ปีการศึกษา 2539

เครื่องวัดกำลังงานมนุษย์
HUMAN POWER MEASUREMENTS

โดย

นาย ธวัชชัย ดำเนินศิลป์

นาย วิรัต เพชรบุรี

อาจารย์ที่ปรึกษา

อาจารย์ พิชิต ลำยอง

ปริญญานิพนธ์ปีการศึกษา 2539

ภาควิชาวิศวกรรมไฟฟ้า

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องวัดกำลังงานมนุษย์

ผู้จัดทำ

1. นาย ธวัชชัย ดำเนินศิลป์
2. นาย วิรัต เพชรบุรี

อาจารย์ที่ปรึกษา

(อาจารย์ พิชิต ลำยอง)

เครื่องวัดกำลังงานมนุษย์

นาย ธวัชชัย ดำเนินศิลป์
นาย วิรัต เพชรบุรี
อาจารย์ พิชิต ลำยอง อาจารย์ที่ปรึกษา
ปีการศึกษา 2539

บทคัดย่อ

โครงงานนี้จะเป็นการศึกษาเกี่ยวกับกำลังงานของมนุษย์ที่ได้รับจากการออกกำลังกาย โดยการนำเอาจักรยานจากการออกกำลังกายมาต่อร่วมกับเครื่องกำเนิดไฟฟ้ากระแสตรง ซึ่งจะเปลี่ยนพลังงานกลเป็นพลังงานไฟฟ้า โดยจะนำค่าที่ได้รับจากเครื่องกำเนิดไฟฟ้าคือค่าแรงดัน ,ค่ากระแส และค่าความเร็วรอบของการปั่นจักรยาน ไปเป็นสัญญาณป้อนให้กับไมโครคอนโทรลเลอร์เพื่อจะประมวลและแสดงผลค่ากำลังงานของมนุษย์ที่จอแสดงผล โดยค่าแรงดัน ,ค่ากระแส และค่าความเร็วรอบในการปั่นจักรยานจะเปลี่ยนแปลงค่าได้ด้วยการเพิ่มโหลดด้วยการเพิ่มหลอดไฟฟ้ามากขึ้น

HUMAN POWER MEASUREMENTS

Thawatchai damnoensin
Wirath Phechburi
Pichit Lumyong Advisor
1996

ABSTRACT

This research project studied about power of human to obtianed from exercise . Take bicycle for exercise to connected with genertor for conversion machanical energy to electrical energy . The output to obtianed from generator is voltage , current and the speed of ride bicycle for exercise is input of microcontroller to calculate and display power of human at monitor . The voltage , current and the speed of ride bicycle for exercise will be variable. with increased load or turn on the lamp .

สารบัญ

สารบัญภาพ

สารบัญตาราง

หน้า

บทที่ 1

บทนำ

การออกกำลังกาย มีหลายอย่าง เช่น วิ่ง ว่ายน้ำ และอื่น ๆเพื่อช่วยให้ร่างกายของคนเรานั้นแข็งแรง ซึ่งการออกกำลังกายนั้นก็ต้องใช้พลังงาน เมื่อเราออกกำลังกาย ก็จะสูญเสียพลังงานส่วนนี้ไปโดยเปล่าประโยชน์ การปั่นจักรยานเป็นการออกกำลังกายอย่างหนึ่ง โดยจะต้องใช้พลังงานในการปั่นให้จักรยานเกิดการหมุนหรือเคลื่อนที่ แต่เราไม่อาจทราบได้ว่าพลังงานที่ใช้ในการปั่นมีค่าเท่าไหรจากแนวความคิดอันนี้ จึงได้เกิดโครงงานนี้ขึ้นโดยการนำเอาจักรยานออกกำลังกายมาต่อกับเครื่องกำเนิดไฟฟ้าเพื่อเปลี่ยนพลังงานกลเป็นพลังงานไฟฟ้าแล้วนำเอาค่าแรงดันกระแสและความเร็วรอบหรือความเร็วที่ใช้ในการปั่น ซึ่งค่าต่าง ๆ เหล่านี้จะป้อนเข้าให้กับชุดไมโครคอนโทรลเลอร์ แต่อินพุทของไมโครคอนโทรลเลอร์ต้องเป็นสัญญาณดิจิตอล เท่านั้น ดังนั้นค่าที่ได้รับจากเครื่องกำเนิดไฟฟ้าซึ่งเป็นสัญญาณอะนาล็อกจะต้องแปลงเป็นสัญญาณดิจิตอล จึงต้องใช้วงจรแปลงอะล็อกเป็นดิจิตอล เป็นวงจรแปลงสัญญาณ เมื่อค่าแรงดัน กระแส และความเร็วรอบหรือความเร็วที่ใช้ในการปั่นเกิดการเปลี่ยนแปลงค่า เพราะใส่โหลดให้กับเครื่องกำเนิด ไมโครคอนโทรเลอร์ก็จะประมวลผลและแสดงผลที่จอมอร์นิเตอร์ ก็จะทำให้ทราบค่าพลังงานของแต่ละคนที่ใช้ในการปั่นจักรยานได้ โดยคนแต่ละคนจะมีพลังงานหรือแรงที่ใช้ในการปั่นไม่เท่ากัน โดยโหลดที่ใช้อาจจะเป็นหลอดไฟ หรือแบตเตอรี่ก็ได้ ซึ่งแบตเตอรี่นี้สามารถชาร์จเก็บพลังงานที่ได้รับไว้ใช้งานในด้านอื่น ๆ ได้อีก แต่ในโครงงานนี้ใช้หลอดไฟเป็นโหลด

บทที่ 2

ทฤษฎี และหลักการของโครงงาน

เพื่อให้เข้าใจการทำงานของโครงงานมากยิ่งขึ้นเราจึงต้องเรียนรู้ทฤษฎีต่างๆที่เกี่ยวข้องกับโครงงานนี้โดยละเอียดซึ่งบทนี้จะอธิบาย หลักการของเครื่องจักรไฟฟ้ากระแสตรง หลักการของเทคโคมิเตอร์ หลักการทำงานของวงจรแปลงอะนาล็อกเป็นดิจิตอล และทฤษฎีของไมโครคอนโทรลเลอร์เบอร์ เอ็มซีเอสห้าหนึ่ง (MCS - 51)

2.1 เครื่องจักรไฟฟ้ากระแสตรง (D.C. Machine)

เครื่องจักรกลไฟฟ้ากระแสตรงเป็นเครื่องจักรที่ทำหน้าที่แปลงรูปพลังงานระหว่างพลังงานไฟฟ้ากระแสตรงกับพลังงานกล โดยที่เครื่องจักรทำงานโดยการหมุนตัวหมุน เมื่อเครื่องจักรแปลงรูปพลังงานกลเป็นพลังงานไฟฟ้ากระแสตรงเรียกว่า เครื่องกำเนิดไฟฟ้ากระแสตรง (D.C. Generator or Dynamo) และถ้าเครื่องจักรแปลงรูปพลังงานไฟฟ้ากระแสตรงเป็นพลังงานกล เรียกว่า มอเตอร์ไฟฟ้ากระแสตรง (D.C. Motor)

เครื่องจักรไฟฟ้ากระแสตรงมีซี่คอมมิวเตเตอร์ (Commutator) กับ แปรงถ่าน (Carbon Brush) ทำหน้าที่จัดเรียงกระแสจากไฟฟ้ากระแสสลับในขดลวดอาร์เมเจอร์ (Armature Coil) ให้เป็นไฟฟ้ากระแสตรงออกจากตัวเครื่องจักร จึงเป็นแหล่งจ่ายกระแสตรง แต่เมื่อศักดาไฟฟ้าที่ขั้วมีค่าสูงกว่าศักดาไฟฟ้าที่เกิดขึ้นภายในจะทำให้กระแสไหลจากภายนอก เข้าสู่ในตัวเครื่องจักร ซึ่งซี่คอมมิวเตเตอร์จะทำหน้าที่ จัดเรียงกระแสที่อยู่ภายใต้ขั้วแม่เหล็กที่แตกต่างกันก็จะมีกระแสไหลตรงข้ามทำให้เกิดแรงบิดเสริมกัน ซึ่งจะเห็นได้ว่าเครื่องจักรไฟฟ้ากระแสตรงตัวหนึ่งจะสามารถนำมาเป็นเครื่องกำเนิดไฟฟ้าหรือนำมาเป็นมอเตอร์ก็ได้

จุดเด่นของเครื่องจักรไฟฟ้ากระแสตรงคือทำงานอยู่ในตำแหน่งที่มุมระหว่างสนามแม่เหล็กจากขั้วแม่เหล็กกับสนามแม่เหล็กจากอาเมเจอร์ทำมุมเท่ากับ 90 องศา ซึ่งเครื่องจักรไฟฟ้ากระแสสลับโดยทั่วไปไม่สามารถทำงานที่ตำแหน่งนี้ได้และมอเตอร์ไฟฟ้ากระแสตรงจะมีแรงบิดขณะเริ่มหมุนสูง

2.1.1 โครงสร้างของเครื่องจักรกลไฟฟ้ากระแสตรง

เครื่องจักรไฟฟ้ากระแสตรงประกอบไปด้วยส่วนสำคัญต่าง ๆ ดังต่อไปนี้

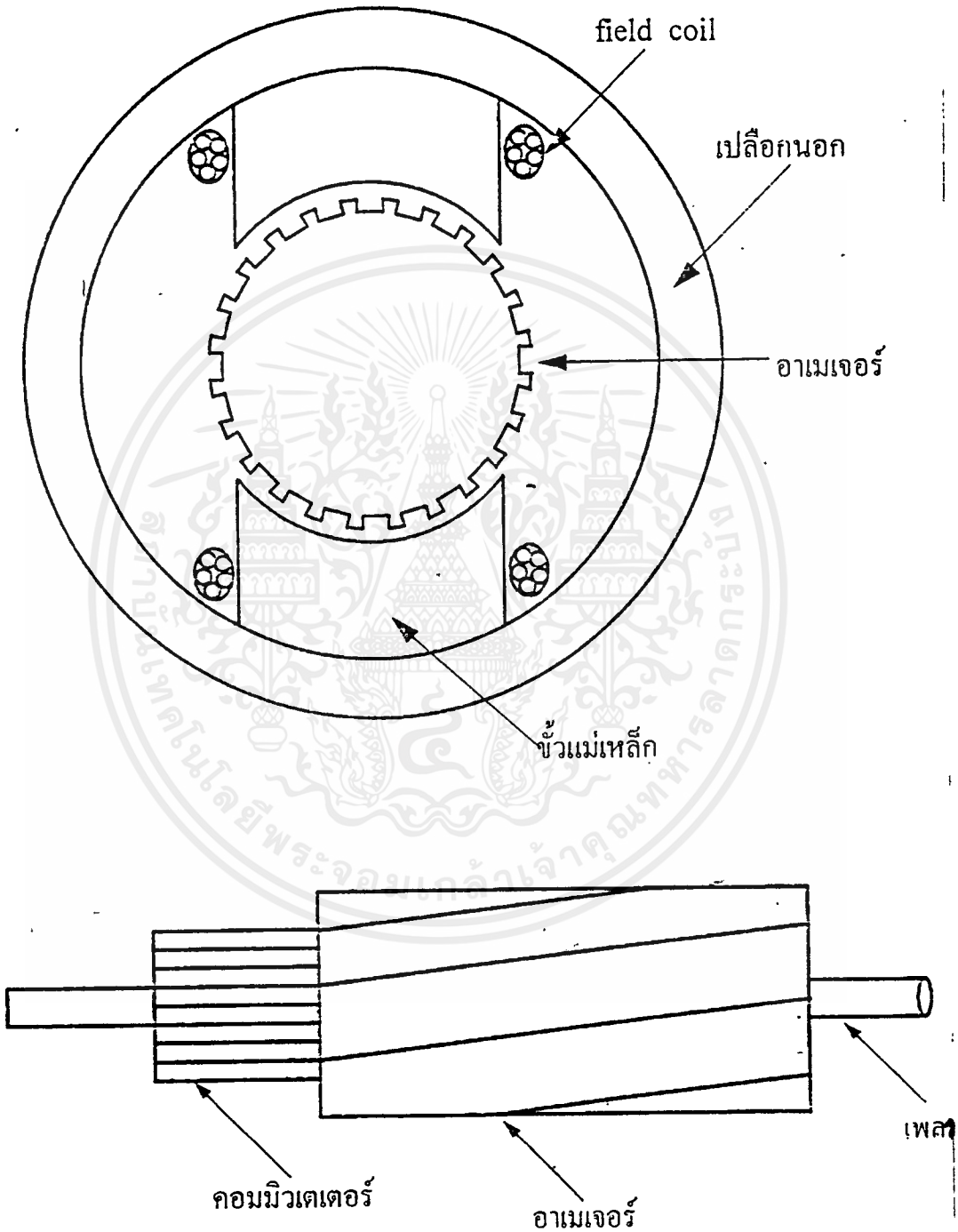
- เปลือกนอก (Yoke Frame) เป็นตัวยึดขั้วแม่เหล็กของส่วนที่อยู่กับที่ พร้อมทั้งเป็นวงจรแม่เหล็กเชื่อมต่อระหว่างขั้วแม่เหล็ก นอกจากนี้เปลือกนอกยังทำหน้าที่เป็นตัวยึดสำหรับติดตั้งเครื่องจักรรวมถึงเป็นแป้นยึดลูกปืน (Bearing) สำหรับเพลาของตัวหมุน การทำเปลือกนอกทำได้โดยการขึ้นรูป โดยการใช้เหล็กหล่อหรือใช้วิธีม้วนเหล็กแผ่นแล้วเชื่อมเป็นวง

- ขั้วแม่เหล็ก (Pole) เป็นส่วนที่ใช้สร้างสนามแม่เหล็กเพื่อให้เกิดการเหนี่ยวนำไฟฟ้า เมื่อหมุนตัวหมุนหรือเกิดแรงบิดเมื่อมีกระแสไหลในอาเมจอร์ โดยที่ขั้วแม่เหล็กประกอบไปด้วยแกนขั้วแม่เหล็ก (Pole Core) และขดสร้างสนามแม่เหล็ก (Field Coil)

- อาเมเจอร์ (Armature) เป็นส่วนที่เกิดการเหนี่ยวนำไฟฟ้า และเกิดแรงบิดจากสนามแม่เหล็กโดยจะประกอบไปด้วยแกนของอาเมเจอร์ (Armature Core) เป็นแผ่นเหล็กบาง (larminate) วางเรียงซ้อนกันเพื่อการเกิดกระแสไหลวน (Eddy Current) บนแกนของอาเมเจอร์จะมีร่องไว้สำหรับฟันขดลวดเรียกว่าร่องสล็อต (Stot) โดยร่องนี้จะไว้สำหรับพันขดลวดอาเมเจอร์ (Armature Winding)

- คอมมิวเตเตอร์ (Commutator) เป็นแท่งตัวนำโดยปกติใช้แท่งทองแดงวางเรียงกันและมีฉนวนกั้นกลางเพื่อป้องกันการลัดวงจรระหว่างแท่งทองแดง ตัวคอมมิวเตอร์จะใช้เป็นจุดต่อระหว่างขดลวดอาร์เมเจอร์ ดังนั้นสมบัติของอุปกรณ์ที่นำมาทำซี่คอมมิวเตอร์ต้องเป็นตัวนำที่ดี มีสัมประสิทธิ์ของความเสียดทานต่ำ เป็นตัวนำความร้อนและมีความสึกกร่อนจากการเสียดสีน้อยหรือไม่สึกกร่อนเลยยิ่งดี

- แปรงถ่าน (Cabon Brush) เป็นตัวนำไฟฟ้าที่ใช้ทำหน้าที่ต่อวงจรระหว่างจุดที่หยุดนิ่งกับส่วนที่หมุนก็คือ ตรงส่วนที่เป็นคอมมิวเตเตอร์ตัวแปรงถ่านจะมีส่วนที่เป็นกราไฟท์ (Graphite) และส่วนที่เป็นสายทองแดงถักเพื่อใช้ต่อกับวงจรไฟฟ้าได้ ช่องยึดแปรงถ่าน (Brush Holder) เป็นตัวยึดแปรงถ่านให้อยู่ในตำแหน่งที่ต้องการคุณสมบัติของแปรงถ่าน ต้องเป็นตัวนำไฟฟ้าที่ดี มีค่าสัมประสิทธิ์ของความเสียดทานต่ำ และต้องไม่แข็งจนเกินไปจนทำให้เกิดความเสียหายที่ซี่คอมมิวเตเตอร์



รูปที่ 2.1 ส่วนประกอบต่าง ๆ ของเครื่องจักรไฟฟ้ากระแสตรง

2.1.2 หลักการทำงานของเครื่องจักรไฟฟ้ากระแสตรง

เมื่อนำเครื่องจักรไฟฟ้ากระแสตรงมาใช้งานเป็นเครื่องกำเนิดไฟฟ้า โดยปกติแล้วการเหนี่ยวนำศักดาไฟฟ้าบนขดลวดที่อยู่บนตัวหมุน ซึ่งก็จะเกิดศักดาไฟฟ้าออกมาเป็นลักษณะของไฟฟ้ากระแสสลับดังแสดงในรูปที่ 2.2 (ก) โดยที่ต้องมีสนามแม่เหล็กจากขั้วแม่เหล็กและแท่งตัวนำเมื่อเคลื่อนตัวตัดสนามแม่เหล็ก เนื่องจากแท่งตัวนำเมื่อเคลื่อนผ่านขั้ว N จะมีทิศทางหนึ่งและเมื่อผ่านขั้ว S สนามแม่เหล็กจะกลับทิศทาง ดังนั้นขดลวดที่ใช้ในการพันในเครื่องจักรไฟฟ้าจะต้องมีด้านทั้ง 2 ด้านอยู่ภายใต้ขั้วแม่เหล็กตรงข้ามกัน เพื่อศักดาไฟฟ้าที่เกิดขึ้นในแต่ละด้านของขดลวดจะได้มีทิศทางเสริมกัน สรุปได้ว่าขดลวดที่ใช้สำหรับเหนี่ยวนำศักดาไฟฟ้าความกว้างของขดลวดจะเท่ากับความกว้างของขั้วแม่เหล็กซึ่งจะเท่ากับ 180̊ (องศาไฟฟ้า)

เมื่อนำปลายของขดลวดต่อเข้ากับวงแหวนครึ่งซีกในแต่ละด้าน แล้วนำแปรงถ่านมากดบนวงแหวนเพื่อต่อดักดาไฟฟ้ามาใช้งานจะปรากฏว่าสัญญาณที่จะออกมาจะเหมือนกับผ่านวงจรเรียงกระแส (Bridge Rectifier) โดยเป็นกระแสตรง ดังนั้นเมื่อได้มีการพัฒนาระบบวงแหวนครึ่งซีกมาเป็นซี่คอมมิวเตเตอร์ แทน ส่วนที่เป็นซี่คอมมิวเตเตอร์กับแปรงถ่านจึงถูกเรียกว่า ตัวเรียงกระแสทางกล (Mechanical Rectifier)

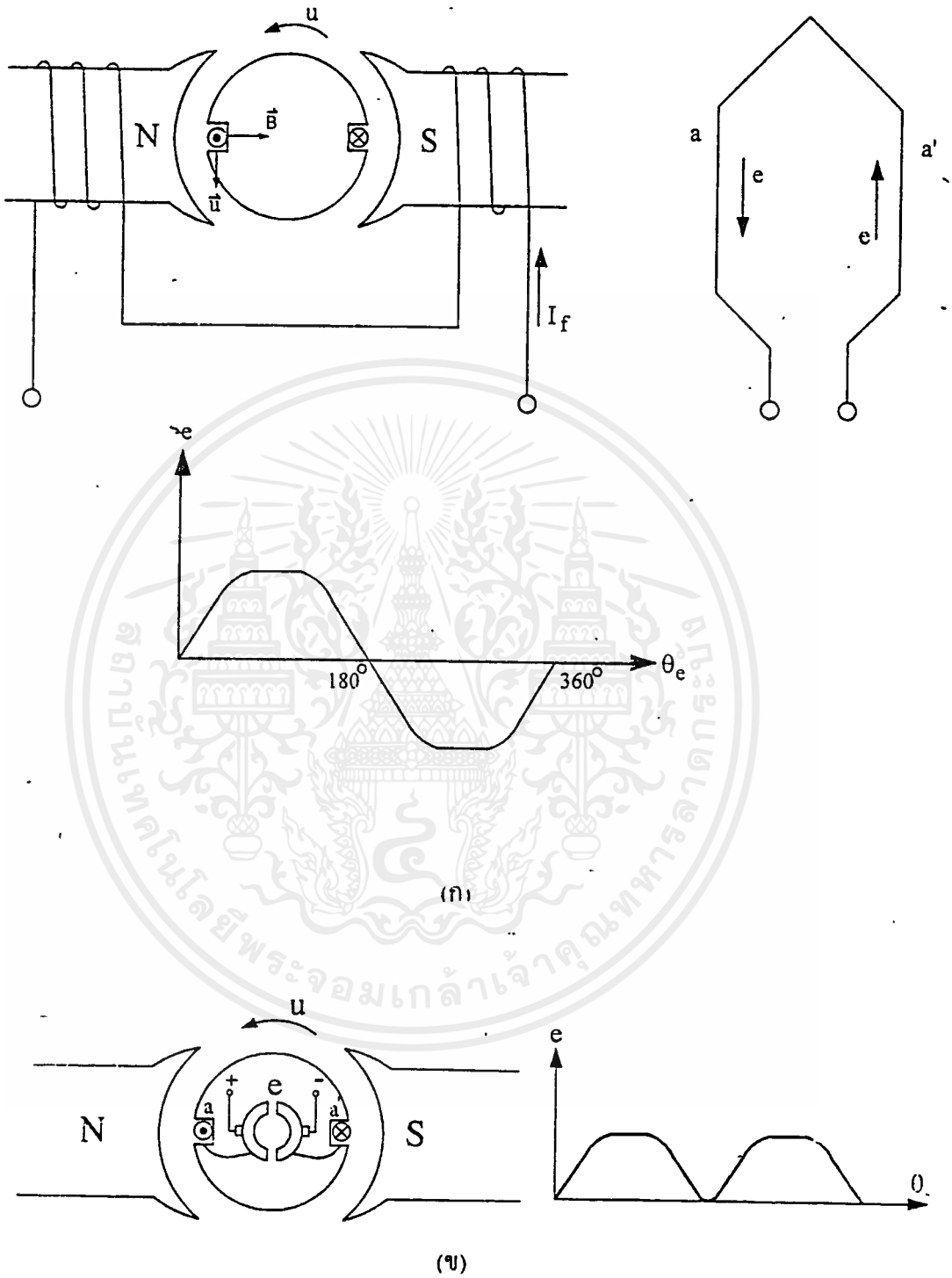
ในทางด้านมุมทางไฟฟ้าดังกล่าวข้างต้นเป็นมุมของสัญญาณไฟฟ้าวัดได้ โดยจะพบว่าเมื่อขดลวดเคลื่อนตัวผ่านขั้วแม่เหล็กครบ 1 คู่ จะทำให้ได้สัญญาณไฟฟ้าออกมา 1 คาบ ดังแสดงตามรูปที่ 2.2 (ก) ซึ่งถ้าเครื่องจักรไฟฟ้ามีจำนวนขั้วแม่เหล็กเป็น P ขั้ว จะได้สัญญาณไฟฟ้าออกมาเป็น P/2 คาบ ซึ่งเขียนเป็นความสัมพันธ์ทางด้านองศาออกมาเป็น

$$\theta_e = \frac{P}{2}\theta_m \qquad (2.1)$$

เมื่อกำหนดให้

- θ_e : เป็นมุมทางไฟฟ้า (Electrical Angle)
- θ_m : เป็นมุมทางกล (Mechanical Angle)

โดยที่ค่ามุมทางกลก็คือมุมที่หัวเพลาของเครื่องจักรไฟฟ้า ถ้าเพลาหมุนครบ 1 รอบ ค่ามุมทางกลเปลี่ยนไปเท่ากับ 360̊ ทางกลดังนั้นถ้าเครื่องจักรมี 2 ขั้วแม่เหล็กจะได้สัญญาณออกมา 1 คาบ หรือ 360̊ ทางไฟฟ้า และถ้าเครื่องจักรไฟฟ้ามี 4 ขั้วแม่เหล็กจะได้สัญญาณออกมา 2 คาบหรือ 2x360̊ ทางไฟฟ้า

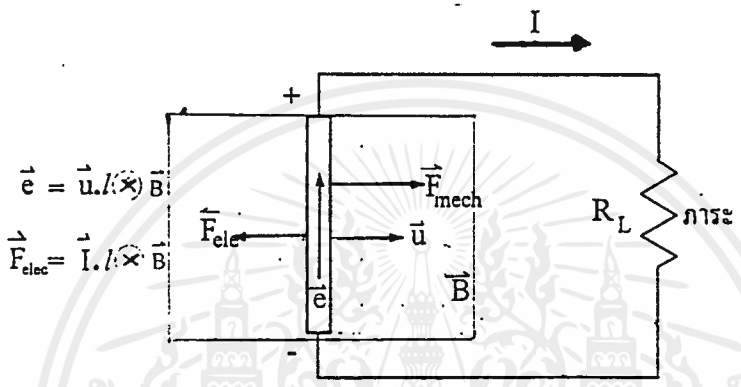


รูปที่ 2.2 ก) แสดงการเหนี่ยวนำศักดาไฟฟ้าในขดลวด

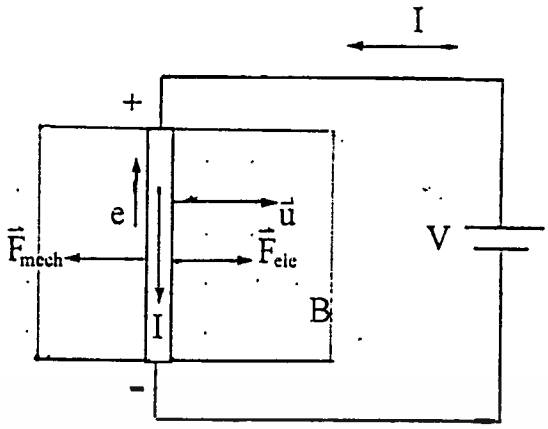
ข) แสดงการใช้วงแหวนครึ่งซีกกับแปรงถ่าน

2.1.3 สมการของเครื่องจักรไฟฟ้ากระแสตรง

ในเครื่องจักรไฟฟ้ากระแสตรงจะมีปรากฏการณ์ที่เกิดขึ้น 2 อย่างคือการเกิดแรงเคลื่อนไฟฟ้า (Induced emf : E_a) และแรงบิดที่เกิดจากสนามแม่เหล็ก (Elector - Magnetic Torque :T_e) โดยที่การเกิดแรงเคลื่อนไฟฟ้า ดังแสดงในรูปที่ 2.3 (ก) เป็นกรณีของเครื่องกำเนิดไฟฟ้า เมื่อแท่งตัวนำที่ยาวเท่ากัน 1 เคลื่อนตัวด้วยความเร็ว U ผ่านสนามแม่เหล็กที่มีความหนาแน่น B จะทำให้เกิดแรงเคลื่อนที่ไฟฟ้า (e)



รูปที่ 2.3 (ก) หลักการเครื่องกำเนิดไฟฟ้า



รูปที่ 2.3 (ข) หลักการมอเตอร์ไฟฟ้า

เมื่อนำภาระมาต่อกับแท่งตัวนำก็จะเกิดกระแสไหลในทิศทางเดียวกับแรงเคลื่อนไฟฟ้า (e) ดังนั้นเมื่อมีกระแสไหลในแท่งตัวนำ (I) อยู่ภายใต้สนามแม่เหล็ก B จะเกิดแรงเนื่องจากสนามแม่เหล็ก (F_{elec}) ถ้าให้แท่งตัวนำเคลื่อนตัวด้วยความเร็วคงที่จะต้องให้แรงทางกล (F_{mech}) จากภายนอกมาช่วยขับในทิศทางความเร็วของตัวนำโดยที่ F_{mech} จะต้องเท่ากับ F_{elec} ถ้าระบบไม่มีความสูญเสีย

$$P_{input} = F_{mech} \cdot U$$

$$= F_{mech} \cdot U \quad = (I/B) \cdot U$$

$$= I \cdot (U/B) \quad = I \cdot e$$

$$P_{input} = P_{output} \qquad (2.2)$$

ในกรณีที่เป็นมอเตอร์ต้องป้อนศักดาไฟฟ้ากระแสตรง (V) เข้าให้กับแท่งตัวนำจะทำให้เกิดกระแสไหลในตัวนำ (I) อยู่ภายใต้สนามแม่เหล็ก (B) จะเกิดแรงเนื่องจากสนามแม่เหล็ก (F_{elec}) เมื่อมีแรงจากสนามแม่เหล็ก เมื่อแท่งตัวนำวิ่งตัดสนามแม่เหล็กจะเกิดการเหนี่ยวนำแรงเคลื่อนไฟฟ้ามีทิศทางตรงข้ามกับศักดาไฟฟ้าป้อนเข้า ระบบจะสมดุลย์เมื่อแรงที่เกิดจากสนามแม่เหล็ก (F_{elec}) เท่ากับแรงที่เป็นภาระทางกล (F_{mech}) ดังแสดงในรูปที่ 2.3 (ข) และแรงเคลื่อนไฟฟ้าในแท่งตัวนำ (e) จะมีค่าเท่ากับศักดาไฟฟ้าป้อนเข้า (V)

จากที่กล่าวมาข้างต้นจะพบว่าไม่ว่าเครื่องจักรไฟฟ้ากระแสตรงจะทำงานเป็นเครื่องกำเนิดไฟฟ้าหรือทำงานเป็นมอเตอร์ไฟฟ้าก็จะเกิดปรากฏการณ์ของ การเกิดแรงเคลื่อนไฟฟ้า

และการเกิดแรงเนื่องจากสนามแม่เหล็ก ซึ่งในเครื่องจักรแบบหมุนจะใช้การเกิดแรงบิดจากสนามแม่เหล็ก

2.1.3.1 การเกิดแรงเคลื่อนไฟฟ้า (E_a) จากหลักการที่ว่าเมื่อแท่งตัวนำเคลื่อนที่ตัดสนามแม่เหล็กจะเกิดแรงเคลื่อนไฟฟ้า จากรูปที่ 2.4 แสดงข้อมูลทางด้านโครงสร้างโดยประมาณ สามารถหาค่าความเร็วของแท่งตัวนำ (U) ที่เคลื่อนตัวในแนวเส้นสัมผัสของวงกลม

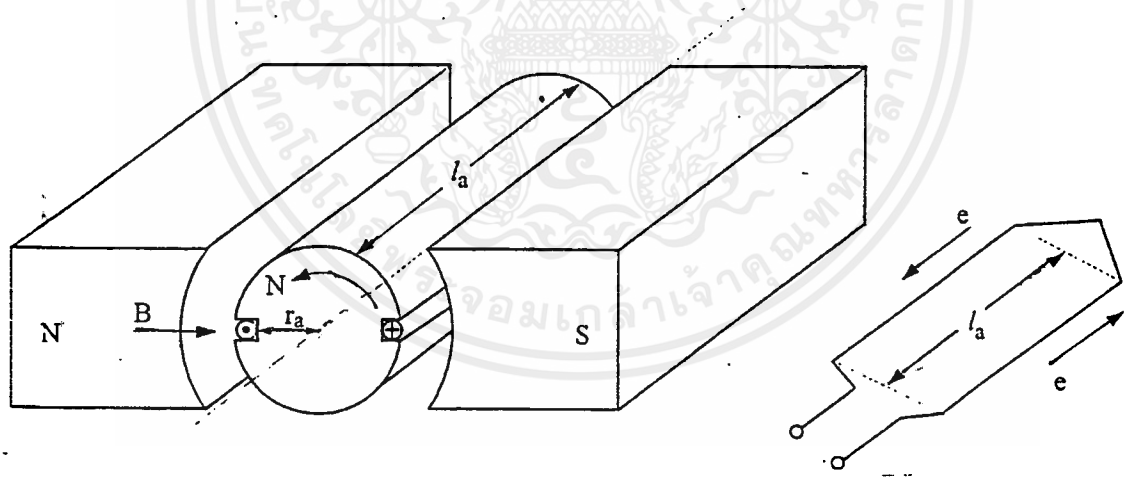
$$U = \frac{2\pi r_a N}{60} \qquad (2.3)$$

เมื่อกำหนดให้

N : เป็นความเร็วรอบของตัวหมุน มีหน่วยเป็น r.p.m.

r_a : เป็นรัศมีของอาเมเจอร์ มีหน่วยเป็น m.

