MOV     A, R4
            MOVC    A, @A+DPTR
            MOV     SEG_DATA, A
            INC  R4
            ACALL   DELAY
            DJNZ    R5, GET_CODE
            SJMP    START

DELAY:          MOV     R5, #5
DELAY1: MOV     R6, #255
DELAY1: MOV     R7, #255
DELAY2: DJNZ    R7, DELAY2
            DJNZ    R6, DELAY1
            DJNZ    R5, DELAY0
            RET
CODE_DATA: DB   3FH,06H,5BH,
      4FH,66H
            DB  6DH,        7DH,        07H,
      7FH,67H
```

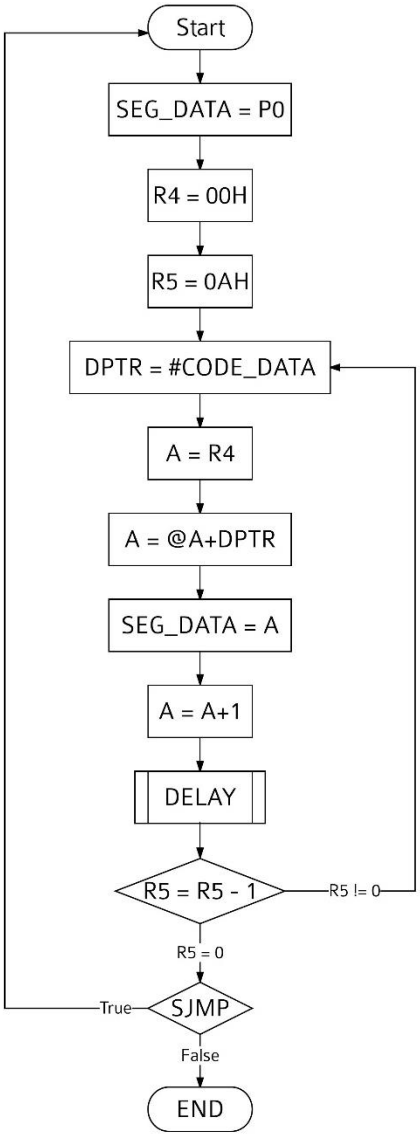
วิธีการทดลอง

1. เขียนโค้ดที่โจทย์กำหนดให้ จากนั้นทำการแปลงไฟล์ให้ได้ไฟล์นามสกุล .hex

2. ทำการรัน ASM Code จากโปรแกรม Flash Magic เพื่อดาวน์โหลดโค้ดลงบอร์ดไมโครคอนโทรลเลอร์ ผ่านสายสื่อข้อมูล

3. สังเกตผลการทดลองและบันทึกผลการทดลอง

Flow Chart:


Start
SEG_DATA = P0
R4 = 00H
R5 = 0AH
DPTR = #CODE_DATA
A = R4
A = @A+DPTR
SEG_DATA = A
A = A+1
DELAY
R5 = R5 - 1
R5 != 0
R5 = 0
True
SJMP
False
END

การทำงานของโปรแกรม :

ORG 0000H	เริ่มต้นการทำงานที่ตำแหน่ง 0000H
SEG_DATA EQU P0	ประกาศตัวแปร SEG_DATA สำหรับเรียกแทนพอร์ต0
START: MOV R4, #00H	เริ่มโปรแกรมย่อย ชื่อ start กำหนดให้ R4 = 00H เพื่อใช้เป็นตัวชี้ข้อมูล
MOV R5, #0AH	กำหนดให้ R5 = 0AH เพื่อใช้เป็นตัวชี้ข้อมูล
GET_CODE: MOV DPTR, #CODE_DATA	เริ่มโปรแกรมย่อย ชื่อ GET_CODE ให้ตำแหน่งของ DPTR เท่ากับตำแหน่งของ CODE_DATA
MOV A, R4	กำหนดให้ A = R4 เพื่อใช้เป็นตัวชี้ข้อมูล
MOVC A, @A+DPTR	นำค่าในตาราง DATA มาไว้ที่รีจิสเตอร์ A
MOV SEG_DATA, A	เก็บค่า A ไว้ใน SEG_DATA
INC R4	เพิ่มค่าของ R4 ไปอีก 1
ACALL DELAY	เรียกใช้โปรแกรมย่อยเพื่อหน่วงเวลา
DJNZ R5, GET_CODE	ถ้า R5 ไม่เท่ากับ 0 ให้ไปที่ GET_CODE
SJMP START	กระโดดไปที่ START
DELAY: MOV R5, #5	โปรแกรมย่อยเพื่อหน่วงเวลา กำหนดให้ R5 = 5
DELAY0: MOV R6, #255	โปรแกรมย่อยเพื่อหน่วงเวลา 0 กำหนดให้ R6 = 255
DELAY1: MOV R7, #255	โปรแกรมย่อยเพื่อหน่วงเวลา 1 กำหนดให้ R7 = 255
DELAY2: DJNZ R7, DELAY2	โปรแกรมย่อยเพื่อหน่วงเวลา 2 ถ้า R7 ไม่เท่ากับ 0 ให้ไปที่ DELAY2
