บทที่ 2
ทฤษฎีและหลักการ

2.1 ไมโครคอนโทรลเลอร์ MCS-51

ไมโครคอนโทรลเลอร์ MCS-51 ได้จัดให้มีส่วนประกอบภายใน เพื่ออำนวยความสะดวกแก่ผู้ใช้ เช่น ไทเมอร์/เคาเตอร์ พอร์ตอนุกรม (Serial Port) และสำหรับไมโครคอนโทรลเลอร์บางตัว อาจมีส่วนอื่นเพิ่มเติมเข้ามาอีก เช่นเบอร์ AT80C515, AT80C535 มีวงจรแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล (Analog to Digital Converter)

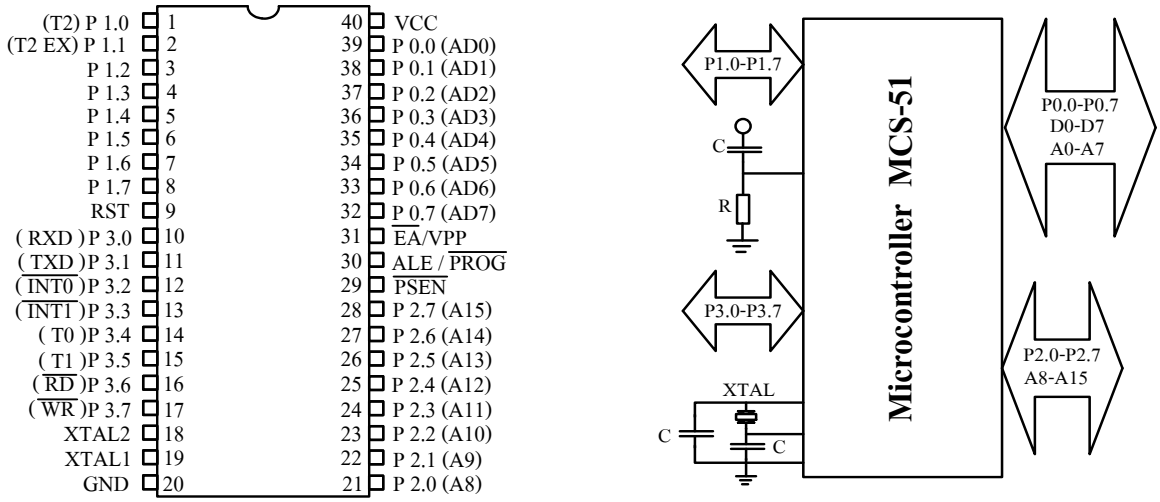
2.1.1 คุณสมบัติทั่วไปของไมโครคอนโทรลเลอร์ตระกูล MCS-51

ไมโครคอนโทรลเลอร์แต่ละเบอร์นั้นจะมีความแตกต่างกันในรายละเอียด ในที่นี้จะขออ้างถึงเบอร์ AT89S52 ของบริษัท Atmel ซึ่งเป็นไมโครคอนโทรลเลอร์ขนาด 8 บิต มีคุณสมบัติดังนี้

- มีหน่วยความจำแบบแฟลช (Flash Memory) ขนาด 8 กิโลไบท์
- โปรแกรมผ่านพอร์ตอนุกรมมาตรฐาน SPI (SPI Serial Interface)
- สามารถเขียนโปรแกรมและลบซ้ำได้นับ 1000 ครั้ง
- มีหน่วยความจำแบบ EEPROM ขนาด 2 กิโลไบต์
- สามารถเขียนโปรแกรมและลบซ้ำได้นับ 100,000 ครั้ง
- ใช้แหล่งจ่ายไฟกระแสตรงขนาด 5 โวลต์ (ทำงานในช่วง 4-6 โวลต์)
- ทำงานด้วยสัญญาณนาฬิกาตั้งแต่ 0-24 MHz
- สามารถป้องกันหน่วยความจำได้ 3 ระดับ
- มีหน่วยความจำข้อมูล (RAM) ขนาด 256 ไบต์
- มี 32 บิตอิสระสามารถเข้าถึงระดับบิตได้
- มีไทเมอร์/ เคาเตอร์ขนาด 16-bit ทั้งหมด 2 ตัว
- รองรับอินเตอร์รัปต์ได้ 8 แหล่ง
- สามารถสื่อสารข้อมูลแบบอนุกรมได้ด้วย UART Channel
- มีโหมดการทำงานแบบ Low Power Idle และ Power Down สำหรับการประหยัดพลังงาน
- มี Watchdog Timer เพื่อให้การทำงานของระบบสามารถ Reset ได้อัตโนมัติ

2.1.2 สถาปัตยกรรมของไมโครคอนโทรลเลอร์ตระกูล MCS-51 (เบอร์ AT89S52)

การจัดเรียงขาของ MCS-51 (เบอร์ AT89S52) เป็นไปตามรูปที่ 2.1



รูปที่ 2.1 การจัดขาของ MCS-51 (เบอร์ AT89S52)

- VCC ต่อไฟเลี้ยง (Supply Voltage 5 v)
- GND ต่อกราวด์ (Ground)
- Port 0 (P0.0-P0.7) เป็นพอร์ต 2 ทิศทางขนาด 8 บิตสามารถทำงานได้ 2 หน้าที่คือเป็นพอร์ตอินพุตเอาต์พุตทั่วไป และใช้เป็นพอร์ตสำหรับติดต่อกับหน่วยความจำภายนอกคือรับส่งข้อมูล และกำหนดแอดเดรสไบต์ต่ำ (A0-A7)
- Port 1 (P1.0-P1.7) เป็นพอร์ต 2 ทิศทางขนาด 8 บิต มีการต่อตัวต้านทานพูลอัพ (Pull-up Resistor) ไว้ภายใน ทำหน้าที่เป็นพอร์ตอินพุตเอาต์พุตทั่วไป นอกจากนี้ยังใช้งานเป็นขาอินพุตเอาต์พุตของไทเมอร์ 2
- Port 2 (P2.0-P2.7) เป็นพอร์ต 2 ทิศทางขนาด 8 บิต มีการต่อตัวต้านทานพูลอัพ (Pull-up Resistor) ไว้ภายใน สามารถทำงานได้ 2 หน้าที่คือเป็นพอร์ตอินพุตเอาต์พุตทั่วไป และใช้เป็นพอร์ตสำหรับติดต่อกับหน่วยความจำภายนอกคือรับส่งข้อมูล และกำหนด แอดเดรสไบต์สูง (A8-A15)
- Port 3 (P3.0-P3.7) เป็นพอร์ต 2 ทิศทางขนาด 8 บิตและ มีการต่อตัวต้านทานพูลอัพ (Pull-up Resistor) ไว้ภายใน ทำหน้าที่คือเป็นพอร์ตอินพุตเอาต์พุตทั่วไป ยังใช้งานเป็นสัญญาณควบคุมการติดต่อกับหน่วยความจำ การอินเตอร์รัปต์ และอื่นๆ
- RST เป็นขาอินพุตที่ใช้รับสัญญาณสำหรับรีเซ็ตซีพียู ซีพียูจะถูกรีเซ็ตเมื่อขานี้เป็นลอจิก “1” นาน 2 แมชินไซเคิล หรือ 24 ไซเคิลของสัญญาณนาฬิกา
- ALE/PROG ทำหน้าที่เป็นขาเอาต์พุตเมื่อซีพียูต้องการติดต่อกับหน่วยความจำภายนอกจะทำการส่งสัญญาณพัลส์ออกมาที่ขานี้เพื่อทำการแอดเดรสไบต์ต่ำของหน่วยความจำ