ในส่วนของความหนาแน่นของสนามแม่เหล็ก (B_{av}) ของแต่ขั้วแม่เหล็ก ซึ่งก็จะเป็นค่าเฉลี่ยของเส้นแรงแม่เหล็กในแต่ละขั้ว (ϕ_p) หารด้วยพื้นที่ของหน้าขั้วแม่เหล็ก โดยที่พื้นที่ของขั้วแม่เหล็ก หาได้จากพื้ นที่ของขั้วแม่เหล็ก หาได้จากพื้นที่ผิวของอาเมเจอร์ ($2\pi r_a l_a$) หารด้วยจำนวนขั้วแม่เหล็ก (P)



รูปที่ 2.4 ขนาดโครงสร้างของเครื่องจักรไฟฟ้ากระแสตรง

จะได้ว่า

$$B_{av} = \frac{\phi_p P}{2\pi r_a l_a} \qquad (2.4)$$

เมื่อกำหนดให้

l_a : เป็นความยาวของแท่งตัวนำ มีหน่วยเป็น m

ระยะทาง l_a เป็นความยาวตามแนวแกนของอาเมเจอร์ ซึ่งอยู่ภายใต้สนามแม่เหล็ก ดังนั้นจะเป็นความยาวของแท่งตัวนำที่เกิดการเหนี่ยวนำศักดาไฟฟ้า หรือการเกิดแรงเนื่องจากสนามแม่เหล็ก

เนื่องจากทิศทางการเคลื่อนที่ของแท่งตัวนำตั้งฉากกับทิศทางสนามแม่เหล็ก (B_{av}) ทำให้สามารถหาค่าแรงเคลื่อนไฟฟ้าที่เกิดขึ้นในแต่ละแท่งตัวนำ (e) จะได้ว่า

$$e = U.l_a.B_{av}$$
$$= N\frac{2\pi r_a l_a}{60}.\frac{\phi_p.P}{2\pi r_a l_a}$$

เพราะฉะนั้น แรงเคลื่อนไฟฟ้าต่อแท่งตัวนำ $= \frac{N.P}{60}\phi_a$ (2.5)

จากการพันขดลวดอาเมเจอร์ซึ่งจะทำให้ทราบถึงจำนวนแท่งตัวนำทั้งหมดในอาเมเจอร์ (Z_a) และจำนวนวงจรขนาด (a) ทำให้สามารถหาจำนวนแท่งตัวนำที่ต่ออนุกรมกัน

จำนวนแท่งตัวนำที่ต่ออนุกรม $= \frac{Z_a}{a}$

ดังนั้นแรงเคลื่อนที่ไฟฟ้าที่เกิดขึ้นในขดลวดอาเมเจอร์ (E_a) จะเท่ากับผลคูณของจำนวนแท่งตัวนำที่ต่อนุกรมกับแรงเคลื่อนไฟฟ้าที่เกิดขึ้นในแต่ละแท่งตัวนำ จะได้ว่า

$$E_a = e.\frac{Z_a}{a}$$
$$= \frac{N.P}{60}\phi_p\frac{Z_a}{a}$$

เพราะฉะนั้น $E_a = \frac{Z_a P}{60a}\phi_p N$ (2.6)

ซึ่งจะเป็นสามการที่ใช้ในการคำนวณหาค่าการเกิดแรงเคลื่อนไฟฟ้าจากอาเมเจอร์ แต่ในบางครั้งการกำหนดข้อมูลในสมการทางด้านความเร็วเชิงมุม (ω) มีหน่วยเป็นเรเดียนต่อวินาที ซึ่งความสัมพันธ์กับความเร็วรอบ (N) ได้ดังนี้

$$\omega = \frac{2\pi N}{60} \quad (2.7)$$

แทนค่าจากสมการที่ (2.7) ลงในสมการที่ (2.6) สรุปได้ว่า

$$\boxed{E_a = \frac{Z_a.P}{2\pi a}\phi_p\omega} \quad (2.8)$$

ซึ่งสมการที่ (2.8) จะสะดวกเมื่อนำมาใช้อธิบายร่วมกับแรงบิดที่เกิดจากสนามแม่เหล็ก

2.1.3.2 แรงบิดที่เกิดจากสนามแม่เหล็ก (T_e)

จากหลักการที่กล่าวมาข้างต้นว่าเมื่อมีกระแสไหลในแท่งตัวนำจะเกิดแรงที่กระทำบนแท่งตัวนำ และจากโครงสร้างของเครื่องจักรไฟฟ้ากระแสตรง กระแสไฟฟ้าที่ไหลในแท่งตัวนำตั้งฉากกับสนามแม่เหล็ก จะได้แรงที่กระทำต่อแท่งตัวนำอยู่ในทิศทางเส้นสัมผัสของวงกลมทำให้เครื่องจักรไฟฟ้ากระแสตรงหมุน

ดังนั้นแรงที่กระทำในแต่ละแท่งตัวนำที่ยาวเท่ากับ l_a และมีกระแสไฟฟ้าที่ไหลในแท่งตัวนำเท่ากับ (I_a) จะได้แรงในแต่ละแท่งตัวนำ

$$F = I_a . l_a . B_{av} \qquad (2.9)$$

โดยที่บนตัวหมุนมีจำนวนแท่งตัวนำทั้งหมดเท่ากับ Z_a และมีวงจรขนานเท่ากับ a ซึ่งมีกระแสไฟฟ้าไหลเข้าขดลวดอาเมเจอร์เท่ากับ I_a ดังนั้นกระแสที่ไหลในแต่ละแท่งตัวนำ (I_c) จะได้ว่า

$$I_c = \frac{I_a}{a} \qquad (2.10)$$

และแรงที่เกิดขึ้นทั้งหมดบนตัวหมุน (F_t) จะเท่ากับจำนวนแท่งตัวนำทั้งหมด (Z_a) คูณด้วยแรงที่เ กิดขึ้นในแต่ละแท่งตัวนำ

$$F_t = Z_a . F$$

แทนสมการด้วยสมการที่ (2.4) , (2.9) , (2.10)

$$F_t = Z_a \frac{I_a}{a} l_a \frac{\phi_p . P}{2\pi . r_a . l_a}$$

$$= \frac{Z_a . P}{2\pi . a} . \frac{1}{r_a} . \phi_p . I_a \qquad (2.11)$$

เนื่องจากเครื่องจักรไฟฟ้ากระแสตรงที่กล่าวถึงเป็นเครื่องจักรกลชนิดหมุนการอธิบายทางด้านแรงจึงนิยมใช้เป็นแรงบิด (Torque) เนื่องจากมุมระหว่างแรงที่กระทำกับแท่งตัวนำกับ รัศมีตั้งฉากกัน จึงสามารถเขียนสมการของแรงบิดที่เกิดจากสนามแม่เหล็ก (T_e) เป็น

$$T_e = F_t r_a$$

$$\boxed{T_e = \frac{Z_a . P}{2\pi a} \phi_p I_a} \qquad (2.12)$$

ซึ่งสามารถสรุปได้ว่าเมื่อเครื่องจักรไฟฟ้ากระแสตรงไม่ว่าจะทำงานเป็นเครื่องกำเนิดไฟฟ้าหรือมอเตอร์ไฟฟ้าเมื่อมีกระแสไหลในขดลวดอาเมเจอร์และมีสนามแม่เหล็ก เหล็กจากขั้วแม่เหล็กจะมีแรงบิดที่เกิดจากสนามแม่เหล็ก

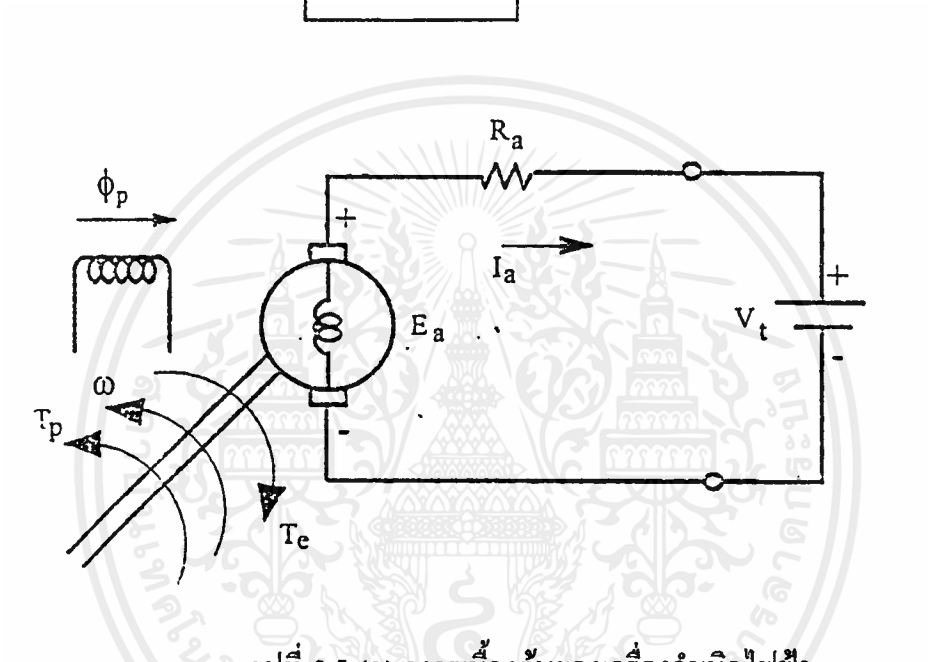
2.1.4 การแปลงรูปพลังงานไฟฟ้าในเครื่องจักรไฟฟ้ากระแสตรง

เครื่องจักรไฟฟ้ากระแสตรงเป็นอุปกรณ์ แปลงรูปพลังงาน โดยเมื่อแปลงรูปพลังงานกลเป็นพลังงานไฟฟ้าเรียกว่า เครื่องกำเนิดไฟฟ้าและเมื่อแปลงรู ปพลังงานไฟฟ้าเป็นพลังงานกลเรียกว่า มอเตอร์ไฟฟ้า ซึ่งเครื่องจักรไฟฟ้ากระแสตรงจะแปลงพลังงานกลในลักษณะของการหมุนจากสมการที่ (2.8) และ (2.12) กำหนดให้

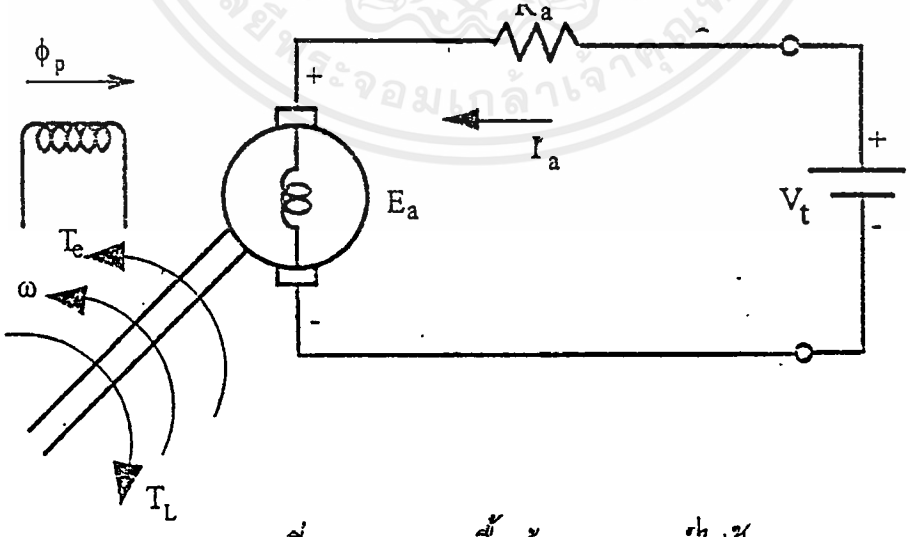
$$k_t = \frac{Z_a . P}{2\pi a}$$

$$\boxed{E_a = k_t \phi_p \omega} \qquad (2.13)$$

และ $$\boxed{T_e = k_t . \phi_p . I_a} \qquad (2.14)$$



รูปที่ 2.5 (ก) วงจรเบื้องต้นของเครื่องกำเนิดไฟฟ้า



รูปที่ 2.5 (ข) วงจรเบื้องต้นของมอเตอร์ไฟฟ้า

ในกรณีที่เป็น เครื่องกำเนิดไฟฟ้า ดังแสดงในรูปที่ 2.5 (ก) เมื่อคิดว่าระบบไม่มีความสูญเสียทางกล $T_p = T_e$ และ ไม่มีความสูญเสียทางไฟฟ้า $R_a = 0$

กำลังทางกล (Mechanical Power) $= T_p \omega$

$= T_e \omega$

$= K_t . \phi_p . I_a . \omega$

กำลังทางไฟฟ้า (Electrical Power) $= E_a . I_a$

$= K_t . \phi . I_a . \omega$

$T_e \omega = E_a I_a$

สรุปได้ว่า

กำลังทางกล	=	กำลังทางไฟฟ้า
$E_a I_a$	=	$Te \omega$

(2.15)

ในทำนองเดียวกันเมื่อเป็ นมอเตอร์ไฟฟ้า ดังแสดงในรูปที่ 2.5 (ข) สามารถสรุปได้ว่า

$$T_e \omega = E_a . I_a$$

ซึ่งเป็นการแสดงการแปลงรูปพลังงานระหว่างพลังงานกลกับพลังงานไฟฟ้าในเครื่องจักรไฟฟ้ากระแสตรงโดยถ้าไม่มีความสูญเสียทางกล และความสูญเสียทางไฟฟ้า ในกรณีเครื่องกำเนิดไฟฟ้ากระแสตรง

กำลังไฟฟ้าป้อนเข้า $(V_t : I_a)$ = กำลังทางกลจ่ายออก $(T_p . \omega)$

และกรณีมอเตอร์ไฟฟ้ากระแสตรง

กำลังไฟฟ้าป้อนเข้า $(V_t . I_a)$ = กำลังทางกลจ่ายออก $(T_L . \omega)$

ส่วนเมื่อมีความสูญเสียทางกลและมีความสูญเสียทางไฟฟ้า ดังนั้นพลังงานที่ป้อนเข้าจะต้องมากกว่าพลังงานที่จ่ายออก โดยแยกเป็นกรณี

ในกรณีที่เป็นเครื่องกำเนิดไฟฟ้ากระแสตรงดังแสดงในรูปที่ (2.5)(ก) แรงบิดทางกลที่ขับเครื่องกำเนิดไฟฟ้า (T_p) จะต้องมากกว่าแรงบิดที่เกิดจากสนามแม่เหล็ ก (T_e) โดยส่วนมากกว่าต้องมาชดเชยแรงบิดที่เกิดจากความสูญเสียทางกล $(T_{mech,loss})$ โดยที่ทั้ง T_e และ $T_{mech,loss}$ จะมีทิศทางตรงข้ามกับทิศทางการหมุนหรือพูดง่าย ๆ ว่าต้านการหมุนจะได้ว่า

$$T_p = T_e + T_{mech,loss} \qquad (2.16)$$

ทางด้านวงจรไฟฟ้าเมื่อเป็นเครื่องกำเนิดไฟฟ้าทิศทางการไหลของกระแสจะต้องไหลจ่ากขั้วบวกของการเกิดแรงเคลื่อนไฟฟ้า (E_a) ที่ขดลวดอาร์เมเจอร์แล้วไหลสู่วงจรภายนอก

โดยที่ E_a จะต้องมากกว่าศักดาไฟฟ้าที่ขั้ว (V_t) และจากรูปที่ 2.5 (ก) จะแสดงถึงแบตเตอรี่ที่ทำเป็นภาระคืออยู่ในกรณีของการประจุแบตเตอรี่และตามกฎของเคอร์ชอฟ (Kirchhoff ' s law) จะได้ว่า

$$E_a = I_a R_a + V_t$$

ส่วนกรณีที่เป็นมอเตอร์ไฟฟ้ากระแสตรงดังแสดงตามรูปที่ 2.5 (ข) ซึ่งเป็นการแปลงรูปพลังงานไฟฟ้าเป็นพลังงานกล เมื่อมีความสูญเสียดังนั้น ค่าของศักดาไฟฟ้าที่ขั้ว (V_t) จะต้องมากกว่า E_a โดยที่กระแสจะไหลออกจากขั้วบวกของแบตเตอรี่เข้าสู่วงจรอาเมเจอร์โดยที่

$$V_t = E_a + I_a . R_a$$

ซึ่งเมื่ออยู่ในรูปพลังงานกลแล้ว แรงบิดที่เกิดจากสนามแม่เหล็ก (T_e) จะต้องมากกว่าแรงบิดที่ขับภาระ (T_L) โดยไปชดเชยความสูญเสียทางกล ($T_{mech,loss}$) ซึ่งสามารถเขียนได้ว่า

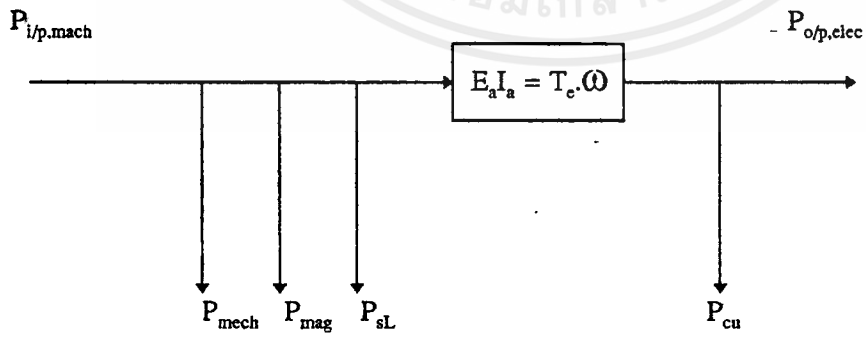
$$T_e = T_L + T_{mech,loss} \qquad (2.17)$$

สรุปได้ว่าถึงแม้ว่าจะมีความสูญเสียทางกลและทางไฟฟ้าเกิดขึ้นในตัวเครื่องจักรไฟฟ้ากระแสตรงก็ตาม ในส่วนของการแปลงรูปพลังงาน แล้วยังคงเดิมตามสมการที่ (2.15) โดยกำลังทาง กลเท่ากับกำลังทางไฟฟ้า

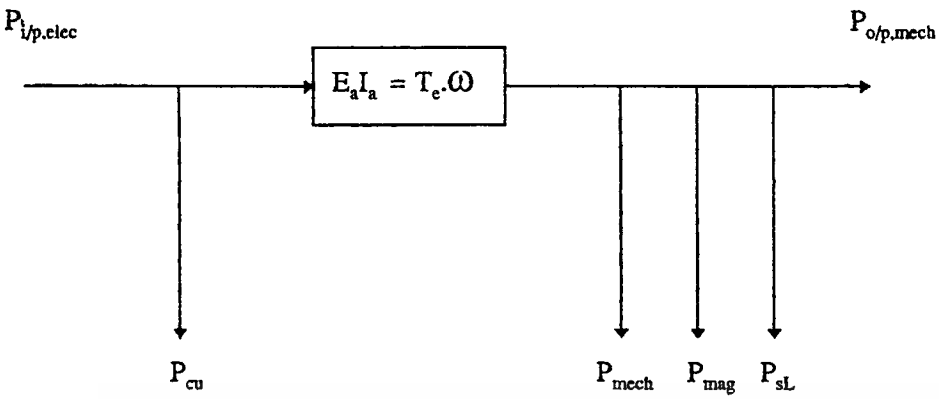
$$\boxed{E_a . I_a = T_e . \omega}$$

2.1.5 ความสูญเสียในเครื่องจักรไฟฟ้าและกระแสตรง (Loss in PC Machine)

ในทางด้านการแปรงรูปพลังงานของเครื่องจักรไฟฟ้ากระแสตรงจะมีความสูญเสียพลังงานทั้งทางด้านพลังงานกลและพลังงานไฟฟ้า ดังแสดงดังรูปที่ 2.6 โดยมีรายละเอียดของความสูญเสียในกรณีที่เป็นเครื่องกำเนิดไฟฟ้าซึ่งกำลังงานป้อนเป็นกำลังทางกล ถ้าเป็นเครื่องกำเนิดไฟฟ้าแบบแยกขดลวดสร้างสนามแม่เหล็กกำลังงานป้อนเข้าทางด้านไฟฟ้าด้วย คือ ป้อนเข้าให้ขดลวดสร้างสนามแม่เหล็กจะมีค่าเท่ากับ ส่วนทางด้านกำลังส่งออกจะเป็นทางด้านไฟฟ้า



รูปที่ 2.6 (ก) ความสูญเสียในเครื่องกำเนิดไฟฟ้ากระแสตรง



รูปที่ 2.6 (ข) ความสูญเสียในมอเตอร์ไฟฟ้ากระแสตรง

ส่วนถ้าเป็นมอเตอร์ไฟฟ้ากระแสตรง แสดงตามรูปที่ 2.6 (ข) ซึ่งจะมีกำลังป้อนเข้าทางไฟฟ้าและกำลังส่งออกทางกลโดยที่ขั้ นตอนการคำนวณของทั่วเครื่องกำเนิดไฟฟ้า กระแสตรง และมอเตอร์ไฟฟ้ากระแสตรงจะขึ้นอยู่กับชนิดของเครื่องจักรนั้น ๆ ในทางด้านรายละเอียดของความสูญเสียในเครื่องจักรไฟฟ้ากระแสตรงประกอบด้วย

2.1.5.1 กำลังสูญเสียทางกล (Machanical loss : P_{mech}) เป็นความสูญเสียที่เกิดจากแรงของลมที่ใบพัดระบายความร้อนเพื่อระบายความร้อนที่เกิดจากขดลวดและแถบเหล็ก ความสูญเสียเนื่องจากความฝืด ระหว่างแปรงถ่านกับคอมมิวเตเตอร์ รวมไปถึงความฝื ดระหว่างตัวรองลื่นและลูกปืน

2.1.5.2 กำลังสูญเสียจากสนามแม่เหล็ก (Magnetic Loss : P_{mag}) เป็นความสูญเสียที่เกิดขึ้นในแกนเหล็ก เนื่องจากสนามแม่เหล็กในแถบเหล็กมีการเปลี่ยนแปลงกับเวลา ดังนั้นแกนเหล็กของ อาเมเจอร์ จะมีค่าความสูญเสีย ที่หน้าขั้วแม่เหล็ก ก็ มีความสูญเสียเนื่องจากร่องสล๊อต (Slot)

2.1.5.3 กำลังสูญเสียคงที่ (Stay Load Loss :P_{SL}) เป็นความสูญเสียเนื่องจากสภาพความไม่เรียบร้อย (Non Uniform) ภายในเครื่องจักรยกตัวอย่างเช่น สกินเอฟเฟ็ก (Skin Effect) ในขดลวดของอาเมเจอร์ทำให้กระแสที่ไหลในตัวนำมีความหนาแน่นไม่ เท่ากันตลอดพื้นที่หน้าตัด ทำให้ มีผลเหมือนกับความต้านทานเพิ่มขึ้นหรือทางด้านสนามแม่เหล็กที่ปิดรูปไปเนื่องจากกระแสที่ไหลในขดลวดอาเมเจอร์ ทำให้เกิดจากความสูญเสียในแกนเหล็กเพิ่มขึ้น ซึ่งยากแก่การคำนวณโดยทั่วไปจะใช้ประมาณ 1% ของกำลังขาออกของเครื่องจักรไฟฟ้ากระแสตรง เพื่อที่จะให้ผลการคำนวณกับการทดสอบออกมาถูกต้องยิ่งขึ้น

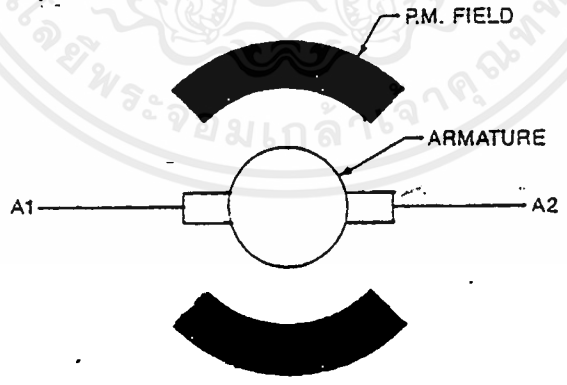
2.1.5.4 ความสูญเสียจากค่าความต้นทาน (Copper Loss : P_{cu}) เป็นความสูญเสียที่เกิดขึ้นทางด้านวงจรไฟฟ้าเมื่อมีกระแสไหลผ่านลวด ตัวนำที่มีค่าความต้านทานจะมีกำลังที่มีค่าความต้านทาน จะมีกำลังที่สูญเสียบางครั้งเรียกว่า ไอกำลังสองอาลอสส์ (I^2R *Loss*)ซึ่งจะประกอบด้วยความสูญเสียของขดลวดต่าง ๆ เช่น ขดลวดอาเมเจอร์ ขดลวดสร้างสนามแม่เหล็ก ขดลวดคอมเพรสเสท (Compensate) ขดลวด อินเตอร์โพนท์ (Inter Pole) ซึ่งจะเป็นผลส่วนหนึ่งที่ทำให้เครื่องจักรไฟฟ้ากระแสตรงร้อนและจำเป็นต้องระบายความร้อนออกจากขดลวด

2.1.5.5 ความสูญเสียจากการหมุน (Rotational Loss : P_{rot}) เป็นการร่วมค่าความสูญเสียกำลัง เนื่องจากการหมุนทั้งหมดซึ่งง่ายต่อการทดสอบและการคำนวณโดยที่

$$P_{rot} = P_{mech} + P_{mag} + P_{SL} \qquad (2.18)$$

2.1.6 การทำงานด้วยแม่เหล็กถาวร (Permanent Magnet Operated)