DJNZ R6, DELAY1	ถ้า R6 ไม่เท่ากับ 0 ให้ไปที่ DELAY1

DJNZ R5, DELAY0	ถ้า R5 ไม่เท่ากับ 0 ให้ไปที่ DELAY0
RET	สิ้นสุดโปรแกรมหน่วงเวลา
CODE_DATA: DB 3FH,06H,5BH,4FH,66H	เรียกใช้โปรแกรมย่อยตาราง
DB 6DH, 7DH, 07H, 7FH,67H	
END	จบการทำงาน

Results and discussion:

7-Segment จะติเป็นเลข 0 และเพิ่มขึ้นทีละเลข 1 ไปเรื่อย ๆ จนถึงเลข 9 แล้วก็กลับมาเริ่มต้นที่เลข 0 เป็นแบบนี้ไปเรื่อย ๆ

EXERCISE

1. หาก PORT 1 (P1.0 - P1.7) ถูกเชื่อมต่อไว้กับแผง Switch (SW0 - SW7) จลดัดแปลงโปรแกรม LAB3-3

ให้มีการทำงานดังนี้

หากสับ SW0 ค้างไว้ ให้โปรแกรมทำการเพิ่มค่าขึ้นเรื่อย ๆ ทีละหนึ่งค่า และแสดงผลออกทาง

7-Segment โดยที่หากข้อมูลเกิน 9 ให้แสดงเลข 9 ค้างไว้

```
            ORG     0000H
            SEG_DATA    EQU P0
            SWITCH  EQU P1
            SW1     EQU P1.7
START:          MOV     R4, #00H
            MOV     R5, #0AH
CHK:            JNB SW1, CHK
            MOV     LOGIC_DATA,
#01H
            LCALL   DELAY
            MOV     LOGIC_DATA,
#00H
            LCALL   DELAY
            SJMP    CHK
GET_CODE:   MOV     DPTR,
#CODE_DATA
            MOV     A, R4
            MOVC    A, @A+DPTR
            MOV     SEG_DATA, A
            INC  R4
            ACALL   DELAY
            DJNZ    R5, GET_CODE
            SJMP    $

DELAY:          MOV     R5, #5
DELAY1: MOV     R6, #255
DELAY1: MOV     R7, #255
DELAY2: DJNZ    R7, DELAY2
            DJNZ    R6, DELAY1
            DJNZ    R5, DELAY0
            RET
CODE_DATA: DB   3FH, 06H, 5BH,
      4FH 66H
```

ORG 0000H	เริ่มต้นการทำงานที่ตำแหน่ง 0000H
SEG_DATA EQU P0	ประกาศตัวแปร SEG_DATA สำหรับเรียกแทนพอร์ต0
SWITCH EQU P1	ประกาศตัวแปร SWITCH สำหรับเรียกแทนพอร์ต 1
SW1 EQU P1.7	ประกาศตัวแปร SW1 สำหรับเรียกแทนพอร์ต 1 บิตที่ 7
START: MOV R4, #00H	เริ่มโปรแกรมย่อย ชื่อ start กำหนดให้ R4 = 00H เพื่อใช้เป็นตัวชี้ข้อมูล
CHK: JNB SW1, CHK	เริ่มโปรแกรมย่อย ชื่อ CHK ถ้า SW1 ไม่ถูกกด ให้กลับไปที่ CHK
