ปีการศึกษา 2533

การควบคุมความเร็วมอเตอร์เหนี่ยวนำ 3 ϕ โดยวิธีการมอดูเลตความกว้างพัลส์แบบสุ่มสม่ำเสมอ

(3 ϕ Induction motor Control by Regular Sampled PWM)

โดย

นาย ทนง รัตนพิทักษ์ 326107

นาย สรรเพชร ทองปาน 326131

อาจารย์ที่ปรึกษา

อาจารย์ วิทยา ทิพย์สุวรรณพร

ปริญญานิพนธ์ปีการศึกษา 2533

ภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การควบคุมความเร็วมอเตอร์เหนี่ยวนำ 3 ϕ โดยวิธีการมอดูเลตความกว้างพัลส์แบบสม่ำเสมอ

ผู้จัดทำ

1 นาย ทนง รัตนพิทักษ์ 326107

2 นาย สรรเพชร ทองปาน 326131

.....อาจารย์ที่ปรึกษา

(อาจารย์ วิทยา ทิพย์สุวรรณพร)

เลขหมู่ T 33120 ท3

เลขทะเบียน 027953

วัน, เดือน, ปี 18 ก.ค. 34

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสาร ห้ามมิให้นำไปใช้

027953

พัลซ์วิดท์มอดดูเลชั่น อินเวอร์เตอร์

นายสรรเพชร ทองปาน
นายทนง รัตนพิทักษ์

อาจารย์ที่ปรึกษา

อาจารย์ วิทยา ทิพย์สุวรรณพร
ปีการศึกษา 2533

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ จะกล่าวถึงการสร้างวงจรกำลังวงจรควบคุมสัญญาณพัลซ์ และวงจรขับทรานซิสเตอร์ที่สมบูรณ์ สำหรับขับมอเตอร์เหนี่ยวนำ 3 ϕ ขนาด 1 แรงม้า หลักของพีดับบลิวเอ็มอินเวอร์เตอร์ โดยการ ใช้ทรานซิสเตอร์ 6 ตัว ต่อแบบบริดจ์ และมีส่วนประกอบของ วงจรป้องกันแรงดันสูงเกิน ป้องกันแรงดันต่ำเกิน วงจรป้องกันกระแสสูงเกิน วงจรหน่วง และวงจรสนับเบอร์ ที่ช่วยลดความเค้นบนทรานซิสเตอร์ ขณะเริ่มนำและหยุดนำกระแส และวงจรขับเบสของทรานซิสเตอร์ โดยเราสามารถ ที่จะนำชิ้นงานนี้ไปใช้ร่วมกับวงจรสร้างสัญญาณ พีดับบลิวเอ็มเพื่อที่จะนำไปควบคุม ความเร็วของมอเตอร์เหนี่ยวนำ หรือ มอเตอร์กระแสตรงก็ได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

PLUSE WIDTH MODULATION INVERTER

Mr. Sanpheat Thongpran

Mr. Tanong Ratanapitag

Advisor

Mr. Vittaya Tipsuvanapon

1990

ABSTRACT

This thesis describes a designing of main-power circuit,control circuit and based-drived circuit for three phase induction motor of Pulse Width Modulation (PWM) inverter using which six bridged transistors. It also reports about those circuits used to protect overvoltage, under-voltage and over-current. While snubber circuit reduce stress on transistors when turning on and off. Besides, the transistor base-drived circuit are inciuded and described by this thesis. By the way, This invention can be used together with the PWM circuit for variable speed of induction motor or D.C motor.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 1
คำนำ

ในการใช้มอเตอร์เป็นต้นกำลังในโรงงานอุตสาหกรรมนั้น ความต้องการโดยทั่วไป คือ วิธีการควบคุมความเร็วของมอเตอร์ที่ประหยัด สามารถควบคุมความเร็วของมอเตอร์ได้ต่อเนื่องแม่นยำ เปลี่ยนความเร็วได้รวดเร็ว มีพิสัยความเร็วตามต้องการและมีเสถียรภาพที่ดีอยู่เป็นเวลานาน

ปัจจุบันในโรงงานอุตสาหกรรมมักจะ ใช้ต้นกำลังเป็นมอเตอร์กระแสตรงสำหรับงานที่ต้องการปรับความเร็ว และใช้มอเตอร์กระแสสลับแบบเหนี่ยวนำกับงานที่ใช้ความเร็วคงที่ เนื่องจากมอเตอร์กระแสตรง มีคุณสมบัติที่เราต้องการทุกอย่าง แต่มีสิ่งหนึ่งที่เป็นข้อเสีย คือ คอมมิวเตเตอร์(COMMUTATOR) กล่าวคือ คอมมิวเตเตอร์ประดิษฐ์ขึ้นจากชิ้นทองแดงหลายชิ้นคั่นด้วยไมก้า ทำให้การสร้างเต็มไปด้วยความลำบากและมีราคาสูง เนื่องจากต้องมีแปรงถ่าน(BRUSHES) ขัดสีอยู่กับคอมมิเตเตอร์เสมอ และประกายไฟที่เกิดขึ้นระหว่างแปรงถ่าน กับคอมมิวเตเตอร์ทำให้มีการสึกหรอมากยิ่งขึ้น จึงต้องการซ่อมบำรุงอยู่เสมอ ทำให้เกิดความยุ่งยากในอุตสาหกรรมที่ใช้มอเตอร์แบบนี้ที่มีขบวนการผลิตที่ไม่สามารถหยุดได้ นอกจากนี้ถ้าติดตั้งมอเตอร์ไว้ในตำแหน่งที่เข้าถึงได้ลำบาก ก็จะทำความยุ่งยากให้กับการซ่อมบำรุง ทำให้เสียค่าใช้จ่ายสูง

เนื่องจากได้มีการพัฒนาอินเวอร์เตอร์(INVERTER) โดยใช้อุปกรณ์สารกึ่งตัวนำ เช่น ทรานซิสเตอร์ ไธริสเตอร์ ขึ้นมาเพื่อนำมาใช้งานในการขับนำมอเตอร์กระแสสลับแบบเหนี่ยวนำให้สามารถปรับความเร็วได้ จึงทำให้การใช้มอเตอร์กระแสสลับแบบเหนี่ยวนำเป็นต้นกำลังในงานที่ต้องการปรับความเร็วเป็นที่น่าสนใจยิ่งขึ้น เนื่องจากมอเตอร์กระแสสลับแบบเหนี่ยวนำมีข้อดีหลายประการ เช่น ลักษณะโครงสร้างที่มีความแข็งแรงทนทานและมีลักษณะมิดชิด ซึ่งประกอบด้วยส่วนสำคัญสองส่วน คือ ส่วนอยู่นิ่ง(STATOR)ซึ่งเป็นตัวสร้างฟลักซ์แม่เหล็กหมุน(ROTATING MAGNATIC FLUX) และส่วนหมุน(ROTOR) ซึ่งอยู่ภายในจะเป็นตัวทำให้แรงบิด (TORQUE) ออกมา จากโครงสร้างที่ง่ายๆนี้ ทำให้มอเตอร์กระแสสลับแบบเหนี่ยวนำมีราคาถูกและมีขนาดเล็กกว่ามอเตอร์กระแสตรงเมื่อเปรียบเทียบที่ความเร็วและกำลังม้าเท่ากันสามารถทำงานได้เป็นเวลายาวนานโดยไม่ต้องการซ่อมบำรุง การบำรุงรักษา

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ง่ายทำให้เสียค่าใช้จ่ายน้อยกว่าและสามารถนำไปใช้งานในบริเวณที่มีฝุ่นละอองมาก บริเวณที่มีความชื้นสูง หรือบริเวณที่อาจมีการระเบิดเนื่องจากประกายไฟได้ ซึ่งข้อดีเหล่านี้ประกอบกับการที่มอเตอร์ชนิดนี้มีขนาดและมาตรฐานซึ่งหาซื้อได้ง่าย ดังนั้นจึงเหมาะที่จะนำไปใช้งานในโรงงานอุตสาหกรรมมากกว่ามอเตอร์กระแสตรง

<u>การควบคุมความเร็วของมอเตอร์กระแสสลับแบบเหนี่ยวนำ</u>

มอเตอร์กระแสสลับแบบเหนี่ยวนำที่นิยมใช้กันมากคือ มอเตอร์เหนี่ยวนำแบบกรงกระรอก(SQUIRREL CAGE INDUCTION MOTOR) การควบคุมความเร็วของมอเตอร์กระแสสลับแบบเหนี่ยวนำนี้ สามารถทำได้หลายวิธี เช่น

1)การควบคุมความเร็วโดยการปรับแรงดันไฟฟ้า ที่จ่ายให้กับมอเตอร์ โดยความถี่ของแหล่งจ่ายยังคงเดิม การปรับแรงดันที่จ่ายให้กับมอเตอร์ในลักษณะนี้ จะทำได้ในช่วงแคบๆเท่านั้น เนื่องจากแรงบิดของมอเตอร์ จะแปรผันไปตามแรงดันของไฟฟ้าที่จ่ายให้กับมอเตอร์ ยกกำลังสอง เมื่อลดแรงดันไฟฟ้าที่จ่ายให้กับมอเตอร์ ขณะที่มอเตอร์รับภาระอยู่ที่โหลดใดๆ ก็ตามจะทำให้ความเร็วของมอเตอร์ลดลง ในขณะเดียวกันการลดแรงดันไฟฟ้าที่จ่ายให้มอเตอร์เพียงเล็กน้อย อาจทำให้แรงบิดน้อยลงมากจนอาจใช้งานไม่ได้ และถ้าเพิ่มแรงดันมากไปจะทำให้มอเตอร์เสียหายได้

2)การควบคุมความเร็ว โดยการเปลี่ยนจำนวน ขั้วแม่เหล็กของมอเตอร์ ความเร็วของมอเตอร์กระแสสลับแบบเหนี่ยวนำ จะแปรตามความถี่ของ แหล่งจ่ายไฟและแปรผกผันกับจำนวนขั้วของมอเตอร์ โดยการออกแบบให้ขดลวดของส่วนอยู่นิ่ง(STATOR) สามารถเปลี่ยนจำนวนขั้วแม่เหล็กได้ เช่น เปลี่ยนจำนวนขั้วแม่เหล็กจาก 2 เป็น 4 หรือจาก 4 เป็น 8 เป็นต้น ซึ่งแต่ละแบบของการต่อขดลวดจะให้ความเร็วได้สองความเร็ว อย่างไรก็ตามการเปลี่ยนขั้วแม่เหล็กของมอเตอร์ จะได้ความเร็วคงที่เป็นช่วงๆ ซึ่งไม่เหมาะสมกับลักษณะงานบางอย่าง ที่ต้องการปรับความเร็วได้ ในช่วงกว้างและต่อเนื่อง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3)การควบคุมความเร็วโดยการปรับค่าแรงดันไฟฟ้าและความถี่ของแหล่งจ่ายไฟที่จ่ายให้กับมอเตอร์ การควบคุมความเร็วด้วยวิธีนี้ จะต้องเปลี่ยนแรงดันไฟฟ้าและความถี่ที่คงที่จากแหล่งจ่ายไฟ ที่มีอยู่ให้เป็นค่าแรงดันไฟฟ้าและความถี่ที่ปรับค่าได้ และเพื่อให้ได้ประสิทธิภาพที่ดี ในการควบคุมความเร็ว จะต้องให้อัตราส่วนของแรงดันต่อความถี่(V/F RATIO) ที่จ่ายให้กับมอเตอร์มีค่าคงที่ หรือ ปรับแต่งเล็กน้อยตามความเหมาะสม จะทำให้มอเตอร์ขับโหลดได้เต็มที่ และค่าแรงบิดที่ได้จะคงที่ตลอดช่วงความเร็วที่ปรับค่า

<u>วัตถุประสงค์ของโครงงาน</u>

1)ออกแบบและสร้างเครื่องขับนำ มอเตอร์ไฟฟ้ากระแสสลับแบบเหนี่ยวนำสามเฟสขนาด 1 แรงม้า

2)ขณะปรับความเร็วของมอเตอร์ สามารถรักษาค่าแรงดันต่อความถี่ (V/F ratio) ให้คงที่ได้ในย่าน 0-50H_z

3)สามารถเริ่มเดินเครื่องแบบ SOFT START ได้

4)เพื่อควบคุมอัตราเร่ง(Acceleration)และอัตราหน่วง(Deceleration)ของมอเตอร์

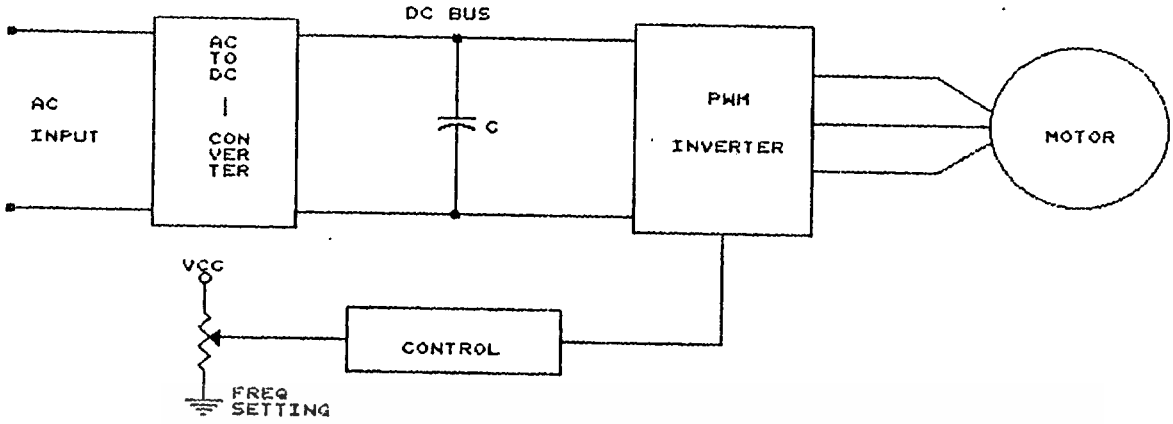
5)ศึกษาการทำงานของไมโครคอนโทรลเลอร์ สำหรับนำไปประยุกต์ใช้ในการควบคุมเครื่องขับนำมอเตอร์

6)ศึกษาการทำงานของชุด base drive power transistor

<u>ขอบเขตของโครงงาน</u>

สร้างเครื่องขับนำมอเตอร์กระแสสลับแบบเหนี่ยวนำสามเฟส ดวัยวิธีการมอดูเลตตามความกว้างพลัส์ (Pule Width Modolation) สามารถปรับความถี่ตั้งแต่ 0-60H_z โดยปรับความถี่ได้ครั้งละ 1H_z โดยมี V/F คงที่ โดยสามารถใช้กับมอเตอร์ขนาด 1 HP ซึ่งประกอบด้วยส่วนสำคัญตามรูปที่ 1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 1 บล็อกไดอะแกรมของเครื่องขับนำมอเตอร์แบบเหนี่ยวนำ

AC TO DC Converter เป็นวงจรเรียงกระแสเต็มคลื่นหนึ่งเฟสหรือสามเฟส โดยใช้ไดโอดซึ่ง จะทำหน้าที่เปลี่ยนแหล่งจ่าย ไฟฟ้ากระแสสลับหนึ่งเฟสหรือหรือสามเฟส ให้เป็นไฟฟ้ากระแสตรง ที่มีระดับแรงดันไฟฟ้าคงที่เพื่อจ่ายให้วงจรอินเวอร์เตอร์โดยที่ตัวเก็บประจุเป็นวงจรกรองคลื่น

PWM Inverter ทำหน้าที่จ่ายพลังงานไฟฟ้ากระแสสลับให้กับมอเตอร์ โดยได้ทำการเปลี่ยนพลังงานไฟฟ้ากระแสตรง ให้เป็นไฟฟ้ากระแสสลับที่สามารถปรับความถี่และแรงดันไฟฟ้าได้ เป็นอินเวอร์เตอร์แบบมอดูเลตความกว้างพัลส์ที่ใช้เพาเวอร์ทรานซิสเตอร์เป็นสวิทช์

Control เป็นวงจรทำหน้าที่สร้างสัญญาณในการควบคุมการสวิทช์ของอินเวอร์เตอร์ เพื่อให้ได้แรงดันไฟฟ้าและความถี่ ตามความต้องการในการขับนำมอเตอร์ โดยใช้ หลักการมอดูเลตความกว้างพัลส์ และรับสัญญาควบคุมการเลือกความถี่จากภายนอกด้วย ซึ่งวงจรส่วนนี้ใช้ไมโครคอนโทรลเลอร์ 8031 เป็นตัวควบคุม

<u>วิธีการดำเนินงาน</u>

ศึกษารายละเอียดของมอเตอร์เหนี่ยวนำสามเฟส โดยเฉพาะคุณลักษณะ (Charecteristic) การเริ่มเดิน (Starting) วิธีควบคุมพารามิเตอร์ต่างๆ เช่น กระแสขณะสตาร์ท เวลาที่ใช้เริ่มเดิน กำลังงานสูญเสีย ตลอดจนประสิทธิภาพของมอเตอร์