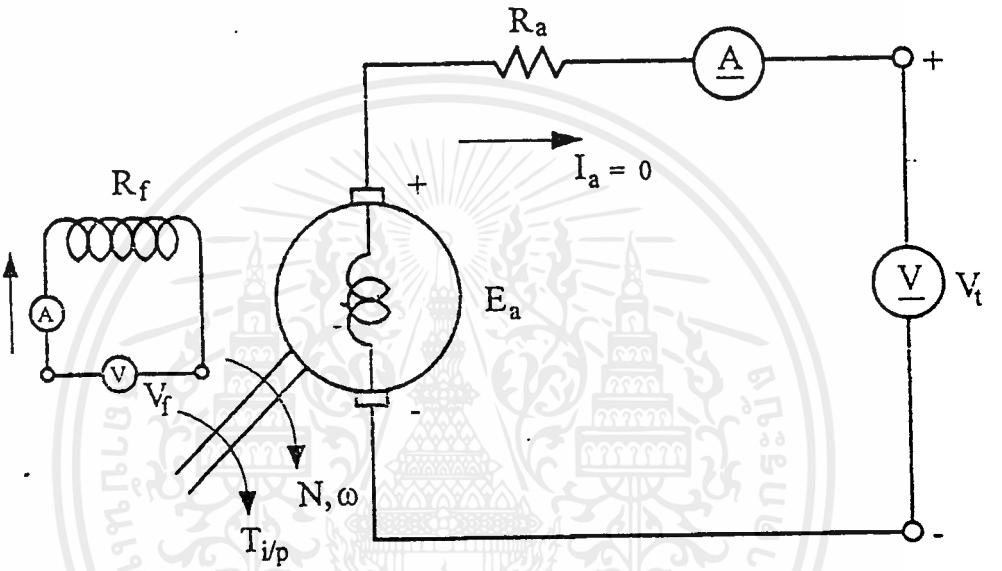
มอเตอร์แบบแม่เหล็กถาวรมีอาร์เมเจอร์แบบวาล์ว (Wound Armature) พร้อมทั้งคอมมิวเตเตอร์ (Commutator) และแปรงถ่าน (Brushes) ดังรูปที่ ซึ่งมอเตอร์ชนิดนี้จะให้ทอล์คคอนสตาร์ท (Starting Torque)ที่ดีเยี่ยมพร้อมด้วยการปรับความเร็ว (Speed Regulation) จำเป็นต้องอยู่ภายใต้เงื่อนไขของมอเตอร์แบบคอมเปาด์ (Compound Motor) เพราะว่าการสูญเสียของสนามแม่เหล็กถาวรและมอเตอร์มีค่าน้อยและทำให้ประสิทธิภาพการทำงานที่ได้รับมีค่ามาก มอเตอร์ชนิดนี้สามารถเบรกได้ด้วยการเบรคแบบไดนามิก (Dynamically Braked) และสามารถกลับทางหมุนได้ที่โวลเตจต่ำ ๆ (10%) ไม่ควรที่จะกลับทางหมุนในขณะที่แรงดันที่อาร์มาเจอร์เต็มพิกัดค่ากระแสตอนกลับทางหมุนใกล้เคียงกับค่ากระแสตอนอาเมเจอร์อยู่กับที่



รูปที่ 2.7 แสดงโครงสร้างภายในของมอเตอร์ดีซีแบบแม่เหล็กถาวร

2.1.7 การทดสอบขณะไร้ภาวะ (No load test)

ในการทดสอบเครื่องจักรไฟฟ้ากระแสตรงนั้นการทดสอบที่สำคัญวิธีหนึ่งที่ทำให้สามารถเข้าใจความสัมพันธ์ระหว่างค่าการเหนี่ยวนำแรงเคลื่อนไฟฟ้า (E_a) กระแสไฟฟ้าสร้างสนามแม่เหล็ก (I_f) และความเร็วรอบ (N) หรือความเร็วเชิงมุม(ω) โดยวิธีนี้กระทำโดยการขับเครื่องไฟฟ้ากระแสตรงให้เป็นเครื่องกำเนิดไฟฟ้ากระแสตรงแบบแยกขดลวดสร้างสนามแม่เหล็กดังแสดงตามรูปที่ 2.8โดยต้องมีแหล่งพลังงานกลภายนอกมาช่วยขับตัวหมุนของเครื่องกำเนิดไฟฟ้าให้หมุนด้วยความเร็วที่พิกัด



รูปที่ 2.8 วงจรการทดสอบขณะไร้ภาระพร้อมเครื่องวัด

ขณะที่กระแส $I_f = 0$ เมื่อขับด้วยความเร็วรอบที่พิกัด แรงบิดที่ป้อนเข้า $T_{i/p}$ จะเป็นส่วนที่มาชดเชยกับความสูญเสียเนื่องจากกำลังความสูญเสียทางกล P_{mach} โดยที่

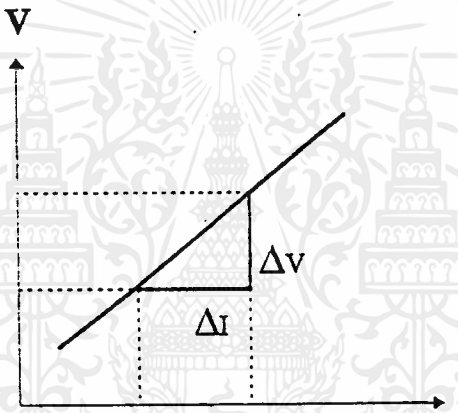
$$P_{mach} = T_{i/p}.\omega$$

และในทางด้านศักดาไฟฟ้าที่ขั้ว V_t จะมีค่าเท่ากับค่าการเกิดแรงเคลื่อนไฟฟ้า E_a เนื่องจากกระแสของขดลวดอาเมเจอร์ I_a เท่ากับศูนย์ ซึ่งค่า V_t ที่วัดได้จากดีซีโวลต์มิเตอร์จะเป็นค่าการเกิดแรงเคลื่อนไฟฟ้า E_r ที่เกิดจากเส้นแรงแม่เหล็กตกค้าง ϕ_r โดยที่ค่าแรงเคลื่อนไฟฟ้า E_r เป็นตัวที่จะทำให้เครื่องกำเนิดไฟฟ้าทำงานในลักษณะกระตุ้นด้วยตัวเอง ถ้าขณะนี้คือ $I_f = 0$ แล้วค่า

ศักดาไฟฟ้าที่ขั้วเท่ากับศูนย์ แสดงว่าเป็นเครื่องจักรไฟฟ้ากระแสตรงไม่ได้ใช้งานนานเกินไปจนสนามแม่เหล็กตกค้างลดลงเป็นศูนย์ หรือถูกต่อผิดทำให้เกิดการทำลายสนามแม่เหล็กตกค้าง หรืออาจจะเป็นเครื่องจักรผลิตใหม่ยังไม่เคยป้อนกระแสสร้างสนามแม่เหล็ก

2.1.8 วิธีโวลต์แอมป์ (VI - Method)

ในการเขียนวงจรเสมือนของเครื่องจักรไฟฟ้ากระแสตรงจำเป็นต้องรู้ค่าความต้านทานของขดลวดทั้งขดลวดอาเมเจอร์ ซึ่งรวมค่าความต้านทานของแปรงถ่านและขดลวดสร้างสนามแม่เหล็ก ดังนั้นจำเป็นต้องวัดค่าความต้านทานของขดลวดเนื่องจากค่าความต้านทานของอาเมเจอร์ต่ำมาก เพื่อความแม่นยำจึงใช้วิธีโวลต์แอมป์ โดยการป้อนศักดาไฟฟ้ากระแสตรงและวัดกระแสไฟฟ้าที่ให้ผ่านขดลวด ทำการบันทึกข้อมูลโดยที่กระแสต้องไม่เกินพิกัดที่ขดลวดทนได้ นำมาเขียนกราฟความสัมพันธ์ระหว่างศักดาไฟฟ้า V และ กระแส I ดังแสดงตามรูปที่ 2.9 ซึ่งสามารถนำมาคำนวณหาค่าความต้านทานได้จากค่าความลาดชันของเส้นตรง



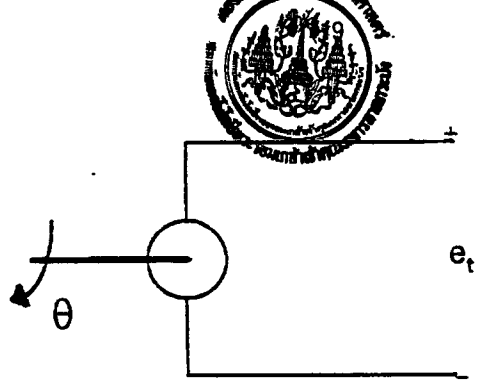
รูปที่ 2.9 เป็นกราฟความสัมพันธ์ระหว่าง V กับ I . ในการทดสอบหาค่าความต้านทาน

ซึ่งได้ค่าความต้านทานของขดลวด(R)

$$R = \Delta V / \Delta I$$

2.2 เทคโคมิเตอร์ (Tachometer)

เทคโคมิเตอร์ (Tachometer) เป็นอุปกรณ์ไฟฟ้าเชิงกล (Electromachnical) ซึ่งแปลงพลังงานเชิงกลไปเป็นพลังงานไฟฟ้า อุปกรณ์นี้จะถือเป็นเสมือนเครื่องกำเนิดไฟฟ้า (Generator) โดยที่แรงดันไฟฟ้าด้านไฟออกเป็นสัดส่วนกับขนาดของความเร็วเชิงมุม ในระบบควบคุมจะใช้เทคโคมิเตอร์ในการตรวจวัดความเร็วของแกนหรือเพลาและใช้ในการปรับปรุงสมรรถนะของระบบให้ดีขึ้นโดยทั่ว ไปแล้ว เทคโคมิเตอร์อาจจะแยกออกได้เป็นสองแบบคือ แบบกระแสสลับและกระแสตรง รูปที่ 2.10 แสดงถึงไดอะแกรมโครงสร้าง (Schematic diagram)ของเทคโคมิเตอร์



รูปที่ 2.10 แสดงไดอะแกรมโครงสร้างของเทคโคมิเตอร์

สำหรับเทคโคมิเตอร์แบบกระแสสลับแรงดันไฟฟ้ารูปคลื่นไซน์จะถูกป้อนให้กับขดลวดปฐมภูมิ (Primary Winding) และขดลวดทุติยภูมิ (Secondary Winding) นั้นจะถูกวางให้ทำมุม 90 องศากับขดลวดปฐมภูมิ เมื่อโรเตอร์ (Rotor) ของเทคโคมิเตอร์อยู่กับที่ แรงดันไฟฟ้าด้านไฟออกที่ขดลวดทุติยภูมิจะมีค่าเท่ากับศูนย์ แต่เมื่อแกนของโรเตอร์หมุนไปแรงดันไฟฟ้าด้านไฟออกของเทคโคมิเตอร์จะเป็นสัดส่วนกับความเร็วของโรเตอร์โดยที่ขั้ว(Polarity) ของแรงดันไฟฟ้าจะขึ้นอยู่กับทิศทางของการหมุน

ความสัมพันธ์ระหว่างเข้าและออกของเทคโคมิเตอร์แบบกระแสสลับสามารถจับแสดงได้ด้วยสมการดิฟเฟอเรนเชียลอันดับหนึ่งต่อไปนี้

$$e_t(t) = k_t \frac{d\theta(t)}{dt} \qquad (2.19)$$

โดยที่ $e_t(t)$ คือ แรงดันไฟฟ้าด้านไฟออก

$\theta(t)$ คือ ระยะแทนที่ของโรเตอร์

k_t คือ ค่าคงที่ของเทคโคมิเตอร์ (โวลท์/ rpm หรือ โวลท์ / 1000 rpm)

ค่าทรานเฟอร์ฟังชั่น (Tranfer Function) ของเทคโคมิเตอร์แบบกระแสสลับจึงมีค่าดังนี้

$$\frac{E_{t(s)}}{\theta_{(s)}} = k_t S \qquad (2.20)$$

สำหรับเทคโคมิเตอร์แบบกระแสตรงก็จะใช้เหมือนกับเทคโคมิเตอร์แบบกระแสสลับที่กล่าวมาข้างต้นแต่ข้อดีข้อหนึ่งของเทคโคมิเตอร์ แบบกระแสตรงก็คือสนามแม่เหล็กของเทคโคมิเตอร์นั้นสามารถทำขึ้นได้โดยใช้แม่เหล็กถาวร จึงไม่จำเป็นจะต้องมีแรงดันไฟฟ้ามากระตุ้นและ

สมการที่ 2.19 กับสมการ 2.20 ก็ใช้กับเทคโคมิเตอร์แบบกระแสตรงได้ในระบบควบคุมนั้น สามารถจะใช้ เทคโคมิเตอร์แบบกระแสตรงแทนเทคโคมิเตอร์แบบกระแสสลับได้โดยการเพิ่มวงจรโมดูเลเตอร์ (Modulator) เพื่อแปลงสัญญาณไฟออกแบบกระแสตรงไปเป็นแบบกระแสสลับในทางกลับกันเทคโคมิเตอร์แบบกระแสตรงก็อาจจะแทนด้วยเทคโคมิเตอร์แบบกระแสสลับได้โดยใช้เฟสเซ็นซิทีฟดิโมดูลเลเตอร์ (Phase Sensitive Demodulator) เพิ่มเข้าไป เพื่อแปลงเอาท์พุทแบบกระแสสลับให้เป็นกระแสตรง

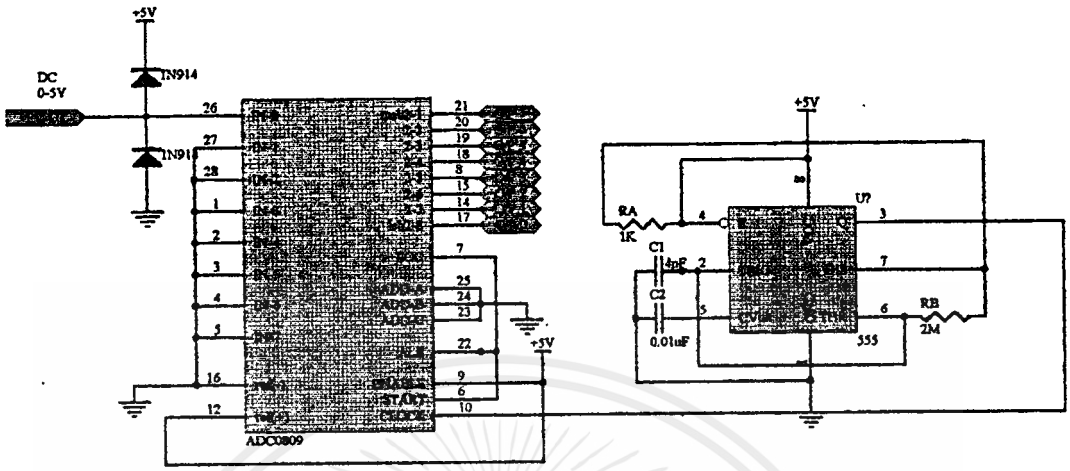
2.3 วงจรแปลงอะนาล็อกเป็นดิจิตอล

วงจรแปลงอะนาล็อกเป็นดิจิตอล(Analog to Digital Converter)จะทำหน้าที่แรงดันหรือกระแสที่เป็นสัญญาณอะนาล็อกไปเป็นตัวเลขหรือสัญญาณดิจิตอล วงจรแปลงอะนาล็อกเป็นดิจิต่อลมีด้วยกันหลายแบบแต่ที่นิยมใช้กันแพร่หลายมี 3 แบบคือ แบบสโลปคู่ (Dual Slope) แบบแปลงแรงดันเป็นความถี่ (V to F Converter) และประมาณทีละบิท (Successive Approximation)

วงจรแปลงอะนาล็กเป็นดิจิตอล แบบสโลปคู่เป็นแบบที่ง่ายที่สุด ไม่จำเป็นต้องใช้อุปกรณ์ที่มีคุณภาพดีมากก็สามารถแปลงสัญญาณได้อย่างแม่นยำ แต่มีข้อเสียนิดเดียวตรงที่ใช้เวลาในการแปลงสัญญาณนานมากไปหน่อย จึงไม่เหมาะในการใช้วัดแรงดันในช่วงเวลาสั้นๆ เช่น การวัดแรงดันของรูปคลื่น ณ. จุดเวลาใดเวลาหนึ่ง แบบสโลปคู่นี้เหมาะสำหรับใช้วัดค่าเฉลี่ยของแรงดันและกระแส จึงนิยมใช้กันมากในมัลติมิเตอร์แบบดิจิตอล และเครื่องวัดแสดงผลเป็นตัวเลขทั่วไป วงจรแปลงอะนาล็อกเป็นดิจิตอลแบบสโลปคู่ที่มีทั้งแบบแปลงเป็นตัวเลขขนาด 3(1/2) หลัก (แสดงผลสูงสุดเป็น 1999) และ 4(1/2) หลัก (แสดงผลสูงสุดเป็น 19999) หลักสุดท้ายจะแสดงค่าเป็น 0 หรือ 1 เท่านั่น จึงเรียกง่าย ๆ ว่า½ หลัก

วงจร แปลงอะนาล็อกเป็นดิจิตอลแบบแปลงแรงดันเป็นความถี่และแบบประมาณทีละบิทนั้นมีข้อดีตรงที่สามารถแปลงสัญญาณได้รวดเร็ว มีความแม่นยำดี เพียงแต่วงจรมีความซับซ้อนมากกว่าจึงมีราคาแพง ในปัจจุบัน เทคโนโลยีในการไอซีเจริญขึ้นมากเราสามารถหาซื้อไอซีที่เป็นวงจรอะนาล็อกเป็นดิจิตอลแบบนี้ได้ง่ายขึ้นและในราคาไม่แพงนัก และในอนาคตราคาจะถูกลงอีกมาก

ไอซีที่เป็นวงจรแปลงอะนาล็อกเป็นดิจิตอล แบบสโลปคู่มีด้วยกันหลายเบอร์ และผลิตกันหลายบริษัท เช่น เบอร์ ICL 7106 ของบริษัทอินเตอร์ซิล (Intersil) เบอร์ MC 14433 บริษัทโมโตโลรา เป็นต้น บางเบอร์ก็ใช้ในไอซีเพียงตัวเดียงบางเบอร์ก็ต้องใช้ไอซี 2-3 ตัวต่อเป็นชุด ปกติมักใช้อุปกรณ์ภายนอก เช่น ความต้านทาน คาปาซิเตอร์ ต่อเพิ่มอีเล็กน้อยก็สามารถใช้งานได้ดี ซึ่งในโครงงานจะไอซีเพียงตัวเดียวคือ เบอร์ 0809 และวงจรสร้างสัญญาณ นาฬิกา (Clock) ไอซีเบอร์ NE 555



รูปที่ 2.11 การต่อวงจรแปลงอะนาล็อกเป็นดิจิตอล

2.4 ทฤษฎีของไมโครคอนโทรลเลอร์ ตระกูล MCS - 51

ไมโครคอนโทรลเลอร์ในตระกูล MCS -51 ได้ถูกพัฒนาผลิตและแนะนำโดยบริษัทอินเทลต่อมาหลายบริษัทได้รับลิขสิทธิ์ไปผลิตโดยใช้เครื่องหมายการค้าของตัวเองเช่น ซีเมนต์ เอ.เอ็ม.ดี ฟิลลิปส์ เป็นต้น บางบริษัทก็ได้เพิ่มเติมฟังชั่นอื่นๆ เข้าไป ในเล่มนี้จะเป็นการศึกษาเกี่ยวกับโครงสร้างภายในหรือฮาร์ดแวร์ 8051

ฮาร์ดแวร์ของ 8051 ไมโครคอนโทรลเลอร์หรือสถาปัตยกรรมภายในของ 8051 จะเป็นดังรูปที่ 2.11 ซึ่งประกอบไปด้วยส่วนสำคัญหลัก ๆ ดังนี้

2.4.1 เป็น ซี.พี.ยู ขนาด 8 บิท ประกอบด้วยรีจิสเตอร์ A (แอคคิวมูเลเตอร์) และรีจิสเตอร์ B

2.4.2 โปรแกรมเคาท์เตอร์ (Program Counter , PC)และดาต้าพอยท์เตอร์ (Data Pointer,DPTR) ขนาด 16 บิท

2.4.3 มีโปรแกรมสเตตัสเวิร์ด (Program Stuatus Ward ,PSW) ขนาด8 บิท

2.4.4 มีสแต็กพอยท์เตอร์ (Stack Pointer ,SP) ขนาด 8 บิท

2.4.5 มีหน่วยความจำรอม (ROM) หรือ อีพรอม (EROM เฉพาะ 8751) ขนาด 0 กิโลไบท์ (8031) ถึง 4 กิโลไบท์ (8051)

2.4.6 มีหน่วยความจำแรมภายใน ขนาด 128 ไบท์ ประกอบด้วย

2.4.6.1 รีจิสเตอร์แบงค์ 4 แบงค์ แต่ละแบงค์ประกอบด้วย รีจิสเตอร์ ขนาด 8 บิท จำนวน 8 รีจิสเตอร์ (R0 - R7)

2.4.6.2 มีหน่วยความจำจำนวน 16 ไบท์ ที่สามารถอ้างแอดเดรสเพื่อควบคุมการทำงานในระดับบิทได้

2.4.6.3 มีหน่วยความจำสำหรับใช้งานทั่วไป 80 ไบท์

2.4.7 มีขารับสัญญาณอินพุท/ เอาท์พุท 32 ขา แบ่งออกเป็นกลุ่ม ๆ ละ 8 บิท ได้สี่กลุ่มคือ P0 ,P1,P2,P3

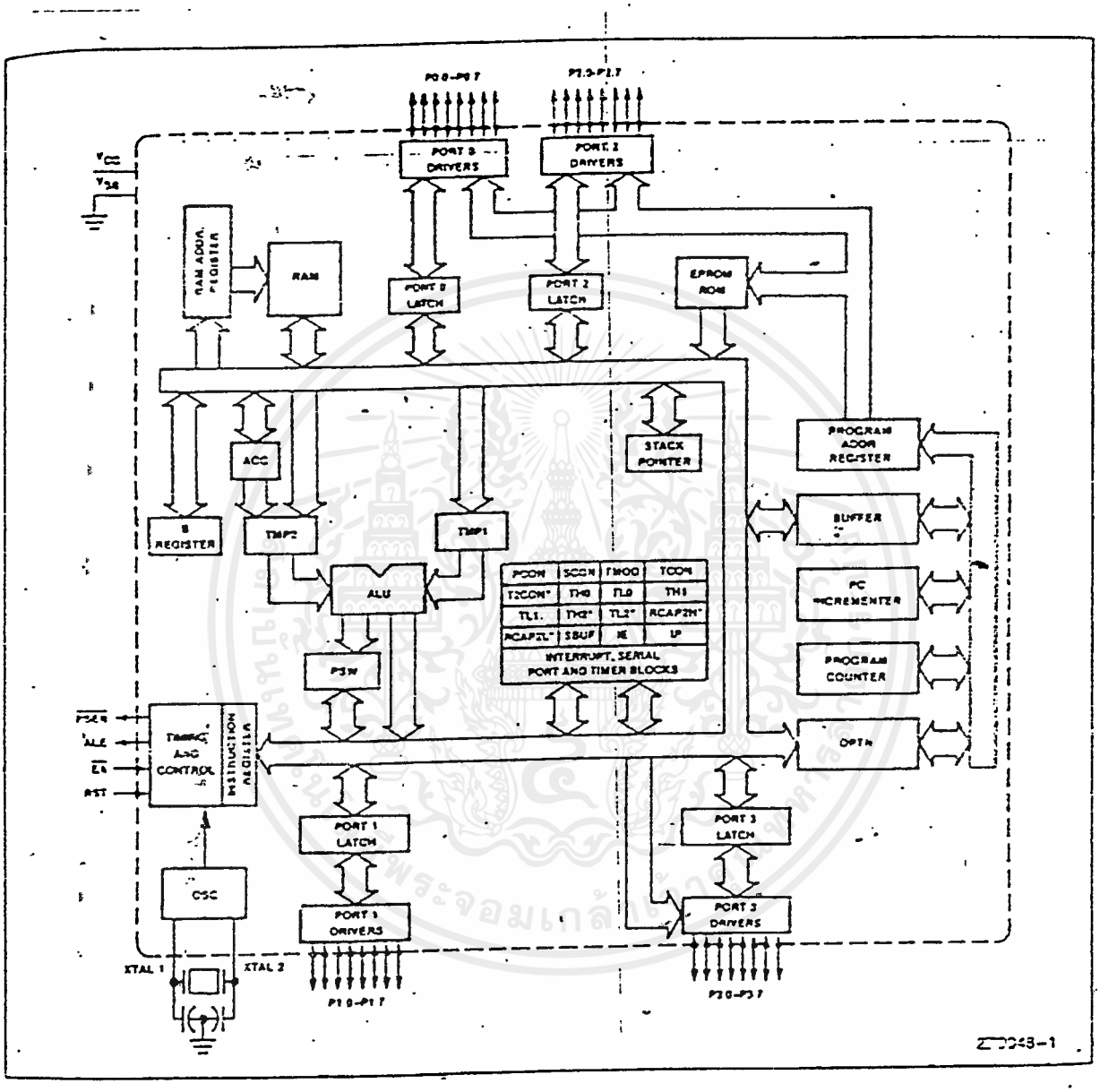
2.4.8 มีไทม์เมอร์/เคาท์เตอร์ ขนาด 16 บิท สองชุดคือ T0และ T1

2.4.9 มีพอร์ทอนุกรมที่ใช้รับส่งสัญญาณแบบฟูลดูเพรก (Full Duplex) เรียกว่า SBUF

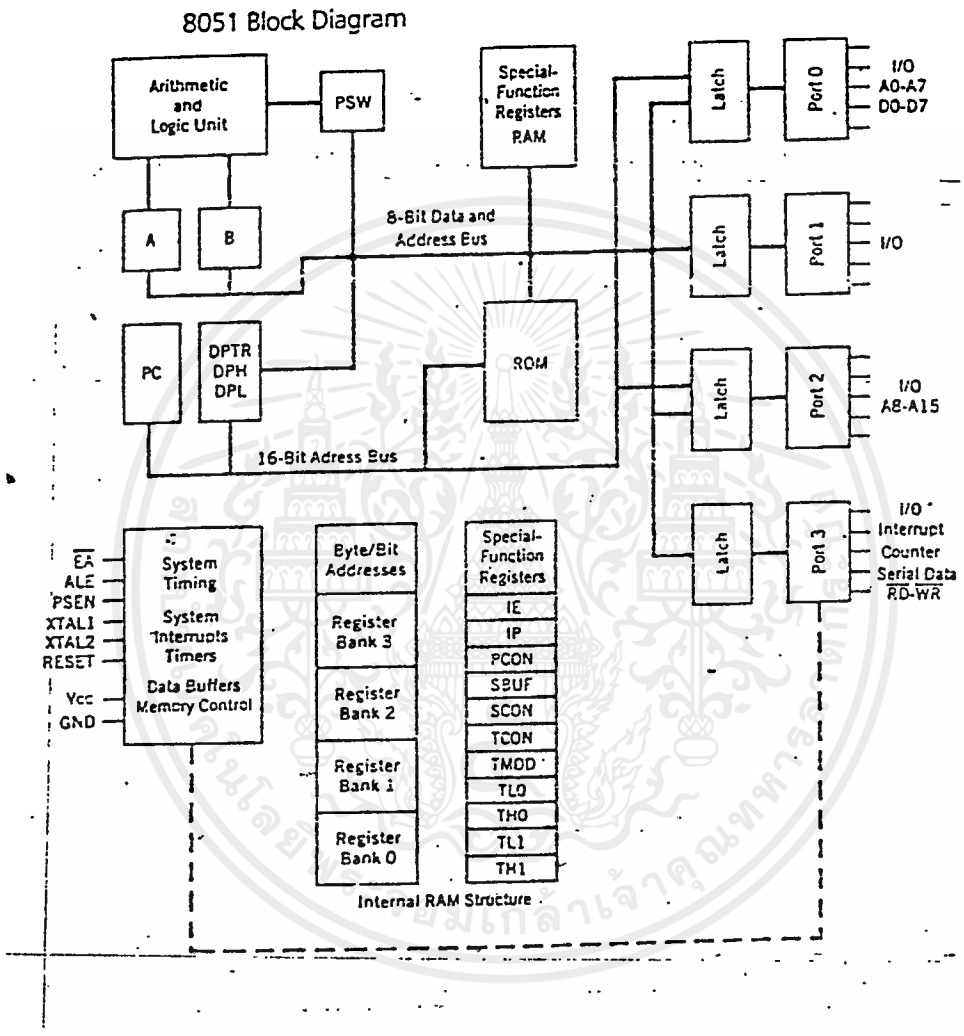
2.4.10 มีรีจิสเตอร์ควบคุมได้แก่ TCON , TMOD , SCON , PCON , IP และIE

2.4.11 สามารถทำการอินเทอร์รัพได้ทั้งภายในและภายนอก การอินเทอรัพท์ภายในได้มาจากแหล่งกำเนิดการอินเทอรัพท์สามแหล่ง การอินเทอรัพท์ภายนอก ได้มาจากแหล่งกำเนิดการอินเทอรัพท์จากภายนอกสองแหล่ง