ภายนอก และขานี้จะเป็นอินพุตเมื่ออยู่ในระหว่างโปรแกรมแฟลช

- PSEN เป็นขาเอาต์พุตใช้ในการติดต่อกับหน่วยความจำโปรแกรมภายนอกคือเมื่อซีพียูทำการประมวลผลกับหน่วยความจำภายนอกขานี้จะแอคทีฟสองครั้ง
- EA/VPP เป็นขาเอาต์พุตและต้องการลอจิก “0” เพื่อยอมให้ซีพียูสามารถเข้าถึงหน่วยความจำภายนอกได้ซึ่งอยู่ที่ตำแหน่ง 0000H ถึง FFFFH นอกจากนี้แล้วขานี้ยังใช้รับไฟ 12 โวลต์เพื่อใช้ในระหว่างที่ทำการโปรแกรมแฟลช
- XTAL1 เป็นขาอินพุตของวงจรออสซิลเลเตอร์แอมปลิไฟเออร์ และยังเป็นอินพุตของวงจรกำเนิดสัญญาณนาฬิกาภายใน
- XTAL2 เป็นขาเอาต์พุตของวงจรออสซิลเลเตอร์แอมปลิไฟเออร์

2.1.3 สัญลักษณ์และคำสั่งของ MCS-51 เป็นไปตามตารางที่ 2.1

ตารางที่ 2.1 สัญลักษณ์และคำสั่งของ MCS-51

สัญลักษณ์	ความหมาย
Rn	หมายถึง รีจีสเตอร์ใช้งานทั่วไป R0-R7 ที่ถูกเลือกใช้ในขณะนั้น
@Ri	หมายถึง รีจีสเตอร์ใช้งานทั่วไป R0 หรือ R1 ที่เข้าถึงข้อมูลได้โดยอ้อม
Direct	หมายถึง ข้อมูลขนาด 8 บิต ที่ใช้กำหนดค่าตำแหน่งหน่วยความจำสำหรับเก็บข้อมูลภายใน ประกอบด้วยหน่วยความจำเก็บข้อมูลทั่วไป ตำแหน่ง 0 - 127 และ หน่วยความจำ เก็บข้อมูลที่ใช้เป็นรีจีสเตอร์ใช้งานเฉพาะ ตำแหน่ง 128 – 255
Bit	หมายถึง ค่าตำแหน่งของบิตข้อมูลในหน่วยความจำที่เข้าถึงในระดับบิต
#Data	หมายถึง ข้อมูลขนาด 8 บิตที่กำหนดในคำสั่ง
#Data16	หมายถึง ข้อมูลขนาด 16 บิตที่กำหนดในคำสั่ง
Rel	หมายถึง ค่าตำแหน่งหน่วยความจำขนาด 8 บิต ที่คิดแบบมีเครื่องหมายในคำสั่ง SJMP และการกระโดดแบบมีเงื่อนไขทุกคำสั่ง
Addr11	หมายถึง ค่าตำแหน่งหน่วยความจำขนาด 11 บิต ใช้เป็นค่าตำแหน่งหน่วยความจำปลายทาง (ภายในขอบเขต 2048 ตำแหน่ง) ในคำสั่ง AJMP และ ACALL

ตารางที่ 2.1 สัญลักษณ์และคำสั่งของ MCS-51 (ต่อ)

สัญลักษณ์	ความหมาย
Addr16	หมายถึง ค่าตำแหน่งหน่วยความจำขนาด 16 บิต ใช้เป็นค่าตำแหน่งหน่วยความจำปลายทาง (ภายในขอบเขต 65536 ตำแหน่ง) ในคำสั่ง LJMP และ LCALL
คำสั่ง	ความหมาย
MOV A,Rn	ย้ายข้อมูลจาก Rn ไป A
MOV A,direct	ย้ายข้อมูลจากหน่วยความจำ Direct ไป A
MOV A,@Ri	ย้ายข้อมูลจากหน่วยความจำที่เก็บอยู่ในตำแหน่ง Ri ไป A
MOV A,#Data	ย้ายค่าคงที่ 8 บิตไปเก็บที่ A
MOV Rn,A	ย้ายข้อมูลจาก A ไป Rn
MOV Rn,Direct	ย้ายข้อมูลจากหน่วยความจำ Direct ไป Rn
MOV Direct,A	ย้ายข้อมูลจาก A ไปยังหน่วยความจำ Direct
MOV Direct,Rn	ย้ายข้อมูลจาก Rn ไปยังหน่วยความจำ Direct
MOV Direct,Direct	ย้ายข้อมูลระหว่างหน่วยความจำภายใน
MOV Direct,@Ri	ย้ายข้อมูลจากหน่วยความจำที่เก็บอยู่ในตำแหน่ง Ri ไปยังหน่วยความจำ Direct
MOV Direct,#Data	ย้ายค่าคงที่ 8 บิต ไปยังหน่วยความจำ Direct
MOV @Ri,A	ย้ายข้อมูลใน A ไปยังหน่วยความจำ Ri
MOV @Ri,Direct	ย้ายข้อมูลจากหน่วยความจำ Direct ไปยังหน่วยความจำ Ri
MOV @Ri,#Data	ย้ายค่าคงที่ 8 บิต ไปยังหน่วยความจำ Ri
MOV DPTR,#Data16	ย้ายค่าคงที่ 16 บิต ไปยัง DPTR
MOVC A,@A+DPTR	ย้ายข้อมูลจากหน่วยความจำข้อมูลที่สัมพัทธ์กับ DPTR ไปยัง A
MOVC A,@A+PC	ย้ายข้อมูลจากหน่วยความจำข้อมูลที่สัมพัทธ์กับ PC ไปยัง A
MOVX A,@Ri	ย้ายข้อมูลจากหน่วยอินพุตที่เก็บอยู่ในตำแหน่ง Ri ไปยัง A
MOVX A,@DPTR	ย้ายข้อมูลจากหน่วยความจำที่เก็บอยู่ในตำแหน่ง DPTR ไปยัง A
MOVX @Ri,A	ย้ายข้อมูลที่เก็บอยู่ใน A ไปยังหน่วยเอาต์พุตตำแหน่ง Ri
MOVX @DPTR,A	ย้ายข้อมูลที่เก็บอยู่ใน A ไปยังหน่วยความจำตำแหน่ง DPTR
MOV C,Bit	ย้ายค่า Bit ไปยัง Carry Flag