MOV LOGIC_DATA, #01H	กำหนดให้ LOGIC_DATA = 1 เพื่อใช้เป็นตัวชี้ข้อมูล
LCALL DELAY	เรียกใช้โปรแกรมย่อยเพื่อหน่วงเวลา
MOV LOGIC_DATA, #00H	กำหนดให้ LOGIC_DATA = 0 เพื่อใช้เป็นตัวชี้ข้อมูล
LCALL DELAY	เรียกใช้โปรแกรมย่อยเพื่อหน่วงเวลา
SJMP CHK	กระโดดไปที่คำสั่ง CHK
MOV R5, #0AH	กำหนดให้ R5 = 0AH เพื่อใช้เป็นตัวชี้ข้อมูล
GET_CODE: MOV DPTR, #CODE_DATA	เริ่มโปรแกรมย่อย ชื่อ GET_CODE ให้ตำแหน่งของ DPTR เท่ากับตำแหน่งของ CODE_DATA
MOV A, R4	กำหนดให้ A = R4 เพื่อใช้เป็นตัวชี้ข้อมูล
MOVC A, @A+DPTR	ให้ A = 0 + ค่าในตาราง
MOV SEG_DATA, A	เก็บค่า A ไว้ใน SEG_DATA
INC R4	เพิ่มค่าของ R4 ไปอีก 1
ACALL DELAY	เรียกใช้โปรแกรมย่อยเพื่อหน่วงเวลา
DJNZ R5, GET_CODE	ถ้า R5 ไม่เท่ากับ 0 ให้ไปที่ GET_CODE
SJMP $	ให้โปรแกรมอยู่กับที่
DELAY:	โปรแกรมย่อยเพื่อหน่วงเวลา

MOV R5, #5	กำหนดให้ R5 = 5
DELAY0: MOV R6, #255	โปรแกรมย่อยเพื่อหน่วงเวลา 0 กำหนดให้ R6 = 255
DELAY1: MOV R7, #255	โปรแกรมย่อยเพื่อหน่วงเวลา 1 กำหนดให้ R7 = 255
DELAY2: DJNZ R7, DELAY2	โปรแกรมย่อยเพื่อหน่วงเวลา 2 ถ้า R7 ไม่เท่ากับ 0 ให้ไปที่ DELAY2
DJNZ R6, DELAY1	ถ้า R6 ไม่เท่ากับ 0 ให้ไปที่ DELAY1
DJNZ R5, DELAY0	ถ้า R5 ไม่เท่ากับ 0 ให้ไปที่ DELAY0
RET	สิ้นสุดโปรแกรมหน่วงเวลา
CODE_DATA: DB 3FH,06H,5BH,4FH,66H	เรียกใช้โปรแกรมย่อยตาราง
DB 6DH, 7DH, 07H, 7FH,67H	
END	จบการทำงาน

หากสับ SW1 ค้างไว้ ให้โปรแกรมทำการลดค่าขึ้นเรื่อยๆ ทีละหนึ่งค่า และแสดงผลออกทาง 7-Segment โดยที่หากข้อมูลต่ำกว่า 0 ให้แสดงเลข 0 ค้างไว้

```
        ORG     0000H
        SEG_DATA    EQU P0
        SWITCH  EQU P1
        SW1     EQU P1.7
START:      MOV     R4, #00H
        MOV     R5, #0AH
CHK:        JNB SW1, CHK
        MOV     LOGIC_DATA,
#01H
        LCALL   DELAY
        MOV     LOGIC_DATA,
#00H
        LCALL   DELAY
        SJMP    CHK
GET_CODE:   MOV     DPTR,
#CODE_DATA
        DEC R4
        MOV     A, R4
        MOVC    A, @A+DPTR
        MOV     SEG_DATA, A

        ACALL   DELAY
        DJNZ    R5, GET_CODE