2.4.12 มีส่วนของออสซิลเลเตอร์และวงจรสร้างสัญญาณนาฬิกาอยู่ภายใน



รูปที่ 2.12 แสดงสถาปัตยกรรมภายในของ 8051



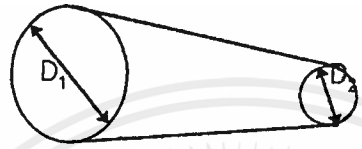
รูปที่ 2.13 แสดงบล็อกไดอะแกรมภายในของ 8051

บทที่ 3
การคำนวณและการสร้างโครงงาน

การคำนวณเพื่อใช้ในการออกแบบวงจรต่างๆ ที่ใช้ในการวัดกำลังงานและออกแบบโครงสร้าง โดยเราสามารถแบ่งการคำนวณเป็นข้อย่อยๆ ดังนี้

3.1 โครงสร้างทางกล

เราจะใช้จักรยานเป็นตัวรับพลังงานที่มนุษย์ออกกำลังมาผ่านเฟืองจานทดรอบเพื่อให้จำนวนรอบสูงขึ้น โดยจะมีลักษณะดังนี้



รูปที่ 3.1 แสดงการทดรอบของจักรยาน

จากรูปเราสามารถคำนวณความเร็วรอบและขนาดของจานได้จากสูตร

$$N_1/N_2 = D_2/D_1$$

โดยที่ N_1 คือความเร็วรอบของจานที่ 1

N_2 คือความเร็วรอบของจานที่ 2

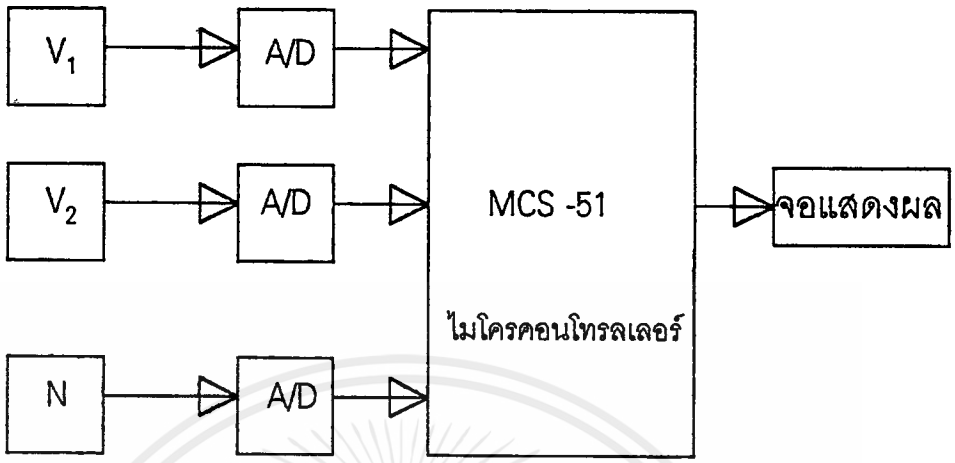
D_1 คือเส้นผ่านศูนย์กลางของจานที่ 1

D_2 คือเส้นผ่านศูนย์กลางของจานที่ 2

หลังจากนั้นเราสามารถออกแบบอัตราทดตามที่เราต้องการได้ เพื่อที่จะได้ค่าความเร็วสูงขึ้นแล้วนำไปติดตั้งกับเครื่องกำเนิดเพื่อได้กำลังไฟฟ้าตามที่เราต้องการ

3.2 การวัดพลังงานจากเครื่องกำเนิดไฟฟ้า

เพื่อที่จะให้เข้าใจในการคำนวณในข้อต่อไป เราจะมาทำความเข้าใจเกี่ยวกับรูปแบบของวงจรวัดเสียก่อน ในการวัดพลังงานของโครงงานนี้จะใช้ไมโครคอนโทรลเลอร์ (Microcontroller) เป็นตัวคำนวณผล ดังนั้นในการวัดจะต้องมีวงจรแปลงสัญญาณอะนาล็อกเป็นดิจิตอล (A/D)เพื่อเข้าไมโครคอนโทรลเลอร์ โดยจะมีลักษณะไดอะแกรมตามรูปที่ 3.2



รูปที่ 3.2 แสดงบล็อกไดอะแกรมการทำงานของโครงงาน

จากบล็อกไดอะแกรมเราสามารถอธิบายบล็อกต่างๆได้ดังนี้

V_1 คือ แรงดันที่ได้รับจากเครื่องกำเนิดไฟฟ้าโดยผ่านวงจรแบ่งแรงดันแล้ว

V_2 คือ แรงดันที่ได้รับจากการแปลงกระแสโดยผ่านวงจรแบ่งกระแสแล้ว

N คือ ความเร็วรอบที่ได้จากเทคโคเจน (Tachogenerator)

A/D คือ วงจรแปลงสัญญาณอะนาล็อกเป็นดิจิตอล

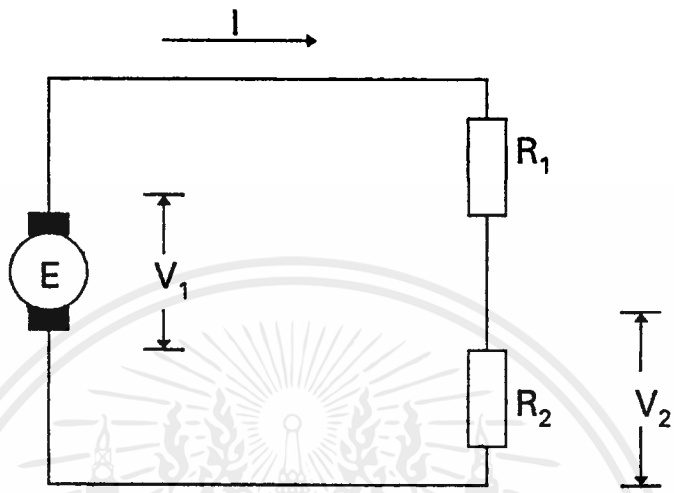
ในการวัดค่าแรงดันและกระแสจากการทดสอบกับผู้ทดสอบในการปั่นจักรยาน โดยจะได้ค่าเฉลี่ยที่ความเร็วรอบสูงสุดจะได้แรงดันจากเครื่องกำเนิดไฟฟ้าประมาณ 20 โวลท์ ซึ่งเป็นไฟกระแสตรง และเมื่อป้อนโหลดให้เครื่องกำเนิดจะได้กระแสสูงสุด 24 แอมแปร์ หรือจะได้กำลังสูงสุดประมาณ 480 วัตต์

จากการทดสอบจะพบว่าแรงดันสูงสุดมีค่าสูงมากเมื่อเทียบกับแรงดันที่จะใช้เป็นอินพุทของวงจรแปลงอะนาล็อกเป็นดิจิตอลเพราะวงจรแปลงอะนาล็อกเป็นดิจิตอลจะทนแรงดันสูงสุดเพียง 5 โวลท์ เท่านั้น ดังนั้นในการวัดแรงดันจึงต้องใช้วงจรแบ่งแรงดัน (Voltage Divider) และวงจรแบ่งกระแส (Current Divider) ซึ่งจะอธิบายต่อไป

3.3 วงจรแบ่งแรงดัน

ในการแปลงสัญญาณอะนาล็อกเป็นดิจิตอล ที่แรงดันอินพุทเข้า 5 โวลท์ เป็นอะนาล็อก จะได้แรงดัน 256 โวลท์ เป็นดิจิตอล ดังนั้นเพื่อให้การคำนวณง่ายขึ้น จึง

กำหนดค่าแรงดันสูงสุดให้หาร 256 โวลท์ ได้ลงตัว และให้ได้ค่าที่ใกล้เคียงกับแรงดันสูงสุดมากที่สุด ดังนั้นเราจะได้ค่าเท่ากับ 32 โวลท์ แล้วนำค่าที่ได้ไปหาอัตราส่วนในวงจรแบ่งแรงดันโดยเมื่อแรงดันเท่ากับ 32 โวลท์ จะแบ่งแรงดันให้เหลือ 5 โวลท์ เพื่อป้อนให้กับวงจรแปลงอะนาล็อกเป็นดิจิตอล



รูปที่ 3.3 วงจรแบ่งแรงดัน (Voltage Divider)

จากรูปวงจรสามารถหาความสัมพันธ์ระหว่างแรงดัน (V) , กระแส (I) และความต้านทาน (R) ซึ่งเขียนเป็นสมการได้ดังนี้

$$V_1 = I(R_1 + R_2) \quad (3.1)$$

และ $$V_2 = IR_2 \quad (3.2)$$

ดังนั้นจะได้ว่า

$$\frac{V_1}{V_2} = \frac{R_1 + R_2}{R_2}$$

$$V_1 = \frac{R_1 + R_2}{R_2}(V_2) \quad (3.3)$$

หรือ $$V_2 = \frac{V_1 R_2}{R_1 + R_2} \quad (3.4)$$

อัตราส่วนของแรงดันที่เราต้องการจะได้เท่ากับ

$$\frac{R_2}{R_1 + R_2} = \frac{V_2}{V_1} = \frac{5}{32}$$

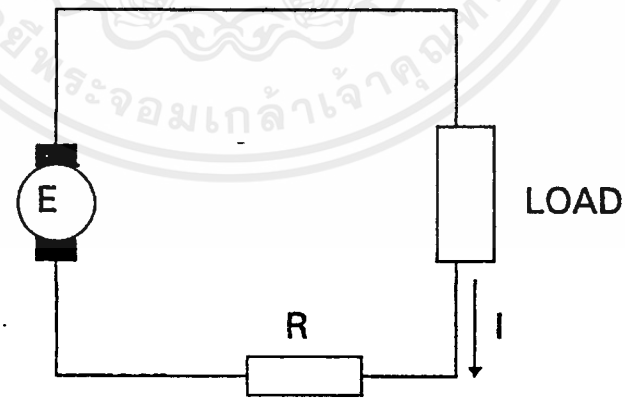
เราสามารถสรุปได้ว่าถ้าค่า R_1 เท่ากับ 5 K โอห์ม ค่าของ R_2 จะเท่ากับ 27K โอห์ม เพื่อให้ได้อัตราส่วนตามที่เราต้องการ

3.4 วงจรแบ่งกระแส (Current Divider)

เช่นเดียวกันกับวงจรแบ่งแรงดัน เราจะต้องลดค่ากระแสที่จะป้อนเข้าวงจรแปลงอะนาล็อกเป็นดิจิตอลที่ทนแรงดันสูงสุดได้ 5 โวลท์ และจากการทดลองเราจะพบว่าค่ากระแสสูงสุดเราจะได้เท่ากับ 24 แอมป์ ดังนั้นเราจะได้ค่าที่ใกล้เคียงที่หาร 256 ลงตัวก็คือ 32 แอมป์ นั้นก็คือเมื่อกระแสมีค่าเท่ากับ 32 แอมป์ จะต้องออกแบบให้วงจรแบ่งกระแสเหลือ 5 โวลท์ แต่ในวงจรแบ่งกระแสจริง เมื่อกระแสเท่ากับ 32 แอมป์ ความต้านทานจะสูงมากไม่ได้ เพราะเมื่อความต้านทานสูง วัตต์ไฟฟ้าก็จะสูงตามไปด้วย จากสูตร

$$P = I^2 R \qquad (3.5)$$

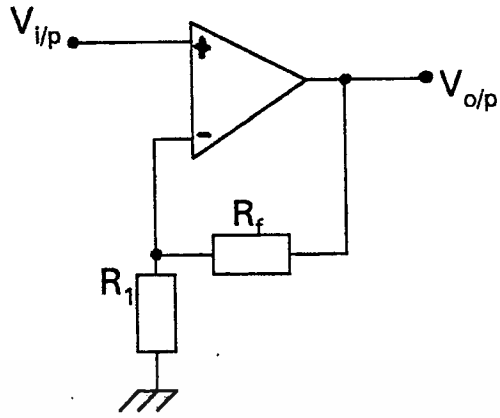
ดังนั้น เราจึงต้องใช้ค่าความต้านทานต่ำ ๆ โดยออกแบบให้ใช้ค่าความต้านทาน 0.1 โอห์ม 10 วัตต์ มาต่อขนานกันให้ได้ความต้านทาน 0.0125 โอห์ม เมื่อกระแสสูงถึง 24 แอมป์ วัตต์ทางไฟฟ้าจะได้เท่ากับ 7.2 วัตต์ ซึ่งไม่เกิน 10 วัตต์ ที่ค่าความต้านทานจะทนได้ โดยจะได้วงจรดังรูป



รูปที่ 3.4 วงจรแบ่งกระแส (Current Divider)

โดยที่ค่าความต้านทาน (R) คือ ความต้านทาน 0.1 โอห์ม 10 วัตต์ ต่อขนานกัน 8 ตัว จะได้ค่าความต้านทานเท่ากับ 0.0125 โอห์ม

แต่เมื่อนำค่ากระแสมาคูณกับค่าความต้านทานแล้วจะได้แรงดันต่ำมาก จึงต้องมีการขยายแรงดันโดยใช้วงจรออปแอมป์แบบค่าเป็นบวก (Non - inverting Amp) รูป



รูปที่ 3.5 วงจร Non-inverting Amp

ค่าอัตราขยาย

$$A_V = \left(\frac{R_F}{R_1} + 1\right) = \frac{V_{o/p}}{V_{i/p}} \qquad (3.6)$$

ดังนั้น

$$V_{o/p} = \left(\frac{R_F}{R_1} + 1\right) V_{i/p} \qquad (3.7)$$

ค่าที่เราต้องการขยาย คือ เมื่อกระแสเท่ากับ 32 แอมป์ จะได้ค่าแรงดันเท่ากับค่า 32 คูณด้วย 0.0125 จะมีค่าเท่ากับ 0.4 โวลท์ แล้วจะต้องขยายแรงดันให้ได้ 5 โวลท์ เท่ากับแรงดันที่ป้อนเข้าวงจรแปลงอะนาล็อกเป็นดิจิตอลสูงสุด ดังนั้นจะได้อัตราขยายดังนี้

$V_{o/p}$ = 5 V

$V_{i/p}$ =0.4 V

แทนค่าในสมการ

A_V = 0.4 A_V

$$A_V = \frac{5}{0.4} = 12.5$$

แต่วงจรในความเป็นจริงไม่สามารถขยายได้ถึง 12.5 เท่า ดังนั้นเราจึงทำการเปลี่ยนค่าจาก 32 แอมป์เป็น 64 แอมป์ แทน เราก็จะได้ค่าแรงดันตกคร่อมความต้านทานเท่ากับ 64 คูณด้วย 0.0125 เท่ากับ 0.8 โวลท์

จะได้

$$V_{o/p} = 5 \quad V$$

$$V_{i/p} = 0.8 \quad V$$

แทนค่าใน (3.7)

$$5 = 0.8A_V$$

$$A_V = 6.25 \quad \text{เท่า}$$

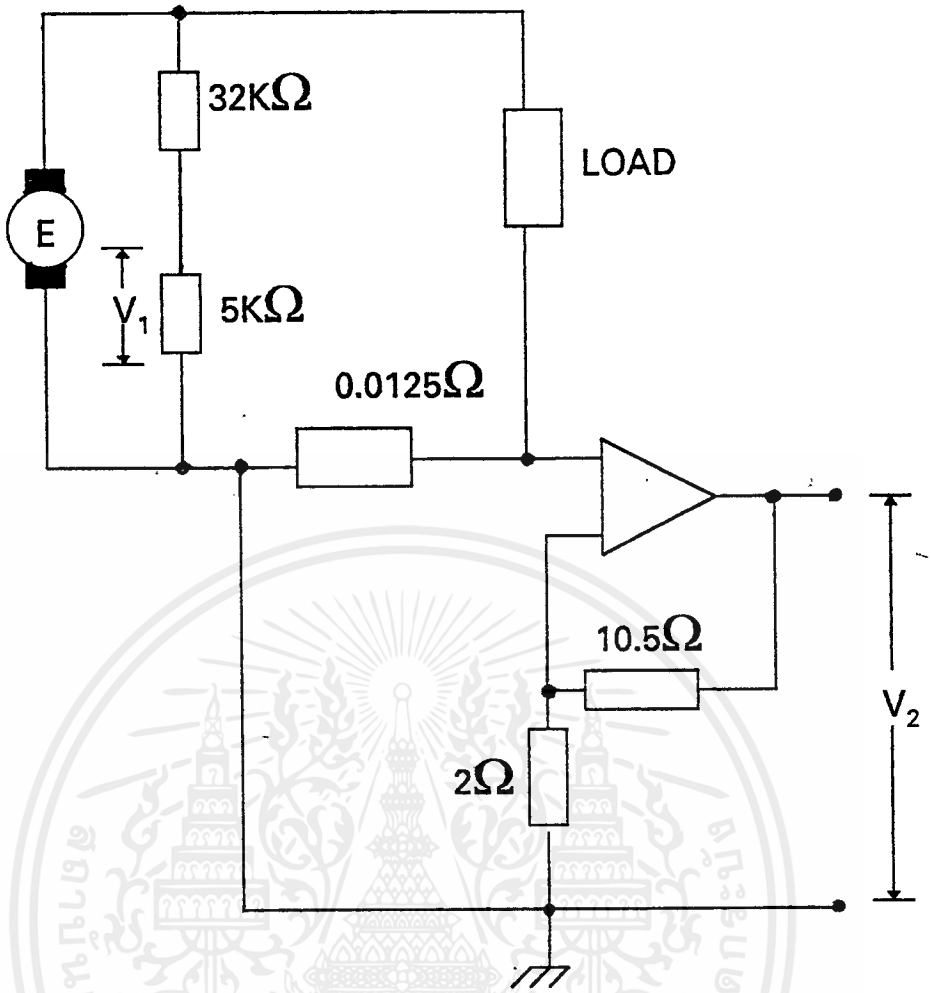
และค่าความต้านทานจะเท่ากับ

$$\frac{R_F}{R_1} + 1 = 6.25$$

$$\frac{R_F}{R_1} = 5.25$$

ดังนั้นเราจะได้อัตราส่วนของความต้านทานเท่ากับ 5.25 โดยเราสามารถนำไปหาค่าความต้านทาน R_1 และ R_2 ได้ ตัวอย่างเช่น ถ้า R_1 เท่ากับ 2 โอห์ม ค่า R_2 จะเท่ากับ 10.5 โอห์ม เป็นต้น

ดังนั้นเราจะได้ที่ใช้ในโครงงาน ดังนี้



รูปที่ 3.6 วงจรที่ออกแบบเพื่อใช้ในโครงงานเพื่อวัดสัญญาณต่าง ๆ

โดยที่ V_1 คือ สัญญาณแรงดันที่ป้อนเข้าวงจรแปลงอะนาล็อกเป็นดิจิตอล

V_2 คือ สัญญาณกระแสที่อยู่ในรูปของสัญญาณแรงดันที่ป้อนเข้าวงจรแปลงอะนาล็อกเป็นดิจิตอล

3.5 การหาค่ากำลังสูญเสียและความเร็วรอบ

จากตารางการทดลองเราจะพบว่า เมื่อนำค่าอัตราส่วนของอัตราส่วนของความเร็วรอบกับแรงดันจากเทคโคเจนมาหาค่าเฉลี่ยคือ

N/V เฉลี่ย = 3988.82/13

= 306.83

หรือประมาณ 307

แต่ในการรับแรงดันของวงจรแปลงอะนาล็อกเป็นดิจิตอล เมื่อแรงดันเท่ากับ 5โวลท์ค่าดิจิตอลจะเป็น 256

นั่นแสดงว่าเมื่อแรงดันอินพุทเท่ากับ 5 โวลท์ ดิจิตอลจะเป็น 256
ถ้าแรงดันอินพุทเท่ากับ 1 โวลท์ ดิจิตอลจะมีค่าเป็น 256/5 = 51.2
แต่เราต้องการให้เมื่อค่าแรงดัน 1 โวลท์ ดิจิตอลจะเป็น 307 รอบต่อนาที
ดังนั่นเมื่อแรงดันเข้ามา 1 โวลท์ เราจะต้องคูณ ค่าดิจิตอลด้วยค่า 307/51.2 = 5.99
หรือ ประมาณ เท่ากับ 6

ดังนั่นในโปรแกรมเราจะเขียนให้เป็นสูตรดังนี้

$$N_{rpm} = 6.\ V_{t.c} \qquad (3.8)$$

และจากการหาค่ากำลังการสูญเสียในระบบเราจะทำการเทียบดังนี้คือ
เมื่อมีความเร็วรอบเท่ากับ 32 รอบต่อนาที เราจะได้ค่ากำลังสูญเสียเท่ากับ 1 วัตต์
และถ้าความเร็วรอบเท่ากับ 1 รอบต่อนาที เราจะได้กำลังสูญเสียเท่ากับ 1/32 วัตต์
ดังนั่นในโปรแกรมจะได้สูตร

$$P_{(W)} = N_{rpm}/32 \qquad (3.9)$$

จากสมการที่ (3.8) จะได้

$$P_{(W)} = 6.V_{t.c}/32 \qquad (3.10)$$

3.6 การนำค่าต่าง ๆ ที่ได้รับไปเขียนโปรแกรม

จากการคำนวณข้างต้น เราสามารถสรุปผลเพื่อนำไปเขียนโปรแกรมได้คือ

3.6.1 ในการวัดแรงดัน เมื่อแรงดันที่ได้รับจากเครื่องกำเนิดมีค่าเท่ากับ 32 โวลท์ แรงดันที่ป้อนเข้าไมโครคอนโทรลเลอร์ มีค่าเท่ากับ 256 เป็นสัญญาณดิจิตอล ดังนั้นค่าแรงดันที่ได้รับจะต้องโปรแกรมให้หารด้วย 8 เพื่อจะให้ได้ค่าเท่ากับ 32 โวลท์ เท่ากับแรงดันจริง

3.6.2 การวัดกระแสก็เหมือนกับการวัดแรงดัน เมื่อกระแสสูงสุดมีค่าเท่ากับ 64 แอมป์ ไมโครคอนโทรลเลอร์ จะได้รับเท่ากับ 256 เป็นสัญญาณดิจิตอล จึงต้องให้หารด้วย 4 เพื่อจะให้ได้ค่าเท่ากับ 64 แอมป์ เท่ากับกระแสจริง

3.6.3 ในการคำนวณหาค่าความสูญเสียนั้น เมื่อเรารู้ค่าแรงดันที่เทคโดเจน จะรู้ค่าโดยทันทีใช้สูตร

$$P_{(W)} = 6 \times \frac{V_{t.c}}{32} \qquad (3.8)$$

โดยที่ $P_{(w)}$ คือ กำลังไฟฟ้าสูญเสีย

V คือ แรงดันที่วัดได้จากเทคโคเจน

ตัวอย่างเช่น ถ้าแรงดันที่เทคโคเจน มีค่าเท่ากับ 5 โวลท์ จะได้แรงดันที่ออกจากไมโครคอนโทรลเลอร์เท่ากับ 256 โวลท์ เป็นสัญญาณดิจิตอล

แทนค่าใน (3.8)

$$P_{(W)} = \frac{6 \times 256}{32}$$

$$= 48 \quad \text{วัตต์}$$

3.6.4ในการคำนวณหาค่าความเร็วรอบนั้นเราจะใช้สูตร

$$N_{rpm} = 6 \times V_{t.c} \qquad (3.9)$$

ตัวอย่างเช่น เมื่อแรงดันที่เทคโคเจนมีค่าเท่ากับ 5 โวลท์ จะได้แรงดันที่ออกจากไมโครคอนโทรลเลอร์เท่ากับ 256 โวลท์ เป็นสัญญาณดิจิตอล

$$N_{rpm} = 6 \times 256$$

$$= 1539 \quad \text{รอบต่อนาที (rpm)}$$

บทที่ 4
การทดลองและผลการทดลอง

ในการทดลองเครื่องวัดกำลังงานมนุษย์จะแบ่งออกเป็น 2 ขั้นตอนดังต่อไปนี้คือ

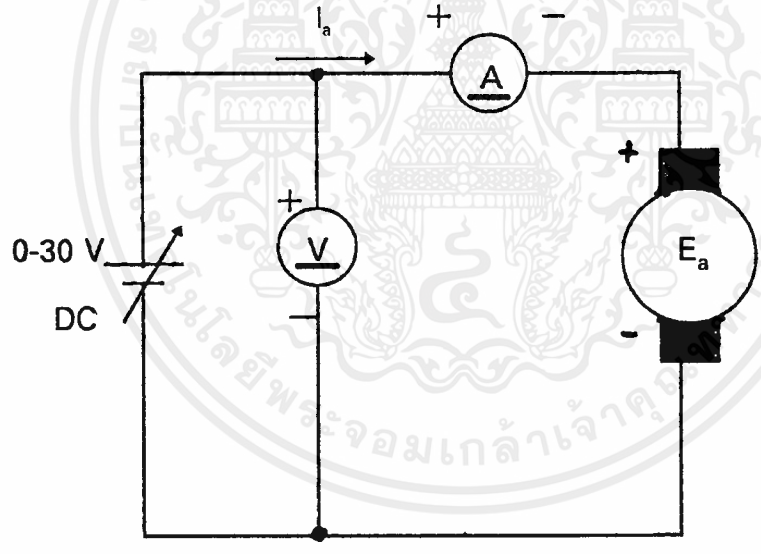
4.1 ทดลองก่อนการออกแบบวงจรอิเล็กทรอนิกส์

ก่อนการออกแบบวงจรอิเล็กทรอนิกส์นั้นเราจะต้องทำการทดสอบหาค่าความสูญเสียของระบบทั้งหมด ซึ่งอาจจะเกิดจากเฟืองทด และเครื่องกำเนิดดังนี้

4.1.1 ขั้นตอนการทดลอง

1.นำแหล่งจ่ายไฟฟ้ากระแสตรงต่อเข้ากับเครื่องกำเนิดกระแสตรง (เปลี่ยนเป็นมอเตอร์กระแสตรง)

2.ทำการวัดค่าแรงดันเข้า ค่ากระแสเข้า ความเร็วรอบที่แกนเพลา และวัดค่าแรงดันที่เทคโคเจน ดังรูป



รูปที่ 4.1 วงจรการทดลองหาค่าความสูญเสีย

3. ทำการคำนวณค่ากำลังงานไฟฟ้า ค่าความสูญเสียทางไฟฟ้า และค่าความสูญเสียทางกล โดยค่าความสูญเสียทางไฟฟ้านั้นหาได้จากการทดลองหาค่าความต้านทานในขดลวดก่อนแล้วนำมาคูณกับค่ากระแสยกกำลังสอง ส่วน

ค่าความสูญเสียทางกลหาได้จากค่ากำลังไฟฟ้าลบด้วยค่าความสูญเสียทางไฟฟ้า

4.ทำการทดสอบหาค่ากำลังงานสูงสุดที่ออกกำลังได้

4.1.2 ผลการทดลอง

$V_{i/p}$(V)	I_a(A)	$V_{t.c}$(V)	N(rpm)	$P_{i/p}$(W)	P_{loss}(W)	P_{mech}(W)
2.00	0.75	0.16	48.00	1.50	0.28	1.22
4.00	0.84	0.35	108.00	3.36	0.35	3.00
6.00	0.88	0.55	169.00	5.38	0.38	5.00
8.00	0.90	0.75	230.00	7.20	0.40	6.80
10.00	0.92	0.96	294.00	9.20	0.42	8.78
12.00	0.94	1.17	360.00	11.28	0.44	10.80
14.00	0.96	1.40	430.00	13.44	0.46	12.98
16.00	0.98	1.63	502.00	15.68	0.48	15.20
18.00	1.00	1.87	576.00	18.00	0.50	17.50
20.00	1.04	2.16	665.00	20.80	0.54	20.26
22.00	1.08	2.47	760.00	23.76	0.58	23.18
24.00	1.10	2.75	844.00	26.40	0.60	25.80
26.00	1.14	3.08	948.00	29.64	0.64	29.00

ตารางที่ 4.1 ตารางบันทึกผลการทดลองที่ 1

เราหาค่ากำลังงานสูงสุดได้โดยทดสอบกับคน 10 คน ผลปรากฏว่าในจำนวน 10 คน นั้นสามารถออกแรงได้ค่าแรงดันเท่ากับ 20 โวลท์ และค่ากระแสเท่ากับ 24 แอมป์ ดังนั้นจึงได้ค่ากำลังงานสูงสุดเท่ากับ 480 วัตต์

การหาค่าความต้านทานในขดลวดอาร์มเจอร์(Armature Resistance) เพื่อหาค่าความสูญเสียด้วยวิธี VI- Method

$V_{i/p}$(V)	I_a(A)	R(Ω)
1	2	0.5
2	4	0.5

ตารางที่ 4.2 ผลการทดลองหาค่าความต้านทานในขดลวดอาร์มเจอร์

จากการทดลองและคำนวณเราจะได้ความสัมพันธ์ระหว่างค่าความเร็วรอบกับค่าความสูญเสียทางกลเรานำความสัมพันธ์นี้ไปคำนวณเพื่อไปเขียนโปรแกรมต่อไป