ตารางที่ 2.1 สัญลักษณ์และคำสั่งของ MCS-51 (ต่อ)

คำสั่ง	ความหมาย
MOV Bit,C	ย้ายค่าแฟล็ก Carry ไปยัง Bit
JNC Rel	กระโดด ถ้าค่าแฟล็ก Carry เป็น 0
JC Rel	กระโดด ถ้าค่าแฟล็ก Carry เป็น 1
JB Bit,Rel	กระโดด ถ้าค่า Bit เป็น 1
JNB Bit,Rel	กระโดด ถ้าค่า Bit เป็น 0
JBC Bit,Rel	กระโดด ถ้าค่า Bit เป็น 1 และเปลี่ยนค่า Bit เป็น 0
JZ Rel	กระโดดไปยังตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าหากค่า A เป็นค่า 00H
JNZ Rel	กระโดดไปยังตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าหากค่า A ไม่เป็นค่า 00H
CJNE A,Direct,Rel	เปรียบเทียบค่า A กับหน่วยความจำ Direct และกระโดดไปยังตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าค่าไม่เท่ากัน
CJNE A,#Data,Rel	เปรียบเทียบค่า A กับค่าคงที่ และกระโดดไปยังตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าค่าไม่เท่ากัน
CJNE Rn,#Data,Rel	เปรียบเทียบค่า Rn กับค่าคงที่ และกระโดดไปยังตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าค่าไม่เท่ากัน
CJNE @Ri,#Data,Rel	เปรียบเทียบค่าใน Ri กับค่าคงที่ และกระโดดไปยังตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าค่าไม่เท่ากัน
DJNZ Rn,Rel	ลดค่าใน Rn และกระโดดไปยังตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าค่าไม่เป็น 0
DJNZ Direct,Rel	ลดค่าในหน่วยความจำ Direct และกระโดดไปยังตำแหน่งสัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าค่าไม่เป็น 0
SJMP Rel	กระโดดไปยังตำแหน่งสัมพัทธ์กับตำแหน่งปัจจุบัน
AJMP Addr11	กระโดดไปยังตำแหน่งจากค่าแอดเดรส 11 บิต
LJMP Addr16	กระโดดไปยังตำแหน่งจากค่าแอดเดรส 16 บิต

ตารางที่ 2.1 สัญลักษณ์และคำสั่งของ MCS-51 (ต่อ)

คำสั่ง	ความหมาย
JMP @A+DPTR	กระโดดไปยังตำแหน่งที่สัมพัทธ์กับ A+DPTR
ACALL Addr11	ไปทำโปรแกรมย่อยจากค่าแอดเดรส 11 บิต
LCALL Addr16	ไปทำโปรแกรมย่อยจากค่าแอดเดรส 16 บิต
RET	สิ้นสุดการทำโปรแกรมย่อย แล้วกลับไปยังตำแหน่งถัดไปที่กระโดดมา
RETI	สิ้นสุดการทำโปรแกรมย่อย อินเตอร์รัปท์
INC A	เพิ่มค่าใน A
INC Rn	เพิ่มค่าใน Rn
INC Direct	เพิ่มค่าในหน่วยความจำ Direct
INC @Ri	เพิ่มค่าในหน่วยความจำที่ตำแหน่ง Ri
INC DPTR	เพิ่มค่าใน DPTR
DEC A	ลดค่าใน A
DEC Rn	ลดค่าใน Rn
DEC Direct	ลดค่าในหน่วยความจำ Direct
DEC @Ri	ลดค่าในหน่วยความจำที่เก็บอยู่ใน Ri
DEC DPTR	ลดค่าใน DPTR
MUL AB	คูณ A กับ B แล้วเก็บค่าใน BA
DIV AB	หาร A กับ B แล้วเก็บค่าใน A เก็บเศษใน B
DA A	ทำ Decimal Adjust ค่าใน A
ADD A,Rn	บวกค่า Rn กับ A
ADD A,Direct	บวกค่าในหน่วยความจำ Direct กับ A เก็บผลลัพธ์ใน A
ADD A,@Ri	บวกค่าในหน่วยความจำตำแหน่ง Ri กับ A เก็บผลลัพธ์ใน A
ADD A,#Data	บวกค่าคงที่ 8 บิต กับ A เก็บผลลัพธ์ใน A
ADDC A,Rn	บวกค่า Rn กับ A พร้อมแฟล็ก Carry เก็บผลลัพธ์ใน A
ADDC A,Direct	บวกค่าในหน่วยความจำ Direct กับ A พร้อมแฟล็ก Carry เก็บผลลัพธ์ใน A
ADDC A,@Ri	บวกค่าในหน่วยความจำตำแหน่ง Ri กับ A พร้อมแฟล็ก Carry เก็บผลลัพธ์ใน A

ตารางที่ 2.1 สัญลักษณ์และคำสั่งของ MCS-51 (ต่อ)

คำสั่ง	ความหมาย
ADDC A,#Data	บวกค่าคงที่ 8 บิต กับ A พร้อมแฟล็ก Carry เก็บผลลัพธ์ใน A
SUBB A,Rn	ลบค่า Rn กับ A พร้อมแฟล็ก Borrow เก็บผลลัพธ์ใน A
SUBB A,Direct	ลบค่าในหน่วยความจำ Direct กับ A พร้อมแฟล็ก Borrow เก็บผลลัพธ์ใน A
SUBB A,@Ri	ลบค่าในหน่วยความจำที่เก็บอยู่ใน Ri กับ A พร้อมแฟล็ก Borrow เก็บผลลัพธ์ใน A
SUBB A,#Data	ลบค่าคงที่ 8 บิต กับ A พร้อมแฟล็ก Borrow เก็บผลลัพธ์ใน A
CLR A	ทำค่าใน A ให้เป็นศูนย์
CPL A	กลับค่าบิตใน A เป็นตรงข้ามทุกบิต
RL A	หมุนบิตใน A ไปทางซ้าย 1 บิตและบิต 0 เป็นค่าจากบิต 7
RLC A	หมุนบิตใน A ไปทางซ้าย 1 บิตและค่าจากบิต 7 นำไปเก็บในแฟล็ก Carry และบิตที่อยู่ในแฟล็ก Carry ไปเป็นบิตที่ 0
RR A	หมุนบิตใน A ไปทางขวา 1 บิตและบิต 7 เป็นค่าจากบิต 0
RRC A	หมุนบิตใน A ไปทางขวา 1 บิตและค่าจากบิต 0 นำไปเก็บในแฟล็ก Carry และบิตที่อยู่ในแฟล็ก Carry ไปเป็นบิตที่ 7
SWAP A	สลับค่าสี่บิตซ้ายกับสี่บิตขวาภายใน A
PUSH Direct	ย้ายข้อมูลหน่วยความจำ Direct ไปเก็บยัง Stack
POP Direct	ย้ายข้อมูลจาก Stack ไปยังหน่วยความจำ Direct
CLR C	ทำค่าแฟล็ก Carry ให้เป็น 0
CLR Bit	ทำค่า Bit ให้เป็น 0
SETB C	ทำค่าแฟล็ก Carry ให้เป็น 1
SETB bit	ทำค่า Bit ให้เป็น 1
CPL C	กลับค่าแฟล็ก Carry ให้เป็นตรงข้าม
CPL Bit	กลับค่า Bit ให้เป็นตรงข้าม
ORL C,Bit	OR ค่า Bit กับแฟล็ก Carry เก็บผลลัพธ์ใน C
ORL C,/Bit	OR ค่า Not Bit กับแฟล็ก Carry เก็บผลลัพธ์ใน C
ANL C,Bit	AND ค่า Bit กับแฟล็ก Carry เก็บผลลัพธ์ใน C
ANL C,/Bit	AND ค่า Not Bit กับแฟล็ก Carry เก็บผลลัพธ์ใน C