        SJMP    $

DELAY:      MOV     R5, #5
DELAY1: MOV     R6, #255
```

ORG 0000H	เริ่มต้นการทำงานที่ตำแหน่ง 0000H
SEG_DATA EQU P0	ประกาศตัวแปร SEG_DATA สำหรับเรียกแทนพอร์ต0
SWITCH EQU P1	ประกาศตัวแปร SWITCH สำหรับเรียกแทนพอร์ต 1
SW1 EQU P1.7	ประกาศตัวแปร SW1 สำหรับเรียกแทนพอร์ต 1 บิตที่ 7
START: MOV R4, #00H	เริ่มโปรแกรมย่อย ชื่อ start กำหนดให้ R4 = 00H เพื่อใช้เป็นตัวชี้ข้อมูล
CHK: JNB SW1, CHK	เริ่มโปรแกรมย่อย ชื่อ CHK ถ้า SW1 ไม่ถูกกด ให้กลับไปที่ CHK
MOV LOGIC_DATA, #01H	กำหนดให้ LOGIC_DATA = 1 เพื่อใช้เป็นตัวชี้ข้อมูล
LCALL DELAY	เรียกใช้โปรแกรมย่อยเพื่อหน่วงเวลา
MOV LOGIC_DATA, #00H	กำหนดให้ LOGIC_DATA = 0 เพื่อใช้เป็นตัวชี้ข้อมูล
LCALL DELAY	เรียกใช้โปรแกรมย่อยเพื่อหน่วงเวลา
SJMP CHK	กระโดดไปที่คำสั่ง CHK

MOV R5, #0AH	กำหนดให้ R5 = 0AH เพื่อใช้เป็นตัวชี้ข้อมูล
GET_CODE: MOV DPTR, #CODE_DATA	เริ่มโปรแกรมย่อย ชื่อ GET_CODE ให้ตำแหน่งของ DPTR เท่ากับตำแหน่งของ CODE_DATA
DEC R4	ลดค่าของ R4 ไปอีก 1
MOV A, R4	กำหนดให้ A = R4 เพื่อใช้เป็นตัวชี้ข้อมูล
MOVC A, @A+DPTR	ให้ A = 0 + ค่าในตาราง
MOV SEG_DATA, A	เก็บค่า A ไว้ใน SEG_DATA
ACALL DELAY	เรียกใช้โปรแกรมย่อยเพื่อหน่วงเวลา
DJNZ R5, GET_CODE	ถ้า R5 ไม่เท่ากับ 0 ให้ไปที่ GET_CODE
SJMP $	ให้โปรแกรมอยู่กับที่
DELAY: MOV R5, #5	โปรแกรมย่อยเพื่อหน่วงเวลา กำหนดให้ R5 = 5
DELAY0: MOV R6, #255	โปรแกรมย่อยเพื่อหน่วงเวลา 0 กำหนดให้ R6 = 255
DELAY1: MOV R7, #255	โปรแกรมย่อยเพื่อหน่วงเวลา 1 กำหนดให้ R7 = 255
DELAY2: DJNZ R7, DELAY2	โปรแกรมย่อยเพื่อหน่วงเวลา 2 ถ้า R7 ไม่เท่ากับ 0 ให้ไปที่ DELAY2
DJNZ R6, DELAY1	ถ้า R6 ไม่เท่ากับ 0 ให้ไปที่ DELAY1
DJNZ R5, DELAY0	ถ้า R5 ไม่เท่ากับ 0 ให้ไปที่ DELAY0
RET	สิ้นสุดโปรแกรมหน่วงเวลา
CODE_DATA: DB 3FH,06H,5BH,4FH,66H	เรียกใช้โปรแกรมย่อยตาราง
DB 6DH, 7DH, 07H, 7FH,67H	
END	จบการทำงาน

วิเคราะห์ผลการทดลอง

จาก LAB3-1 เป็นการเขียนโปรแกรมเพื่อแสดงผลติดสลับเป็นสีเขียวและสีแดง ตรง LED 0