4.2 การทดลองหลังการออกแบบวงจรอิเล็กทรอนิกส์

การทดลองหลังการออกแบบนั้นเพื่อให้แน่ใจว่าเครื่องวัดพลังงานมนุษย์ใช้งานได้จริงเราจึงต้องทำการทดสอบเครื่องโดยการวัดกำลังงานจริงกับค่าที่อ่านได้จากจอแสดงผล การวัดค่ากำลังงานจริงทำได้โดยการนำค่ากำลังงานไฟฟ้าบวกกับค่าความสูญเสียเมื่อเทียบกับความเร็วรอบโดยจะได้ผลการทดลองดังนี้

ค่ากำลังงานไฟฟ้า	ความเร็วรอบ	ค่าความสูญเสีย	ค่าที่อ่านได้
194	640	20	252
267	640	20	316
456	640	20	562
395	640	20	436

ตารางที่ 4.1 ตารางการทดลองการวัดกำลัง

จากการทดลองเราจะเห็นว่าค่าที่แตกต่างละหว่างกำลังไฟฟ้าจริงกับจอแสดงผลต่างกันเพียงเล็กน้อย เพราะเนื่องจากในโปรแกรมมีการปัดเศษทำให้ค่าที่ได้คลาดเคลื่อน

บทที่ 5
บทวิจารณ์และสรุปผล

เมื่อเราทำการทดลองและออกแบบแล้วเราก็ได้เครื่องวัดกำลังงานมนุษย์โดยเราจะนำค่าจากการคำนวณ ไปใช้ในการเขียนโปรแกรมในภาคผนวก ก. ซึ่งจะแสดงผลออกบนจอภาพ และใช้วัดในงานจริง ซึ่งจะมีประโยชน์กับบุคคลโดยทั่วไปที่ไม่มีสถานที่ออกกำลังกายหรือต้องการทราบว่าตัวเองมีกำลังงานมากน้อยเท่าไร

โครงงานเครื่องวัดกำลังงานมนุษย์ที่ได้ทำการทดสอบและวัดค่าแล้วนั้น สรุปได้ว่าได้ผลตามขอบเขตของงานที่ได้ตั้งไว้ ซึ่งเราจะสามารถวัดกำลังงานที่มนุษย์ออกมาได้ โดยเครื่องวัดจะแสดงผลที่จอ เป็นค่ากำลังงานและค่าความเร็วรอบซึ่งจะมีรูปร่างดังนี้

POWER □ W

SPEED □ rpm

ซึ่งสามารถใช้วัดกำลังงานได้จริง และค่าที่ได้ตรงกับการคำนวณและการทดสอบ ซึ่งเราสามารถรู้ว่ากำลังงานที่ได้นั้นแต่ละคนจะไม่เท่ากันซึ่งจะอธิบายความแตกต่างได้ในภาคผนวก ข.ที่อธิบายพลังงานของมนุษย์แต่ละคน

คำแนะนำในการค้นคว้าต่อไป เราสามารถนำค่าที่ได้จากเครื่องวัดกำลังงานมนุษย์ คือค่ากำลังงานเราสามารถทำให้เป็นค่าพลังงานได้โดยใช้สูตร

$$W = Pt$$

โดยที่ W คือ ค่าพลังงาน (มีหน่วยเป็น จูล)

P คือ ค่ากำลังงาน (มีหน่วยเป็น วัตต์)

t คือ เวลาที่ใช้ออกกำลัง (มีหน่วยเป็น วินาที)

โดยเราจะสามารถหาค่าพลังงานได้โดยบอกค่ากำลังงานที่ได้จากเครื่องวัดทุกๆ 1 วินาที โดยจะใช้การคำนวณหรือแก้ไขที่โปรแกรมได้

เมื่อเราได้พลังงานแล้วเราสามารถรู้ได้อีกอย่างว่าเมื่อเรารับประทานอาหาร 1 จาน เราจะได้พลังงานเท่าไร โดยเทียบกับตารางในภาคผนวก ข. หรือ เราปั่นเครื่องวัด 1 วัตต์จะเท่ากับข้าวกี่จาน เป็นต้น

การปรับปรุงโครงงานให้ดีขึ้นได้ โดยปรับปรุงโปรแกรมให้มีความละเอียดมากขึ้น คือออกแบบวงจรให้ใช้สัญญาณดิจิตอลที่เป็นสัญญาณอินพุทเข้าไมโครคอนโทรลเลอร์ให้ใช้ได้ครบทั้ง 8 บิทจะทำให้ค่าที่ได้ละเอียดมากขึ้น และยังเห็นได้ว่าโครงงานนี้จะใช้แหล่งจ่ายไฟจากภายนอก เพื่อให้สะดวกในการเคลื่อนย้ายควรออกแบบให้ใช้แบตเตอรี่เป็นแหล่งจ่ายจะสะดวกมากยิ่งขึ้น นอกจากนั้นเรายังสามารถนำกำลังงานที่ได้จากการปั่นจักรยานมาชาร์จแบตเตอรี่นี้ได้อีกด้วย

ภาคผนวก ก.

เพื่อให้เข้าใจในโปรแกรมที่เขียนได้ดียิ่งขึ้นเราจึงอธิบายคำสั่งและเนื้อหาเรื่องของแอดเดรสซิ่งดังต่อไปนี้

แอดเดรสซิ่ง (Addressing)

ในขั้นตอนของบทนี้กล่าวถึงวิธีการจัดหน่วยความจำของ 8051 แต่วิธีการที่จะติดต่อกับหน่วยความจำเหล่านี้ในชุดคำสั่งซึ่งเรียกว่าการกำหนดที่อยู่ (Addressing) จะสามารถทำได้หลายวิธีดังนี้

1. Direct Addressing เป็นการกำหนดตำแหน่งที่อยู่ของหน่วยความจำที่จะติดต่อเข้าไปใน Operand ของคำสั่งโดยตรง วิธีการนี้สามารถใช้กับการติดต่อหน่วยความจำสำหรับข้อมูลในตัวของ 8051 จำนวน 256 ตำแหน่ง เท่านั้นเช่นคำสั่ง

DEC Direct

ซึ่งเป็นคำสั่งให้ลดค่าข้อมูลในหน่วยความจำสำหรับข้อมูลภายใน 8051 ลง 1 ใน Operand ของคำสั่ง Mnemonic จะใส่ค่า Address โดยตรง ถ้าใน Operand มีคำว่า Direct หมายความว่ามีการระบุตำแหน่งของหน่วยความจำได้โดยตรงดังตัวอย่าง

DEC 20H

เป็นการลดข้อมูลของหน่วยความจำสำหรับข้อมูลภายใน 8051 ที่ตำแหน่ง 20H ลง 1 ถ้าเดิมข้อมูลในหน่วยความจำสำหรับข้อมูลภายใน 8051 ตำแหน่ง 20H มีค่า 11H เมื่อสิ้นสุดคำสั่งนี้ค่าในหน่วยความจำจะเป็น 10H แต่ถ้าค่าเดิมเป็น FFH เมื่อทำคำสั่งนี้สิ้นสุดลงจะทำให้ข้อมูลเป็น 00H เพราะแต่ละตำแหน่งเก็บข้อมูลได้ 8 บิท เท่านั้น

2. Indirect Addressing เป็นการกำหนดที่อยู่ของหน่วยความจำสำหรับข้อมูลภายใน 8051 โดยอ้อม วิธีการระบุตำแหน่งของหน่วยความจำที่ต้องการติดต่อวิธีนี้ จะใช้รีจิสเตอร์ตัวหนึ่งเป็นตัวชี้ (Pointer) ไปยังหน่วยความจำที่ต้องการ รีจิสเตอร์ที่ใช้เป็นตัวชี้ได้แก่ R0, R1, DPTR เป็นต้น ใน Operand ของชุดคำสั่ง 8051 ที่ติดต่อกับหน่วยความจำโดยอ้อมจะมีสัญลักษณ์ @ นำหน้ารีจิสเตอร์ที่เป็นตัวชี้ เช่นคำสั่ง

DEC @R1

เป็นคำสั่งลดค่าข้อมูลในหน่วยความจำตำแหน่งที่ชี้ค่าด้วยค่าของรีจิสเตอร์ Ri (Ri หมายถึงรีจิสเตอร์ R0 หรือ R1) ลง 1

ตัวอย่างโปรแกรมภาษาแอสเซมบลี

```
MOV     R0, #10H
DEC     @R0
```

คำสั่งแรกเป็นคำสั่งกำหนดค่าให้กับรีจิสเตอร์ R0 โดยตรงให้มีค่าเท่ากับ 10H

คำสั่งที่สองเป็นคำสั่งลดค่าในหน่วยความจำสำหรับข้อมูลที่ชี้โดยรีจิสเตอร์ R0 ซึ่งจากคำสั่งแรกนั้น R0 มีค่า 10H ดังนั้นคำสั่งที่ 2 จึงมีการลดค่าในหน่วยความจำที่ตำแหน่ง 10H ดังไดอะแกรม

ADDRESS	DATA		ADDRESS	DATA
10H	15H	⟶	10	14H
11H	16H	EXECUTE	11H	16H

3. Register instruction เป็นคำสั่งที่ใช้ติดต่อกับรีจิสเตอร์ R0 ถึง R7 ของรีจิสเตอร์ Bank ที่กำลังใช้งานอยู่ใน Operand จะมีชื่อของรีจิสเตอร์ที่ต้องการใช้อยู่ เมื่อแปลเป็นภาษาเครื่องจะพบว่า ในภาษาเครื่องของคำสั่งติดต่อกับรีจิสเตอร์ เหล่านี้จะมี 3 บิทที่เป็นตัวบอก รีจิสเตอร์ R0 ถึง R7 ดังนั้นเมื่อคำสั่งนั้นถูกอ่าน (Fetch) เข้าไปประมวลผล (Execute) ก็จะแยกเอาตัวชี้รีจิสเตอร์ที่ต้องการมาจากคำสั่งภาษาเครื่อง (OP-CODE) ที่อ่านเข้าไปนั่นเอง เช่นคำสั่ง MOV A,Rn โดยค่า Rn คือ R0,R1, ..., R7 คำสั่งนี้จะเปลี่ยนเป็นภาษาเครื่องได้ 1 ไบท์ดังตาราง

คำสั่ง Mnemonic	ภาษาเครื่อง	
MOV A,R0	E8H	(01110100B)
MOV A,R1	E9H	(11101001B)
MOV A,R2	EAH	(11101010B)
MOV A,R3	EBH	(11101011B)
MOV A,R4	ECH	(11101100B)
MOV A,R5	EDH	(11101101B)
MOV A,R6	EEH	(11101110B)
MOV A,R7	EFH	(11101111B)

จะเห็นว่าภาษาเครื่องในรูปเลขฐาน 2 (ในวงเล็บ) ของคำสั่งทั้ง 8 นั้นจะแตกต่างกันที่ 3 บิทสุดท้าย ซึ่งจะเป็นตัวชี้ตำแหน่งความจำสำหรับข้อมูลภายในสำหรับรีจิสเตอร์ เช่นขณะที่เรียกการใช้งานรีจิสเตอร์ BANK 1 จะทำให้คำสั่ง MOV A,R0 ซึ่งมี 3 บิทสุดท้ายเป็น 000B มีความ

หมายถึง การอ่านข้อมูลจากหน่วยความจำสำหรับข้อมูลที่ตำแหน่ง 08H เข้ามาเก็บไว้ยังรีจิสเตอร์ A

Register-Specific Instruction เป็นคำสั่งที่มีการทำงานเฉพาะกับรีจิสเตอร์บางตัวเช่น Accumulator หรือ DPTR คำสั่งเหล่านี้จะไม่มีตำแหน่งหน่วยความจำในส่วนของ OP-CODE ที่จะไปชี้รีจิสเตอร์ใด ๆ เพราะ CPU จะรู้ว่าเป็นรีจิสเตอร์ใดโดยอัตโนมัติ เช่น คำสั่ง CLR A จะมีภาษาเครื่อง 1 ไบท์คือ E4H ซึ่งเป็นคำสั่งที่จะกระทำกับรีจิสเตอร์ A โดยอัตโนมัติ

4. Immediate Constant เป็นคำสั่งเกี่ยวกับค่าคงที่โดยตรง คำสั่งนี้จะมีการกำหนดค่าคงที่ในส่วน Operand เช่นคำสั่ง

MOV A, #100

คำสั่งนี้เป็นการกำหนดค่า 100 ไปเก็บในรีจิสเตอร์ A เครื่องหมาย # ในสำหรับบอกว่าค่าที่ตามมาหลังเครื่องหมายนี้เป็นค่าคงที่ไม่ใช่ตำแหน่งของหน่วยความจำ ตัวชี้ตำแหน่งหน่วยความจำก็คือรีจิสเตอร์ Program Counter นั่นเองที่จะชี้ตำแหน่งของ Operand เช่นคำสั่งข้างบนอยู่ที่หน่วยความจำตำแหน่ง 0000H ก็จะมีภาษาเครื่องคือ 74H 64H อยู่ที่ตำแหน่ง 0000 และ 0001 ตามลำดับข้อมูลที่จะนำเอามาใส่ให้กับรีจิสเตอร์ A ก็คือข้อมูลในหน่วยความจำตำแหน่ง 0001 นั่นเอง

5. Index Addressing การกำหนดเลขที่อยู่โดยดรรชนี การอ้างอิงหน่วยความจำวิธีนี้ใช้ได้เฉพาะกับการติดต่อหน่วยความจำสำหรับโปรแกรมเท่านั้น ซึ่งวิธีการอ้างอิงหน่วยความจำแบบนี้จะใช้รีจิสเตอร์ DPTR หรือ Program Counter ขนาด 16 บิท บวกด้วยรีจิสเตอร์ A ขนาด 8 บิท แล้วนำผลลัพธ์ไปชี้ตำแหน่งหน่วยความจำสำหรับโปรแกรมเพื่ออ่านข้อมูลออกมา คำสั่งนี้มีประโยชน์ในการอ่านข้อมูลซึ่งเก็บไว้เป็นตาราง (Table) ในหน่วยความจำโปรแกรม เช่นคำสั่ง

MOVC A,@A+DPTR

ถ้า DPTR มีค่า 1000H และรีจิสเตอร์ A มีค่า 18H การทำงานของคำสั่งนี้จะอ่านข้อมูลจากตำแหน่ง 1018H มาเก็บไว้ที่รีจิสเตอร์ A

ADD A,<byte>

ในช่องของ Operation แสดงว่าการกระทำที่เกิดขึ้น คือเอาข้อมูลใน Accumulator บวกด้วยข้อมูล <byte> แล้วเก็บผลลัพธ์ไว้ใน Accumulator ข้อมูล <byte> ที่นำมาบวกนั้นได้จากการชี้ตำแหน่ง (Addressing) หน่วยความจำด้วยวิธีในช่อง Addressing Mode ซึ่งมีเครื่องหมาย X ปรากฏอยู่ในทุกช่องของ Dir, Ind, Reg และ Imm ดังนั้นในคำสั่ง ADD A,<byte> อาจเป็นดังนี้

ADD A,FH

ADD A,@R0

ADD A,R7

ADD A,7FH

ในระหว่างที่ทำการบวกนั้น บิท Carry หรือ Auxiliary-Carry Flag ใน PSW จะมีค่าเป็น 1 ถ้าเกิดตัวทดจากหลักที่ 7 หรือ 3 ของ Accumulator ตามลำดับ

บิท OV (Overflow) จะถูกตั้ง (Set) ให้มีค่าเป็น 1 ถ้าในระหว่างการใช้คำสั่งคณิตศาสตร์มีตัวทดจากบิท 6 ไปยังบิลท 7 แต่ไม่ม่ตัวทดเกิดขึ้นจาก บิท 7 หรือมีตัวทดออกจากบิท 7 โดยไม่มีตัวทดออกมาจากบิท 6 นอกนั้นแล้วจะทำให้ OV เป็น 0 บิทนี้มีประโยชน์มากเช่นในการบวกเลขจำนวนเต็มบวก Signed Integer (Signed Integer หมายถึงระบบจำนวนเลขที่ใช้บิทหน้าสุดแสดงเครื่องหมายของเลขจำนวนนั้น ถ้าบิทหน้าสุดเป็น 0 แสดงว่าเป็นจำนวนเต็มบวกถ้าเป็น 1 แสดงว่าเป็นจำนวนเต็มลบ) ถ้า OV ถูก set เป็น 1 แสดงว่าผลการบวกเลข Signed Integer จำนวนบวก 2 จำนวนให้ผลลัพธ์เป็นลบ หรืออาจเกิดจากเลขจำนวนลบ 2 จำนวนบวกกันแล้วให้ผลลัพธ์เป็นบวก คำสั่งในกลุ่มของ ADD A, <byte> มีดังนี้

ADD A, Rn

ADD A, direct

ADD A, #Ri

ADD A, #data

ตัวอย่าง

โปรแกรมบวกเลข Signed Integer 2 จำนวน คือ 00010000B และ 01110000B เข้าด้วยกันดังนี้

MOV A,#00010000B

ADD A,#01110000B

ซึ่งผลการบวกจะได้

A = 10000000B

จะเห็นว่าผลลัพธ์มีบิทหน้าสุดเป็น 1 แสดงว่าเป็นเลขจำนวนลบและในการบวกจะมีตัวทดจากบิท 6 ไปยังบิท 7 โดยไม่มีตัวทดเกิดจากบิท 7 ดังนั้นในการทำงานบ้างบนจะทำให้ OV ใน PSW มีค่าเป็น 1 Auxiliary carry bit มีค่าเป็น 0

ADDC A,<byte>

คำสั่งนี้เป็นคำสั่งที่มีการกระทำดังในช่อง Operation เป็นการบวกข้อมูลใน Accumulator ด้วย <byte>และตัวทดอันเกิดจากการกระทำทางคณิตศาสตร์ก่อนหน้าคำสั่งนี้ แล้วเก็บผลลัพธ์ไว้ใน Accumulator ข้อมูล <byte> จะสามารถชี้ได้ด้วยวิธีการ Dir,Ind, Reg หรือImm ในการกระ

ทำคำสั่ง ADDC จะทำให้ flag แต่ละบิทของ PSW เปลี่ยนแปลงเช่น เดียวกับคำสั่ง ADD A, <byte>

ตัวอย่าง

```
MOV   A,#00011111B
ADD   A,#11110000B
ADDC  A,#00000010B
```

ในการบวกของคำสั่งบรรทัด ที่ 2 จะทำให้บิท Carry ถูก set เป็น 1 ได้ผลลัพธ์เก็บไว้ในริจิสเตอร์ A ทำให้ข้อมูลในรีจิสเตอร์ A มีค่าเท่ากับ 00001111B ในบรรทัดที่ 3 จะทำการบวกข้อมูลในรีจิสเตอร์ A ด้วย 00000010 และ carry ซึ่งมีค่าเป็น 1 ทำให้ผลลัพธ์มีค่าเท่ากับ

```
00001111
00000010   +
00000001   +
00010010
```

คำสั่งในกลุ่มของ ADDC A,<byte> จะมีรูปแบบดังนี้

```
ADDC   A, Rn
ADDC   A, direct
ADDC A,# data
```

SUBB A,<byte>

เป็นคำสั่งทำการลบข้อมูลโดยเอาข้อมูล <byte> และตัวทดไปลบบออกจากข้อมูลใน Accumulator (รีจิสเตอร์ A) แล้วเก็บผลการลบไว้ใน Accumulator ข้อมูล <byte> สามารถชี้ตำแหน่ง (Addressing) ได้ 4 วิธีดังตารางช่อง Addressing Mode คำสั่งทำการลบนี้จะต้องลบด้วยตัวทดเสมอ ในการลบกันถ้าต้องใช้ตัวยืมในบิท 7 จะทำให้บิท Carry Flag ใน

PSW มีค่าเป็น 1 ถ้าไม่มีตัวยืมก็จะให้บิท Carry Flag เป็น 0 ส่วนบิท Auxiliary Carry จะเป็น 1 ถ้าเกิดตัวยืมในบิท 3 ของ Accumulator ระหว่างทำการลบ และบิท OV ของ PSW จะเป็น 1 ถ้ามีตัวยืมจากบิท 7 ไปยังบิท 6 แต่ไม่มีตัวยืมเข้าไปที่บิท 7 หรือมีตัวยืมเข้าไปที่บิท 7 แต่ไม่มีตัวยืมเข้าไปที่บิท 6

ตัวอย่าง

```
MOV   A,#0C9H
SUBB  A,#054H
```

ถ้าการคำนวณก่อนโปรแกรมนี้ทำให้บิท Carry Flag เป็น 1 ดังนั้นเมื่อสิ้นสุดการทำโปรแกรมจะได้ผลลัพธ์

```
11001001
01011000   -
00000001   -
00110000
```

ผลลัพธ์นี้จะเก็บไว้ใน Accumulator ในการทำงานข้างต้นจะมีตัวยืมจากบิท 7 ไปยังบิท 6 แต่ไม่มีตัวยืมเข้าไปยังบิท 7 ดังนั้นบิท OV จะมีค่าเป็น 1 ส่วนบิท Carry Flag และ Auxilary Carry Flag เป็น 0 คำสั่ง SUBB A, <byte< มีรูปแบบดังนี้

```
SUBB    A, Rn
SUBB    A, direct
SUBB    A, @Ri
SUBB    A,# data
```

INC DPTR

เป็นคำสั่งในเพิ่มค่าของรีจิสเตอร์ DPTR ไปอีก 1 DPTR นี้มีขนาด 16 บิท ดังนั้นรีจิสเตอร์ DPTR จะเก็บค่าได้ตั้งแต่ 0000H ถึง FFFFH ถ้า DPTR มีค่าเป็น FFFFH แล้วถูก

เพิ่มค่าไปอีก 1 ก็จะมีค่าเป็น 0000H โดยไม่มี Flag ใดเปลี่ยนแปลง รีจิสเตอร์ DPTR นี้ใช้เป็นรีจิสเตอร์สำหรับชี้ตำแหน่งของหน่วยความจำที่ต้องการอ่านหรือเขียนข้อมูลได้

ตัวอย่าง

```
MOV     DPTR,#2500H
INC     DPTR
```

ผลจากการทำงานของโปรแกรมข้างต้นจะทำให้ DPTR มีค่าเป็น 2501H

DEC A

เป็นคำสั่งให้ทำการลดค่ารีจิสเตอร์ A ลง 1 ถ้ารีจิสเตอร์ A มีค่าเป็น 0 อยู่ แล้วถูกลดค่าลง 1 จะทำให้รีจิสเตอร์ A มีค่าเป็น FFH คำสั่งนี้ไม่มีผลทำให้ Flag ใด ๆ ใน PSW เปลี่ยนแปลง

ตัวอย่าง

```
MOV     A,#15H
DEC     A
```

จะทำให้ Accumulator มีค่าเป็น 14H เมื่อสิ้นสุดการทำงาน

DEC <byte>

เป็นคำสั่งลดข้อมูลของหน่วยความจำ <byte> ลง 1 ถ้าข้อมูลเดิมเป็น 0 จะทำให้มีค่าเป็น FFH<byte> ที่ชี้ตำแหน่งหน่วยความจำสามารถทำได้ 3 วิธี คือ Dir,Ind และ Reg คำสั่งที่มีรูปแบบดังข้างบนคือ

```
DEC     Rn
DEC     Direct
DEC     @Ri
```

ตัวอย่าง

```
DEC     @R1
```

เป็นคำสั่งให้ลดข้อมูลในหน่วยความจำสำหรับข้อมูลภายใน 8051 ที่ชี้ตำแหน่งด้วยค่าของรีจิสเตอร์ R1 เช่น R1 มีค่า 15H ก็จะเป็นการลดค่าในหน่วยความจำสำหรับข้อภายใน 8051 ที่ตำแหน่ง 15H ลง 1

MUL AB

คำสั่งนี้ใช้สำหรับการคูณข้อมูลในรีจิสเตอร์ A ด้วยข้อมูลในรีจิสเตอร์ B รีจิสเตอร์ทั้ง 2 มีขนาด 8 บิท ดังนั้นผลลัพธ์ที่ได้จากการคูณจึงมีได้สูงสุด 16 บิท จึงต้องเก็บ 8 บิทบน (Highest order byte) ไว้ที่ รีจิสเตอร์ B และเก็บ 8 บิทล่าง (Lowest order byte) ไว้ที่รีจิสเตอร์ A ถ้าผลลัพธ์จากการคูณมีค่ามากกว่า 255 จะทำให้บิท 0V มีค่าเป็น 1 บิท Carry และ Auxiliary Carry ถูกเคลียร์ให้มีค่าเป็น 0 เสมอ
ตัวอย่าง

```
MOV     A,#25H
MOV     B,#30H
MUL     AB
```

โปรแกรมข้างบนจะเริ่มด้วยการกำหนดให้รีจิสเตอร์ A มีค่า 25H และรีจิสเตอร์ B มีค่า 30H ผลลัพธ์จากการคูณจะมีค่า 6F0H โดย 6 จะเก็บไว้ในรีจิสเตอร์ B และ F0H จะเก็บไว้ในรีจิสเตอร์ A บิท Overflow Flag มีค่าเป็น 1 เพราะผลลัพธ์ ของการคูณมีค่าเกิน 255

DIV AB

เป็นคำสั่งให้หารข้อมูลในรีจิสเตอร์ A ด้วยข้อมูลในรีจิสเตอร์ B ผลลัพธ์ของการหารจะถูกเก็บไว้ในรีจิสเตอร์ A และเศษของการหารจะเก็บไว้ในรีจิสเตอร์ B การทำงานของคำสั่งนี้จะทำให้บิท OV, Carry และ Auxilary Carry ถูกเคลียร์เป็น 0 เสมอ

ตัวอย่าง โปรแกรมเปลี่ยนเลขฐาน 16 เป็นฐาน 10

```
MOV     A,# 58H
```

```
MOV     B,# 10
DIV     AB
```

โปรแกรมนี้เปลี่ยนเลขฐาน 16 ค่าไม่เกิน 58H ที่เก็บไว้ในรีจิสเตอร์ A ให้เป็นเลขฐาน 10 ลัพธ์ที่เป็นหลักสิบจะเก็บไว้ที่รีจิสเตอร์ A มีค่าเท่ากับ 9 และผลลัพธ์ที่เป็นหลักหน่วยจะเก็บไว้ในรีจิสเตอร์ B มีค่าเท่ากับ 8

ANL A,<byte>

เป็นคำสั่งให้ทำการ AND ข้อมูลใน Accumulator กับข้อมูล <byte> ซึ่งชี้ตำแหน่งได้ 4 วิธีดังในตาราง Addressing Mode ผลลัพธ์ที่ได้จากการ AND ของข้อมูลจะเก็บไว้ใน Accumulator คำสั่งนี้ไม่ทำให้ Flag Bit ใด ๆ ใน PSW เปลี่ยนแปลง

ตัวอย่าง

```
MOV     A, # 01100110B
ANL     A, # 01000111B
```

คำสั่งแรกเป็นการกำหนดค่าให้กับรีจิสเตอร์ A มีค่า 01100110B หรือ 66H และในคำสั่งที่ 2 เมื่อ AND ข้อมูลในรีจิสเตอร์ Accumulator ด้วย 01000111B หรือ 47H ผลลัพธ์ที่ได้คือ

```
01100110
01000111
01000110 หรือ 46H
```

ผลลัพธ์นี้จะถูกเก็บไว้ในรีจิสเตอร์ A คำสั่งที่มีรูปแบบดังข้างบนคือ

```
ANL     A, Rn
ANL     A, direct
ANL     A, @Ri
ANL     A,# data
```

ANL <byte> ,A

คำสั่งนี้จะนำเอาข้อมูลใน <byte> และข้อมูลใน Accumulator มา AND กันแล้วเก็บไว้ใน <byte> โดยที่ข้อมูลเดิมในรีจิสเตอร์ A ไม่เปลี่ยนแปลง <byte> นี้จะชี้ตำแหน่งของข้อมูลได้วิธีเดียวคือ Direct Addressing ดังในตาราง Addressing Mode คำสั่ง ANL นี้ไม่มีผลต่อบิทของ flag ใด ๆ ใน PSW คำสั่งที่มีรูปแบบดัง ข้างบนมีคำสั่งเดียวคือ

ANL direct, A

ตัวอย่าง

ANL 2FH,A

คำสั่งนี้จะเอาข้อมูลในรีจิสเตอร์ A มา AND กับข้อมูลในหน่วยความจำที่ตำแหน่ง 2 FH แล้วเก็บผลลัพธ์ การ AND ไว้ยังตำแหน่ง 2FH ถ้าหน่วยความจำตำแหน่ง 2FH มีข้อมูล 10111011B และรีจิสเตอร์ A มีข้อมูล 11101110B เมื่อทำคำสั่ง ANL ,2FH,A จะทำให้ข้อมูลในหน่วยความจำตำแหน่ง 2FH จะมีข้อมูลเท่ากับ 10101010B และรีจิสเตอร์ A ยังคงมีค่าเป็น 11101110B