2.2 ภาษา C

ภาษา C เป็นภาษาระดับกลาง (Middle Language) ที่นำมาเขียนเป็นโปรแกรมระบบปฏิบัติการยูนิกซ์ หรือใช้เป็นภาษาในการโปรแกรมทางระบบ (System Programming) ภาษา C มีความสามารถเหมือนภาษาแอสแซมบลี (Assembly) คือมีคำสั่งที่สามารถเข้าถึง บิต , ไบต์ และตำแหน่งต่างๆ ของหน่วยความจำของเครื่อง
คอมพิวเตอร์ได้ แต่มีวิธีการเขียนโปรแกรมง่ายเหมือนกับภาษาระดับสูงทั่วไป ซึ่งในการเขียนโปรแกรมภาษา C จะแบ่งเป็นเมนูชุดคำสั่งต่างๆ ที่ใช้ในการเขียนโปรแกรม และมีการนิยามแบ่งเป็นส่วนต่างๆ สำหรับการเรียกใช้ ซึ่งแต่ละเมนูจะประกอบไปด้วย คำสั่งเรียงกันเป็นกลุ่ม และแต่ละเมนูจะมีชื่อกำกับไว้เพื่อเรียกทำงาน โดยเมนูหลักจะมีชื่อว่า “Main ( )” เป็นส่วนหัวที่ใช้ในการเริ่มเขียนโปรแกรม และเมนูในส่วนที่เหลือจะเรียกเป็นฟังก์ชันโดยในแต่ละฟังก์ชันประกอบไปด้วยชุดคำสั่งต่างๆ

2.2.1 ประเภทของข้อมูลในภาษา C
ในภาษา C นั้นได้มีการแบ่งกลุ่มของชนิดข้อมูลออกเป็น 4 กลุ่มด้วยกัน คือ

2.2.1.1 ข้อมูลที่เป็นเลขจำนวนเต็ม (Integer Type) ชนิดข้อมูลแบบนี้ มักเป็นตัวเลขที่เป็นจำนวนเต็ม เช่น ค่าของ 10, 20, 15, -5, 0, -1 เป็นต้น

2.2.1.2 ข้อมูลที่เป็นตัวอักษร (Character Type) ชนิดข้อมูลแบบนี้มักนิยมใช้ ในการจัดเก็บข้อมูลที่เป็นตัวอักษร เช่น A, B, C, a,b,c, ก, ข, ค, เป็นต้น

2.2.1.3 ข้อมูลชนิดอื่นๆ (Other Type) เป็นชนิดข้อมูล ที่นอกเหนือไปจาก 3 ชนิดข้างต้นอันได้แก่ตัวแปรอาร์เรย์ (Array) และตัวแปรแบบพอยน์เตอร์ (Pointer)

2.2.1.4 ข้อมูลที่เป็นตัวเลขนั้นจะมีรูปแบบเฉพาะอีก 2 แบบ คือ ชนิดของข้อมูลที่มีค่าได้ทั้งค่าลบ และค่าบวก หรือที่เรียกว่า “Signed” กับชนิดของข้อมูลที่เก็บได้เฉพาะตัวเลขจำนวนเต็มศูนย์กับค่าที่เป็นค่าบวกเท่านั้น หรือที่เรียกว่า “Unsigned”

ตารางที่ 2.2 ขนาดของบิตที่ใช้และช่วงค่าจำนวนตัวเลขของแต่ละบิต

ชนิด	ขนาด(บิต)	ช่วงของค่าที่เก็บ
Int	16	-32,768 ถึง 32,767
Unsigned	16	0 ถึง 65,535
Char	8	-128 ถึง127และใช้เก็บตัวเก็บอักษร
Unsigned Char	8	ถึง 255 และใช้เก็บตัวเก็บอักษร

ส่วนตัวแปรแบบอื่นๆ ที่มีใช้ในภาษานั้นได้แก่

1. ชนิดข้อมูบลแบบอาร์เรย์ (Array Type) ตัวแปรชนิดนี้จะมีขนาดเท่ากับจำนวนชุดของอาร์เรย์คูณกับขนาดของชนิดข้อมูลที่เก็บในอาร์เรย์นั้น

2. ชนิดของข้อมุลแบบพอยน์เตอร์ (Pointer Type) ตัวแปรชนิดนี้จะเป็นตัวแปรที่ทำหน้าที่ชี้ หรือเรียกได้ว่าตัวแปรชนิดนี้จะเป็นตัวเก็บค่าของแอดเดรสของตัวแปรอื่นๆ ปกติในโปรแกรมภาษา C จะมีตัวแปรแบบตัวชี้โดยมีขนาด 4 ไบต์ หรือ 32 บิต

2.2.2 นิพจน์ในภาษา C

นิพจน์ (Expression) หรือประโยคกระทำการในภาษา c นั้น เรียกได้ว่าเป็นภาษาที่เต็มไปด้วยเครื่องหมายพิเศษมากมาย ทั้งนี้เพราะเพื่อความยืดหยุ่นในการพัฒนาซอฟต์แวร์นั่นเอง ดังนั้นการที่สามารถเข้าใจเครื่องหมายต่างๆ ที่สามารถจะนำมาประยุกต์ใช้กับภาษา C ได้เหมาะสมมากขึ้น และเนื่องจากมีเครื่องหมายดำเนินการมากมาย จึงได้ทำการแบ่งกลุ่มเครื่องหมายต่างๆ ออกเป็นกลุ่มๆ ดังต่อไปนี้

2.2.2.1เครื่องหมายการดำเนินการทางคณิตศาสตร์

เป็นเครื่องหมายที่ใช้ในการคำนวณทางภาษา C ซึ่งได้แก่ตารางที่ 2.3

ตารางที่ 2.3 เครื่องหมายที่ใช้ในการคำนวณทางภาษา C

เครื่องหมาย	ความหมาย
+	บวก (Add)
-	ลบ (Subtract)
*	คูณ (Multiply)
/	หารแบบไม่เอาเศษมาใช้ (Div)
%	หารแบบเอาเศษจากการหารมาใช้ (Mod)