ศึกษาวิธีการควบคุมมอเตอร์ในที่นี้จะใช้วิธี pulse width modulation ซึ่งเป็นการควบคุมการมอดูเลตตามความกว้างพัลส์ เพื่อปรับความถี่และแรงดันให้สัมพันธ์กัน (V/F คงที่) และยังสามารถทำให้รูปคลื่นใกล้เคียงรูปซายน์มากที่สุด

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 2
ทฤษฎีและหลักการ

มอเตอร์เหนี่ยวนำ

มอเตอร์เหนี่ยวนำประด้วยส่วนสำคัญสองส่วน คือ ส่วนอยู่นิ่ง(Stator) และส่วนหมุน(Rotor) ส่วนหมุนจะทำหน้าที่รับพลังงานจากแหล่งจ่ายไฟฟ้ากระแสสลับผ่านส่วนอยู่นิ่งโดยอาศัยหลักการเหนี่ยนำ ซึ่งขดลวดของส่วนอยู่นิ่งทำหน้าที่สร้างสนามแม่เหล็กที่มีค่าคงที่ขึ้นในช่องอากาศ(Airgep) ระหว่างส่วนอยู่นิ่งกับส่วนหมุนและหมุนด้วยความเร็วซิ่งโครนัส(Synchronous)

$$N_s = 120\ f/p \quad \text{รอบต่อนาที} \qquad \text{---------- (1)}$$

เมื่อ N_s = ความเร็วซิงโครนัสของสนามแม่เหล็ก
f = ความถี่ของแหล่งจ่ายไฟฟ้ากระแสสลับ
p = จำนวนขั้วแม่เหล็กบนสเตเตอร์

เมื่อมอเตอร์หมุน กระแสในส่วนหมุน(I_2) จะมีเฟสตามหลังแรงเคลื่อนไฟฟ้าในส่วนหมุนด้วยเฟส θ_2 ซึ่งมอเตอร์นี้มีแรงหมุนเป็นสัดส่วนกับกระแสองค์ประกอบของส่วนที่มีเฟสทับกันในส่วนหมุนคือ $I_2.\cos.\theta_2$ และแรงหมุนนี้ยังมีค่าเป็นสัดส่วนกับฟลั๊กซ์ในช่องอากาศ ดังนั้น โดยทั่วๆไป เราจะเขียนสมการของแรงบิดได้ว่า

$$T = K_T.\Phi.I_2.\cos\theta_2 \qquad \text{---------- (2)}$$

เมื่อ K_T = ค่าคงที่
Φ = ค่าฟลั๊กสูงสุดต่อขั้วแม่เหล็กของมอเตอร์
I_2 = ค่ากระแสในส่วนหมุน
θ_2 = มุมระหว่างแรงดันและกระแสในส่วนหมุน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เมื่อจ่ายแรงดันไฟฟ้ากระแสสลับรูปซาย์ที่มีขนาดเท่ากับ $V_m \sin wt$ เข้าที่ขดลวดของส่วนอยู่นิ่ง จะทำให้เกิดฟลั๊กซ์แม่เหล็กหมุนมีขนาดตามสมการ(ไม่คิดความต้านทานของขดลวด)

$$V_m \sin wt = -K.d\phi/dwt \quad \text{---------- (3)}$$

จะได้

$$d\phi/dwt = -V/_K \sin wt \quad \text{---------- (4)}$$

$$d\phi = -V/_K \sin wt.dwt \quad \text{---------- (5)}$$

$$\phi = K_1.v/_f.\cos wt \quad \text{---------- (6)}$$

นั้นคือ ขนาดของฟลั๊กซ์แม่เหล็กหมุนในช่องอากาศจะเท่ากับ

$$\phi = K_2.V/_F \quad (7)$$

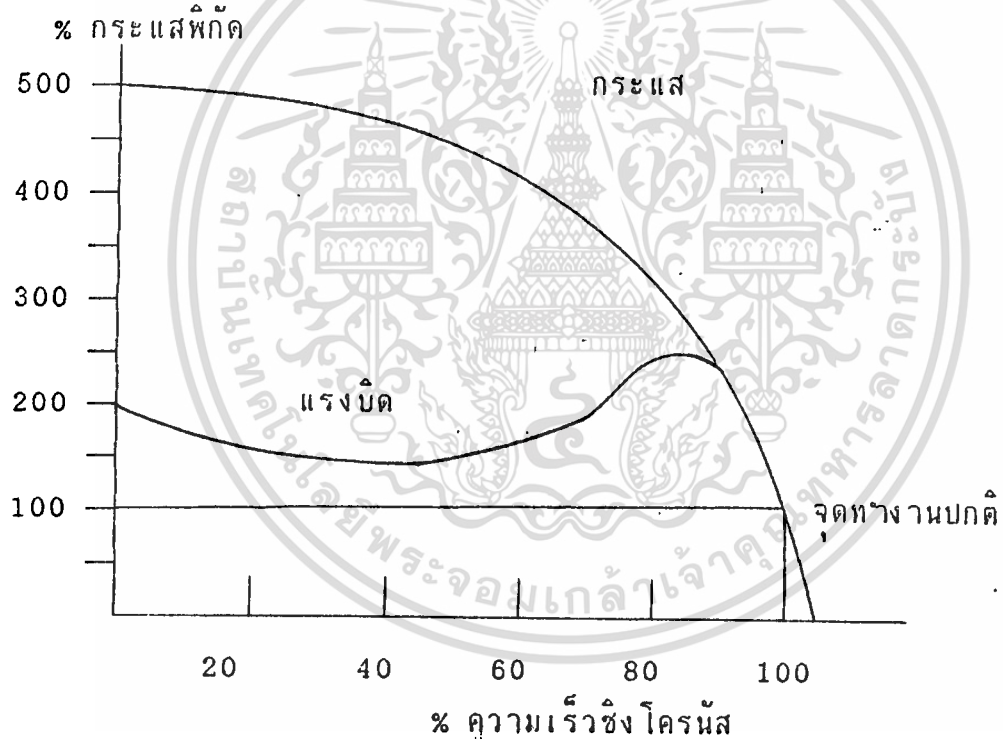
สมการนี้แสดงว่า ฟลั๊กซ์แม่เหล็กจะเป็นสัดส่วนกับแรงดันต่อความถี่ และเพื่อรักษาให้ฟลั๊กซ์คงที่ เราต้องปรับค่าแรงดันให้เป็นโดยตรง กับความถี่หรือรักษาอัตราส่วนแรงดันต่อความถี่ให้คงที่ วิธีการแบบนี้เรียกว่า "วิธีโวลท์ต่อเฮิรซ์คงที่" จะทำให้มอเตอร์เหนี่ยวนำสามารถทำงานที่ แรงบิดที่เหมาะสมที่สุด

ในกรณีที่ความต้านทานของ ขดลวดของมอเตอร์มีค่าน้อยมากสามารถตัดทิ้งได้ แรงเคลื่อนตกคร่อมความต้านทาน ของขดลวดจะมีค่าเท่ากับศูนย์โดยประมาณ แรงดันกระแสสลับที่ป้อนเข้ามอเตอร์ จะมีค่าเท่ากับแรงดันไฟฟ้าที่เกิดขึ้นในมอเตอร์โดยการเหนี่ยวนำ แต่ในกรณีความต้านทานของขดลวดมีค่ามากก็จะไม่สามารถตัดทิ้งแรงดันตกคร่อมความต้านทานนี้ได้ และถ้าความถี่มีค่าลดต่ำลง แรงดันที่ตกคร่อมความต้านทานมีค่ามากขึ้นเมื่อ เปรียบเทียบกับขนาดของแรงเคลื่อนไฟฟ้าเหนี่ยวนำที่เกิดขึ้นซึ่งเป็นสาเหตุที่ทำให้ฟลักซ์ในช่องอากาศมีค่าลดลงและเป็นผลทำให้มอเตอร์มีแรงบิดลดลงด้วย ดังนั้นในทางปฏิบัติสำหรับความถี่ที่ต่ำกว่า 20 เฮิรซ์ แล้วเราต้องรักษาแรงบิดที่ความเร็วต่ำไว้ให้ดีด้วยการเพิ่มค่า"โวลท์ต่อเฮิรตซ์" ให้มีค่าสูงขึ้นที่ความถี่ต่ำๆ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ลักษณะคุณสมบัติของมอเตอร์เหนี่ยวนำ

ถ้าเราเริ่มสตาร์ตมอเตอร์ โดยการป้อนกระแสไฟฟ้าตามอัตราปกติเข้ามอเตอร์เหนี่ยวนำแบบกรงกระรอกขณะอยู่นิ่ง โดยทั่วๆไปแล้วจะมีกระแสไหลเข้ามอเตอร์ประมาณ 5 หรือ 6 เท่าของกระแสพิกัด และจะทำให้เกิดแรงบิดขณะเริ่มหมุนตามปกติประมาณ 1.5 ถึง 2 เท่าของแรงบิดพิกัดตามรูปที่ 2 แสดงลักษณะคุณสมบัติระหว่างแรงบิด ความเร็วและกระแสโดยปกติของมอเตอร์เหนี่ยวนำซึ่งทำงานที่แรงดันไฟฟ้าและความถี่คงที่

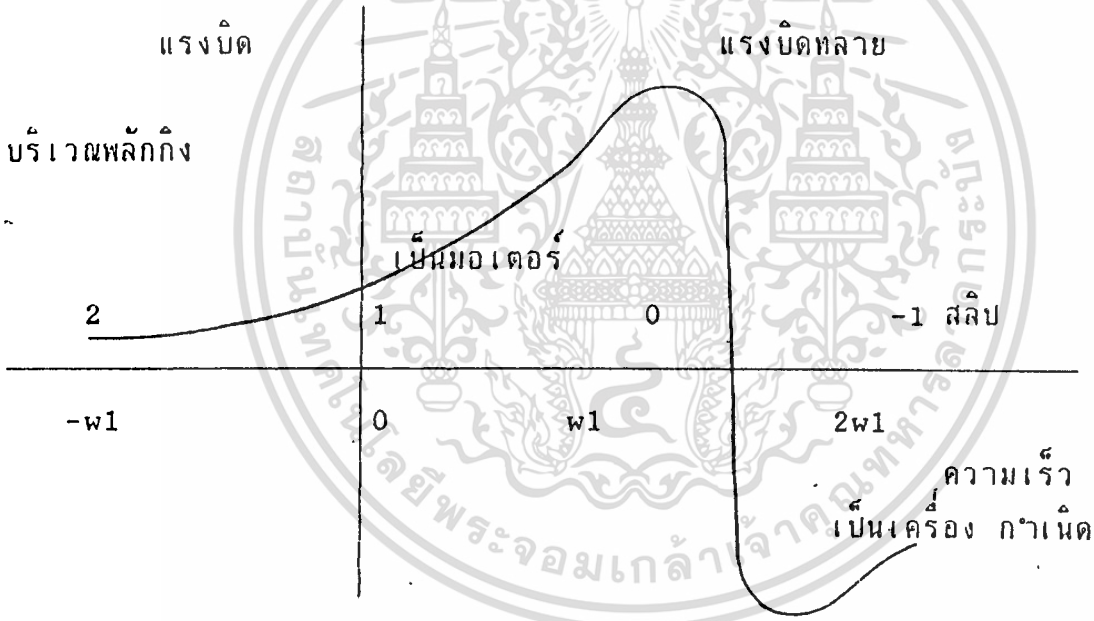


รูปที่ 2 แสดงลักษณะคุณสมบัติความเร็ว แรงบิดและกระแสของมอเตอร์

รูปที่ 2 แสดงเส้นโค้งลักษณะคุณสมบัติระหว่างแรงบิดกับความเร็วของมอเตอร์เหนี่ยวนำ ขณะกำลังทำงานด้วยแรงดันคงที่จากแหล่งจ่ายไฟฟ้ากระแสสลับที่มีความถี่คงที่ ตรงจุดที่ให้แรงบิดสูงสุด เรียกว่า "แรงบิดทลาย" (Breakdown Torque) และค่าความถี่ของส่วนหมุนตรงจุดทลาย เรียกว่า "ความถี่ทลายของการหมุน" ถ้าแรงบิดมีค่ามากเกินกว่าค่าแรงบิดสูงสุดจะทำให้มอเตอร์มีความเร็วลดค่าต่ำลงจนกระทั่งหยุดนิ่ง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ถ้ามอเตอร์หมุนกลับในทิศที่สวนทางกับการหมุนในทิศฟอร์เวิร์ดของสนามแม่เหล็ก สลิปจะมีค่ามากกว่าหนึ่ง และแรงหมุนโต้กลับจะเกิดขึ้นต่อต้านกับการหมุนของมอเตอร์ ปรากฏการณ์นี้จะเกิดขึ้นเมื่อ เรากลับลำดับของเฟส(Phase Sequence) ของสายไฟฟ้ากระแสสลับสามเฟสเป็นเหตุให้สนามแม่เหล็กในช่องอากาศหมุนกลับทิศทาง เป็นผลให้มอเตอร์หยุดหมุนอย่างรวดเร็วและถ้ายังคงปล่อยสายไฟฟ้าไว้ในสภาวะเช่นนี้ ในไม่ช้ามอเตอร์จะเริ่มหมุนกลับทางต่อไปด้วยความเร็วเพิ่มสูงขึ้น วิธีการนี้เป็นวิธีการเบรคหรือวิธีการกลับทางหมุนมอเตอร์อย่างรวดเร็วซึ่งเราเรียกว่า"พลักกิ่ง(Plugging) และบริเวณที่เส้นโค้ง ลักษณะนี้เรียกว่า "อาณาบริเวณพลักกิง"



รูปที่ 3 แสดงเส้นโค้งลักษณะคุณสมบัติระหว่างแรงบิด
กับความเร็วของมอเตอร์เหนี่ยวนำ

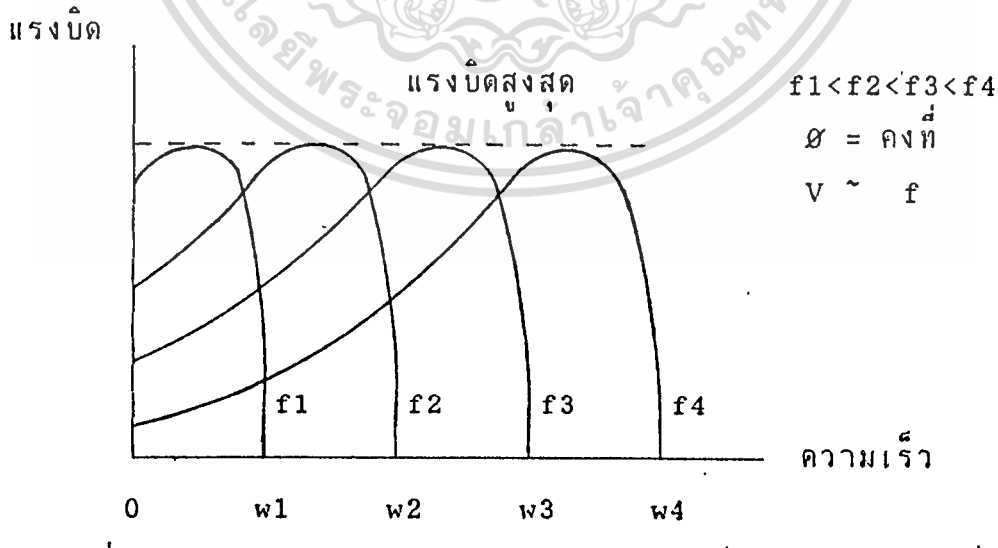
ขณะที่มอเตอร์กำลังหมุนในทิศฟอร์เวิร์ดตามปกติ ถ้าเราใช้วิธีทางกลเข้าช่วยขับเคลื่อนให้ส่วนหมุนของมอเตอร์หมุนเร็วขึ้นกว่าอัตราเร็วปกติ เข้าสู่ความเร็วซูเปอร์ซิงโครนัส(Super Synchronous Speed) สลิปมีค่าเป็นลบแล้วมอเตอร์นี้จะเปลี่ยนหน้าที่กลับเป็นเครื่องกำเนิดไฟฟ้า โดยเปลี่ยนพลังงานป้อนเข้าที่เพลาให้กลายเป็นพลังงานไฟฟ้า ส่งกลับเข้าสู่แหล่งจ่ายไฟฟ้ากระแสสลับ ดังนั้นเราอาจต่อเส้นโค้งออกไป ในทิศทางที่มีความเร็วสูงกว่าความเร็วซิงโครนัส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้
027953

เข้าสู่อาณาบริเวณแห่งการปฏิบัติการ เป็นเครื่องกำเนิดไฟฟ้าถ้าหากสามารถเปลี่ยนความถี่ของแหล่งจ่ายไฟฟ้าได้ เราอาจปรับมอเตอร์ให้เข้าสู่อาณาบริเวณแห่งการปฏิบัติการเป็นเครื่องกำเนิดไฟฟ้า ได้อีกวิธีหนึ่งด้วยการลดความถี่ลงอย่างรวดเร็ว โดยมอเตอร์จะถูกฉุดให้หมุนต่อไปด้วยความเฉื่อย พลังงานจลย์ของมวลที่อยู่ในส่วนหมุน(รวมถึงโหลด) จะเปลี่ยนกลับเป็นพลังงานไฟฟ้าป้อนกลับเข้าแหล่งจ่ายไฟฟ้า ทำให้เกิดเป็นแรงหมุน โต้กลับขึ้นในมอเตอร์แล้วมอเตอร์จะหมุนช้าลงหรือหยุด ซึ่งเป็นการหยุดแบบรีเจนเนอเรตีฟเบรกกิง(Regenerative Breaking) แต่ถ้าทำให้พลังงานไฟฟ้าป้อนกลับมาสูญเสียไปในความต้านทาน แทนที่จะป้อนกลับเข้าแหล่งจ่ายไฟฟ้าก็จะเรียกว่า การหยุดแบบพลวัต(Dynamic Breaking)