ANL <byte> , # data

เป็นคำสั่งให้นำเอาข้อมูลใน <byte> มา AND กับข้อมูลค่าหนึ่งโดยตรงแล้วเก็บผลลัพธ์ไว้ใน <byte> <byte> จะชี้ตำแหน่งได้วิธีเดียวคือ Direct Addressing คำสั่งนี้ไม่มีผลใด ๆ ต่อ Flag Bit ใน PSW คำสั่งที่มีรูปแบบดังข้างบนมีคำสั่งเดียวคือ

ANL direct, #data

ตัวอย่าง

ANL 2FH,# 10H

คำสั่งนี้จะนำเอาข้อมูลในหน่วยความจำสำหรับข้อมูลใน 8051 ตำแหน่ง 2FH มา AND กับ 10H แล้วเก็บผลลัพธ์ไว้ในหน่วยความจำสำหรับข้อมูลภายใน 8051 ที่ตำแหน่ง 2FH

ORL A, <byte>

เป็นคำสั่งให้ทำการ OR ข้อมูลใน Accumulator กับข้อมูล <byte> แล้วเก็บผลลัพธ์ไว้ใน Accumulator <byte> จะชี้ตำแหน่งหน่วยความจำได้ 4 วิธีคือ Dis, Ind, Reg และ Imm คำสั่งที่มีรูปแบบดังข้างบนคือ

```
ORL     A,direct
ORL     A,@Ri
ORL     A,Rn
ORL     A,# data
```

คำสั่งนี้ไม่มีผลใด ๆ ต่อ flag bit ใน PSW การกระทำของคำสั่ง ORL คือการกระทำแบบเดียวกับ OR operator

ตัวอย่าง

```
MOV     A,#10011001B
ORL     A,#01000010B
```

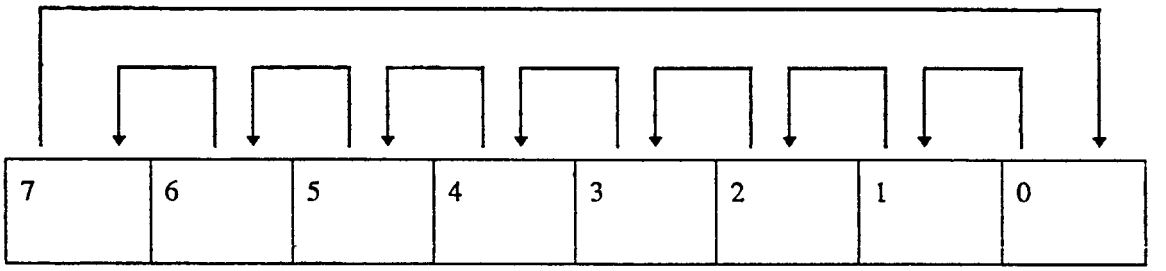
ผลลัพธ์ที่ได้จากการทำงานของโปรแกรมนี้คือ Accumulator จะมีค่าเท่ากับ 11011011B หรือ DDB

CLR A

คำสั่ง Clear Accumulator เป็นคำสั่งกำหนดค่า 00H ให้กับ Accumulator คำสั่งนี้ไม่มีผลต่อ Flag Bit ใด ๆ ใน PSW ถ้าเดิม Accumulator มีค่าเป็น 5CH อยู่เมื่อทำคำสั่งนี้แล้วจะทำให้ Accumulator มีค่าเป็น 00H คำสั่งนี้จะมีการทำงานเหมือนกับคำสั่ง MOV A,00H แต่ใช้เวลาในการทำงานคำสั่งนี้สั้นกว่า คือใช้เพียง 1 Machine Cycle เท่านั้น

RL A

คำสั่ง Rotate Accumulator to Left เป็นคำสั่งเลื่อนข้อมูลใน Accumulator ไปทางซ้าย 1 บิท เช่นข้อมูลในบิท 0 จะถูกเลื่อนไปที่บิท 1 ข้อมูลทั้ง 8 บิท จะถูกเลื่อนไปพร้อมกัน ข้อมูลจากบิท 7 ซึ่งเป็นบิท สุดท้ายจะถูกเลื่อนมายังบิท 0 ของ Accumulator ดังรูป



คำสั่งนี้ไม่มีผลต่อการเปลี่ยนแปลง flag bit ใน PSW

ตัวอย่าง

ให้ A มีค่า 10100000B เมื่อทำคำสั่ง RL A สิ้นสุดจะทำให้ Accumulator มีข้อมูล 01000001B

SWAP A

คำสั่งนี้จะทำการสลับข้อมูลระหว่าง 4 บิทบน (Highest nibble) และ 4 บิทล่าง (Lowest nibble) ของ Accumulator คำสั่งนี้ไม่มีผลต่อการเปลี่ยนแปลงของ flag bit ใด ๆ ใน PSW

ตัวอย่าง

```
MOV   A, #75H
SWAP  A
```

คำสั่งแรกจะทำให้ Accumulator มีค่า 75H คำสั่งที่ 2 จะทำให้เกิดการสลับข้อมูลระหว่างบิท 0-3 กับ บิท 4-7 ทำให้ Accumulator มีค่าเป็น 57H

MOV A,<src>

คำสั่งให้นำข้อมูลในตำแหน่ง <src> มาใส่ยัง Accumulator โดยข้อมูลใน <src> ก็ยังไม่เปลี่ยนแปลง <src> ใช้ชี้ตำแหน่งหน่วยความจำได้ 4 วิธีคือ Dir, Ind, Reg และ Imm คำสั่งที่มีรูปแบบดังข้างบนคือ

```
MOV     A, direct
MOV     A, @Ri
MOV     A, Rn
MOV     A, #data
```

ตัวอย่าง

```
MOV     A,27H
```

เป็นคำสั่งให้นำข้อมูลจากหน่วยความจำสำหรับตำแหน่ง 27H มาใส่ยัง Accumulator เช่นเดิมตำแหน่งหน่วยความจำ 27H มีข้อมูล 15H เมื่อสิ้นสุดการทำงานของคำสั่งดังกล่าวจะทำให้ Accumulator และหน่วยความจำตำแหน่ง 27H มีค่า 15H โดยไม่สนใจว่าค่าเดิมของ Accumulator เป็นเท่าใด คำสั่งดังกล่าวข้างบนจะเสมือนว่า Accumulator เป็น Destination ของคำสั่งเคลื่อนย้ายข้อมูล

MOV <dest> ,A

คำสั่งเคลื่อนย้ายข้อมูลทางเดียวโดยจะนำเอาข้อมูลใน Accumulator ไปเก็บที่ตำแหน่ง <dest> เสมือน A เป็น Source ของการเคลื่อนย้ายข้อมูล <dest> เป็นการชี้ตำแหน่งของหน่วยความจำที่จะเก็บข้อมูลซึ่งมีวิธีการชี้ตำแหน่งได้ 3 วิธี คือ Dir, Ind และ Reg คำสั่งเคลื่อนย้ายข้อมูลในกลุ่มนี้คือ

```
MOV     direct, A
MOV     Rn, A
MOV     @Ri, A
```

ตัวอย่าง

MOV 25H,A เป็นคำสั่งให้เก็บข้อมูลของ Accumulator ไปเก็บยังหน่วยความจำสำหรับข้อมูลภายใน 8051 ที่ตำแหน่ง 25H ถ้าก่อนทำคำสั่งนี้ Accumulator มีค่า 18H เมื่อสิ้นสุดการทำงานของคำสั่งนี้จะทำให้หน่วยความจำภายใน 8051 ตำแหน่ง 21H และ Accumulator มีค่าเป็น 18H

MOV <dest>,<src>

คำสั่งเคลื่อนย้ายข้อมูลจาก <src> ไปยังตำแหน่ง <dest> คำสั่งนี้จะมีการชี้ตำแหน่งหน่วยความจำของ <src> และ <dest> ได้ทั้ง 4 แบบ คือ Dir, Ind, Reg และ Imm คำสั่งนี้เป็นการเคลื่อนย้ายข้อมูลแบบทิศทางเดียว ซึ่งแตกต่างจากคำสั่งเคลื่อนย้ายข้อมูล 2 คำสั่งแรกที่มี Source หรือ Destination เป็น Accumulator แต่ในคำสั่งนี้จะทำการเคลื่อนย้ายข้อมูลจาก Source ไปยัง Destination ที่ไม่ใช่ Accumulator คำสั่งนี้ใช้เวลาในการทำงาน 2 Machine Cycle หรือ 24 ไซเคิลของสัญญาณนาฬิกา คำสั่งในกลุ่มนี้คือ

MOV	Rn, direct
MOV	Rn, # data
MOV	direct, Rn
MOV	direct, direct
MOV	Direct @Ri
MOV	Direct, # data
MOV	@Ri, direct
MOV	@Ri, #data

ตัวอย่าง

การทำงานของคำสั่งในกลุ่มนี้เช่น

MOV 25H,@R1

ถ้า R1 มีค่า 20H ดังนั้นการทำงานของคำสั่งนี้ก็คือการเคลื่อนย้ายข้อมูลจากหน่วยความจำภายใน 8051 ตำแหน่ง 20H ไปยังตำแหน่ง 25H โดยไม่เปลี่ยนแปลงข้อมูลในตำแหน่ง 20H เดิม

MOV DPRT, #data16

คำสั่งกำหนดข้อมูลให้กับ Register DPTR ให้มีค่า data 16 ซึ่ง data16 หมายถึงข้อมูลจำนวน 16 บิททั้งนี้เพราะ DPTR เป็นรีจิสเตอร์สำหรับชี้ตำแหน่งหน่วยความจำของ 8051 โดยข้อมูล 16 บิทใน DPTR สามารถอ้างอิงตำแหน่งได้ถึง 64× 1024 ตำแหน่ง

ตัวอย่าง

MOV DRTR,#1234H

จะทำให้รีจิสเตอร์ DPTR มีค่าเป็น 1234H

PUSH <src>

คำสั่งนี้เป็นคำสั่งเคลื่อนย้ายข้อมูลแบบหนึ่ง ซึ่ง <src> จะชี้หน่วยความจำเป็นแบบ Direct Addressing เท่านั้น เช่น

ตัวอย่าง

PUSH 25H

การทำงานของคำสั่งนี้จะเริ่มจากรีจิสเตอร์หนึ่งซึ่งชื่อ Stack Pointer หรือ SP เพิ่มค่าไป 1 ตัวอย่าง เช่นเดิม SP มีค่า 08H ก็จะเปลี่ยนเป็น 09H แล้วนำเอาข้อมูลจากหน่วยความจำสำหรับข้อมูลตำแหน่ง 25H ภายใน 8051 ไปเก็บยังตำแหน่ง 09H โดยที่ข้อมูลในตำแหน่ง 25 H เดิมจะไม่เปลี่ยนแปลง

POP <dest>

เป็นคำสั่งที่ทำงานเคลื่อนย้ายข้อมูลตรงข้ามกับคำสั่ง PUSH การชี้ตำแหน่งความจำของ <dest> เป็นแบบ direct addressing เท่านั้น

ตัวอย่าง

POP 20H

การทำงานของคำสั่งนี้ จะเริ่มต้นจากการอ่านข้อมูลของหน่วยความจำสำหรับข้อมูลภายใน 8051 ที่ชี้ตำแหน่งด้วย SP ถ้า SP มีค่า 09H ก็จะทำการอ่านข้อมูลจากตำแหน่ง 09H แล้วนำไปเก็บที่ตำแหน่ง <dest> ในที่นี้คือ 20H เสร็จแล้ว SP จะลดค่าตัวเองลง 1 เหลือ 08H

XCH A, <byte>

คำสั่งนี้เป็นการเคลื่อนย้ายข้อมูลแบบ 2 ทาง คือเป็นการแลกเปลี่ยนข้อมูลซึ่งกันและกัน โดยข้อมูล 8 บิทใน Accumulator จะแลกเปลี่ยนกับข้อมูลในตำแหน่ง <byte> <byte> จะชี้ตำแหน่งหน่วยความจำได้ 3 วิธีคือ Dir, Ind และ Reg คำสั่งในกลุ่มนี้คือ

```
XCH     A, direct
XCH     A, @Ri
XCH     A,R
```

ตัวอย่าง

```
XCH     A,@R0
```

ถ้า Accumulator มีค่า 80H และ RO มีค่า 12H หน่วยความจำสำหรับข้อมูลภายใน 8051 ตำแหน่ง 12H มีค่า 78H การทำงานของคำสั่งตัวอย่างคือการแลกเปลี่ยนข้อมูลจาก Accumulator กับข้อมูลในหน่วยความจำที่ชี้โดยค่าใน RO ซึ่งในที่นี้คือค่า 12H ดังนั้นเมื่อสิ้นสุดการทำงานคำสั่งข้างบนจะทำให้ Accumulator มีค่า 78H และหน่วยความจำสำหรับข้อมูลภายใน 8051 ตำแหน่ง 12H มีค่า 80H

MOVX A, @DPTR

เป็นคำสั่งอ่านข้อมูลจากหน่วยความจำสำหรับข้อมูลภายนอกที่ชี้ตำแหน่งโดย รีจิสเตอร์ DPTR ทำให้สามารถอ่านข้อมูลได้จากภายนอกถึง 64x1024 ตำแหน่ง ข้อมูลที่ชี้ตำแหน่งใน DPTR จะส่ง Address Low Byte ออกไปทาง Port 0 และ Address High Byte ไปทาง Port 2 จากนั้นจะอ่านข้อมูลจากหน่วยความจำข้อมูลภายนอกเข้ามาเก็บไว้ยัง Accumulator คล้ายกับคำสั่ง MOVX A, @Ri

ตัวอย่าง

```
MOVX    A,@DPTR
```

ให้ DPTR มีค่า 13F8H และที่หน่วยความจำสำหรับข้อมูลตำแหน่ง 13F8H มีข้อมูล 17H อยู่ เมื่อทำคำสั่งข้างต้นจะทำให้ Accumulator มีค่าเป็น 17H

MOVX @DPTR,A

เป็นคำสั่งส่งข้อมูลใน Accumulator ไปเก็บยังหน่วยความจำภายนอกที่ชี้ตำแหน่งด้วยข้อมูลใน DPTR การส่งตำแหน่งหน่วยความจำไปชี้หน่วยความจำภายนอกจะเหมือนกับคำสั่ง MOVX @DPTR,A

ตัวอย่าง

. ให้ DPTR มีค่า 1234H และ Accumulator มีข้อมูล 13H เมื่อทำงานคำสั่ง MOVX@DPTR,A จะเป็นคำสั่งให้เก็บค่า 13H ไปยังหน่วยความจำสำหรับข้อมูลภายนอกที่ตำแหน่ง 1234H

MOVC A,@A+DPTR

คำสั่งนี้จะอ่านข้อมูลจากหน่วยความจำสำหรับโปรแกรม ตำแหน่งของหน่วยความจำมีค่าเท่ากับค่าใน Accumulator บวกกับรีจิสเตอร์ DPTR ทำให้ได้ตัวชี้ตำแหน่ง 16 บิท ค่าที่อ่านได้จากหน่วยความจำตำแหน่งนั้นจะถูกนำไปเก็บไว้ที่ Accumulator

ตัวอย่าง ให้ Accumulator มีค่า 10H และ DPTR มีค่า 1234H การทำงานของ

MOVC A,@A+DPTR

จะอ่านข่อมูลจากหน่วยความจำสำหรับโปรแกรมตำแหน่ง 10H + 1234H = 1244H มาเก็บไว้ที่ Accumulator

JC rel

ถ้าบิท Carry เป็น 1 จะทำให้ข้ามการทำงานของโปรแกรมไปเท่ากับ rel ตำแหน่ง แต่ถ้าบิท Carry Flag เป็น 0 จะทำคำสั่งถัดไป

ตัวอย่าง

0000	MOV	A, # 25H
0002	ADD	A, # 83H
0004	JC	15H

ตัวอย่าง 4 หลักที่อยู่หน้าแต่ละแถวนั้น เป็นตำแหน่งของคำสั่งในหน่วยความจำสำหรับโปรแกรมนั้น ๆ เช่น คำสั่ง MOV A,25H ซึ่งมีภาษาเครื่องคือ 74H 25H จะถูกเก็บไว้ที่

ตำแหน่ง 0000 และ 0001 ตามลำดับ คำสั่ง JC 15H ก็เป็นคำสั่งที่มีภาษาเครื่อง 2 ไบท์ ดังนั้นคำสั่งที่อยู่ถัดจากคำสั่ง JC จะเริ่มต้นที่ตำแหน่ง 0006H

การทำงานของคำสั่งก่อนหน้า JC 15H ถ้าทำให้บิท Carry เป็น 1 เมื่อสิ้นสุดการทำงานคำสั่ง JC 15H ก็จะทำให้ข้ามไปทำงานยังตำแหน่ง 0006H + 15H คือตำแหน่ง 001BH แต่ถ้าคำสั่งการทำงานของโปรแกรมก่อนคำสั่ง JC ทำให้บิท Carry เป็น 0 จะไม่ทำให้เกิดการกระโดดของโปรแกรม โดยจะทำคำสั่งที่ต่อจากคำสั่ง JC นั่นเอง

JMP addr

เป็นคำสั่งให้ข้ามไปทำงานยังตำแหน่ง addr โดยไม่มีเงื่อนไข คำสั่งที่มีรูปแบบดังข้างต้นมี 3 คำสั่ง

1. AJMP addr

คำสั่งนี้จะให้ข้ามไปทำงานยังตำแหน่ง addr คำสั่งนี้มีภาษาเครื่อง 2 ไบท์ โดย 5 บิทล่างของไบท์แรกเป็นคำสั่ง (Operation Code) ส่วน 3 บิทบนและไบท์ที่ 2 รวม 11 บิทเป็นตำแหน่งของโปรแกรมที่ต้องการข้ามให้ไปทำงาน ดังนั้นคำสั่งนี้สามารถข้ามไปทำงานได้ในช่วงหน่วยความจำ 2048 ตำแหน่งจากตำแหน่งที่ปรากฏคำสั่งนี้ ซึ่งเรียกว่า Block 2K เมื่อเกิดการทำงานในคำสั่งนี้ตำแหน่งโปรแกรมที่ต้องการข้ามไปทำงานจะได้มาจาก 11 บิทในคำสั่งภาษาเครื่องของคำสั่ง AJMP และ 5 บิทจาก Program Counter ของคำสั่ง AJMP

ตัวอย่าง

0200	0125	AJMP	25H

ที่ตำแหน่ง 200H มีคำสั่ง AJMP 25H ดังนั้นเมื่อทำงานคำสั่งนี้จะเกิดการกระโดดข้ามไปยังตำแหน่งที่คำนวณได้ดังนี้

PC ของคำสั่งที่อยู่ถัดจาก AJMP= 0000 0010 0000 0010 = 0202

5 บิทบนของจาก PC มีค่า = 0000 0

ของคำสั่ง AJMP 25H = 0000 0001 0010 0101 = 0125

OP-CODE

3 บิทบน OP-CODE = 000

8 บิทล่างของ OP-CODE = 0010 0101

ตำแหน่งที่จะข้ามไปทำงานจึงเท่ากับ

0000 0 000 0010 0101 = 0025H

จาก PC Opcode Opcode

2. LJMP ADDR

Long Jump คำสั่งนี้มีภาษาเครื่อง 3 ไบท์โดย 1 ไบท์แรกเป็น Operation Code และ 2 ไบท์ที่เหลือคือ 16 บิทเป็นตำแหน่งของหน่วยความจำสำหรับโปรแกรมที่ต้องการข้ามไปทำงาน โดยไม่มีเงื่อนไข

ตัวอย่าง

LJMP 1500H

หมายถึงให้เริ่มไปทำงานที่ตำแหน่ง 1500H

3. SJMP rel

Short Jump เป็นคำสั่งที่มีภาษาเครื่อง 2 ไบท์ ในไบท์แรกเป็น Operation Code ส่วนไบท์ที่ 2 จะเป็นตำแหน่งของหน่วยความจำแบบอ้างอิงกับตำแหน่งของคำสั่งถัดไป ดังนั้นคำสั่งนี้สามารถข้ามการทำงานไปข้างหน้าได้ + 127 ตำแหน่งหรือย้อนหลังได้ - 128 ตำแหน่ง

ตัวอย่าง

0200 80E5 SJMP -25

จากคำสั่งภาษาเครื่องข้างบนจะเห็นว่า 80H เป็น OP-CODE และ E5H เป็นค่า 2's complement ของ -25

CALL addr

คำสั่งเรียกการทำงานของโปรแกรมย่อยการทำงาน (Subroutine) โปรแกรมย่อยนี้เป็นคำสั่งที่จะถูกเรียกทำงานได้จากหลาย ๆ ตำแหน่งในโปรแกรมเมื่อเสร็จสิ้นการทำงานของโปรแกรมย่อยแล้วจะสามารถกลับมาทำงานยังโปรแกรมเดิมในตำแหน่งที่ต่อจาก CALL ได้ การใช้โปรแกรมย่อยทำให้ไม่ต้องเขียนโปรแกรมนั้นซ้ำ ๆ กัน สามารถประหยัดหน่วยความจำได้ คำสั่งที่มีรูปแบบ CALL addr มี 2 คำสั่ง

ACALL

คำสั่งนี้มีภาษาเครื่อง 2 ไบท์ 5 บิทล่างของไบท์แรกเป็นส่วน Operation Code ส่วน 3 บิทบนและไบท์ที่ 2 รวม 11 บิท จะใช้ร่วมกับ 5 บิทบนของตำแหน่งหน่วยความจำของคำสั่งที่ต่อจากคำสั่ง ACALL รวมกันชี้ไปยังตำแหน่งเริ่มต้นของโปรแกรมย่อยที่จะต้องทำงาน การชี้ตำแหน่งหน่วยความจำของคำสั่งนี้จะเหมือนกับคำสั่ง AJMP

ตัวอย่าง

2000 3125 ACALL 125H

จะเป็นการเรียกโปรแกรมย่อยเริ่มต้นจากตำแหน่ง 2125H

LCALL addr เป็นคำสั่ง 3 ไบท์ Addr มีขนาด 16. บิทใช้เป็นตัวชี้ตำแหน่งเริ่มต้นของโปรแกรมย่อยที่ต้องการเรียกใช้งาน

ตัวอย่าง

2002 12 0125 ACALL 125H

การทำงานของคำสั่ง CALL นอกจากจะข้ามการทำงานไปยังตำแหน่งที่ต้องการแล้วยังเก็บตำแหน่งหน่วยความจำเดิมก่อนที่จะข้ามไปทำงาน เพื่อจะสามารถกลับมาทำงานต่อหลังจากคำสั่ง CALL แล้ว ตำแหน่งหน่วยความจำที่มีขนาด 16 บิท จึงต้องใช้หน่วยความจำ 2 ตำแหน่งสำหรับเก็บข้อมูล 16 บิทนี้ ตัวชี้ตำแหน่งเพื่อเก็บข้อมูลนี้คือ SP ซึ่งจะชี้ตำแหน่งในหน่วยความจำสำหรับข้อมูลภายใน 8051 โดย SP จะเพิ่มค่าไป 1 แล้วเก็บข้อมูลไบท์ล่างคือ บิท 0 ถึง 7 ของ PC จากนั้น SP จะเพิ่มค่าอีก 1 แล้วเก็บค่าบิท 8 ถึง 15 ของ PC

RET

เป็นคำสั่งที่จะใส่ไว้บรรทัดสุดท้ายของโปรแกรมย่อย เพื่อสั่งให้โปรแกรมย่อยกลับไปทำงานตำแหน่งที่ต่อจากคำสั่ง CALL ก่อนจะข้ามการทำงานมายังโปรแกรมย่อย การทำงานของคำสั่ง RET จะอ่านค่าจากหน่วยความจำสำหรับข้อมูลภายใน 8051 ที่ชี้โดย SP ไปไว้ยัง Program Counter บิท 8 ถึง 15 (PC8-7) ทำให้กลับไปทำงานยังตำแหน่งต่อจากคำสั่ง CALL ได้

NOP

คำสั่งนี้เป็นคำสั่ง No operation คือไม่มีการทำงานใด ๆ เลย คำสั่งนี้จะใช้เพื่อเสมือนเป็นการหน่วงเวลาในโปรแกรมเท่านั้น

JZ rel

คำสั่ง Jump on Zero การทำงานของคำสั่งนี้จะตรวจสอบ Accumulator ก่อน ถ้า Accumulator มีค่าไม่เป็น 0 ก็จะทำงานในคำสั่งถัดไป แต่ถ้า Accumulator มีค่าเป็น 0 ก็จะข้ามไปทำงานยังตำแหน่ง rel ซึ่งเป็นตำแหน่งหน่วยความจำแบบอ้างอิง

ตัวอย่าง

```
001000                      ORG     1000H
001000   54FF               ANL     A, # 0FFH
001002   6002               JZ      NEXT
001004   00                 NOP
001005   00                 NOP
001006   00          NEXT:  NOP
```

จากโปรแกรมข้างบน ถ้าเมื่อทำงานที่คำสั่ง ANL สิ้นสุดแล้วทำให้ Accumulator มีค่าเป็น 0 การทำงานของคำสั่ง JZ NEXT จะเกิดการกระโดดไปทำงานที่ตำแหน่ง 1006H แต่ถ้าการทำงานของคำสั่ง ANL ก่อนหน้านี้ไม่ทำให้ Accumulator เป็น 0 ก็จะไม่เกิดการข้ามการทำงานเนื่องมาจากคำสั่ง JZ NEXT โดยจะทำงานคำสั่งที่ต่อจากคำสั่ง JZ NEXT

ให้ดูภาษาเครื่องไบท์ที่ 2 ของคำสั่ง JZ จะเป็นค่า OFFSET ของตำแหน่งที่ต้องการข้ามไปทำงานโดยที่ OFFSET นี้นับจากตำแหน่งของคำสั่งที่อยู่ต่อจากคำสั่ง JZ

JNZ rel

คำสั่ง Jump on Zero การทำงานของคำสั่งนี้จะตรวจสอบ Accumulator ก่อนถ้า Accumulator มีค่าเป็น 0 ก็จะทำงานในคำสั่งถัดไป แต่ถ้า Accumulator มีค่าไม่เป็น 0 ก็จะข้ามไปทำงานยังตำแหน่ง rel ซึ่งเป็นตำแหน่งหน่วยความจำแบบอ้างอิง

ตัวอย่าง

```
002000                      ORG     2000H
002000 54FF                 ANL     A, # 0FFH
002002 7002                 JNZ     NEXT2
```

```
002004 00                       NOP
002005 00                       NOP
002006 00           NEXT2:      NOP
000000                          END
```

จากโปรแกรมข้างบน เมื่อทำงานที่คำสั่ง ANL แล้วทำให้ Accumulator มีค่าไม่เป็น 0 แล้ว การทำงานของคำสั่ง JZ NEXT2 จะเกิดการกระโดดไปทำงานที่ตำแหน่ง 1006H แต่ถ้าการทำงานของคำสั่ง ANL ก่อนหน้านี้ทำให้ Accumulator เป็น 0 ก็จะไม่เกิดข้ามการทำงานเนื่องมาจากคำสั่งที่ต่อจากคำสั่ง JNZ NEXT2

CJNE <byte> , # data , rel

คำสั่งให้เปรียบเทียบข้อมูลในตำแหน่ง <byte> กับ data ถ้าเท่ากันให้ทำคำสั่งถัดไป ถ้าไม่เท่ากันให้ข้ามไปทำที่ rel <byte> จะชี้ตำแหน่งได้ 2 วิธีคือ Ind และ Reg โดยคำสั่งที่มีรูปแบบดังกล่าวได้แก่

```
CJNE        Rn, # data , rel
CJNE        @Ri # data , rel
```

คำสั่งแรกจะเปรียบเทียบระหว่างข้อมูลในรีจิสเตอร์ Rn กับข้อมูล data และในคำสั่งที่ 2 จะเปรียบเทียบข้อมูลระหว่าง data กับข้อมูลในหน่วยความจำสำหรับข้อมูลที่ชี้ตำแหน่งโดย Ri (R0 หรือ R1)

ตัวอย่าง

```
CJNE        R7, #  25H, -45H
```

จะทำการเปรียบเทียบข้อมูลของรีจิสเตอร์ R7 กับข้อมูลในหน่วยความจำสำหรับข้อมูลภายใน 8051 ที่ตำแหน่ง 25H ถ้าค่าที่สองไม่เท่ากันก็ให้ข้ามการทำงานไปข้างหลัง 45H ตำแหน่ง แต่ถ้าค่าที่สองเหมือนกันก็จะทำงานที่คำสั่งต่อจากคำสั่ง CJNE

END

คำสั่ง Assembly Directive นี้อาจเขียนหรือไม่เขียนในโปรแกรมก็ได้ ถ้าเขียนจะต้องใส่ไว้ที่บรรทัดสุดท้ายของโปรแกรม เพื่อเป็นการบอกการสิ้นสุดของโปรแกรม สิ่งที่อยู่ต่อจากคำสั่ง END นั้นโปรแกรมแอสเซมเบลอร์จะไม่สนใจ