2.2.2.2 เครื่องหมายที่ใช้ในการเปรียบเทียบ

เครื่องหมายในกลุ่มนี้ มักจะใช้ในการตรวจสอบเงื่อนไข หรือใช้ในการวนรอบเท่านั้น เวลาใช้ระวังจะสับสนกับเครื่องหมายในกลุ่มอื่นๆ ด้วย

ตารางที่ 2.4 เครื่องหมายที่ใช้ในการเปรียบเทียบสำหรับภาษา C

เครื่องหมาย	ความหมาย
=	เท่ากับ
!=	ไม่เท่ากัน
!	ตรงกันข้าม (Not)
&&	และ (And)
>=	มากกว่า หรือ เท่ากับ
<=	น้อยกว่า หรือ เท่ากับ
II	หรือ (OR)
<	น้อยกว่า
>	มากกว่า

2.2.2.3 เครื่องหมายดำเนินการทางบิต

เครื่องหมายกลุ่มนี้จะเกี่ยวข้องกับการนำค่าตัวเลขฐานสอง มาทำการกระทำต่อกันแบบทีละบิตซึ่งเครื่องหมายที่เกี่ยวข้องนั้นได้แก่ในตารางที่ 2.5

ตารางที่ 2.5 เครื่องหมายที่ใช้ดำเนินการทางบิตสำหรับภา C

เครื่องหมาย	ความหมาย
& , I	การ AND บิต และ การ OR บิต
^	การ XOR บิต
<<	เลื่อนบิตไปทางซ้าย (Left Shift)
>>	เลื่อนบิตไปทางขวา (Right Shift)
~	ทำ One ‘s Complement (Inverse)

2.2.2.4 เครื่องหมายที่ใช้ในการแปลงค่า

เครื่องหมายเหล่านี้ จะใช้ในการถ่ายคาไปมาของตัวแปร แต่จะไม่ใช้เครื่องหมายเหล่านี้กับการตรวจเงื่อนไขดังความหมายในตารางที่ 2.6

ตารางที่ 2.6 เครื่องหมายที่ใช้ในการแปลงค่าสำหรับภาษา C

เครื่องหมาย	ความหมาย
=	นำค่าทางขวามาให้ทางซ้าย
++	เพิ่มค่าขึ้น 1
-	ลดค่าลง 1
+=	เพิ่มค่าเท่ากับค่าทางขวา
-=	ลดค่าเท่ากับค่าทางขวา
*=	นำตัวเองคูณกับค่าทางขวา
/=	นำตัวเองหารกับค่าทางขวา
%=	นำตัวเองหารเอาค่าเศษกับค่าทางขวา
&=	นำตัวเองมาทำการ AND กับค่าทางขวา
I=	นำตัวเองมาทำการ OR กับค่าทางขวา

2.2.3 รูปแบบในการเขียนโปรแกรมภาษา C

ในการเขียนซอร์สโค้ด (Source Code) โปรแกรมภาษา C นั้น สามารถแบ่งลำดับการสั่งการ หรือลำดับที่ต้องทำได้คร่าวๆ ดังนี้

ลำดับที่ 1. ทำการเรียกไฟล์ที่เก็บชื่อฟังก์ชันที่ต้องใช้ในโปรแกรม โดยสั่งว่า #Include ชื่อไฟล์ .h

ลำดับที่ 2. ทำการสร้างตัวแปรภายนอกที่ต้องการใช้

ลำดับที่ 3. เขียนฟังก์ชันที่ทำหน้าที่เป็น โปรแกรมหลัก ซึ่งมีชื่อว่า “Main ( )” ที่ต้องกำหนดเป็นชื่อนี้เพราะว่า ในภาษา C นั้นได้ถูกกำหนดว่า โปรแกรมจะเริ่มทำงานในฟังก์ชันที่ชื่อว่า “Main” ก่อนเป็นลำดับแรก ซึ่งรูปแบบของฟังก์ชันในภาษา C เป็นดังนี้

ชนิดค่าที่คืนกลับ ชื่อฟังก์ชัน (ค่าที่ส่งให้ฟังก์ชัน)

ดังนั้นฟังก์ชันที่ชื่อว่า “Main” สามารถเขียนออกมาได้ดังนี้

“Main ( )”

2.2.3.1 เงื่อนไขและการวนรอบ

สิ่งที่ทำให้ภาษา C สะดวกต่อการเขียนโปนแกรมมากกว่าภาษาแอสแซมบลี ก็คือเรื่องของการสร้างเงื่อนไขในการทำงานกับการวนรอบการทำงาน ถ้าต้องการทำงานในส่วนที่มีการตรวจสอบเงื่อนไขและการวนรอบมากๆ จำนวนของรีจิสเตอร์ก็อาจจะไม่พอ ทำให้ต้องอาศัยการของหน่วยความจำเอาไว้เป็นที่เก็บข้อมูลในการวนรอบการทำงาน และคำสั่งที่มีไว้สำหรับการตรวจสอบเงื่อนไขและการวนรอบนั้น ในภาษาแอสแซมบลีก็ตระเตรียมเอาไว้ให้แค่เพียงพอสำหรับการใช้งานแต่ไม่เพียงพอต่อการใช้งานของผู้เขียนโปรแกรมเท่าไรนัก เมื่อการเขียนโปรแกรมที่ซับซ้อนมากๆ โอกาสที่จะผิดพลาดในเรื่องของการออกแบบการใช้รีจิสเตอร์ก็อาจจะเกิดขึ้น ปัญหาเหล่านี้จะลดลงเมื่อทำการเขียนโปรแกรมโดยใช้ภาษา C เนื่องจากภาษา C มีโครงสร้างของการตรวจสอบเงื่อนไขและการวนรอบ ที่เป็นธรรมชาติมากกว่าภาษาแอสแซมบลี แต่ต้องระวังในเรื่องของหน่วยความจำการที่ใช้หน่วยความจำเกินกว่าที่ออกแบบไว้อาจจะเกิดขึ้นในภาษา C ได้เพราะว่าภาษา C ก็คือโปรแกรมที่ถูกแปลงมาจากภาษาแอสแซมบลี โดยมีเงื่อนไขที่ใช้อยู่ในภาษา C ก็คือ

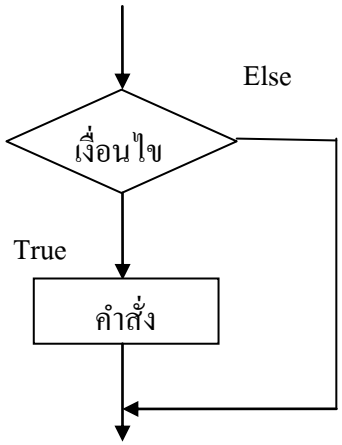
2.2.3.2 เงื่อนไข If

If เป็นคำสั่งแรกที่ต้องทำความรู้จัก ทั้งนี้เพื่อมีการตรวจสอบเงื่อนไขการทำงานคำสั่ง if จะเข้าถึงรูปแบบการตรวจสอบเงื่อนไขและการวนรอบในรูปแบบต่างๆ ได้ด้วยหลักการทำงานของคำสั่ง If ก็คือ ทำการตรวจสอบเงื่อนไขในวงเล็บที่อยู่ต่อจาก If ว่าเงื่อนไขนั้นเป็นจริง (ผลลัพธ์ของเงื่อนไขเป็น “1”) หรือไม่ ก็จะทำงานตามคำสั่งที่ต่อจาก If โดยรูปแบบการใช้ If นั้นมีอยู่หลากหลายมาก ในที่นี้ขอยกตัวอย่างที่เห็นได้ชัดๆ ด้วยกัน 4 รูปแบบดังต่อไปนี้