<u>ลักษณะคุณสมบัติของเครื่องควบคุมความเร็วโดยการปรับค่าแรงดันไฟฟ้าและความถี่ของแหล่งจ่ายไฟ</u>

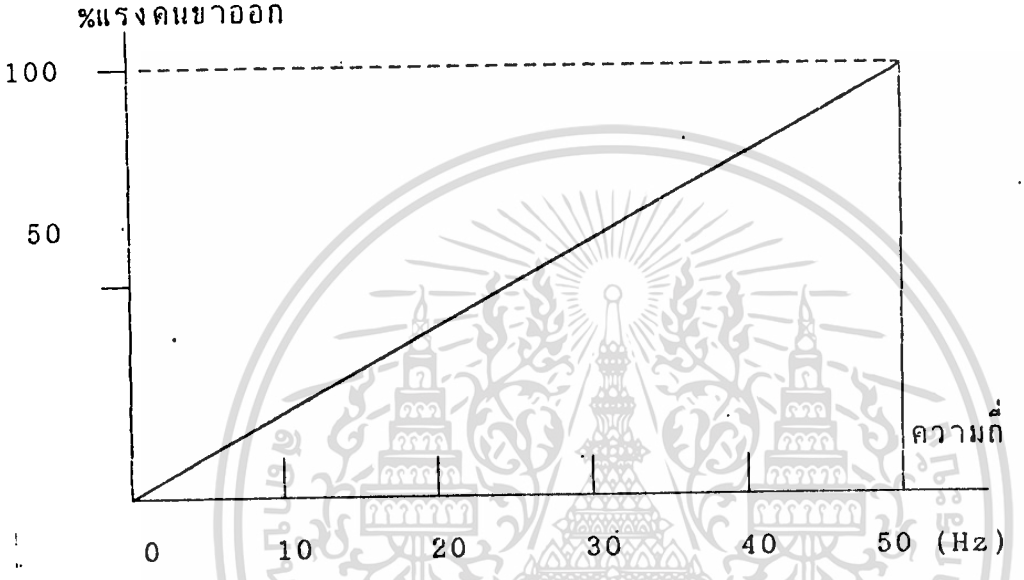
เพื่อให้กำเนิดแรงบิดที่คงที่ เครื่องควบคุมความเร็วจะต้องทำให้ค่าของฟลั๊กซ์ในอากาศคงที่ ตามรูปที่ 4 แสดงลักษณะคุณสมบัติของแรงบิดของมอเตอร์ที่ความถี่ต่างๆ โดยรักษาแรงบิดพลายให้คงที่ด้วยปรับค่า "โวลท์ต่อเฮิรตซ์" ให้คงที่ ลักษณะคุณสมบัติเช่นนี้ทำให้มอเตอร์เหนี่ยวนำเหมาะสมกับงานขับเคลื่อนโหลดที่มีแรงบิดคงที่ให้มีอัตราเร็วต่างๆ



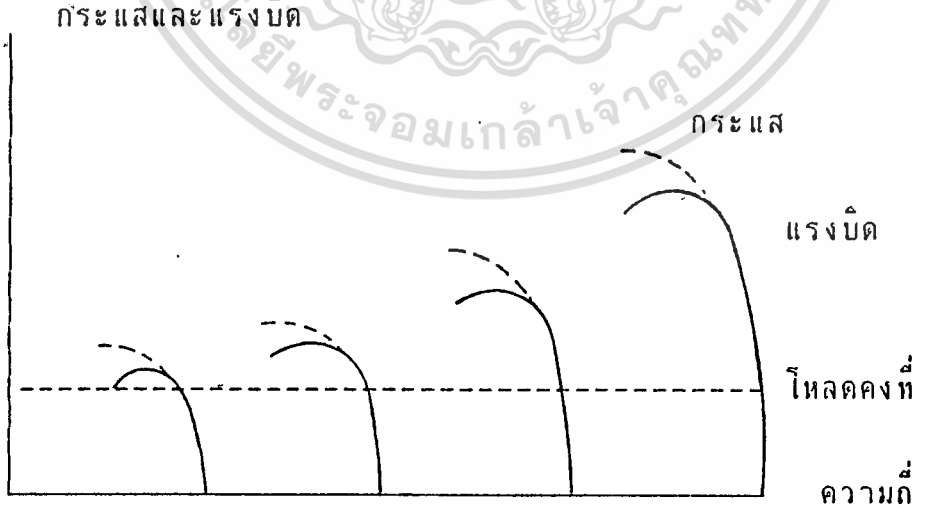
รูปที่ 4 ลักษณะสมบัติระหว่างแรงบิดกับความเร็วของมอเตอร์เหนี่ยวนำที่ความถี่ต่างๆของแหล่งจ่ายไฟฟ้าเมื่อฟลั๊กซ ในช่องอากาศมีค่าคงที่

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในทางปฏิบัติเมื่อเราทำให้เครื่องควบคุมมีอัตราส่วนของ "โวลท์ต่อเฮิรตซ์"คงที่ดังรูปที่ 5 เมื่อความถี่เปลี่ยนไป มอเตอร์จะมีลักษณะของความเร็ว แรงบิด และกระแสไฟฟ้าสำหรับแต่ละความถี่แสดงในรูปที่ 6



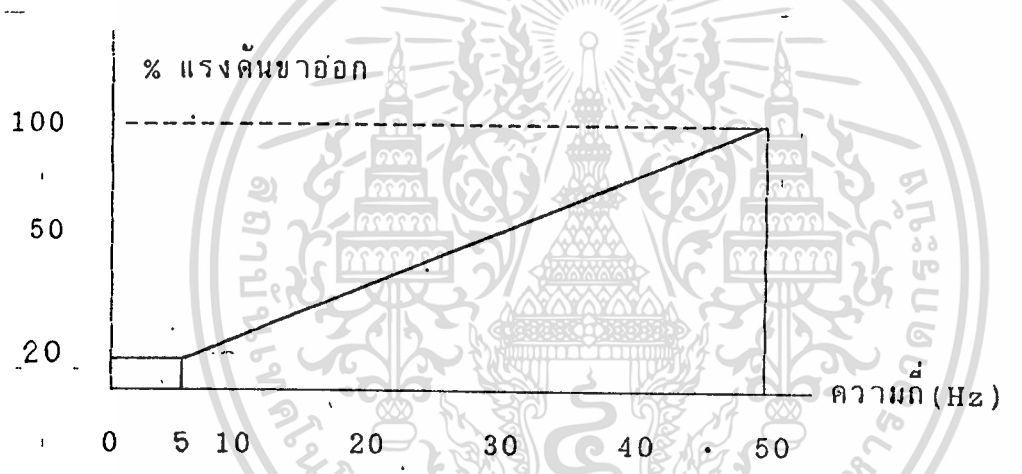
รูปที่ 5 ลักษณะของ "โวลท์ต่อเฮิรตซ์"



รูปที่ 6 แรงบิดและกระแสกับความถี่ที่โหลดคงที่และไม่มีแรงดันออฟเซ็ท

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เมื่อความถี่ลดลงค่าแรงบิดสูงสุดจะลดลง สาเหตุเกิดจากแรงดันตกคร่อมที่ความต้านทานของขดลวดบนตัวอยู่กับที่(Stator) จะมีขนาดมากขึ้น เมื่อเปรียบเทียบกับแรงดันขาออกที่ความถี่ต่ำ มีผลไปทำให้ฟลั๊กซ์ในช่องอากาศมีค่าลดลง และแรงบิดจะลดลงด้วย การลดลงของแรงบิดสูงสุดนี้เราสามารถเอาชนะได้โดยการเพิ่มแรงดันไฟฟ้าที่ความถี่ต่ำขึ้นเรียกว่า"แรงดันออกเซ็ท" (Offset voltage) ค่าแรงดันไฟฟ้านี้จะไปชดเชยแรงดันตกคร่อม ในขดลวดของส่วนอยู่กับที่จะได้ลักษณะของ"โวลต์ต่อเฮิรตซ์"ดังรูปที่ 7 เมื่อปรับค่าของแรงดันออกเซ็ทแล้วมอเตอร์จะกำเนิดแรงบิดคงที่ตลอดช่วงความเร็ว



รูปที่ 7 "โวลต์เฮิรตซ์"เมื่อมีการชดเชยแรงดัน

<u>ส่วนประกอบของเครื่องควบคุมความเร็วแบบปรับความถี่</u>

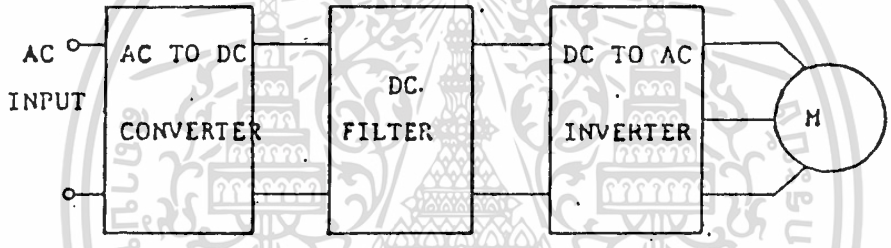
เครื่องควบคุมความเร็วแบบปรับความถี่ จะประกอบไปด้วยคอนเวอร์เตอร์ วงจรกรอง และอินเวอร์เตอร์ ตามรูปที่ 8 เป็นบล๊อกไดอะแกรมของภาคจ่ายกำลังของเครื่องควบคุมความเร็ว

-คอนเวอร์เตอร์ (Converter) เป็นวงจรเปลี่ยนแปลงกำลังไฟฟ้าให้เป็นรูปแบบต่างๆ ซึ่งในที่นี้คอนเวอร์เตอร์จะหมายถึง เครื่องเรียงกระแส (Rectifier) ซึ่งเปลี่ยนกำลังไฟฟ้ากระแสสลับให้เป็นกำลังไฟฟ้ากระแสตรง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

-วงจรกรอง (DC Filter) ทำหน้าที่กรองกระแสและแรงดันที่ออกจากวงจรเรียงกระแสก่อนที่จะนำไปใช้ เนื่องจากกระแสและแรงดันที่ได้จากวงจรเรียงกระแสจะยังมีส่วนประกอบกระแสสลับ หรือระลอกคลื่นปนอยู่เป็นปริมาณสูงไม่เหมาะที่จะนำไปใช้เราจะนำมาผ่านวงจรกรอง เพื่อให้เหลือแต่ส่วนประกอบของไฟตรง

-อินเวอร์เตอร์ (Inverter) เป็นเครื่องเปลี่ยนกำลังไฟฟ้ากระแสตรงให้เป็นกำลังไฟฟ้ากระแสสลับ



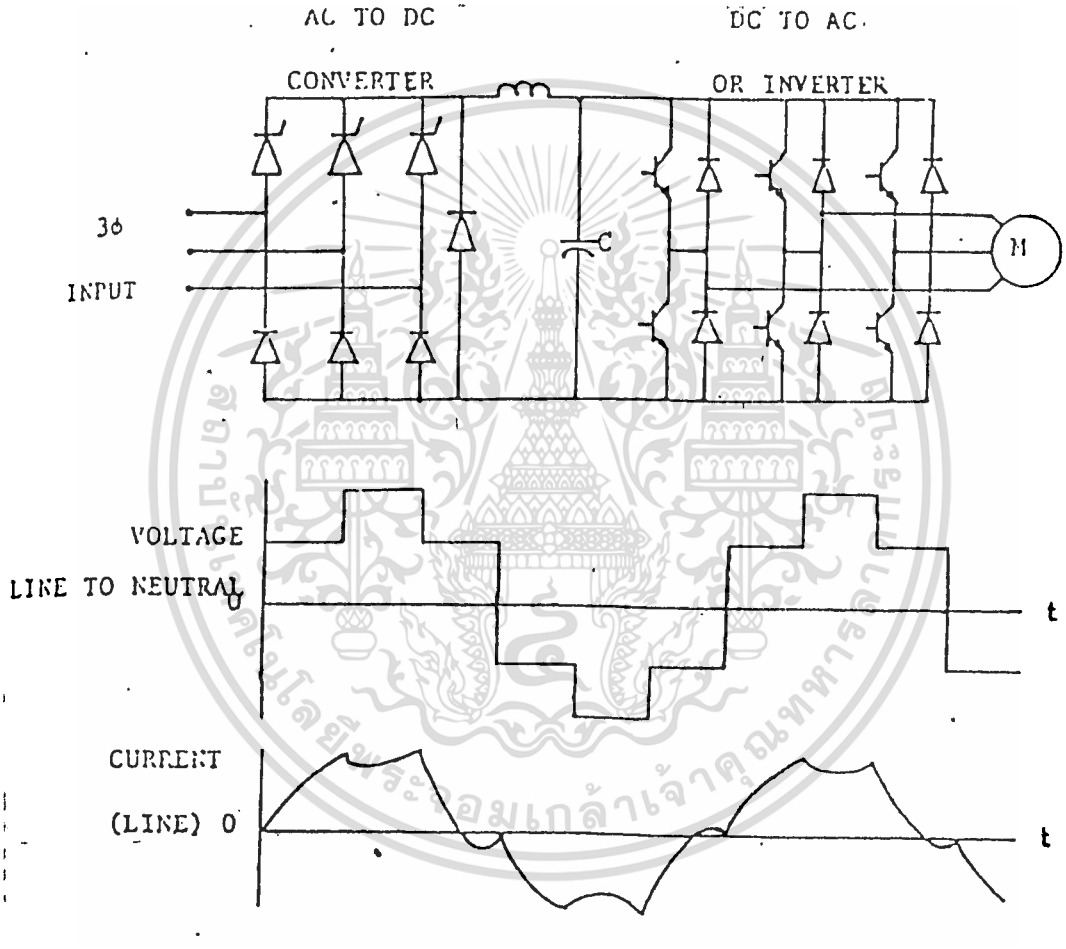
รูปที่ 8 บล๊อกไดอะแกรมภาคจ่ายกำลังของเครื่องควบคุมแบบปรับความถี่

<u>เครื่องควบคุมความเร็วแบบปรับความถี่โดยทั่วไปจะมีอยู่ 3 แบบ</u>

1) เครื่องควบคุมความเร็วแบบแหล่งพลังงานเป็นแรงดันปรับค่าได้ (Variable-Voltage Input Controller) เครื่องควบคุมความเร็วแบบนี้จะใช้อินเวอร์เตอร์ที่ใช้แหล่งพลังงานเป็นแหล่งแรงดัน โดยใช้วงจรเรียงกระแสที่มีการควบคุมเป็นคอนเวอร์เตอร์ วิธีการนี้จะใช้ ไทริสเตอร์เป็นคอนเวอร์เตอร์ ที่ทำหน้าที่เรียงกระแสซึ่งมีการควบคุมการจุดชนวนดวัยวิธีควบคุมเฟส เป็นผลให้แรงดันกระแสตรงที่เปลี่ยนค่าได้ เพื่อให้อัตราส่วนของโวลต์ต่อเฮิรตย์ตามต้องการ แรงดันกระแสตรงนี้จะถูกกรองโดยวงจรกรองซึ่งประกอบไปด้วยตัวเก็บประจุและตัวเหนี่ยวนำซึ่งมีขนาดใหญ่ เพื่อทำให้แรงดันกระแสตรง ที่ได้มีค่าเรียบก่อนป้อนเข้ากับวงจรอินเวอร์เตอร์ ความถี่ของแรงดันขาออกจะถูกกำหนดโดย การสวิทช์ของทรานซิส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เตอร์ หรือไทริสเตอร์ ในส่วนของวงจรอินเวอร์เตอร์ในรูปที่ 9 แสดงวงจรเครื่องควบคุมชนิดนี้ พร้อมทั้งรูปคลื่นของแรงดันและกระแสขาออก ซึ่งเป็นรูปขั้นบันไดหกขั้น