รูปแบบของคำสั่ง

label : END expression ; comment

Expression เป็นตัวเลขค่าหนึ่งอาจเขียนหรือไม่เขียนก็ได้ ถ้าเขียนจะถูกนำไปใช้เป็นตำแหน่งหน่วยความจำของคำสั่ง END ในแฟ้มของการแปลเป็นภาษาเครื่องที่เป็นรูปแบบ Hexidecimal และเป็นตำแหน่งหน่วยความจำนี้จะถูกนำไปใช้เป็นตำแหน่งของโปรแกรมที่ปรากฏเป็นคำสั่ง END ในรูป Intel และ Motorola Hexadecimal Machine Code File

ตัวอย่าง

```
        CPU     "8051.TBL"
        ORG     0000H
        MOV     A, # 25H
        ADD     A, #54H
        DWM     1, 2, 3
        NOP
        END
```

ในคำสั่ง END ไม่เขียน Expression จะให้ผลลัพธ์เป็นรูปแบบ Hexadecimal ของ Intel ได้ดังนี้

```
:020000020000FC
:0B0000074252454000100020003000DE
:00000001FF
```

ซึ่งในบรรทัดสุดท้ายจะมีค่าตำแหน่งในหลักที่ 4, 5, 6 และ 7 เป็น 0000H (ดูรายละเอียดในบทที่ 2.1) ถ้าโปรแกรมในตัวอย่างที่ 1 ถูกแก้ไขให้ Expression มีค่า 100H ผลจากการแปลภาษาแอสเซมบลีจะได้ดังนี้

```
000000                  CPU     "8051.TBL"
000000                  ORG     0000H
000000 7425             MOV     A, # 25H
```

```
000002 2454                ADD          A, # 54H
000004 000100020003  DWM          1, 2, 3
000100                     END          100H
```

และผลลัพธ์ที่เป็น Hexadecimal จะมีดังนี้

```
:020000020000FC
:0B0000074252454000010002000300DE
:00010001FE
```

ซึ่งจะเห็นว่าตำแหน่งของคำสั่ง END ในโปรแกรมจะเปลี่ยนไปเป็นค่า Expression ที่ต่อจาก END

EQU - Equate Lable

ใช้สำหรับกำหนดค่าจำนวนเต็มให้กับตัวแปร (label)

รูปแบบของคำสั่ง

label : EQU Expression ; comment

label เป็นตัวแปรที่ต้องการกำหนดค่าให้ เมื่อมีการอ้างอิงด้วยชื่อของตัวแปรในโปรแกรมแอสเซมเบลอร์จะถือเป็นการอ้างอิงค่าของตัวแปรนี้

expression เป็นค่าจำนวนเต็มที่จะกำหนดให้ label expression นี้จะต้องมีค่าเดียว

ตัวอย่าง

```
0000                       CPU          "8051.TBL"
0000                       HOF          "INT8"
0200                       ORG          0200H
0001=          NAME1:      EQU          1
0003=          NAME2:      EQU          3
0003           END
```

LABEL ชื่อ NAME1 จะถูกกำหนดให้มีค่า 0001H และ NAME2 จะถูกกำหนดให้มีค่าเป็น 003H เมื่อมีการอ้างอิงถึงชื่อ NAME1,NAME2 ในโปรแกรม ก็จะหมายถึงการอ้างอิงถึง LABEL นั่นเอง

ORG - Program Counter Origin

เป็นคำสั่ง ที่ใช้กำหนดว่าโปรแกรมที่อยู่ต่อจากคำสั่งนี้ จะเริ่มต้นที่ตำแหน่งหน่วยความจำเท่าไร คำสั่งนี้จะใช้กี่ครั้งในโปรแกรมก็ได้ เพราะผู้เขียนอาจไม่ต้องการให้โปรแกรมที่เขียนขึ้นอยู่ต่อเนื่องกัน

รูปแบบของคำสั่ง

label : ORG Expression ; comment

Expression เป็นตำแหน่งของหน่วยความจำเริ่มต้นที่ต้องการของโปรแกรมที่อยู่ตามมา

ตัวอย่าง

```
0000                CPU     "8051.TBL"
0000                HOF     "INT8"
0000                ORG     0000H
0000    7401        MOV     A, # 1
0002    7802        MOV     R0, # 2
0100                ORG     0100H
0100    7903        MOV     R1 # 3
0102    7A04        MOV     R2,# 4
0104                END
```

ในตัวอย่างบรรทัดที่ 3 เขียนไว้ว่า ORG 0 หมายความว่าคำสั่งแรกที่อยู่ตามมาจาก ORG จะถูกแปลเป็นภาษาเครื่องเก็บที่ตำแหน่ง 0 ในบรรทัดต่อมาทางซ้ายสุดมีค่าแอสแดสเป็น 0 และคำสั่ง ORG ที่ 2 จะกำหนดให้โปรแกรมเริ่มต้นที่ตำแหน่ง 100H ทำให้ค่าแอสเดสของคำสั่งที่อยู่ถัดไปเริ่มที่ตำแหน่ง 100H

โปรแกรมที่ใช้ในโครงงาน เครื่องวัดกำลังมนุษย์

```
;   PORT_I (V0) ---> V0/CONS1 ----------------\
;                                              *---\
;   PORT_V (V1) ---> V1/CONS2 ----------------/      |
;                                                    |
;     >>>>> POWER  <<<<<                          <--+
;                                                    |
;   PORT_TKGEN (V2) ---> V2*CONS3 ---------> /CONS4 ----/
;                  |
;     >>>>> RPM <<<<<     <--

;******** set constances  *********
CONS1          EQU    4
CONS2          EQU    8
CONS3          EQU    6
CONS4          EQU    32
PORT_I         equ    0F800h
PORT_V         equ    0F801h
PORT_TKGEN     equ    0F802h
CONTROL_PORT   equ    0F803h
RATE_REFRSH    equ    0          ;wait diaplay
CONS_I_UP      equ    2
CONS_I_CONVRT  equ    2
CONS_V_UP      equ    2
CONS_TKGEN     equ    5
CONS_LOSS      equ    3
CONS_K         equ    21
LCD_Write_I    equ    0FA00h
LCD_Read_I     equ    0FA01h
LCD_Write_D    equ    0FA02h
LCD_Read_D     equ    0FA03h
RESULT_CONS    equ    2
               org    8100h
start:         ljmp   main
```

```
;-------------------
init8255:
        mov     dptr,#CONTROL_PORT
        mov     a,#80h          ;all port out
        ret

;-------------------
; multiple 16 with 16 to 16
; AB * R0 -> AB (Limit 4 byte)
mul16:
        xch     a,r2
        push    acc
        mov     a,b
        xch     a,r3
        push    acc

        mov     a,r2
        mov     b,r0
        mul     ab
        push    acc             ;store lsb only
        mov     a,r3
        mov     b,r0
        mul     ab
        xch     a,b
        mov     r0,a
        pop     acc
        add     a,r0
        mov     r0,a

        pop     acc
        mov     r3,a
        pop     acc
        mov     r2,a
        mov     a,r0
        ret
```

```
;-------------------
; read data from 8255 since I,V,Gen
; T can read from 'V' cause convert to T
; R0R1 = I
; R2R3 = V
; R4R5 = Gen
get_data:
        ;push   dpl
        ;push   dph
        ;push   acc
        ;push   b

        ;mov    dptr,#PORT_I
        ;movx   a,@dptr
        ;mov    b,#CONS_I_CONVRT
        ;div    ab              ;convert to "I"
        ;mov    b,#CONS_I_UP
        ;mul    ab              ;up voltage
        ;mov    r0,b
        ;mov    r1,a            ;store

        ;mov    dptr,#PORT_I    ;PORT_V
        ;movx   a,@dptr
        ;mov    b,#CONS_V_UP
        ;mul    ab
        ;mov    r2,b
        ;mov    r3,a

        ;mov    dptr,#PORT_I    ;PORT_TKGEN
        ;movx   a,@dptr
        ;mov    b,#CONS_TKGEN
        ;mul    ab
        ;mov    r4,b
        ;mov    r5,a

        ;pop    b
```

```
        ;pop    acc
        ;pop    dph
        ;pop    dpl
        ;ret

;-------------------
; get data
; AB = Round per min.
; R0R1 = Power
get_data2:
        push    dph
        push    dpl

        mov     dptr,#PORT_TKGEN
        movx    a,@dptr             ;get data from port TK
        mov     b,#CONS3
        mul     ab                  ;find RMP
        push    acc
        push    b

        mov     dptr,#PORT_TKGEN
        movx    a,@dptr
        mov     b,#(CONS3/CONS4)        ;find value loss for multi to Power
        mul     ab
        push    acc
        push    b

        mov     dptr,#PORT_I
        movx    a,@dptr             ;find I
        mov     b,#CONS1
        div     ab
        push    acc

        mov     dptr,#PORT_V
        movx    a,@dptr
        mov     b,#CONS2            ;find V
```

```
        div     ab

        mov     b,a
        pop     acc             ;find Power
        mul     ab

        pop     0               ;get Loss
        pop     1
        call    add16           ;AB+R0R1->AB

        pop     0
        pop     1
        pop     dpl
        pop     dph
        ret

;-----------------
delay:  push    dpl
        push    dph
        push    acc
        mov     dptr,#LCD_Read_I
loop_delay:
        movx    a,@dptr
        anl     a,#10000000b
        jnz     loop_delay      ;LCD busy cause wait
        pop     acc
        pop     dph
        pop     dpl
        ret

;-----------------
; a is posittion
gotoxy:
        push    dpl
        push    dph
        mov     dptr,#LCD_Write_I
```

```
            call    delay
            movx    @dptr,a
            pop     dph
            pop     dpl
            ret

;--------------------
; a is data to LCD
display_LCD:
            push    dpl
            push    dph
            mov     dptr,#LCD_Write_D
            call    delay
            movx    @dptr,a
            pop     dph
            pop     dpl
            ret
;--------------------
: initial LCD
init_LCD:
            mov     dptr,#LCD_Write_I       ;PORT control LCD

            mov     a,#00111000b            ;init type LCD
            call    delay
            movx    @dptr,a

            mov     a,#00001100b            ;cursor on
            call    delay
            movx    @dptr,a

            mov     a,#01h                  ;function clear LCD
            call    delay
            movx    @dptr,a
            ret

;--------------------
```

```
; display string until data is  (ASCII 254)
: DPTR pointer by
; R3 amount of data
; R0 x axis
; R1 y axis
display_string:
        push    acc
        cjne    r1,#00h.display_s_line2
        mov     a,#80h
        jmp     set_position
display_s_line2:
        mov     a,#0C0h
set_position:
        orl     a,r0
        call    gotoxy
loop_next_display:
        mov     a,r3
        jz      end_display_string
        mov     a,#00h
        movc    a,@a+dptr
show_datatoLCD:
        call    display_LCD
        inc     dptr
        dec     r3
        jmp     loop_next_display
end_display_string:
        pop     acc
        ret

:-------------------
; display integer 8 bit to LCD
: A = data 1 byte
: R0 = x axis (0h..13h)
: R1 = y axis (0h..1h)
display8:
        push    acc
```

```
        push    b               ;use all
        xch     a,r2
        push    acc             ;use all
        mov     a,r3
        push    acc             ;counter
        push    dph
        push    dpl
        mov     r3,#00h
        mov     a,r2
repeat1:
        mov     b,#10           ;constance convert to BCD
        div     ab
        jz      next_display1
        push    b               ;push result to stack
        inc     r3              ;inc amount of data in stack
        jmp     repeat1
next_display1:
        cjne    r1,#00,line2_display
        mov     a,#080h
        jmp     convert_display
line2_display:
        mov     a,#0C0h
convert_display:
        add     a,r0
        call    gotoxy
        clr     c
        mov     a,b
next_display2:
        addc    a,#30h
        call    display_LCD
        mov     a,r3
        jz      end_display
        dec     r3
        pop     acc
        jmp     next_display2
end_display:
```

```
        pop     dpl
        pop     dph
        pop     acc
        mov     r3,acc
        pop     acc
        mov     r2,acc
        pop     b
        pop     acc
        ret

;-------------------------
; ADD-BCD 16 bit
: (Hight R7)AB + (Lower R7)R0R1 => (Hight R7)AB
adder16:
        xch     a,r2
        push    acc
        mov     a,r3
        push    acc
        push    dph
        push    dpl

        mov     a,b             ;simulate digit 0 (bit 3-0)
        anl     a,#0Fh
        mov     r3,a            ;store register
        mov     a,r1            ;with add
        anl     a,#0Fh
        add     a,r3
        push    acc
        clr     c
        subb    a,#0Ah          ;check less than 10
        pop     acc
        jc      adder16_next1
        push    acc
        mov     a,r1
        add     a,#10h
        mov     r1,a
```

```
        pop     acc
        clr     c
        subb    a,#0Ah
adder16_next1:
        push    acc
        mov     a,b
        anl     a,#0F0h
        swap    a
        mov     r3,a
        mov     a,r1
        anl     a,#0F0h
        swap    a
        add     a,r3
        push    acc
        clr     c
        subb    a,#0Ah
        pop     acc
        jc      adder16_next2
        inc     r0
        clr     c
        subb    a,#0Ah
adder16_next2:
        swap    a
        inc     dptr
        mov     r3,a
        pop     acc
        orl     a,r3
        push    acc

        mov     a,r2            ;simulate digit 2
        anl     a,#0Fh
        mov     r3,a            ;store register
        mov     a,r0            ;with add
        anl     a,#0Fh
        add     a,r3
        push    acc
```

```
        clr     c
        subb    a,#0Ah          ;check less than 10
        pop     acc
        jc      adder16_next3
        push    acc
        mov     a,r0
        add     a,#10h
        mov     r0,a
        pop     acc
        clr     c
        subb    a,#0Ah
adder16_next3:
        push    acc
        mov     a,r2            ;digit 4
        anl     a,#0F0h
        swap    a
        mov     r3,a
        mov     a,r0
        anl     a,#0F0h
        swap    a
        add     a,r3
        push    acc
        subb    a,#0Ah
        pop     acc
        jc      adder16_next4
        clr     c
        subb    a,#0Ah
        inc     r7
adder16_next4:
        swap    a
        mov     r3,a
        pop     acc
        orl     a,r3
        push    acc
        mov     a,r7
        anl     a,#0F0h
```

```
        swap    a
        mov     b,a
        mov     a,r7
        anl     a,#0Fh
        add     a,b
        swap    a
        mov     r7,a
        pop     acc
        mov     r0,a
        pop     b

        pop     dpl
        pop     dph
        pop     acc
        mov     r3,a
        pop     acc
        mov     r2,a
        mov     a,r0
        ret

;-------------------------
; display integer 16 bit to LCD
; AB = data
; R0 = x axis
; R1 = y axis
cons_table:  db   00h,00h,01h,00h, 00h,02h,00h,00h, 04h,00h,00h,08h
             db   00h,00h,16h,00h, 00h,32h,00h,00h, 64h,00h,01h,28h
             db   00h,02h,56h,00h, 05h,12h,00h,10h, 24h,00h,20h,48h
             db   00h,40h,96h,00h, 81h,92h,01h,63h, 84h,03h,27h,68h
display16:
        xch     a,r2        ;R2 store value of A
        push    acc
        mov     a,r3
        push    acc
        mov     a,r4
        push    acc
```

```
        mov     a,r5
        push    acc
        mov     a,r7
        push    acc
        push    dph
        push    dpl

        mov     dptr,#cons_table
        mov     a,r1            ;now no use R0 then push to stack
        push    acc             ;use R0 for addetion
        mov     a,r0            ;now no use R1 then push to stack
        push    acc             ;now R1 for addetion
        mov     r3,b            ;R3 store value of B
        mov     r0,#00h
        mov     r1,#00h
        mov     r5,#01h         ;init R5
        mov     r7,#00h
loop_check:
        mov     a,r3            ;check bit
        anl     a,r5            ;check digit
        jz      check_endloop
        mov     b,r1
        mov     a,r0
        push    acc
        mov     a,#00h
        movc    a,@a+dptr
        orl     a,r7
        mov     a,r7
        mov     a,#01h
        movc    a,@a+dptr
        mov     r0,a
        mov     a,#02h
        movc    a,@a+dptr
        mov     r1,a
        pop     acc
        call    adder16         ;adder BCD
```

```
        mov     r0,a
        mov     r1,b
check_endloop:
        inc     dptr
        inc     dptr
        inc     dptr
        mov     a,r5
        rl      a
        mov     r5,a
        anl     a,#11111110b
        jz      next_adder
        jmp     loop_check
next_adder:                     ;high byte
        mov     r5,#01h
high_adder:
        mov     a,r2
        anl     a,r5
        jz      next_check_loop
        mov     b,r1
        mov     a,#00h
        movc    a,@a+dptr
        orl     a,r7
        mov     r7,a
        mov     a,#02h
        movc    a,@a+dptr
        mov     r1,a
        mov     a,r0
        push    acc
        mov     a,#01h
        movc    a,@a+dptr
        mov     r0,a
        pop     acc
        call    adder16
        mov     r0,a
        mov     r1,b
next_check_loop:
```

```
        inc     dptr
        inc     dptr
        inc     dptr
        mov     a,r5
        rl      a
        mov     r5,a
        anl     a,#11111110b
        jz      complete_change_BCD
        jmp     high_adder
complete_change_BCD:            ;now result store to (Hight R7)R0R1
                                ;old R0R1 in stack
        pop     acc
        mov     b,a
        pop     acc
        jz      show_line2
        mov     r2,#0C0h
        jmp     next_show
show_line2:
        mov     r2,#80h
next_show:
        mov     a,b
        orl     a,r2
        call    gotoxy
        mov     r2,#00H         ;MSB is zero
        mov     a,r7            ;Write msb
        jz      show_digit4
        anl     a,#0F0h
        swap    a
        add     a,#30h
        call    display_LCD
        inc     r2
show_digit4:
        mov     a,r0
        anl     a,#0F0h
        jz      show_digit4_2
show_digit4_1:
```

```
            swap    a
            add     a,#30h
            call    display_LCD
            inc     r2
            jmp     show_digit3
show_digit4_2:
            cjne    r2,#00,show_digit4_1
            ;push   acc
            ;mov    a,r2
            ;pop    acc
            ;jnz    show_digit4_1
show_digit3:
            mov     a,r0
            anl     a,#0Fh
            jz      show_digit3_2
show_digit3_1:
            add     a,#30h
            call    display_LCD
            inc     r2
            jmp     show_digit2
show_digit3_2:
            cjne    r2,#00,show_digit3_1
            ;push   acc
            ;mov    a,r2
            ;pop    acc
            ;jnz    show_digit3_1
show_digit2:
            mov     a,r1
            anl     a,#0F0h
            jz      show_digit2_2
show_digit2_1:
            swap    a
            add     a,#30h
            call    display_LCD
            inc     r2
            jmp     show_digit1
```

```
show_digit2_2:
        cjne    r2.#00.show_digit2_1
        ;push   acc
        ;mov    a,r2
        ;pop    acc
        ;jnz    show_digit2_1
show_digit1:
        mov     a,r1
        anl     a,#0Fh
        add     a,#30h
        call    display_LCD

        pop     dpl
        pop     dph
        pop     acc
        mov     r7,acc
        pop     acc
        mov     r5,acc
        pop     acc
        mov     r4,acc
        pop     acc
        mov     r3,acc
        pop     acc
        mov     r2,acc
        ret

;---------- adder 16 bit ----------
; adder binary 16
; AB + R0R1 --> AB & carry flag
add16:
        push    acc
        mov     a,b
        add     a,r1
        mov     b,a
        pop     acc
        addc    a,r0
```

```
        ret

;------------ MAIN ---------------
;*************** data ********************
Line1:      db '                ' ;Power       W   '
Line2:      db '                ' ;Round       rpm '
Blank:      db '    '
main:
;*************** Header *****************
        call   init_LCD
        call   init8255

        mov    r3,#20
        mov    r0,#0
        mov    r1,#0
        mov    dptr,#Line1
        call   display_string      ;write Line 1

        mov    r3,#20
        mov    r0,#0
        mov    r1,#1
        mov    dptr,#Line2
        call   display_string      ; write Line 2
;*************** display data ***************
again:
        ;call   get_data2
        ;push   acc
        ;push   b                  ;push Round per min.
        ;mov    a,r0
        ;push   acc
        ;mov    a,r1
        ;push   acc                ;push Power

        ;mov    a,r2
        ;push   acc
        ;mov    a,r3
```

```
        ;push   acc
        ;mov    a,r4
        ;push   acc
        ;mov    a,r5
        ;push   acc                 ; store all data to stack

        mov     r0,#00h
        mov     r1,#00h
        mov     r3,#16
        mov     dptr,#Line1
        call    display_string      ; write blank
        ;pop    b
        ;pop    acc
;*******
        mov     dptr,#PORT_I
        movx    a,@dptr
        mov     b,#00
        xch     a,b
;*******
        mov     r0,#00
        mov     r1,#00h
        mov     r7,#00h
        call    display16           ; write V

        ;mov    r0,#06
        ;mov    r1,#01h
        ;mov    r3,#6
        ;mov    dptr,#Blank
        ;call   display_string      ; write blank
        ;pop    b
        ;pop    acc
;*******
        mov     dptr,#PORT_V
        movx    a,@dptr
        mov     b,#00
        xch     a,b
```

```
;*******
        mov     r0,#05
        mov     r1,#00h
        mov     r7,#00h
        call    display16           ; write V

;*******
        mov     dptr,#PORT_TKGEN
        movx    a,@dptr
        mov     b,#00
        xch     a,b
;*******
        mov     r0,#10
        mov     r1,#00h
        mov     r7,#00h
        call    display16           ; write Gen

        mov     r0,#00h
        mov     r1,#01h
        mov     r3,#16
        mov     dptr,#Line1
        call    display_string      ; write blank

        call    get_data2
        push    acc
        push    b                   ;push Round per min.
        mov     a,r0
        push    acc
        mov     a,r1
        push    acc                 ;push Power

        mov     r0,#00
        mov     r1,#01h
        mov     r7,#00h
        pop     acc
        pop     b
```

```
        call    display16           ; write Gen

        mov     r0,#08
        mov     r1,#01h
        mov     r7,#00h
        pop     b
        pop     acc
        call    display16           ; write Gen

;***************** delay *********************
delay22:
        mov     r1,#RATE_REFRSH
delay23:
        dec     r1
        nop
        nop
        nop
        cjne    r1,#00,delay23
        dec     r0
        nop
        nop
        nop
        cjne    r0,#00,delay22
        jmp     again

        end.
```

ภาคผนวก ข.

สารอาหาร

สารอาหารที่ร่างกายต้องมีอยู่หลายชนิดแบ่งได้ดังนี้

1. คาร์โบไฮเดรท

1.1 คาร์โบไฮเดรทมีหน้าที่ให้พลังงาน คาร์โบไฮเดรท 1 กรัมให้พลังงาน 4 แคลอรี่

1.2 ควรกินอาหารที่มีคาร์โบไฮเดรทประมาณ 50 % ของความต้องการพลังงานทั้งหมดของร่างกาย

1.3 ความต้องการคาร์โบไฮเดรทของร่างกายต่อวัน

ความต้องการคาร์โบไฮเดรทของร่างกายขึ้นอยู่กับน้ำหนัก ขนาดของร่างกาย งานที่ทำเป็นประจำของคนนั้นๆ เช่นชาวนา ชาวสวน มีความต้องการมากกว่าคนอาชีพอื่นๆ 6-7 กรัมต่อน้ำหนักตัว 1กิโลกรัม

อย่างไรก็ดีไม่ว่าจะเป็นคนในอาชีพใด อย่างน้อยที่สุดวันหนึ่งๆ ควรจะได้รับประมาณ 4-6 กรัมต่อน้ำหนักตัว 1 กิโลกรัม

2. ไขมัน

2.1ไขมันให้พลังงานมากกว่าสารอื่น ๆ ไขมัน 1 กรัม ให้พลังงาน 9 แคลอรี่

2.2 ไม่ควรกินไขมันเกิน 30% ของความต้องการพลังงานทั้งหมดของร่างกาย

2.3 ความต้องการของร่างกายต่อวัน

ความต้องการอาหารไขมัน หรือพลังงานจะไม่เท่ากันทุกคน ซึ่งจะขึ้นอยู่กับเพศ อายุ และที่สำคัญก็คือ ประเภทของกิจกรรมที่ทำอยู่ประจำวัน รวมทั้งการใช้แรงกายเป็นประจำหรืองานเบามากน้อยเพียงใดด้วย

กิจกรรมแบ่งเป็น 3 ประเภท

1. งานเบา ได้แก่ ผู้ทำงานในสำนักงาน ผู้ชำนาญการทางด้านอาชีพต่าง ๆ เช่น แพทย์ ครู นักบัญชี สถาปนิก แม่บ้าน ที่ทำงานโดยมีเครื่องผ่อนแรง เสมียนหน้าร้าน และผู้ที่กำลังหางานทำ

2. งานหนักปานกลางได้แก่ ผู้ทำงานในโรงงานอุตสาหกรรมประเภทเบา นักศึกษา พนักงานร้านขายยา ชาวประมง พนักงานหญิงในห้างสรรพสินค้า แม่บ้านที่ทำงานโดยไม่มีเครื่องผ่อนแรง

3. งานหนัก ได้แก่ ชาวนา ชาวไร่ ชาวสวน กรรมกรแบกห้าม พนักงานป่าไม้ ทหารประจำการ กรรมกรขุดแร่ กรรมกร และนักกีฬา

วิธีหาจำนวนแคลอรี่ที่ต้องการให้เอาน้ำหนักตัว คูณด้วยจำนวน แคลอรี่ในประเภทที่อยู่ในอาชีพของตนเอง จากตารางข้างล่าง

เพศ	งานเบา	งานปานกลาง	งานหนัก
ชาย	1.55	1.78	2.10
หญิง	1.56	1.64	1.82

3. โปรตีน

3.1 โปรตีน 1 กรัม ให้พลังงาน 4 แคลอรี่

3.2 ความต้องการโปรตีนของร่ากายแต่ละคนจะไม่เท่ากันขึ้นอยู่สภาพร่างกาย อายุ เพศ แตกต่างกันออกไป โดยทั่วไป ผู้ใหญ่ควรได้รับประมาณ 15% ของความต้องการพลังงานทั้งหมดของร่างกาย

ผู้ใหญ่ 0.75 กรัมต่อน้ำหนักตัว 1 กิโลกรัม

หญิงมีครรภ์ เพิ่มขึ้นอีกวันละ 7 กรัม

หญิงให้นมลูก เพิ่มขึ้นอีกวันละ 19 กรัมในระยะ 6 เดือนแรก และ 14 กรัมในระยะ 6 เดือนหลัง

เด็กอายุ 3-5 เดือน 1.85 กรัม ต่อน้ำหนักตัว 1 กิโลกรัม

เด็กอายุ 9-11 เดือน 1.50 กรัม ต่อน้ำหนักตัว 1 กิโลกรัม

เด็กอายุ 1-3 ปี 1.2 กรัม ต่อน้ำหนักตัว 1 กิโลกรัม

เด็กอายุ 10-12 ปี 1.00 กรัม ต่อน้ำหนักตัว 1 กิโลกรัม

เด็กอายุ 16-19 ปี 0.80 กรัม ต่อน้ำหนักตัว 1 กิโลกรัม

อาหาร	ปริมาณ	น้ำหนักประมาณ (กรัม)	พลังงานปริมาณ (แคลอรี่)
เนย	1 ช้อนโต๊ะ	-	100
ครีมใส่กาแฟ	1 ช้อนโต๊ะ	-	30
นมอย่าไม่เอาครีมออก	1 ถ้วย	-	150
โยเกิร์ต	1 กล่อง	-	125
ไข่	1 ฟอง	-	80
น้ำมันถั่วเหลืองสำหรับทอด	1 ช้อนโต๊ะ	-	120
ปลาทอด	1 ชิ้น	65	135
เนื้อปู	1 ถ้วย	-	135

อาหาร	ปริมาณ	น้ำหนักประมาณ (กรัม)	พลังงานปริมาณ (แคลอรี่)
เนื้อวัวไม่ติดมัน	1 ชิ้น	85	140
เนื้อหน้าอกไก่	1 ชิ้น	80	160
กล้วยหอม	1 ผล	-	100
กล้วยไข่ กล้วยน้ำว้า	1 ผล	-	น้อยกว่า 100
ตับต้ม	1 ชิ้น	85	195
หมูติมัน	1ชิ้น	85	320
บะหมี่ลวกสุกแล้ว	-	160	200
ถั่วลิสงต้ม	1ช้อนโต๊ะ	180	840
แยมส้ม แยมสตรอเบอรี่	1ถ้วย	-	50
บร็อกคอลี่ตัดเป็นชิ้น	1 ถ้วย	180	45
กะหล่ำปลีตัดเป็นชิ้น	1 ถ้วย	180	45
ข้าวโพดหวานต้ม	1 ฝัก (กลาง)	-	70
ผัดกาดแก้ว	1 หัว	-	25
ฟักทองต้ม	1 ชิ้น	245	80
มะเขือเทศ	1 ลูก (2 นิ้ว)	-	20
หมูเนื้อแดงล้วน	1 ชิ้น	85	160
ลูกองุ่น	1 ถ้วย	-	50
ส้มเขียวหวานขนาดกลาง	1 ผล	100	60
มะละกอหั่นเป็นชิ้น	1 ถ้วย	140	55
ส้มโอ	1 กลีบ	90	110
ขนมปัง	1 แผ่น	-	80
ขนมเค้กไม่มีหน้า	1 ชิ้น	80	315