รูปแบบที่ 1

If (เงื่อนไข) {คำสั่งที่ต้องการให้ทำเมื่อเงื่อนไขเป็นจริง}

ในรูปแบบนี้จะเป็นการตรวจสอบเงื่อนไขว่าเป็นจริงตามที่กำหนดหรือไม่ถ้าใช่จึงจะกระทำคำสั่งภายใต้ {…} ไป

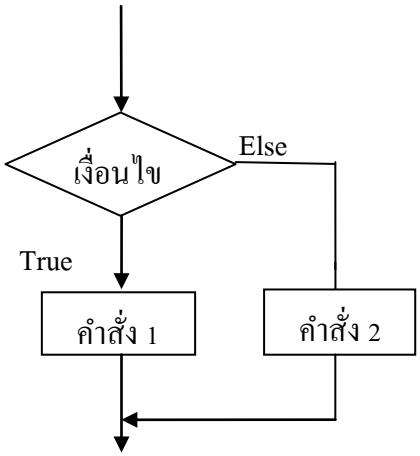


รูปที่ 2.2 ผังการทำงานโดยใช้คำสั่ง If

การใช้ If ในรูปแบบที่ 1 นี้จะเห็นว่าเน้นการตรวจสอบให้เงื่อนไขมีค่าเป็นจริงเท่านั้นแต่ถ้าเกิดผลลัพธ์ที่ออกมาแล้วนั้นเป็นเท็จ หรือไม่เป็นจริงตามเงื่อนไข ก็จะถูกข้ามไป แต่ในบางโปรแกรมนั้น ผู้เขียนอาจจะต้องตรวจสอบเพิ่มด้วยว่าเมื่อไม่เป็นจริงจะเป็นอย่างไร ด้วยเหตุนี้ในการเขียนโปรแกรมด้วยภาษาระดับสูงนั้น จะเตรียมความสามาถรองรับเงื่อนไขที่ซับซ้อนเอาไว้แล้ว ซึ่งจะขอกล่าวรายละเอียดในแบบต่างๆ ต่อไป

รูปแบบที่ 2

If (เงื่อนไข) {คำสั่งที่ต้องการให้ทำเมื่อเงื่อนไขเป็นจริง} Else {คำสั่งที่ต้องการให้ทำเมื่อเงื่อนไขเป็นเท็จ} ในรูปแบบที่ 2 นี้แตกต่างกับรูปแบบที่ 1 ตรงที่มีการใช้คำสั่ง Else มารองรับในกรณีที่เงื่อนไขของ If ไม่เป็นจริง นั้นคือ เมื่อโปรแกรมเริ่มทำงานก็จะตรวจสอบเงื่อนไขของ If ได้ผลลัพธ์เป็นจริงก็จะทำในส่วนของ “คำสั่งที่ต้องการให้ทำเมื่อเงื่อนไขเป็นจริง” พอเสร็จก็จะออกจากการตรวจสอบเงื่อนไข แต่ถ้าเงื่อนไขของ If เป็นเท็จ โปรแกรมก็จะกระโดดมาทำในส่วนของ Else หรือคำสั่งในส่วนของ “คำสั่งที่ต้องการให้ทำเมื่อเงื่อนไขเป็นเท็จ”ผังการทำงานของคำสั่งแสดงในภาพที่ 2.3



รูปที่ 2.3 ผังการทำงานโดยใช้คำสั่ง If / Else

2.2.3.3 การใช้ Switch

การตรวจสอบเงื่อนไขด้วย Switch นั้นจะแตกต่างจากการใช้ If ทั้งนี้เพราะ Switch นั้นจะนำค่าจากตัวแปรที่ผู้เขียนโปรแกรมกำหนดไปทำการเปรียบเทียบกับค่าที่ต้องการตรวจสอบ ถ้าตรงกันก็จะทำงานภายใต้เงื่อนไขนั้น เมื่อตรวจสอบเสร็จก็จะไปตรวจสอบเงื่อนไขเลื่อยๆ จนสุดท้ายถ้าไม่ตรงกับค่าใดเลยก็จะไปทำในส่วนของ Default การใช้ Switch จะมีคำสั่งคู่การทำงานอีก 2 คำสั่งคือ Continue และ Break

รูปแบบของคำสั่ง “Switch” เป็นดังนี้

Switch (ตัวแปรที่เราจะนำมาทดสอบ)

Case ค่าคงที่ 1 : คำสั่งที่ทำเป็นจริง 1

Case ค่าคงที่ 2 : คำสั่งที่ทำเป็นจริง 2

Case ค่าคงที่ 3 : คำสั่งที่ทำเป็นจริง n

Default : คำสั่งเมื่อเป็นกรณีอื่น

จากรูปแบบด้านบนจะมีหลักการทำงานคือ นำค่าของ “ตัวแปรที่จะนำมาทดสอบ” มาทำการตรวจสอบว่ามีค่าเท่ากับ “ค่าคงที่ 1 ” หรือไม่ถ้าใช่ก็จะทำ “คำสั่งที่ทำเมื่อเป็นจริง 1 ” เสร็จแล้วก็จะไปตรวจสอบ “ค่าคงที่ 2 ” ถ้าเท่ากันก็จะทำ “คำสั่งที่ทำเมื่อเป็นจริง 2 ” และจะตรวจสอบเช่นนี้ไปเรื่อยๆ และถ้าไม่เป็นจริง หรือไม่เท่ากันกับค่าใดๆ ตาม “Case” เลย ก็จะทำที่ส่วนของ “Default” ส่วนคำสั่ง “Break” หมายความว่าเมื่อทำมาจนถึงคำสั่งนี้ให้ออกจากการตรวจสอบของ “Switch” เมื่อเปรียบเทียบระหว่างการใช้ if กับ Switch จะเห็นได้ว่า การใช้ Switch จะลดจำนวนจะลดบรรทัดในการเขียนได้มากกว่าการใช้ If แต่ไม่เสมอไป การใช้ Switch จะทำงานได้รวดเร็วกว่าการใช้ If ซึ่งจะจะใช้อะไรเมื่อไรนั้นก็ขึ้นอยู่กับความนิยมและประสบการณ์ของผู้เขียน