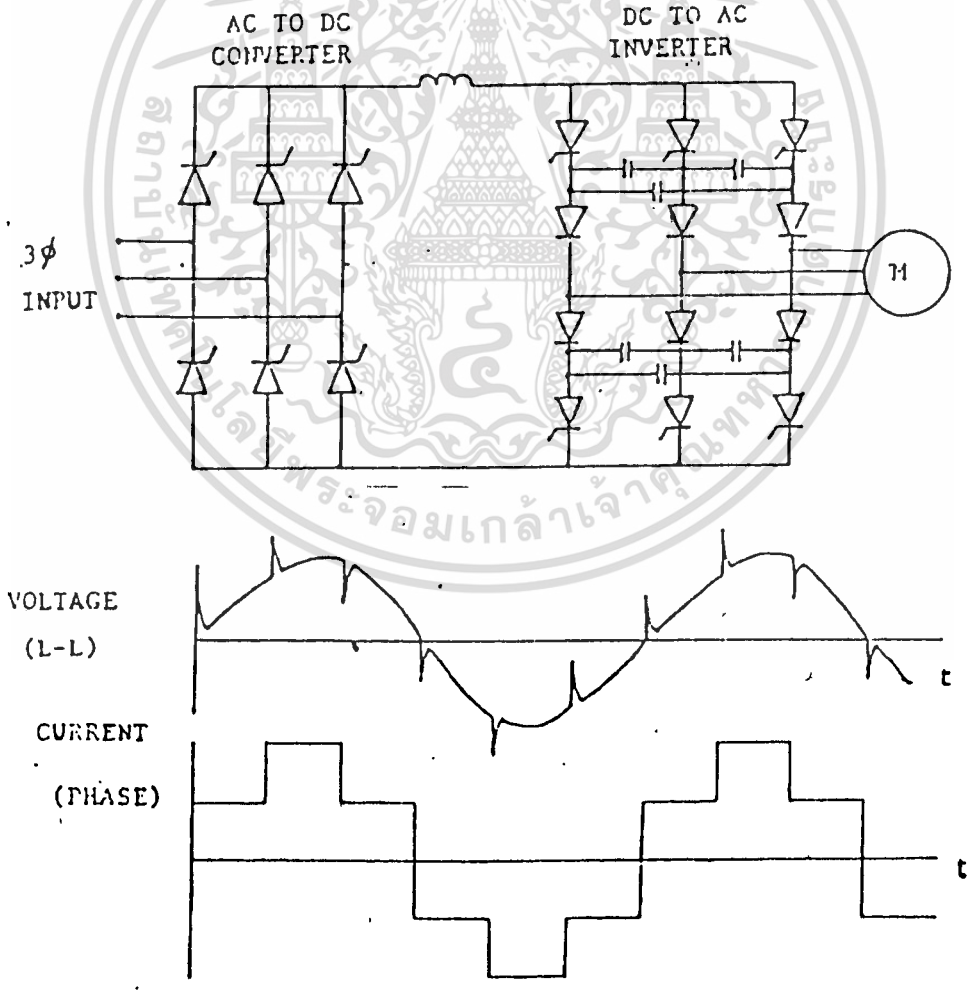


รูปที่ 9 เครื่องควบคุมความเร็วแบบแหล่งพลังงานเป็นแหล่งแรงดันปรับค่าได้

เครื่องควบคุมแบบนี้จะมีแรงดันฮาร์มอนิคส์เกิดขึ้น และผลลัพธ์กระแสฮาร์มอนิคส จะขึ้นอยู่กับอิมพิแแดนซ์ ของโหลดที่ความถี่ของฮาร์มอนิคส์ กระแสของฮาร์มอนิคส์นี้ จะถูกกำจัดโดยค่ารีแอกแตนซ์รั่ว (Leakaeg Reactance) ของมอเตอร์เหนี่ยวนำ สำหรับมอเตอร์เหนี่ยวนำชนิดที่มีค่ารีแอกแตนซ์รั่วสูงจะเกิดกระแสฮาร์มอนิคส์ต่ำทำให้เกิดการสูญเสีย เนื่องจากฮาร์มอนิคส์น้อย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2) เครื่องควบคุมความเร็วแบบแหล่งพลังงานเป็นแหล่งกระแส(Current Souse Input Control) เครื่องควบคุมความเร็ว แบบนี้จะใช้อินเวอร์เตอร์ชนิดที่ใช้แหล่งพลังงานเป็นกระแส และใช้วงจรเรียงกระแสที่มีการควบคุมเป็นคอนเวอร์เตอร์โดยการใช้ไทริสเตอร์ จำนวน 6 ตัว เป็นคอนเวอร์เตอร์ทำหน้าที่เรียงกระแส ซึ่งเป็นการควบคุมการจุดชนวนด้วยวิธีการควบคุมเฟส ทำให้ได้แรงดันกระแสตรงที่ต้องการ หลังจากนั้นจะผ่านวงจรกรองซึ่งมีตัวเหนี่ยวนำขนาดใหญ่ เพื่อให้ได้กระแสคงที่และทำหน้าที่เสมือนเป็นแหล่งจ่ายกระแส วงจรอินเวอร์เตอร์จะทำหน้าที่สวิทช์ ทำให้เกิดกระแสไฟสลับที่แปรความถี่ได้ และแรงดันของมอเตอร์จะเป็นไปตามกระแส ในรูปที่ 10 แสดงวงจรของเครื่องควบคุมชนิดนี้ พร้อมทั้งรูปคลื่นของแรงดันและกระแสขาออก ซึ่งเป็นขั้นบันไดหกขั้น

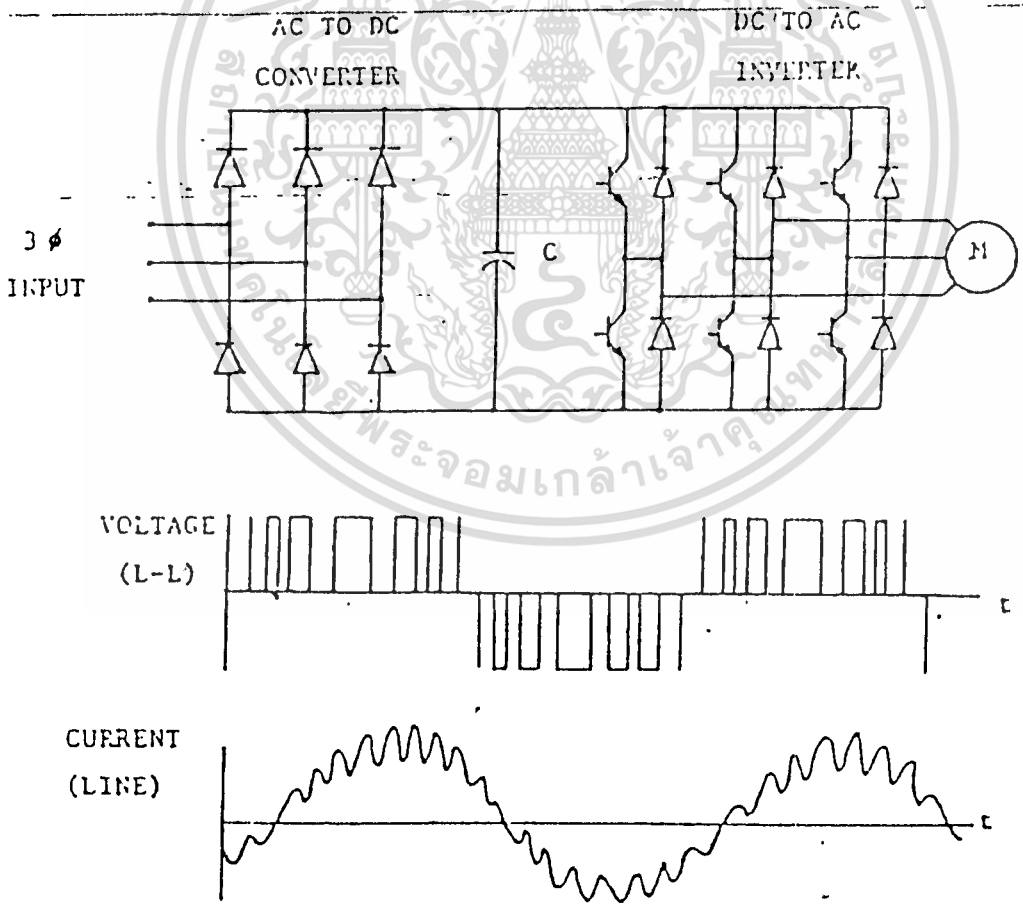


รูปที่ 10 เครื่องควบคุมความเร็วแบบแหล่งพลังงานเป็นแหล่งกระแส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เครื่องควบคุมแบบนี้ จะทำให้เกิดกระแสฮาร์มอนิคส์ ซึ่งทำให้มีแรงดันฮาร์มอนิคส์เกิดขึ้นและแรงดันฮาร์มอนิคส์นี้ จะถูกจำกัดโดยค่ารีแอกแตนซ์รั่วของมอเตอร์เหนี่ยวนำ สำหรับมอเตอร์เหนี่ยวนำชนิดที่มีค่ารีแอกแตนซ์รั่วต่ำจะทำให้ค่าแรงดันฮาร์มอนิคส์ต่ำ และการสูญเสียเนื่องจากฮาร์มอนิคส์จะน้อย

3) เครื่องควบคุมความเร็วแบบมอดูเลตความกว้างพัลส์ (Pulse Width Modulation Controller) เครื่องควบคุมความเร็วแบบนี้ จะใช้อินเวอร์เตอร์ที่ควบคุมการสวิทช์โดยวิธีการมอดูเลตความกว้างพัลส์ วงจรเรียงกระแสจะใช้ไดโอดจำนวน 6 ตัว และใช้ตัวเก็บประจุเป็นวงจรกรองเพื่อกำเนิดแรงดันกระแสตรงที่คงที่ ในภาคของอินเวอร์เตอร์จะถูกควบคุมการสวิทช์เพื่อให้แรงดันและความถี่ที่แปรค่าได้ตามต้องการ ในรูปที่ 11 แสดงวงจรของเครื่องควบคุมความเร็วแบบมอดูเลตความกว้างพัลส์ พร้อมทั้งรูปคลื่นของแรงดันและกระแสขาออก



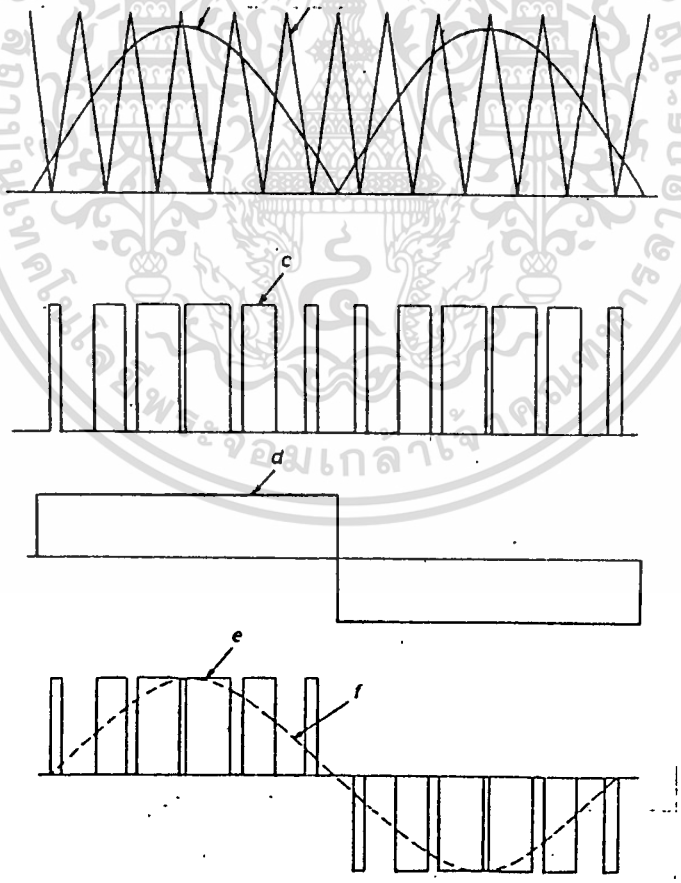
รูปที่ 11 เครื่องควบคุมความเร็วแบบมอดูเลตความกว้างพัลส์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เครื่องควบคุมชนิดนี้ จะทำให้เกิดแรงดันฮาร์มอนิคส์เช่นเดียวกับเครื่องควบคุมความเร็วแบบแหล่งพลังงานเป็นแหล่งแรงดัน ฮาร์มอนิคส์ที่เกิดขึ้น ขึ้นอยู่กับความถี่ของการสวิทช์ ซึ่งเราสามารถกำหนดได้ โดยทั่วๆไป ฮาร์มอนิคส์ของเครื่องควบคุมความเร็วแบบแหล่งพลังงานเป็นแหล่งแรงดันแปรค่าได้มาก และถ้านำไปใช้กับมอเตอร์เหนี่ยวนำที่มีค่ารีแอกแตนซ์รั่วสูง จะทำให้เกิดกระแส ฮามอนิคส์น้อยมาก ทำให้การสูญเสียน้อย

วิธีการต่างๆในการสร้างสัญญาณพัลส์แบบมอดูเลตความกว้าง

1)การมอดูเลตความกว้างพัลส์ โดยการสุ่มแบบตอบสนองตามธรรมชาติ (Natureal Sampled PWM) โดยส่วนมาก PWM Inverter ที่มีการควบคุมแบบอนาลอก(Analoge) จะใช้วิธีนี้ดังแสดงในรูปที่ 12



รูปที่ 12 สัญญาณพัลส์ที่เกิดจากการมอดูเลต
โดยวิธีการสุ่มแบบตอบสนองตามธรรมชาติ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

a คือ สัญญานอ้างอิงของการมอดูเลตความกว้างพลัส์

b คือ สัญญานพาหะ

c คือ สัญญานพัลส์แบบมอดูเลตความกว้าง

f คือ องค์ประกอบหลักมูลของสัญญานพัลส์

จากรูปที่ 12 สัญญานพาหะสามเหลี่ยม (Trianglar Carrier Wave) จะถูกนำมาเปรียบเทียบโดยตรงกับสัญญานรูปซายน์ (Sinusoidal Modulating Wave) เพื่อจะกำหนดช่วงเวลาหรือมุมการสวิทช์ ทำให้ได้สัญญานพัลส์ (Pule Width) ออกมาตามรูป จุดที่สำคัญของวิธีการนี้ก็คือขอบการสวิทช์ของสัญญานพัลส์จะเกิดจากจุดตัดระหว่างสัญญานสามเหลี่ยมกับรูปซายน์ ซึ่งจะเป็นผลทำให้ความกว้างของพัลส์เป็นสัดส่วนโดยตรงกับขนาด (Amplitude) ของสัญญานซายน์ ณ ตรงจุดที่เกิดการสวิทช์ ความกว้างของพัลส์สามารถกำหนดได้จากสมการต่อไปนี้

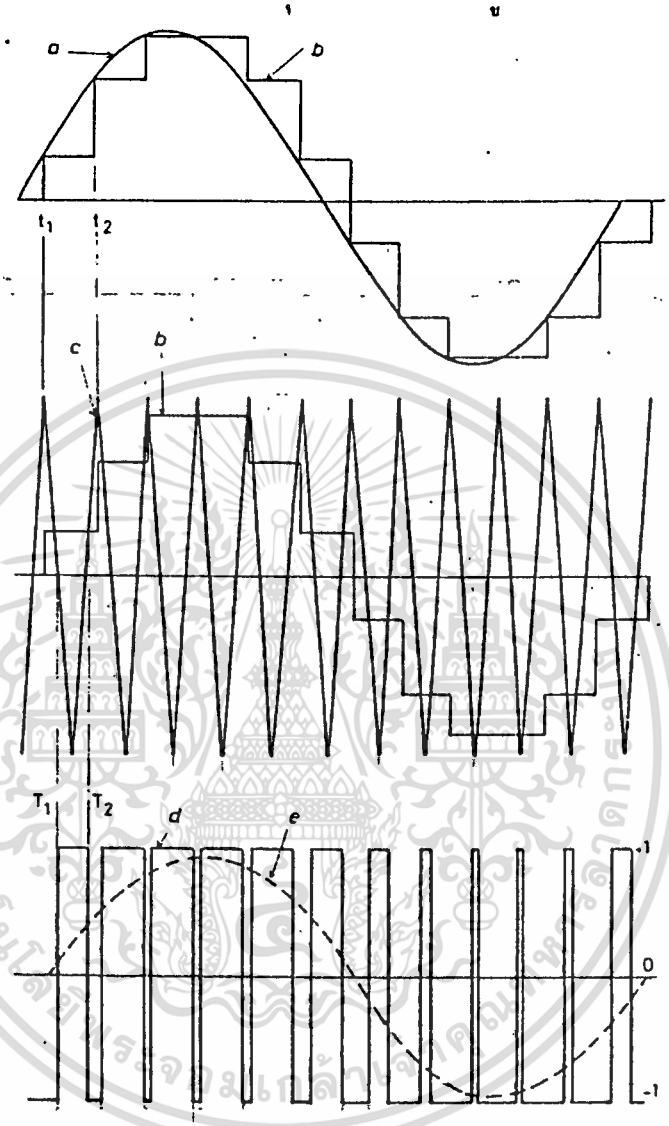
$$t_p = T/2[1+M/2(\sin w_m t_1+\sin w_m t_2)]$$

M = Amplitude factor

จากสมการ จะเห็นได้ว่า เป็นสมการที่แสดงถึงความสัมพันธ์ของเวลาในการสวิทช์กับขนาดของซายน์ เราจึงไม่สามารถที่จะคำนาณความกว้างของสัญญานพัลส์ได้โดยตรงเราต้องใช้วิธีการของ Bessel function ซึ่งอาจจะต้องใช้ไมโครคอมพิวเตอร์ช่วยในการคำนวณ

2)การมอดูเลตความกว้างพัลส์แบบสุ่มสม่ำเสมอ (Requlr Sampled PWM) อินเวอร์เตอร์ที่มีการควบคุมแบบนี้ส่วนมากจะเป็นแบบดิจิตอล หรือใช้ไมโครโปรเซสเซอร์เป็นตัวควบคุม ดังแสดงในรูปที่ 13