สูตรการเทียบพลังงานมีดังต่อไปนี้

1 British thermat unit (Btu) = 1054.8 joules , 1 joules = 10 Mergs

1 Calories (cal) = 4.1840 joules

1 Kilowatt hour (Kwh) = 3413 Btu = 3.6 Mjoules

1 Horsepower (hp) = 2545 Btu/h = 178.2 csl/sec = 0.74570 kw

1 Kilowatt (kw) = 1000 watts = 3413 Btu/h = 238.9 cal/sec

DMC202

• Display Format(20character ×2line) • Display Fonts(5×8dots) • Driving Method(1/16D)

ABSOLUTE MAXIMUM RATINGS

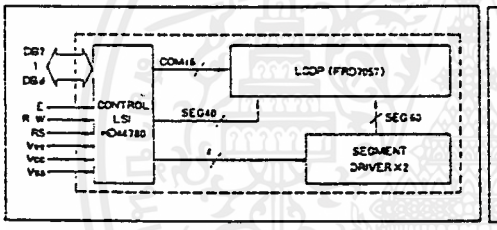
Item	Symbol	Test Condition	Standard Value min.	typ.	max.	Unit
Power Supply Voltage for Logic	Vcc−Vss	——	0	—	7	V
Power Supply Voltage for LCD Drive	Vcc−Vee	——	0	—	13.5	V
Input Voltage	Vi	——	Vss	—	Vcc	V
Operating Temperature	Ta	——	0	—	+50	℃
Storage Temperature	Tstg	——	−20	—	+70	℃

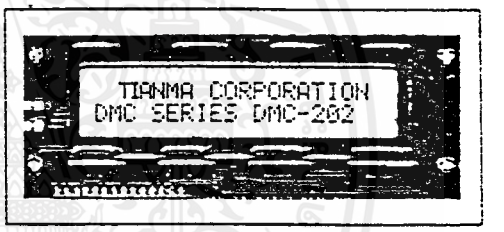
ELECTRICAL CHARACTERISTICS

Item	Symbol	Test Condition	Standard Value min.	typ.	max.	Unit
Input "High" Voltage	V_{IH}	——	2.2	—	Vcc	V
Input "Low" Voltage	V_{IL}	——	−0.3	—	0.6	V
Output "High" Voltage	V_{OH}	$-I_{OH}$=0.205mA	2.4	—	—	V
Output "Low" Voltage	V_{OL}	I_{OH}=1.2mA	—	—	0.4	V
Power Supply Current	Icc	Vcc=5.0V	—	1.5	3.0	mA

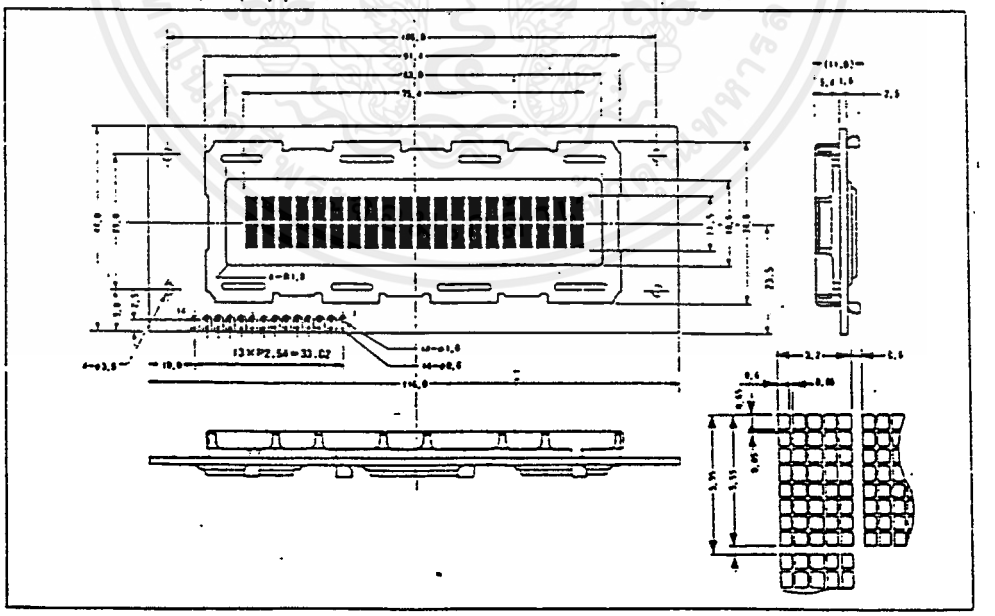
※Vcc=5.0V±5%, Ta=25°C

Block diagram





External dimensions / Display pattern



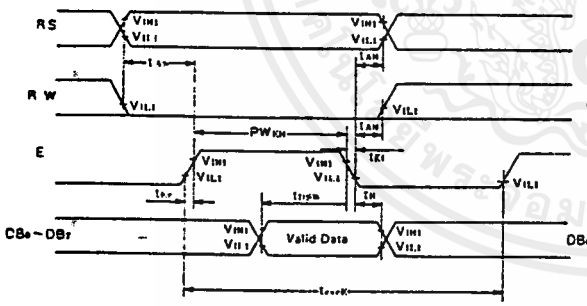
ภาพแสดงผังเวลาของขาสัญญาน

TIMING CHART 時序

Item （項目）	Symbol 符號	Measuring Condition 測試條件	（標準值） Standard Value min.	typ.	max.	Unit 單位
Enable Cycle Time （允許時間周期）	TcycE	Figs. 1、2	1000	—	—	nS
Enable Pulse Width, High Level （允許脈沖寬度、高電平）	PWEH	Figs. 1、2	450	—	—	nS
Enable Rise and Decay Time （允許上升和下降時間）	tEr,tEf	Figs. 1、2	—	—	25	nS
Address Setup Time,RS,R/W－E 地址建立時間）	tAS	Figs. 1、2	140	—	—	nS
Data Delay Time （數據延遲時間）	tDDR	Fig.2	—	—	320	nS
Data Setup Time （數據設置時間）	tDSW	Fig. 1	195	—	—	nS
Data Hold Time （數據保持時間）	tH	Fig. 1	10	—	—	nS
Data Hold Time	tDHR	Fig.2	20	—	—	nS
Address Hold Time（地址保持時間）	tAH	Figs. 1、2	10	—	—	nS

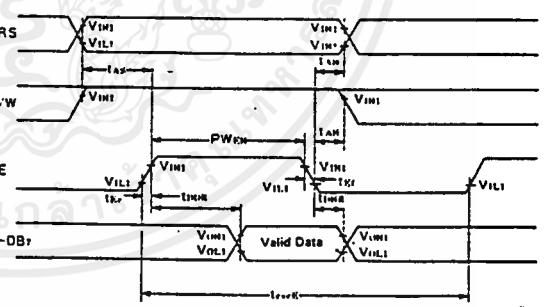
※ Vcc=5.0V ± 5%、Ta=25℃

FIG. 1 WRITE OPERATION （寫操作）



(Write Data from MPU to MODULE)

FIG. 2 READ OPERATION （讀操作）

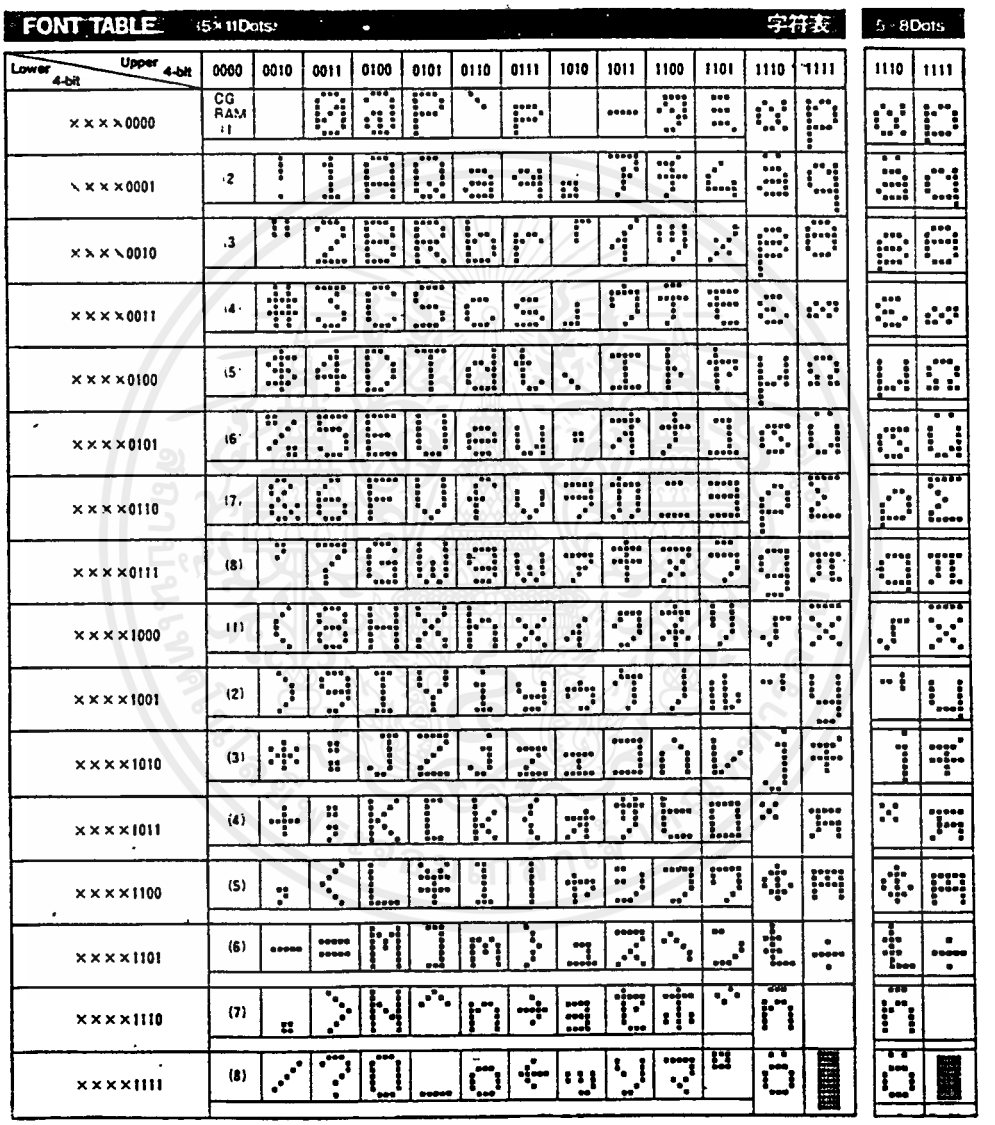


(Reading Data from MODULE to MPU)

DMC Series

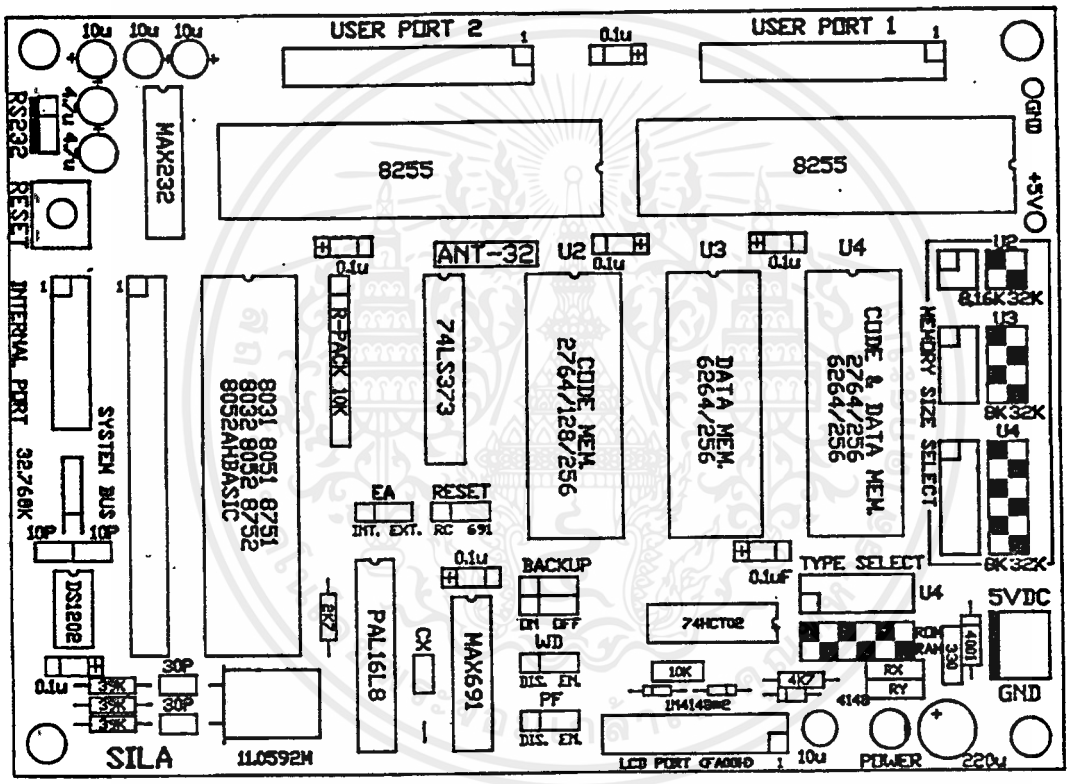
Type No.		DMC161C	DMC162	DMC164	DMC202	DMC204	DMC402	DMC404
Display Format		16 charactersX1 line	16 charactersX2 lines	16 charactersX4 lines	20 charactersX2 line	20 charactersX4 lines	40 charactersX2 lines	40 charactersX4 lines
Display Fonts		5X8 dots	5X8 dots	5X8 dots	5X8 dots	5X8 dots	5X8 dots	5X8 dots
Module Size (WXHXTmm)		80X36X10	80X36X11	87X60X11	116X37X11	98.0X60X11	182X33.5X11	190X54X10
View Area (WXHmm)		64.5X13	64.5X13.8	61.8X25.2	83X18.6	76X25.2	152.5X16.5	147X29.5
Character Size (WXHmm)		3.2X5.95	2.95X4.35	2.95X4.75	3.2X5.55	2.95X4.75	3.2X5.55	2.78X4.89
Dot Size (WXHmm)		0.6X0.7	0.55X0.5	0.55X0.55	0.6X0.65	0.55X0.55	0.6X0.65	0.5X0.55
Recommended Power Supply	Vcc~Vss(V)	+5	+5	+5	+5	+5	+5	+5
	Vee~Vss(V)	—	—	—	—	—	—	—
Driving Method (Duty)		1/8D-1/4B	1/16D-1/5B	1/16D-1/5B	1/16D-1/5B	1/16D-1/5B	1/16D-1/5B	1/16D-1/5B
Power Consumption typ.(mW)		10	10	20	10	25	25	50
Weight(g)		30	40	60	30	65	75	100
Operating Temp.(℃)		0~+50	0~+50	0~+50	0~+50	0~+50	0~+50	0~+50
Storage Temp.(℃)		-20~+70	-20~+70	-20~+70	-20~+70	-20~+70	-20~+70	-20~+70

ตารางตัวอักษร

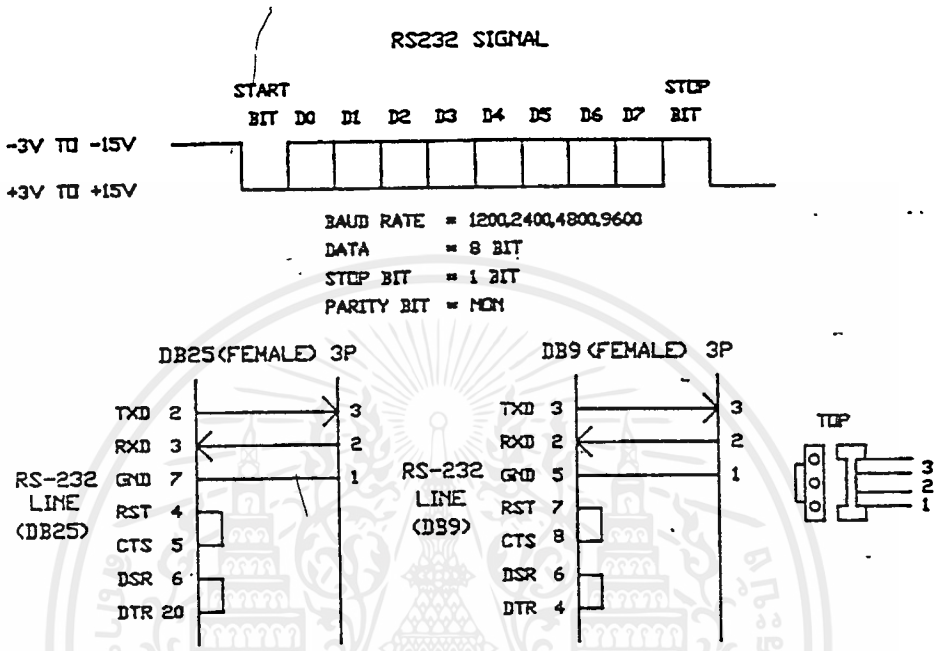


※CG RAM : Character pattern area can be rewritten by program.

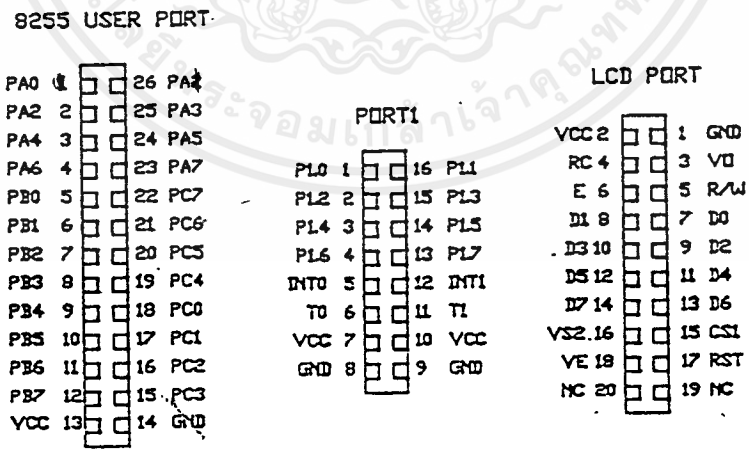
ภาพแสดงบอร์ดและตำแหน่งจัมพ์เปอร์



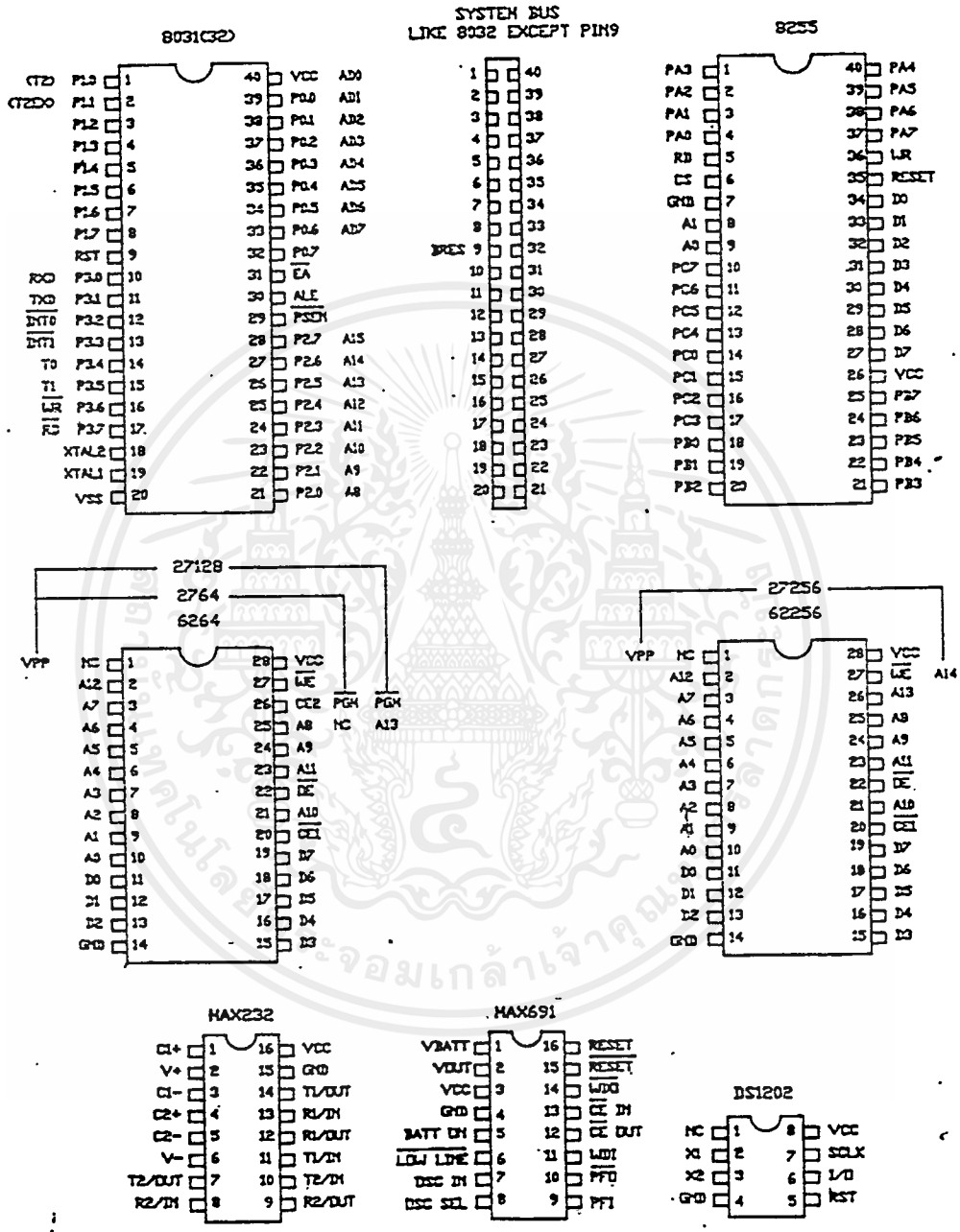
ภาพแสดงลักษณะสัญญาณ RS232 และการต่อสาย

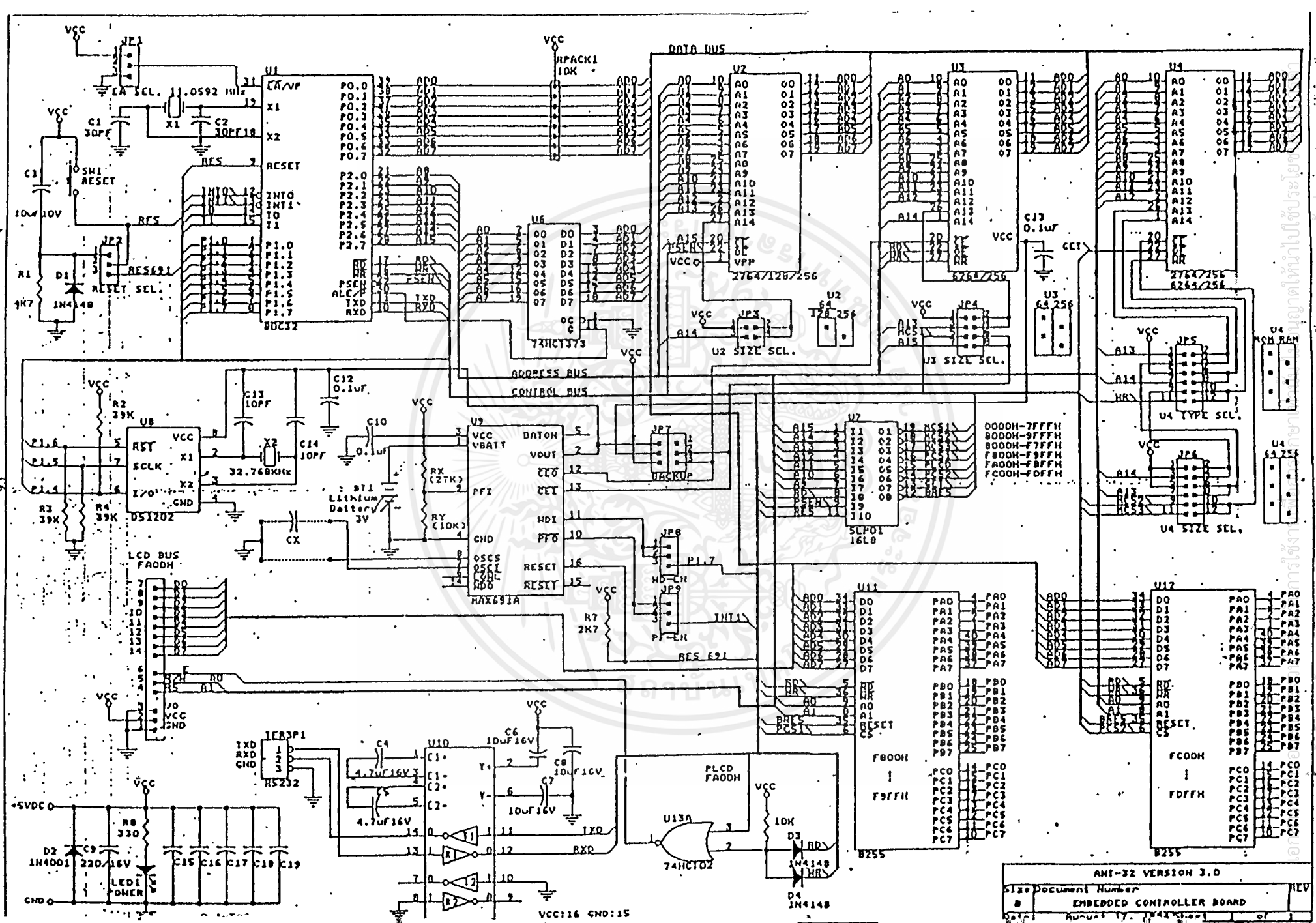


ภาพแสดงรายละเอียด CONNECTOR และ CHIP



ภาพแสดงรายละเอียดของ CONNECTOR และ CHIP (ต่อ)




DATA BUS
ADDRESS BUS
CONTROL BUS
U1
80C32
U2
27647128/256
U3
6764/256
U4
27647256
6764/256
U6
74HCT373
U7
SLP01
16L8
U8
DS1202
U9
MAX691A
U10
MAX232
U11
8255
U12
8255
U13A
74HCT02
U2 SIZE SEL.
U3 SIZE SEL.
U4 TYPE SEL.
U4 SIZE SEL.
BACKUP
RES 691
LCD BUS
F400H
PLCD
F400H
F800H
F9FFH
FC00H
FDFFH
0000H-7FFFH
8000H-87FFH
8000H-F7FFH
F800H-F9FFH
FA00H-FBFFH
FC00H-FDFFH
ANT-32 VERSION 3.0
EMBEDDED CONTROLLER BOARD
VCC:16 GND:15

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ สำหรับลุล่วงไปด้วยดีทั้งด้านวงจร ด้านโปรแกรม และโครงสร้าง ทั้งนี้ต้องขอขอบพระคุณท่านอาจารย์ พิชิต ลำยอง ที่ได้ช่วยให้คำปรึกษาและคำแนะนำที่ดีและต้องขอขอบคุณ คุณ เกรียงไกร ประทุมพร ที่ได้ให้การช่วยเหลือในการเขียนโปรแกรมและคำปรึกษาด้านอิเล็กทรอนิกส์

ที่ขาดมิได้ในการทำงานคือ ปัจจัยด้านทุนทรัพย์ ต้องขอกราบขอบพระคุณ คุณพ่อ คุณแม่ และพี่ๆ ที่อุปการะสนับสนุน ทางด้านการเงินกำลังใจด้วยดีตลอดมาและขอขอบคุณเพื่อน ๆ ทุกคนไว้ ณ.โอกาสนี้ด้วย

นาย ธวัชชัย ดำเนินศิลป์

นาย วิรัต เพชรบุรี

เอกสารอ้างอิง

[1] Ramakant A.Gayakwad ," Op-Amp and Linear Integrated Circuits",Prentice Hall International Editions,1993

[2] กิตติ ตีรเศรษฐ,"พื้นฐานวิศวกรรมระบบควบคุม", คณะวิศวกรรมศาสตร์ , สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,2539

[3] กฤษดา วิศวธีรานนท์,"ไอซี ดิจิตอล", ซีเอ็ดยูเคชั่น, 2531

[4] ตุ้ย ชุมสาย หม่อมหลวง ," กินดีเพื่ออยู่ดี",เยลโล่การพิมพ์,2536

[5] เปรมจิตต์ สิทธิศิริ และสุทิน เกตุแก้ว," กินอยู่เพื่อสุขภาพ เล่ม 2", เยลโล่การพิมพ์,2538

[6] พิชิต ล้ายอง ," เครื่องจักรกลไฟฟ้า 1", คณะวิศวกรรมศาสตร์,สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,2539

[7] สุนทร วิทูสุรพจน์ ," การโปรแกรมภาษาแอสแซมบลีของไมโครคอนโทรลเลอร์ ตระกูล 8051 ", ซีเอ็ดยูเคชั่น,2537