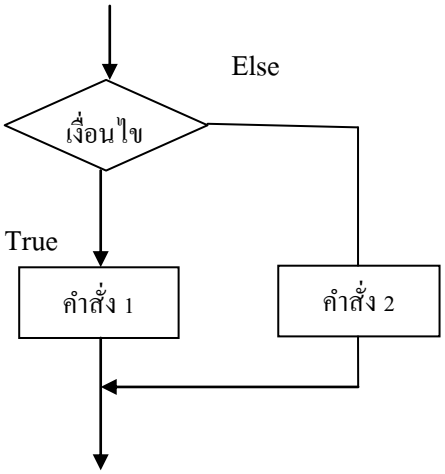
สิ่งหนึ่งที่คำสั่ง “Switch” ทำไม่ได้คือ ตรวจสอบค่าเป็นช่วง เช่น อย่างตรวจว่า “Menu_Choice” เป็นตัวอักษร “0-9” หรือไม่ อย่างนี้จะทำไม่ได้ด้วยคำสั่ง “Switch” แต่จะสามารถทำได้โดยการใช้คำสั่งตรวจสอบเงื่อนไขเป็นคำสั่ง ”If”

2.2.3.4 การใช้ While

เมื่อทำความรู้จักกับการตรวจสอบเงื่อนไขโดยคำสั่ง “If” และ “Switch” ไปแล้วเรื่องต่อไปนี้จะเป็น เรื่องของการวนรอบ หรือที่เรียกกันว่า ลูป (Loop) ประโยชน์ของการเขียนโปรแกรมด้วยการใช้คำสั่งวนรอบก็คือ ช่วยลดจำนวนบรรทัดของคำสั่งที่เขียนคล้ายๆ กัน การเขียนโปรแกรมวนรอบก็มีข้อเสียในเรื่องของการใช้หน่วยความจำแบบสแต็ก สำหรับการเก็บค่าการทดรอบ ซึ่งถ้าหน่วยความจำของบอร์ดไมโครคอนโทรลเลอร์ที่ออกแบบไว้นั้นมีไม่เพียงพอก็จะเกิดปัญหาที่เรียกว่า “หน่วยความจำแบบสแต็กเต็ม (Stack Overflow)” ได้ดังนี้ ในการเขียนโปรแกรมจะต้องคำนึงถึงเรื่องความจำเป็นที่จะต้องวนรอบด้วย

หลักการทำงานของคำสั่ง ”While” ก็คือ ถ้าเงื่อนไขของคำสั่งเป็นจริงก็จะทำ คำสั่งที่กำหนดเอาไว้เมื่อทำคำสั่งเสร็จครบทุกคำสั่งแล้วก็จะกลับมาตรวจสอบเงื่อนไขใหม่อีกรอบหนึ่ง ถ้าเงื่อนไขเป็นจริงก็จะทำตามคำสั่งอีกรอบหนึ่ง จะทำอย่างนี้เรื่อยๆ จนกว่าเงื่อนไขจะเป็นเท็จต่อไปนี้เป็นรูปแบบของคำสั่ง “While”

While (เงื่อนไข) {คำสั่งที่ทำ เมื่อเป็นจริง}

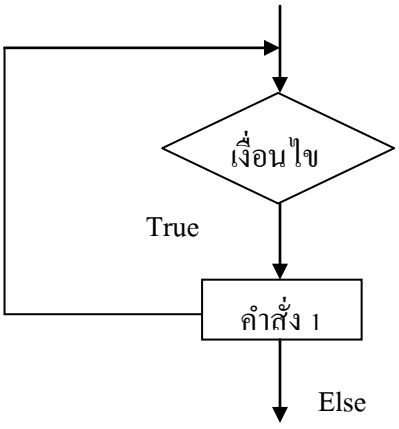


รูปที่ 2.4 ผังงานการทำงานในการใช้คำสั่ง "While"

2.2.3.5 การใช้ "Do…While"

ความแตกต่างของคำสั่ง" Do…While" ที่แตกต่างไปจากคำสั่ง "While" ก็คือคำสั่ง "While" จะทำการตรวจสอบเงื่อนไขก่อนที่จะทำตามคำสั่งที่ต้องการ แต่ในคำสั่ง "Do…While" นั้นจะทำคำสั่งในเงื่อนไขก่อน 1 รอบ แล้วค่อยตรวจสอบเงื่อนไข ถ้าเงื่อนไขเป็นจริงก็จะกลับไปทำคำสั่งในช่วงของ "Do…While" อีกรอบหนึ่งและจะออกจากการวนรอบก็ต่อเมื่อเงื่อนไขหลัง "Do…While" เป็นเท็จ รูปแบบของคำสั่ง "Do…While" เป็นดังนี้

Do{คำสั่งที่ทำ} While (เงื่อนไข) ;



รูปที่ 2.5 ผังงานการทำงานโดยใช้คำสั่ง "Do…While"

2.3 LCD (Liquid Crystal Display)

ในโมดูล LCD มีส่วนประกอบหลัก ๆ 3 ส่วน ดังนี้

- ตัวแสดงผล ภายในเป็นผลึกเหลวที่สามารถแสดงผลให้เห็นโดยอาศัยแสงภายนอก ดังนั้นจึงต้องมีมุมมองที่เหมาะสมในการมองข้อมูลที่แสดงผลบนจอ LCD
- ตัวควบคุม เป็นตัวรับข้อมูลจากอุปกรณ์ภายนอกมาควบคุมการทำงานของโมดูล LCD เช่น ลบจอภาพ แสดงตัวอักษร เป็นต้น โดยใช้ชิปควบคุมที่นิยมใช้ คือเบอร์ HD44780
- ตัวขับ เป็นตัวรับสัญญาณจากตัวควบคุมมาขับให้ตัวแสดงผลแสดงข้อมูลตามที่กำหนด

2.3.1 การแสดงผลที่จอ LCD

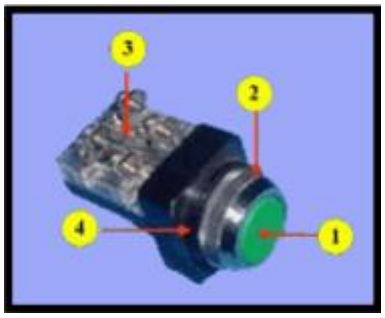
สามารถแสดงผลได้ 2 แบบ คือ

1. ASCII แสดงผลได้แบบตัวเลข, ตัวอักษรภาษาอังกฤษตัวพิมพ์ใหญ่และตัวพิมพ์เล็ก Space หรืออักขระใด ๆ ตามรหัส ASCII
2. CGRAM แสดงผลได้ตามรูปแบบที่สามารถกำหนดขึ้นเอง เช่นภาษาไทยหรือรูปภาพ ต่างๆ

2.4 สวิตช์ปุ่มกด (Push Button Switch)

สวิตช์ปุ่มกดที่ใช้ในการเริ่มเดิน (Start) เรียกว่าสวิตช์ปกติเปิด (Normally Open) หรือที่เรียกว่า เอ็น โอ (N.O.)