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 13 สัญญานพัลส์ที่เกิดจากการมอดูเลตโดยวิธีสุ่มสม่ำเสมอ

a คือ สัญญานอ้างอิง

b คือ สัญญานแซมเปิลและโฮลด์(Sampled-Hold)

c คือ สัญญานพาหะ

d คือ สัญญาณ์พัลส์แบบมอดูเลตความกว้าง

e คือ องค์ประกอบหลักมูลของสัญญานพัลส์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จากรูปที่ 13 ขนาดของสัญญานมอดูเลต a ที่มีการสุ่ม(Sample) ที่เวลา t_1 จะถูกเก็บไว้โดยวงจรแซมเปิลโอลด์ (Sampled and Hold Circuit) และก็คงไว้ในช่วง t_1 ถึง t_2 จนกว่าจะมีสัญญานแซมเปิลลูกใหม่เข้ามา สัญญานแซมเปิลและโฮลด์ b จะถูกเปรียบเทียบกับสัญญานพาหะสามเหลี่ยม c จุดที่ตัดกันของสัญญานทั้งสองจะใช้กำหนดช่วงเวลาการสวิทช์ (ในช่วง T_1 ถึง T_2) จะได้สัญญานรูป d ออกมา สัญญานพัลส์ d จะมีความกว้างเปลี่ยนไปตามจุดที่มีการตัดกันของสัญญาน c และ d ขนาดของสัญญานพัลส์จะคงที่ไว้ในแต่ละจุดที่มีการแซมเปิล (Sample) ความกว้างของสัญญานพัลส์จะเป็นสัดส่วนโดยตรงกับขนาดของสัญญานรูปซายน์ a ความกว้างของสัญญานพัลส์สามารถกำหนดได้จากสมการ ข้างล่างนี้

$$t_d = T/2(1+M \sin w_m t_1)$$

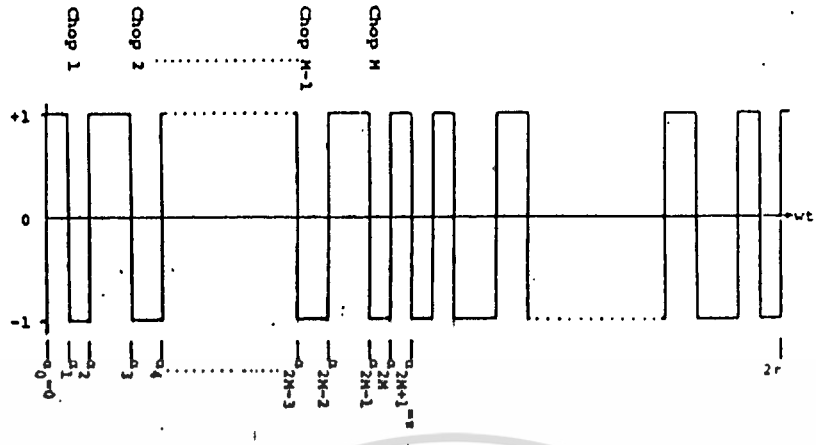
M =Amplitude factor

เทอมแรกของสมการจะสมนัยกับ ความกว้างของสัญญานพาหะสามเหลี่ยม (Unmodulated Carier-Frequency Pulse Width) และเทอมที่สองจะสมนัยกับสัญญานรูปซายน์ (Sinusiodal Modulation) ที่เวลา t_1 สมการนี้เราสามารถที่จะคำนวณความกว้างของพัลส์ได้ โดยตรงโดยใช้ไมโครโปรเซสเซอร์ วิธีการ Regular Sampling นี้สามารถกำเนิดความถี่หลักมูลของ PWM Inverter ได้จาก 0 ถึง 100 H_z

3)การมอดูเลตความกว้างพัลส์แบบดีที่สุด (Optimised PWM)

ข้อได้เปรียบของวิธีการ Optimissed PWM นี้ก็คือสามารถที่จะกำจัดฮามอนิคส์(Harmonics Elemination) ที่ไม่ต้องการได้ ซึ่งต้องใช้วิธีการควบคุมแบบดิจิตอลเท่านั้น เราจะต้องคำนวณรูปแบบ (Patten) ของการสวิทช์ออกมาก่อนโดยใช้ Fourier Serie Method แล้วจึงนำไปเก็บไว้ในหน่วยความจำของไมโครโปรเซสเซอร์ แล้วให้หน่วยประมวลผลกลาง(CPU) สแกนหาค่า Address ของความถี่หลักมูลที่เก็บเอาไว้ออกมาใช่ ตามรูปที่ 14

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 14 สัญญานพัลส์ที่เกิดจากวิธี Optimised

จากวิธีการกำเนิดสัญญาน PWM ทั้งสามแบบ เรามาพิจารณากันว่าจะใช้วิธีควบคุมแบบอนาลอกหรือดิจิตอล ซึ่งถ้าใช้อนาลก เมื่อจะมีการเปลี่ยนแปลงสัญญานเอาต์พุต ก็อาจจะต้องเปลี่ยนฮาร์ฟแวร์ ทำให้ยุ่งยากแต่ถ้าใช้ดิจิตอลเราจะเปลี่ยนเฉพาะซอร์ฟแวร์เท่านั้น โดยเฉพาะอุปกรณ์ที่ใช้สร้างสัญญาน PWM แบบอนาลอกนั้นหายากมากในประเทศไทย เพราะฉะนั้นในปริญญานิพนธ์นี้จะใช้การควบคุมแบบดิจิตอลถ้าเราเลือกการควบคุมแบบดิจิตอลวิธี Natural Sampled PWM ก็ตัดทิ้งไปได้เลยเหลือแต่ Regular Sample กับ Optimised สองวิธีนี้มีข้อดีข้อเสียต่างๆกันคือ

Regulal Sample PWM

- สามารถคำนวณ pattern การสวิทช์ได้ง่าย
- สามารถกำหนดมุมการสวิทช์ได้โดยตรง
- เป็นเทคนิคที่ไม่ต้องใช้คณิตศาสตร์ชั้นสูง
- ไม่สามารถกำจัดฮาร์มอนิคบางตัวออกได้
- อัตราส่วนของสัญญาณพาหะกับสัญญานมอดูเลตจะสูง
- ใช้หน่วยความจำมากกว่า

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

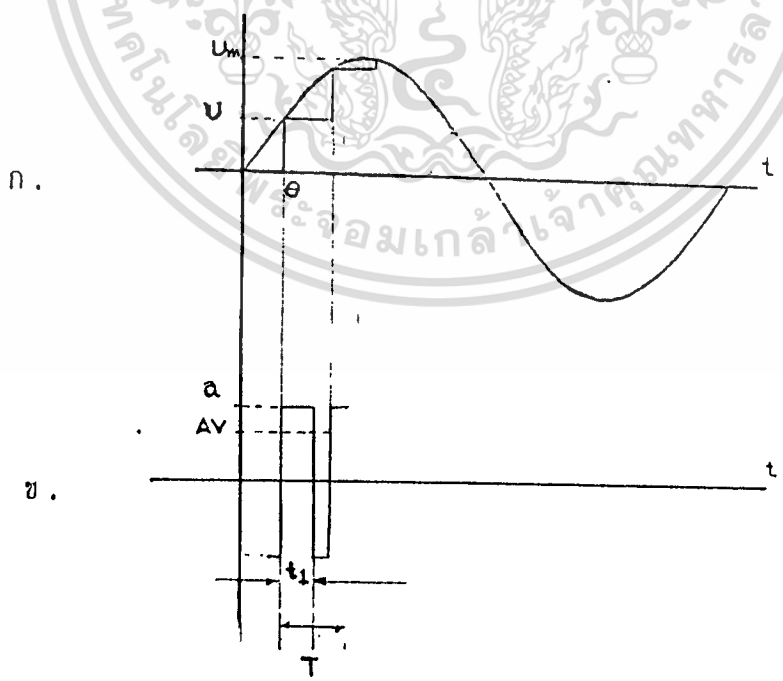
Optimised PWM

- สามารถกำจัดฮาร์มอนิคที่สำคัญออกได้
- อัตราส่วนของสัญญาณพาหะกับสัญญาณมอดูเลตต่ำ
- ใช่หน่วยความจำน้อยเพราะจะเก็บค่าเพียง 1/4 ไซเคิลเท่านั้น
- ต้องใช้คณิตศาสตร์ชั้นสูง
- การคำนวณ pattern การสวิทช์ทำได้ยาก

จากข้อดีและข้อเสียทั้งสองวิธีผู้จัดทำได้พิจารณาแล้วจึงเลือก Regular Sample PWM ซึ่งเหมาะกับระดับปริญญาตรี

พิจารณาการสร้างสัญญาณพัลส์แบบมอดูเลตความกว้างด้วยวิธี Regular Sample

พิจารณารูปคลื่นในรูปที่ 15 ซึ่งใช้หลักการหาค่าเฉลี่ยของสัญญาณพัลส์ให้มีขนาดเท่ากับขนาดของสัญญาณรูปซายน์ ณ จุดที่ต้องการดังแสดงในรูปที่ 15



รูปที่ 15 สัญญาณซายน์และพัลส์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จากรูปที่ 15 เราจะหาค่าเฉลี่ยของสัญญาณพัลส์ได้จากสมการข้างล่างนี้

$$AV = 1/_{T} [a.dt - a.dt] \quad \text{----------(8)}$$

$$= 1/_{T} [at_1 - aT + at_1]$$

$$= 1/_{T} [2at_1 - aT]$$

$$= a/_{T} [2t_1 - T] \quad \text{----------(9)}$$

จากรูปที่ 15ก ที่จุด t_1 สมการของรูปคลื่นซายน์

$$U = Um . \sin \theta \quad \text{----------(10)}$$

ที่จุด t_1 U = Av จะได้ว่า

$$Um.\sin \theta = a/_{T} [2t_1 - T]$$

$$2t_1 - T = T/_{m} [Um.\sin \theta]$$

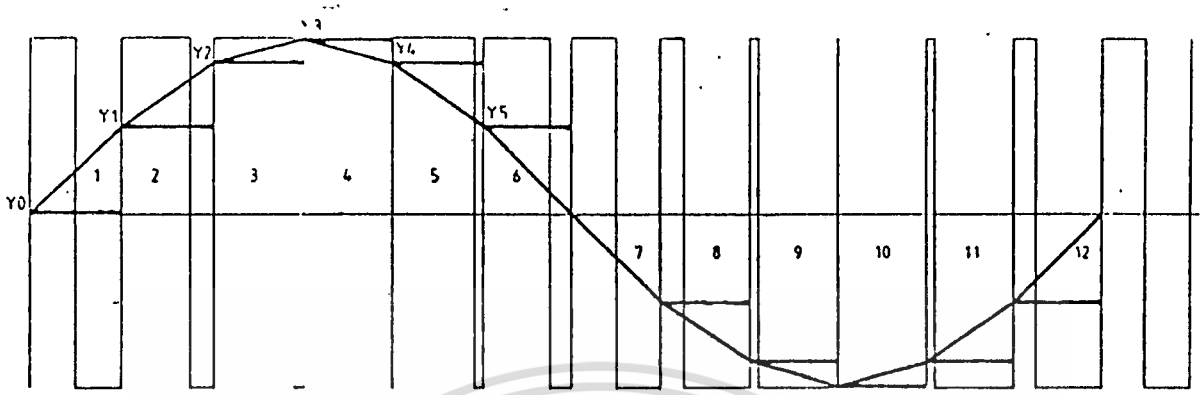
$$t_1 = T/_{2m} [Um.\sin \theta] + T/2 \quad \text{----------(11)}$$

ซึ่ง Um/a ก็คือ Amplitude Factor = M

$$t_1 = T/_{2} [1 + M.\sin \theta] \quad \text{----------(12)}$$

พิจารณาจากสมการที่ 12 T จะเป็นตัวกำหนดความถี่ของสัญญาณสวิทช์ และ M จะเป็นตัวกำหนดขนาดของสัญญาณซายน์ โดยปกติการควบคุมมอเตอร์กระแสสลับ จะต้องทำให้ อัตราส่วนของแรงดันต่อความถี่คงที่ ($V/_{F}$ =constant) ซึ่งเป็นตัวกำหนดแรงบิดของมอเตอร์ หมายความว่า ถ้าแรงดันเพิ่มขึ้นความถี่ก็จะต้องเพิ่มขึ้นตาม และกลับกันถ้าแรงดันลดความถี่ก็ต้องลดตามไปด้วย $V/_{F}$ ถึงจะคงที่ จากสมการที่ 12 จะเห็นได้ว่าเราสามารถควบคุมแรงดันและความถี่แยกกันได้ คือ ถ้าจะควบคุมความถี่ก็จะควบคุมที่ T แต่ถ้าจะควบคุมขนาดของแรงดันก็ควบคุมที่ M

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 16 แสดงสัญญาณรูปซายน์ที่แบ่งเป็น 12 จุดสังเคราะห์

จากรูปที่ 16 เราจะแบ่งสัญญาณรูปซายน์ ออกเป็นจุดต่างๆ (Synthesis Point) ซึ่งจะต้องมีจำนวนจุดสังเคราะห์ = 6n เมื่อ n=1,2,..,40 จะได้สมการของความถี่เอาท์พุทดังนี้

$$f_T = f_s / Nst.Z \quad \text{----------(13)}$$

หรือ $t_T = Nst.Z.t_s$

f_T = ความถี่เอาต์พุทของอินเวอร์เตอร์

f_s = ความถี่การสวิทช์

Nst = จำนวนจุดสังเคราะห์

Z = จำนวนพัลส์ใน 1 จุดสังเคราะห์

หมายความว่า ถ้าเราปรับความถี่เอาต์พุทของอินเวอร์เตอร์เราสามารถปรับได้ที่ f_s,Nst และ Z เช่นจากรูปที่ 16 ซึ่งมีค่า Z = 1,Nst = 12 ถ้าเราให้ fs = 10 KHz จะได้

$$f_T = 10 \times 10^3 / 12 \times 1 \text{ Hz}$$

$$= 833.33 \text{ Hz}$$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

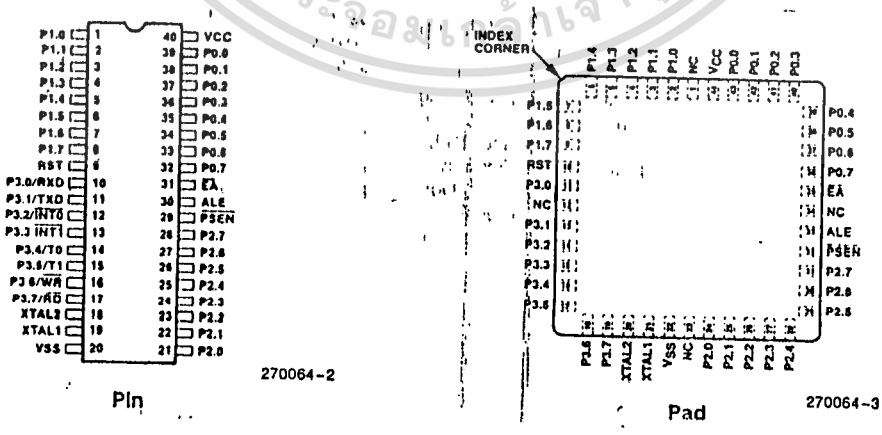
ไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ คือ ไมโครคอมพิวเตอร์แบบชิพเดียว มีความสะดวกในการใช้งาน ที่ใช้ในปริญญานิพธ์ฉบับนี้จะใช้ของอินเทล เบอร์ 8031 ซึ่งอยู่ในตระกูล MCS-51 ซึ่งเป็นอุปกรณ์ที่ออกแบบมาสนองความต้องการของผู้ใช้ คือมีสายอินพุทภายในตัวเอง พอร์ตของอิพุท บัฟเฟอร์อินเตอร์เฟสและสายควบคุมอื่นๆ ที่ใช้สำหรับแยกข้อมูลกับแอดเดรสและยังมีชุดคำสั่งเพิ่มขึ้นเป็นพิเศษ เพื่อจัดการข้อมูลแถมท้ายด้วยวงจรตั้งเวลากับวงจรนับด้วย

ลักษณะทั่วไป

- สร้างโดยใช้ CHMOS ทำงานด้วยแหล่งจ่ายเดียว 5 โวลท์
- ซีพียูขนาด 8 บิท
- มีแรมบรรจุไว้ภายในขนาด 128 ไบต์
- วงจรตั้งเวลา วงจรนับขนาด 16 บิท 2 ชุด
- อินเตอร์รัพต์แบ่งออกเป็น 2 ระดับ 5 แหล่ง
- มีคำสั่งคูณ หาร ทางฮาร์ดแวร์
- ตัวเลขทางคณิตศาสตร์ใช้ได้ทั้งแบบไบนารีและเดซิมอล

ตระกูลของMCS-51 จะมีทั้งแบบมี ROM ในตัว หรือไม่มี ROM หรือมี EPROM บนชิพเดียวกัน และจะมีตำแหน่งขาที่เหมือนกัน



รูปที่ 17 การจัดวางขาและหน้าที่ของพอร์ตต่างๆ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตารางที่ 1ไมโครคอนโทรลเลอร์ตระกูล MCS-51