จาก LAB3-2 เป็นการควบคุมการทำงานของ LED โดยที่มี 4 รูปแบบ คือ ถ้า SW1 และ SW2 มีลอจิกเป็น 0 LED0 - LED7 จะติดเป็นสีเขียว ถ้า SW1 มีลอจิกเป็น 1 และ SW2 มีลอจิกเป็น 0 LED จะติดทีละดวงเป็นสีแดงวิ่งจาก LED7 ไป LED0 ไปเรื่อย ๆ ถ้า SW1 มีลอจิกเป็น 0 และ SW2 มีลอจิกเป็น 1 LED จะติดทีละดวงเป็นสีแดงวิ่งจาก LED0 ไป LED7 ไปเรื่อย ๆ ถ้า SW1 และ SW2 มีลอจิกเป็น 0 LED จะติดทีละดวงเป็นสีแดงวิ่งจาก LED0 ไป LED7 และ ติดทีละดวงเป็นสีแดงวิ่งจาก LED7 ไป LED0 เป็นแบบนี้ไปเรื่อย ๆ

จาก LAB3-3 เป็นการเขียนโปรแกรมเพื่อควบคุมการทำงานของ 7-Segment โดยจะแสดงเป็นเป็นตัวเลขเรียงจากเลข 0 ไปเลข 9

มีการใช้สวิตช์หลายตัวทำให้สามารถเกิดเหตุการณ์ได้หลายเหตุการณ์ มีคำสั่ง JB และ JNB ใช้ในการเปรียบเทียบ

สรุปผลการทดลอง

ถ้ามีสวิตช์ n ตัว เหตุการณ์ที่สามารถเกิดได้ จะเท่ากับ 2 ยกกำลัง n เหตุการณ์ การใช้งานคำสั่งกระโดดแบบมีเงื่อนไข (Conditional Jump) ทำให้ผู้เรียนได้ฝึกใช้คำสั่งในการกระโดดไปยังโปรแกรมย่อยหลังจากการพิจารณาเงื่อนไขตามที่กำหนดแล้ว (Check Condition Before Jump) เช่นในการทดลองที่ 3-1 ได้มีการใช้คำสั่ง JNB ซึ่งคือการกระโดดไปยังตำแหน่งอื่นที่กำหนด หากค่าในบิตที่กำหนดเป็น 0 ยกตัวอย่างเช่น JNB SW1, CHK เป็นการเช็คว่า SW1 ถูกกดเปลี่ยนหรือไม่ ถ้าจาก 1 ถูกกดเป็น 0 ก็จะไปที่ CHK ถ้ายังไม่มีการเปลี่ยนก็จะอยู่ที่ตำแหน่งเดิมไม่ขยับไปไหนนั่นเอง ส่วนการทดลองที่ 3-2 ใช้คำสั่ง JB คือการกระโดดไปยังตำแหน่งอื่นที่กำหนด หากค่าในบิตที่กำหนดเป็น 1 หลักการทำงานยังคงคล้ายกับ JNB ที่ว่าจะกระโดดเมื่อ SW1 หรือ SW2 ถูกกด จาก 0 เป็น 1 เพียงแค่ต่างเงื่อนไขกันเท่านั้น และการทดลองที่ 3-3 มีการใช้ 7-Segment มาต่อเป็นเอาท์พุตเพื่อแสดงผลเป็นตัวเลข มี Look-up Table เพื่อช่วยให้แสดงผลไล่จาก 0 ถึง 9 บน 7-Segment

เอกสารอ้างอิง

พนัส นัถฤทธิ์. (2560). **ไมโครคอนโทรลเลอร์และการประยุกต์ใช้งานควบคุมหุ่นยนต์.** พิมพ์ครั้งที่ 1. พิษณุโลก: รัตนสุวรรณการพิมพ์ 3

http://www.mind-tek.net/8051.php

http://www.ett.co.th/article/Robot/et_robot_rd2/rd2_001.html

http://www.olearning.siam.edu/2011-11-28-08-10-01/593-100-101-

http://www.thaiall.com/assembly/asmcommand.htm