สวิตช์ปุ่มกดหยุดการทำงาน (Stop) เรียกว่า สวิตช์ปกติปิด (Normally Close) หรือที่เรียกว่า เอ็น ซี (N.C.)



รูปที่ 2.6 โครงสร้างภายนอกของสวิตช์ปุ่มกด

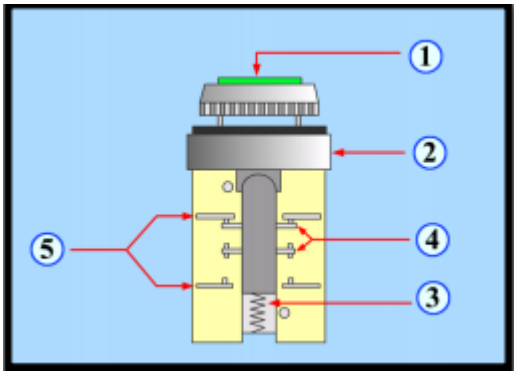
2.4.1 โครงสร้างภายนอกของสวิตช์ปุ่มกด

2.4.1.1 ปุ่มกดทำด้วยพลาสติก อาจเป็นสี เขียวแดงหรือเหลือง ขึ้นอยู่กับการนำไปใช้งาน

2.4.1.2 แหวน ล็อค

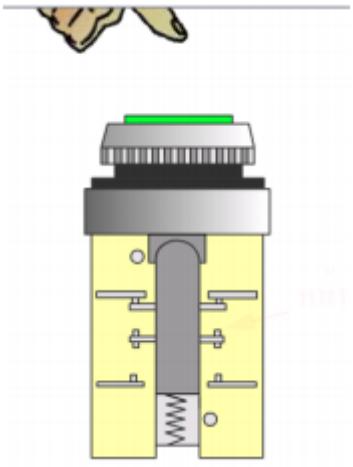
2.4.1.3 ชุดกลไกลหน้าสัมผัส

2.4.1.4 ยางรอง



รูปที่ 2.7 ลักษณะโครงสร้างของสวิตช์ปุ่มกดโดยทั่วไป

การทำงานของสวิตช์ปุ่มกด



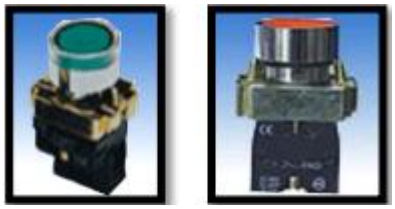
รูปที่ 2.8 การทำงานของสวิตช์ปุ่มกด

สรุปการทำงานของสวิตช์ปุ่มกด

ใช้นิ้วกดที่ปุ่มกดทำให้มีแรงดันหน้าสัมผัสให้เคลื่อนที่ หน้าสัมผัสที่ปิดจะเปิดส่วน หน้าสัมผัส ที่เปิดจะปิด เมื่อปล่อยนิ้วออกหน้าสัมผัส จะกลับสภาพเดิม ด้วยแรงสปริง การนำไปใช้งานใช้ในการ ควบคุมอุปกรณ์เริ่มต้นการทำงาน ควบคุมการเดินและหยุดหมุนมอเตอร์

2.4.2 ชนิดของสวิตช์ปุ่มกด

สวิตช์ปุ่มกดที่นิยมใช้มีอยู่ด้วยกันหลายชนิดเช่น



รูปที่ 2.9 สวิตช์ปุ่มกดแบบธรรมดา

สวิตช์ปุ่มกดแบบธรรมดา ใช้ในงานเริ่มเดิน (Start) และหยุดหมุน (Stop) สวิตช์สีเขียวใช้ในการสตาร์ท หน้าสัมผัส เป็นชนิดปกติเปิด (Normally Open) หรือที่เรียกว่า เอ็น โอ (N.O.) สวิตช์สีแดงใช้ในการหยุดการทำงาน (Stop) หน้าสัมผัสเป็นชนิดปกติปิด (Normally Close) หรือที่เรียกว่าเอ็น ซี (N.C.)

รูปที่ 2.10 สวิตช์ปุ่มกดที่ใช้ในการเริ่มเดิม

สวิตช์ปุ่มกดที่ใช้ในการเริ่มเดิม (Start) และหยุดหมุนนี้อยู่ในกล่องเดียวกันปุ่มสีเขียวสำหรับกดเริ่มเดินมอเตอร์ (Start) ปุ่มสีแดง สำหรับกดหยุดหมุน (Stop) เหมาะกับการใช้งานมอเตอร์ขนาดเล็กใช้งานธรรมดาที่ใช้กระแสไม่สูงสามารถต่อได้โดยตรง ใช้กับมอเตอร์ไฟฟ้าขนาดใหญ่กว่า 1/2 แรงม้าต้องใช้ร่วมกับอุปกรณ์อื่นเช่นสวิตช์แม่เหล็ก (Magnetic Contactor) และอุปกรณ์ป้องกันมอเตอร์ทำงาน เกินกำลัง (Over Load Protection) ดั้งนั้นจึงทำให้ระบบควบคุมการเริ่มเดินมอเตอร์เป็นไปอย่างมีประสิทธิภาพมากยิ่งขึ้น

รูปที่ 2.11 สวิตช์ปุ่มกดฉุกเฉิน

สวิตช์ปุ่มกดฉุกเฉิน (Emergency Push Button) สวิตช์ปุ่มกดฉุกเฉินหรือเรียกทั่วไปว่าสวิตช์ดอกเห็ดเป็นสวิตช์หัวใหญ่กว่าสวิตช์แบบธรรมดาเป็นสวิตช์ที่เหมาะกับงานที่ที่เกิดเหตุฉุกเฉินหรืองานที่ต้องการหยุดทันที

รูปที่ 2.12 สวิตช์ปุ่มกดที่มีหลอดสัญญาณติดอยู่

สวิตช์ปุ่มกดที่มีหลอดสัญญาณติดอยู่ (Illuminated Push Button) เมื่อกดสวิตช์ปุ่มกดจะทำให้หลอดสัญญาณสว่างออกมา

รูปที่ 2.13 สวิตช์ปุ่มกด ET-TEST10P/INP

ET-TEST10/INP เป็นบอร์ดสวิทช์ที่ใช้สำหรับงานทดสอบ Input ของ Port ต่างๆ ของบอร์ด จาก อีทีที ที่เป็นขั้ว 10 PIN ET BUS I/O หรือ 10 PIN 12 CIN/OUT เช่น บอร์ด CP-A VR V4, CP-JR908 GP32 V2, CPPIC V4, CP-JR51AC2 V2, CP-BS2P40, CP-JRBS2P40 ….ฯลฯ โดยเป็น SW Input กดติดปล่อยดับ จำนวน 8 ตัว

- Input ใช้ SW 8 ตัวแบบกดติดปล่อยดับ
- ต่อกับขั้ว 10 PIN ET BUS I/O
- PCB SIZE 8.3 x 2.5 CM
- พร้อมสายแพร์ 10 PIN 1 เส้น ต่อเข้าบอร์ดที่จะทดสอบ