เบอร์	หน่วยความจำภายใน		ตัวตั้งเวลา/ตัวนับ	INT
	โปรแกรม	ข้อมูล		
8052 AH	8K X 8 ROM	256 X 8 RAM	3 X 16 Bit	6
8051 AH	4K X 8 ROM	128 X 8 RAM	2 X 16 Bit	5
8051	4K X 8 ROM	128 X 8 RAM	2 X 16 Bit	5
8032 AH	ไม่มี	256 X 8 RAM	3 X 16 Bit	5
8031 AH	ไม่มี	128 X 8 RAM	2 X 16 Bit	5
8031	ไม่มี	128 X 8 RAM	2 X 16 Bit	5
8751 H	4K X 8 EPROM	128 X 8 RAM	2 X 16 Bit	5
8751H-12	4K X 8 EPROM	128 X 8 RAM	2 X 16 Bit	5

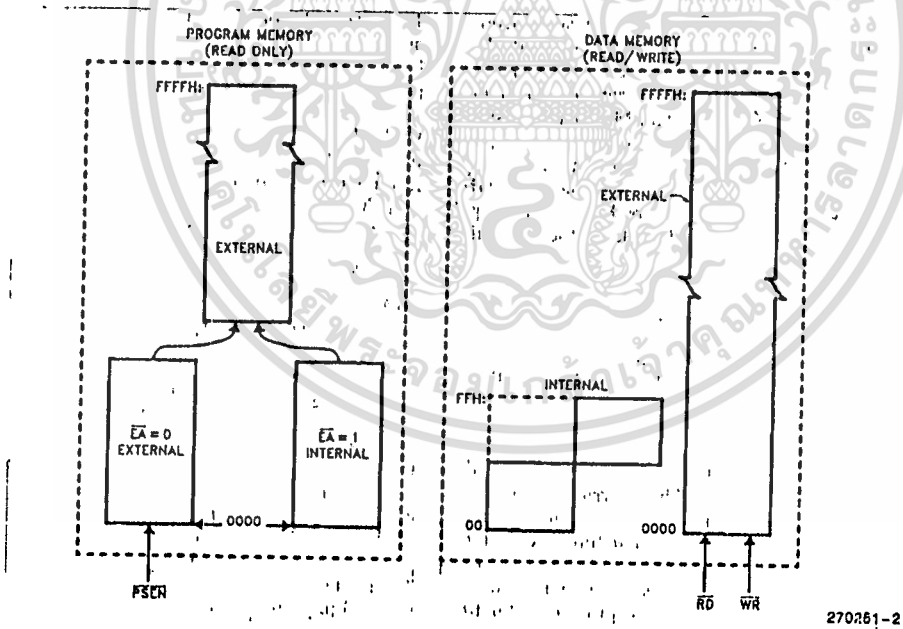
จากรูปที่ 17 ก และ ข นั้นแสดงจำนวนขาและหน้าที่ของ MCS-51 เบอร์ 8051 8051AH และ 8052AII เป็นอุปกรณ์ที่ทำงานโดยการควบคุมจากโปรแกรมในหน่วยความจำ (RAM) โดยการโปรแกรมด้วยต้นแบบเพื่อให้เกิดข้อมูลบนรอมครั้งหนึ่ง เป็นจำนวนมากจึงเหมาะกับการผลิตเพื่อใช้งานจำนวนมากๆ อีกสามเบอร์คือ 8031AH และ 8031 สามารถใช้แทน 8051 และ 8052ได้โดยไม่ต้องส่งให้โรงงานโปรแกรม เพราะเราจะเขียนและทดสอบโปรแกรมด้วยหน่วยความจำภายนอกแทน สำหรับท่านที่ต้องการการป้องกันการอ่านจากภายนอก ก็สามารถเลือก 8751 และ 8752 ที่มีอีพรอม(EPORM) ภายในที่จัดค่าเวลาในการอ่านข้อมูล เพื่อกันการลอกเลียนโปรแกรม

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หน่วยความจำ

จากรูปที่ 19 เป็นหน่วยความจำภายใน MCS-51 หน่วยความจำนี้แบ่งได้เป็น 2 กลุ่มคือ หน่วยความจำสำหรับเก็บโปรแกรมและหน่วยความจำสำหรับเก็บข้อมูล หน่วยความจำแรกมีแอดเดรสที่ต่ำกว่า 4 หรือ 6 กิโลไบต์บรรจุอยู่ภายในรอม สำหรับเบอร์ที่ไม่มีรอมภายในจะใช้หน่วยความจำภายนอกแทน ซึ่งอาจจะเป็นรอม แรม และอีพรอมแทนก็ได้

MCS-51 จะอ่านหน่วยความจำสำหรับเก็บโปรแกรมเข้ามาเป็นภาษาเครื่องตามลำดับ ส่วนหน่วยความจำสำหรับเก็บข้อมูลจะใช้เป็นที่เก็บตัวแปรการคำนวณหาผลลัพท์ทันทีกับข้อมูลที่มี 16 บิท(WORD)ตารางที่ใช้ค้นหาค่าต่างๆ และหน้าที่อื่นที่คล้ายกันหน่วยความจำสำหรับเก็บข้อมูลใช้ร่วมกับหน่วยความจำภายนอกได้ถึง64 กิโลไบต์ ซึ่งเลือกใช้รอมหรือแรมก็ได้และยังมีรีจิสเตอร์พิเศษ ที่ใช้หน่วยความจำภายนอกของแรมได้ 128 หรือ 256 ไบต์



รูปที่ 18 โครงสร้างภายในหน่วยความจำของ MCS-51

รีจิสเตอร์ภายใน MCS-51

MCS-51 มีรีจิสเตอร์ที่อำนวยความสะดวกในการใช้งานตามคำสั่งต่างๆประกอบด้วยแอคคิวมูเลเตอร์ รีจิสเตอร์ B ที่ใช้ในการคูณและหาร รีจิส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เตอร์สถานะ สแต็กพอยน์เตอร์ ดาต้าพอยน์เตอร์(2 X 8 Bit) หรือ 1 X 16 Bit)พอร์ตหมายเลขศูนย์ถึงหมายเลขสามเป็นแบบคู่(Bidirectional) ซึ่งใช้ส่งและรับข้อมูลชนิดอนุกรม รีจิสเตอร์ 16 Bit ที่เป็นวงจรตั้งเวลาและวงจรนับ รีจิสเตอร์ซึ่งจองไว้สำหรับใช้นับตัวที่ 3 รีจิสเตอร์คำสั่งสำหรับหน้าที่พิเศษ(เช่น การอินเตอร์รัพต์ RTC:Real Time Clock) และอินพุทเอาท์พุทแบบอนุกรม

สายต่างๆของบัสและพอรต์

จะเห็นโครงสร้างภายในของ MCS-51 ในรูปที่ 19 เราจะสมมุติให้มีบัสสองทิศทาง 4 เส้น และพอรต์ขนาด 8 บิทตามทฤษฎี แต่ที่จริงนั้นจะใช้ได้ก็ต่อเมื่อมีการใช้หน่วยความจำภายในตัว(ROM OR RAM)

เมื่อไม่ใช้หน่วยความจำภายในพอร์ต 0 และ 2 จะถูกใช้เป็นบัสของข้อมูลและแอดเดรส ดังนั้นพอร์ต 2 พอร์ต ยังคงใช้งานเป็นอินพุทและเอาต์พุทพอร์ต 2 ทำหน้าที่เป็นสายสัญญาณแอดเดรส A15 - A8 ส่วนพอร์ต 0 ทำหน้าที่เป็นสายแอดเดรส A7-A0 ออกจากบิตข้อมูล D7-D0 ส่วนเอาพุทของขา RD และ WR มาจากสายเอาท์พุตของพอรต์ 3 โดยโปรแกรมภายในใช้ RD และ WR เพื่อการเขียนและอ่านข้อมูลกับข้อมูลของหน่วยความจำภายนอก

(1) ขา PSEN เป็นขารับสัญญาณสำหรับเปิดให้มีการอ่านหน่วยความจำภายนอก ในทุกๆรอบคำสั่ง ระหว่างการทำงานด้วยโปรแกรมในรอมหรืออีพรอม จะเห็นว่าสัญญาณ PSEN ทำงานถึงสองครั้ง เพราะว่ามีการอ่านข้อมูล 2ไบต์ในแต่ละรอบคำสั่ง

(2) ขา EAเป็นขาอินพุตใช้ร่วมกับแอดเดรสภายนอกโดยจะมีค่าลอจิก "0" เมื่อไมโครโปรเซสเซอร์อ่านคำสั่งจากหน่วยความจำภายนอก ขา EA ยังมีหน้าที่เป็นอินพุทสำหรับป้อนแรงดัน 21 โวลท์ เพื่อโปรแกรมให้กับอีพรอม(สำหรับกรณีที่ใช้ 8751 หรือ 8752)

วงจรนับ /วงจรตั้งเวลา

จากตารางที่ 1 จะเห็นว่า 8052 มีวงจรนับและวงจรตั้งเวลาชนิด 16 Bit มากกว่า 8051 อยู่ในตัวต่อไปเราจะศึกษาการทำงานอย่างย่อๆ ของวงจรและวงจรตั้งเวลา

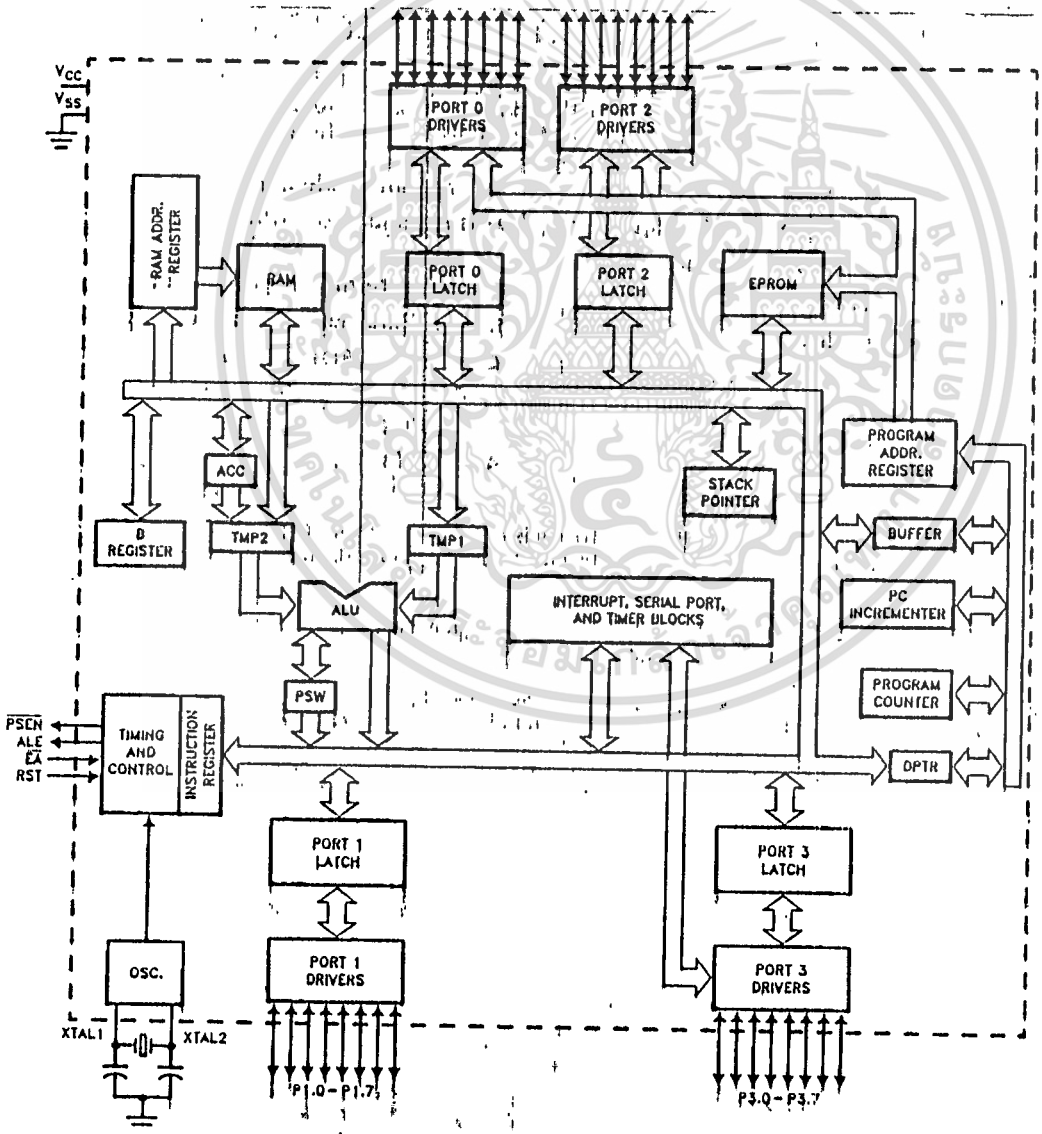
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

(1) เมื่อทำงานเป็นวงจรตั้งเวลา

รีจิสเตอร์ทำหน้าที่ตั้งเวลาจะเพิ่มค่าขึ้นหนึ่งค่าของทุกๆ รอบคำสั่งของเครื่อง และจะนับด้วยอัตราสูงสุดที่ 1/12 ของความเร็วสัญญาณนาฬิกาของไมโครโปรเซสเซอร์

(2) เมื่อทำงานเป็นวงจรนับ

รีจิสเตอร์วงจรนับจะเพิ่มขึ้นหนึ่ง เมื่อมีสัญญาณป้อนให้อินพุต T0 T1 หรือ T2 (T2 มีเฉพาะ 8052) เป็นขอบสัญญานขาลงอัตราการนับสูงสุดคือ 1/24 ของความเร็วสัญญานนาฬิกา ของ CPU



รูปที่ 19 โครงสร้างของ MCS-51

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

พอร์ชนิดอนุกรม

ไมโครคอนโทรลเลอร์ทั้งหมดในตระกูล MCS-51 เป็นชิปที่มีอินเตอร์เฟสอนุกรมชนิด สองทิศทาง มีบัฟเฟอร์สำหรับรับข้อมูลเป็นพิเศษ เพื่อเพิ่มความเร็วในการสื่อสาร พอร์ตชนิดอนุกรมนั้น เราสามารถเลือกโปรแกรมใช้งานได้ 4 แบบ อัตราการส่งข้อมูลที่เลือกใช้ได้จะสูงถึง 19,200 บิต/วินาที ด้วยความเร็วของสัญญานนาฬิกา 1 MHz

อินเตอร์รัพต์และชุดคำสั่ง

8051, 8031 รับการอินเตอร์รัพต์ได้ 5 แหล่งส่วน 8052 ได้ 6 แหล่ง โดยรับอินเตอร์รัพต์ได้ 2 ระดับ โดยไม่ต้องอาศัยวงจรภายนอกช่วย

ตัวจับเวลา/ตัวนับ (Timer/Counter)

MCS-51 มี 16 บิตตัวจับเวลาหรือตัวนับ 2 ตัวคือ Timer/Counter0 และ Timer/Counter1 ขณะที่ Timer/Counter แต่ละตัวสามารถที่จะกำหนดให้ทำงานได้เป็นตัวจับเวลาหรือตัวนับ

ตัว Timer/Counter0 และ Timer/counter1

แต่ละตัวจะถูกกำหนดให้ทำงานเป็นตัวจับเวลาหรือเป็นตัวนับ ได้โดยการเซตหรือเคลียร์ในรีจีสเตอร์ TMOD ในกลุ่ม SFR

ในฟังก์ชั่นตัวจับเวลา ตัวรีจีสเตอร์จะเพิ่มค่าทุกๆในวัฏจักรแมชชีนดังนั้นตัวเลขในรีจีสเตอร์จะเป็นจำนวน วัฏจักรแมชชีนเนื่องจากแต่ละวัฏจักรแมชชีน ประกอบด้วย 12 คาบของออสซิลเลเตอร์ อัตราการนับแต่ละครั้งจะกินเวลาเป็น 1/12 ของความถี่ ออสซิลเลเตอร์

ในฟังก์ชั่นตัวนับ รีจีสเตอร์จะมีการเพิ่มค่าทุกครั้ง ที่มีการเปลี่ยนสถานะจาก "1" เป็น "0" ที่เข้ามาที่ขา T0 หรือ T1ในฟังก์ชั่นนี้สัญญานภายนอกที่เข้ามาจะถูกรับแซมปลิ้ง (Sampling)ระหว่างช่วง S5P2ของทุกวัฏจักรแมชชีนโดยถ้าแซมปลิ้งสัญญานเข้าเป็นระดับสูงในวัฏจักรหนึ่งดังนั้นถ้าในวัฏจักรตัวต่อมาของสัญญาณเข้าเป็นระดับต่ำรีจีสเตอร์จะนับเพิ่มขึ้นหนึ่งค่า โดยที่ค่าใหม่ของตัวนับ จะปรากฏที่รีจีสเตอร์ช่วง S3P1ของวัฏจักรซึ่งค่าหนึ่งที่รับเข้าไป จะใช้ช่วง 2 วัฏจักรแมชชีน (เท่ากับ 24 คาบ)ในการรับค่าช่วงการเปลี่ยน 1 เป็น 0 ดังนั้นค่าสูงสุดในการนับ

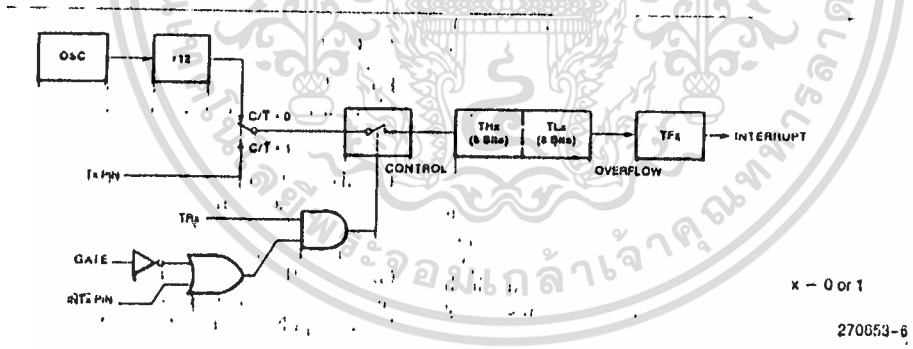
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จะมีอัตรา 1/24 ของความถี่ออสซิลเลตเตอร์ และสัญญาณอินพุทที่นับนั้นจะไม่มีช่วงระยะห่างที่แน่นอนของ Duty Cycle แต่จะถูกนับเมื่อระดับแรงดันที่ถูกแซมปลิ้งในแต่ละครั้ง จะต้องมีช่วงคงที่อย่างน้อย 1 วัฏจักรแมชชีนก่อนที่จะมีการเปลี่ยนค่าระดับแรงดันใหม่

ในการเลือกทำงานระหว่างตัวนับเวลากับตัวจับเวลา จะเลือกได้ 4 โหมดคือ โหมด 0,1 และ 2 เลือกได้ทั้งสองตัวของ Timer/Counter ส่วนโหมด 3 จะทำงานแตกต่างออกไป

โหมด 0

การใช้ Timer/Counter 0 หรือ 1ให้อยู่ในโหมด 0 จะทำงานคล้ายกับ MCS 48 โดยตัวจับเวลาของ MCS 48 มีขนาด 8 บิต มีตัว Prescaler เป็นตัวหาร 12 ดังรูปที่ 20 จะเป็นการ แสดงการทำงานใน โหมด 0 ของตัวจับเวลาตัวนับ 1



รูปที่ 20 แสดงการทำงานโหมด 0

ในโหมดนี้ รีจีสเตอร์ตัวจับเวลาจะถูกกำหนดให้มี 13 บิต ด้วยการนับขึ้นเมื่อเป็น " 1 " หมดทุกบิต จะกลับมาที่ "0" ทุกบิตใหม่ ซึ่งก็คือการเกิด over flow ไปทดให้แฟลกอินเทอร์รัพท์ TF1) ปรับเป็น "1" การควบคุมให้เริ่มนับอินพุท จะควบคุมด้วยการ ENABLE TR1 = 1 ; GATE = 0 และขา INT1 = 1 การปรับ GATE = 1 เป็นการตั้งตัวนับให้ถูกควบคุมด้วยสัญญาณจากภายนอก เข้าขา INT1 TR1 จะเป็นบิทควบคุมในรีจีสเตอร์ TMOD ของ SFR

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

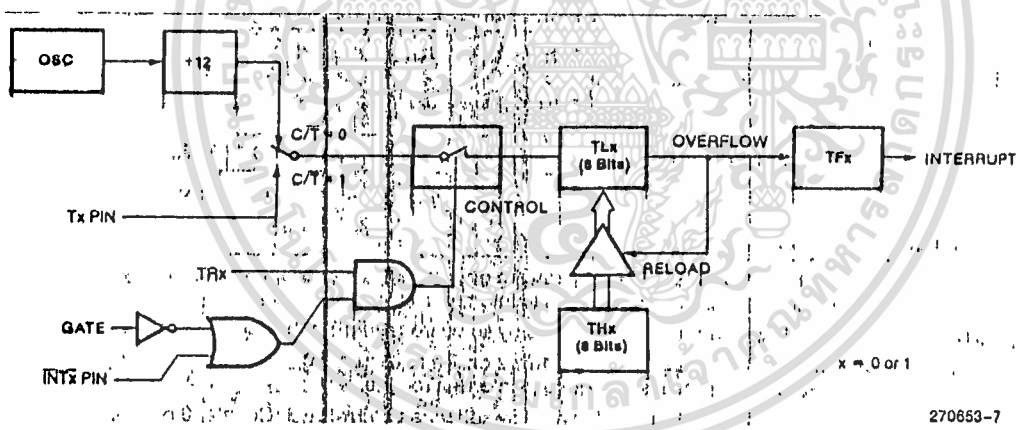
การทำงานในโหมด 0 ใน Timer/Count0 ก็เหมือนกับ Timer /Counter1 ผิดกันตรงที่การ SET ค่าใน TMOD ของ FSR เท่านั้น

โหมด 1

โหมด 1 ทำงานเหมือนกับ โหมด 0 ต่างกันแต่ เฉพาะการใช้ รีจีสเตอร์ ตัวจับเวลาตัวนับ จะทำงานนับด้วยขนาด 16 บิท

โหมด 2

โหมด 2 มีการทำงานโดยการกำหนดให้ตัวนับ 8 บิท ของ TL1 และจะโหลดใหม่โดยอัตโนมัติ จากค่าที่ตั้งไว้ใน TH1 ทุกครั้งเมื่อมีการ overflow ซึ่งค่าใน TH1 จะไม่หายไป โหมดนี้จะมีการทำงานตามรูปที่ 21 " โหมด 1 โหมด 2 จะนำไปใช้ใน Project " ส่วนโหมด 3 จะไม่ขอกล่าวในที่นี้



รปที่ 21 แสดงการทำงานโหมด 3

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การใช้ทรานซิสเตอร์เป็นตัวสวิทช์

ในปัจจุบันการใช้ทรานซิสเตอร์ในงานอิเล็คทรอนิคส์กำลัง ได้ขยายตัวไปอย่างกว้างขวาง ทั้งนี้เพราะในปัจจุบันทรานซิสเตอร์ได้รับการพัฒนาไปอย่างมาก ทำให้สามารถทนกระแสและแรงดันได้สูงขึ้น อีกทั้งความเร็วในการสวิทช์ก็สูงขึ้นด้วย การเพิ่มความเร็วของทรานซิสเตอร์ทำให้เราสามารถทำงานที่ความถี่สูงขึ้น คุณสมบัติทางด้านความเร็วของทรานซิสเตอร์นี้นับเป็นจุดเด่นที่สำคัญที่สุดอันหนึ่งของทรานซิสเตอร์ และเนื่องจากทรานซิสเตอร์ PNP จะทนแรงดันได้ไม่สูงนัก อีกทั้งความเร็วในการสวิทช์ยังต่ำกว่าทรานซิสเตอร์แบบ NPN ดังนั้น ในวงจรอิเล็คทรอนิคส์กำลังจึงใช้เฉพาะทรานซิสเตอร์แบบ NPN

การขับนำเบสของทรานซิสเตอร์

ในวงจรอิเล็คทรอนิคส์กำลังที่ใช้ทรานซิสเตอร์เป็นสวิทช์นั้น การขับนำเบสของทรานซิสเตอร์นับเป็นสิ่งสำคัญมากอย่างหนึ่ง เพราะคุณสมบัติที่สำคัญบางอย่างของทรานซิสเตอร์ เช่น แรงดันสูงสุด(BV_{ceo}) เวลาหยุดนำกระแส(t_{off}) เวลาเริ่มนำกระแส(t_{on}) และย่านทำงานที่ปลอดภัย(SOA) จะขึ้นอยู่กับลักษณะการขับนำเบส ในปัจจุบันได้มีการเสนอวิธีการขับนำเบสไว้หลายวิธี ซึ่งส่วนใหญ่มีวัตถุประสงค์ที่จะลดเวลาในการสวิทช์ เพื่อลดกำลังสูญเสียในการสวิทช์และเพิ่มความถี่ในการทำงานให้สูงขึ้น อย่างไรก็ดีคุณสมบัติของการขับนำเบสที่ดีควรมีลักษณะดังนี้คือ

1) ในการเริ่มขับนำทรานซิสเตอร์ควรที่จะให้การเพิ่มของกระแสเบส $dI_{B/dt}$ มีค่าสูง และควรจะให้กระแสเบสมีค่ายอดสูงกว่ากระแสเบสซึ่งเพียงพอที่จะทำให้ทรานซิสเตอร์อิ่มตัวประมาณ 2-3 เท่า อันเป็นการลด t_{on} ให้สั้นลงได้

2) ควรจะใช้กระแสเบสที่เหมาะสมกับกระแสคอเล็คเตอร์ คือให้เพียงพอที่จะทำให้ทรานซิสเตอร์อิ่มตัวพอดี แต่ไม่ควรให้มีค่ามากเกินไป เพราะจะเป็นการเพิ่ม t_{off} อันเนื่องมาจาก storage time ซึ่งเป็นเวลาที่ใช้ในการกวาดพาหะข้างน้อยส่วนเกินออกจากเบส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3) การทำให้ทรานซิสเตอร์หยุดนำกระแส ควรจะมีการดึงกระแสออกจากเบสโดยให้มีค่า dI_{B}/dt สูง แต่ไม่ควรเกิน 40 $A/\mu s$ เพราะอาจจะทำให้เกิด secondary breakdown ขึ้นได้ การดึงกระแสออกจากเบสจะเป็นการช่วยกวาดพาหะข้างน้อยส่วนเกินออกจากเบสของทรานซิสเตอร์ อันเป็นการลด t_{off} ของทรานซิสเตอร์

4) ในระหว่างที่ทรานซิสเตอร์อยู่ในสถานะปิดกั้นกระแส(off-state) ควรที่จะไบอัสย้อนหัวต่อเบส-อิมิเตอร์ เพื่อป้องกันการนำกระแสเนื่องจากสัญญาณรบกวน และการเปลี่ยนแปลงแรงดัน dV_{CE}/dt ที่มีค่าสูง

5) วงจรขับเบสควรมีประสิทธิภาพสูง เนื่องจากทรานซิสเตอร์กำลังสูงจะมีอัตราขยายกระแสต่ำ คือ ประมาณ 5-10 เท่า ซึ่งทำให้จำเป็นต้องใช้กระแสขับนำเบสมาก ดังนั้นแรงดันในวงจรเบสไม่ควรจะสูงเกินไป และไม่ควรจะให้มีกระแสไหลในวงจรเบสมากเมื่อทรานซิสเตอร์อยู่ในสถานะปิดกั้นกระแส

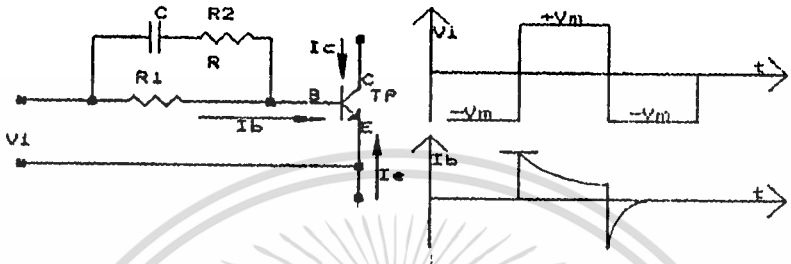
6) วงจรขับนำเบสควรที่จะเป็นวงจรง่ายๆและมีความเชื่อถือได้สูง ถ้าสามารถเข้าได้กับ TTL (TTL Compatible) จะทำให้การควบคุมสะดวกมากขึ้น นอกจากนี้ถ้าหากมีการแยกกันทางไฟฟ้าระหว่างวงจรขับกับทรานซิสเตอร์กำลัง หรือระหว่างสัญญาณควบคุมกับวงจรขับ จะเป็นการลดปัญหาเรื่องสายดิน การแยกอาจทำได้โดยใช้หม้อแปลงความถี่สูง(pluse transformer) หรือ ถ้าในระดับวงจรขับกับสัญญาณควบคุมซึ่งมีกำลังต่ำอาจจะใช้ opto coupler ก็ได้

วงจรขับนำเบส

การที่จะให้ได้มาซึ่งลักษณะของวงจรขับนำเบสที่ดีนั้น เราจำเป็นต้องรู้จักวงจรพื้นฐานที่จะให้ได้มาซึ่งฟังค์ชั่นที่เราต้องการ และตอนต่อไปนี้เราจะแนะนำให้รู้จักกับวงจรพื้นฐานต่างๆที่ใช้ในวงจรขับนำเบสทั่วๆไป

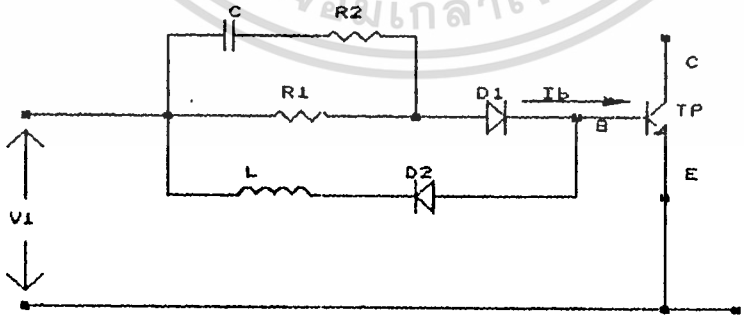
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1) วงจรเร่งกระแสเบส



รูปที่ 22 วงจรเร่งกระแสเบส (ก) และรูปคลื่นของกระแสเบส

จากวงจรรูปที่ 22 ก C ซึ่งเป็นตัวเก็บประจุเร่งกระแส (Speed up capacitor) จะทำตัวเสมือนกับวงจรลัดในตอนที่เกิดการเปลี่ยนแปลง ดังนั้น ในตอนที่ Vi เพิ่มจาก $-V_m$ ไปเป็น $+V_m$ กระแสเบสจะมีค่ายอดสูงสุด และในสภาวะอยู่ตัว ตัวเก็บประจุ C จะทำตัวเสมือนเป็นวงจรเปิด ในกรณีที่เราต้องการจำกัดค่า $d I_{b/dt}$ ช่วงที่ V_i เปลี่ยนจาก $+V_m$ เป็น $-V_m$ เราอาจทำได้โดยใช้ตัวเหนี่ยวนำและไดโอดต่อดังในวงจรในรูปที่ 23



รูปที่ 23 วงจรขับกระแสเบสที่มีการเร่งกระแสเบสในตอนทำให้ TP เริ่มนำกระแสและมีการหน่วงกระแสในตอนทำให้ TP หยุดนำกระแส

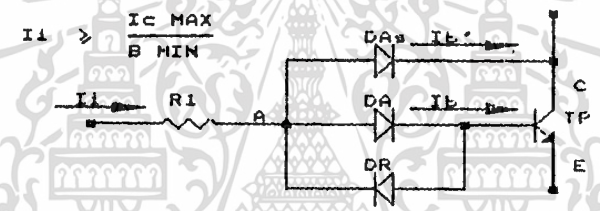
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2) การปรับกระแสเบสให้เหมาะสม

ในการขับนำเบสของทรานซิสเตอร์กำลังเพื่อ ใช้เป็นสวิทช์เรามักใช้สัญญาณขั้นที่มีค่าคงที่ ทำให้กระแสที่ไหล มีค่าคงที่ดังนั้นเราจึงจำเป็นต้องมีตัวปรับ I_B ให้เหมาะสมซึ่งจะทำได้โดย

ก) ใช้ไดโอดป้องกันการอิ่มตัวเกินควร

วิธีนี้เราใช้วิธีปรับ I_B ให้เหมาะสมโดยใช้ไดโอดป้องกันการอิ่มตัวเกินควร(Antisaturation diode)



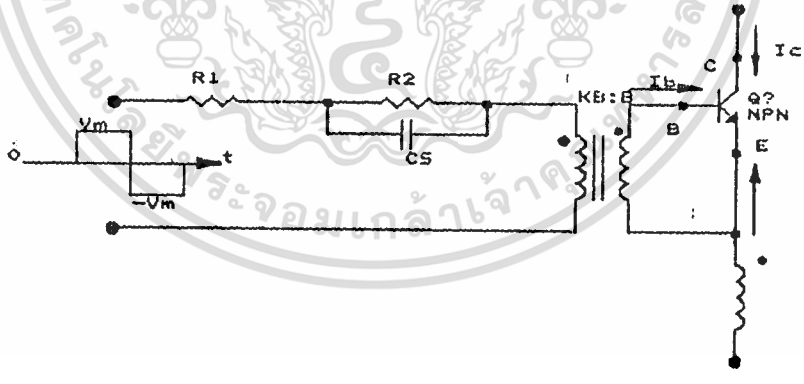
รูปที่ 24 วงจรปรับ I_B ให้เหมาะสมกับกระแสคอลเล็กเตอร์

ในการออกแบบวงจรขับนำเบสนั้น ถ้ากระแส I_i มีค่าคงที่และมีค่ามากกว่าหรือเท่ากับอัตราส่วนของกระแสคอลเล็กเตอร์สูงสุดที่จะมีได้ ($I_{c\ max}$) กับอัตราการขยายกระแสต่ำสุดของทรานซิสเตอร์(B_{min}) แล้ว TP จะอยู่ในสภาวะอิ่มตัวอยู่เสมอถ้า $V_{CB} > V_{AK}$ ไดโอด D_{AS} ซึ่งเป็นไดโอดป้องกันการอิ่มตัวของ TP จะไม่นำกระแส แต่ถ้า I_i มีค่ามากเกินไปจะทำให้ V_{CE} ลดลง จนกระทั่ง V_{AK} สูงกว่า V_{CE} เท่ากับแรงดันเริ่มนำกระแสของไดโอด (Cutin voltage มีค่าประมาณ 0.5V) D_{AS} จะเริ่มนำกระแส ทำให้กระแสเบส I_B ลดลง โดยที่แรงดันตกคร่อมไดโอด D_{AS} และ D_A ใกล้เคียงกัน จากรูปที่ 24 เราจะเห็นได้ว่า จะรักษาศักดาไฟฟ้าของคอลเล็กเตอร์ให้เท่ากับศักดาไฟฟ้าของทรานซิสเตอร์กำลัง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ข) ใช้การขับนำตามกระแสขาออก (Regenerative or proportional feedback drive)

เป็นการส่งผ่านหม้อแปลงกลับมาขับนำเบสหลังจากที่วงจรเริ่มขับนำเบสได้ ทำให้ทรานซิสเตอร์เริ่มนำกระแสในรูปที่ 25 เราจะให้กระแสอีมิเตอร์ไหลผ่านขดลวดซึ่งมีเพียงหนึ่งขดของหม้อแปลง กระแสอีมิเตอร์นี้จะเหนี่ยวนำให้เกิดกระแสไหลในขดลวดในวงจรเบส เพื่อที่จะให้กระแสที่เกิดจากการเหนี่ยวนำมีค่ามากพอที่จะทำให้ทรานซิสเตอร์อยู่ในสถานะนำกระแสได้ เราจะให้ขดลวดของวงจรเบสมีจำนวนรอบน้อยกว่าอัตราขยายกระแสของทรานซิสเตอร์ในสภาวะอิ่มตัว (β) และมีขั้วแสดงดังในรูปที่ 25 การเริ่มขับนำเบสและการทำให้ทรานซิสเตอร์หยุดนำกระแสจะทำได้โดยการใช้ขดลวดชุดที่ 3 ซึ่งมีขดลวดมากกว่า β เพื่อเป็นการลดกระแสในวงจรขับ C_s จะต้องมีค่าใหญ่มากพอที่จะสามารถทำให้ทรานซิสเตอร์เริ่มหรือหยุดนำกระแสได้ R_2 มีไว้สำหรับที่จะทำให้มีกระแสขับนำเบสในตอนที่มีกระแสโหลดน้อย การใช้สัญญาณขับบวกลบจะทำให้เวลาในการสวิทช์ลดลง



รูปที่ 25 วงจรขับนำเบสตามกระแสขาออก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 3
การคำนวณและการสร้าง

ส่วนประกอบของโครงงานนี้แบ่งออกเป็น 2 ส่วน

3.1 ส่วนสร้างสัญญาณพัลส์แบบมอดูเลตความกว้าง

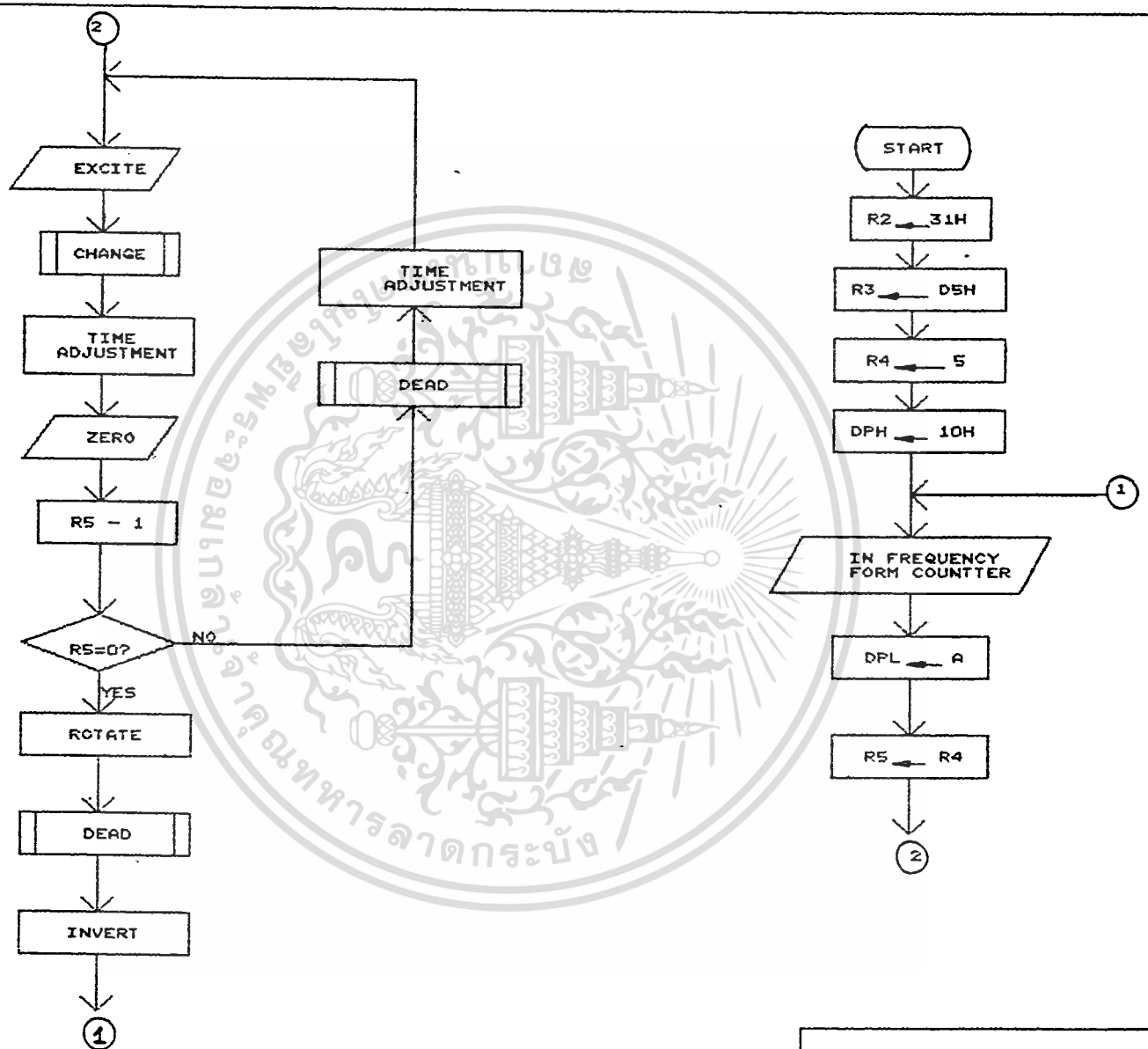
วงจรชุดควบคุมความเร็ว และ สร้างสัญญาณ PWM

วงจรสร้างสัญญาณ PWM และควบคุมความเร็วมอเตอร์ จะแสดงไว้หลังรายละเอียดที่จะกล่าวต่อไปนี้

การสร้างสัญญาณ PWM ใน project นี้เราใช้ ตัว microprocessor เป็นตัวสร้างสัญญาณ โดยที่ CPU จะรับ prarameter ที่สามารถเปลี่ยนความถี่ของรูปคลื่นซาย์ ที่ได้จากสัญญาณ pwm จาก counter up-down การสร้างสัญญาณ PWM ทำได้โดยการเขียน program โดยทำให้สัญญาณที่ได้ออกมามีมุมต่างกัน 120° ทางไฟฟ้า

สำหรับ flowchart ของ program สร้าง PWM ของ inverter เป็นดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



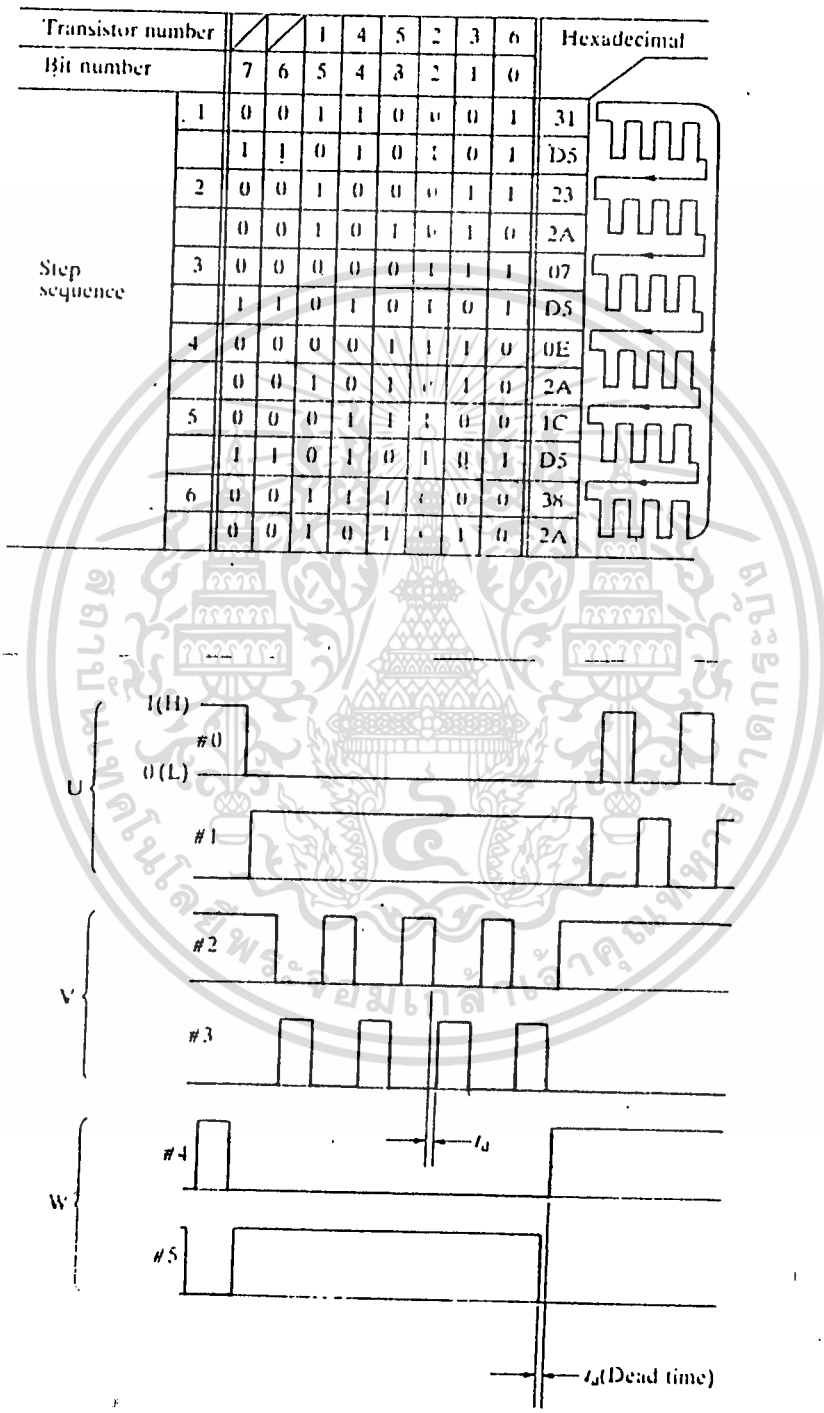
Size	Document Number	REV
A		
Date: April 3, 1991	Sheet of	

หลักการทำงานของ program

- เริ่มจากนำค่า 31H เก็บใน R2 ค่า 31H คือค่าสภาวะ แรก ของ switching
- นำค่า 5 เก็บใน R4 (R4 จะเป็นตัวกำหนดจำนวน พัลซ์)
- นำค่า D5H เก็บใน R3 ค่า D5H นี้จะเป็น code switch สภาวะที่สองของ switching เมื่อส่งค่านี้ให้กับ inverter Tr1, Tr3 และ Tr5 จะ off ส่วน Tr2,Tr4 และ Tr6 จะ on ดังนั้นก็จะทำให้เกิดจำพัลซ์ ใน 1/6 ของ หนึ่ง cycle
- รับ prarameter ที่สามารถ เปลี่ยนความถี่ของสัญญาณ out put จาก counter และนำค่าที่ได้ไปหน่วงเวลา ให้กับความกว้าง พัลซ์
- นำค่าใน R4 มาเก็บใน R5
- ต่อมาส่ง code data switch ให้ out put ออกเป็น 0v
- ลด R5 ลงมาทีละ 1 แล้วตรวจดูว่า เป็น 0 หรือไม่ถ้าไม่เป็น 0 ให้กลับไปสร้าง dead time ในการ switch แล้วกลับไปทำกระบวนการเดิมจนกว่าจะครบจำนวนพัลซ์ ที่กำหนดไว้ใน R4
- จากนั้นเมื่อครบจำนวนพัลซ์ก็ มีการ ROTATE เพื่อให้เกิดสัญญาณ 3 เฟสต่อจากนั้น ก็เป็นกระบวนการ กลับรูปคลื่นเดิมให้เป็นตรงข้าม และสร้าง dead time กลับไปรับ prarameter มาใหม่ เป็นอย่างนี้ตลอดไป

รายละเอียดดูได้จากตาราง และรูปคลื่นที่ได้แสดงไว้ในรูปที่ 26

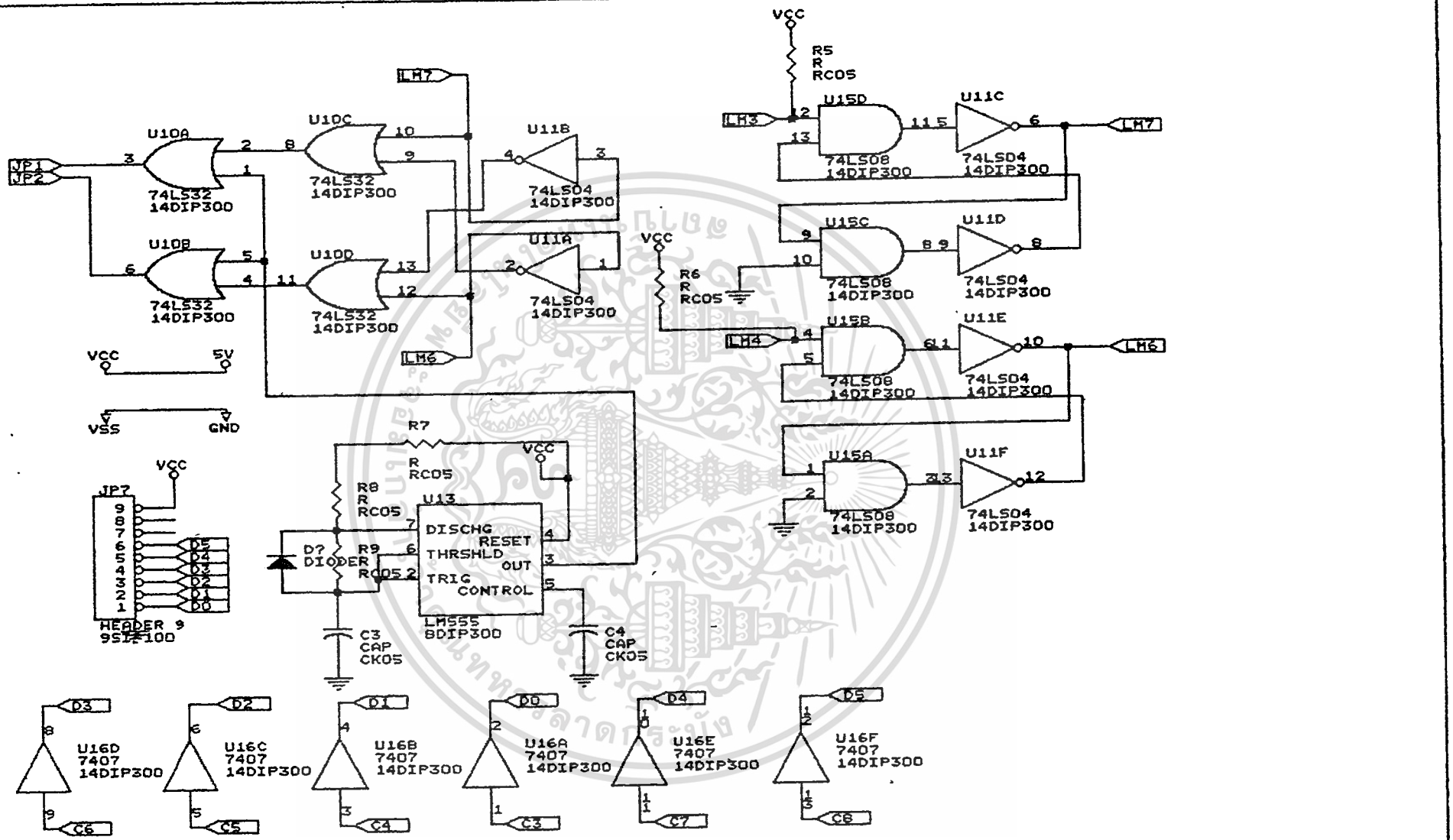
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



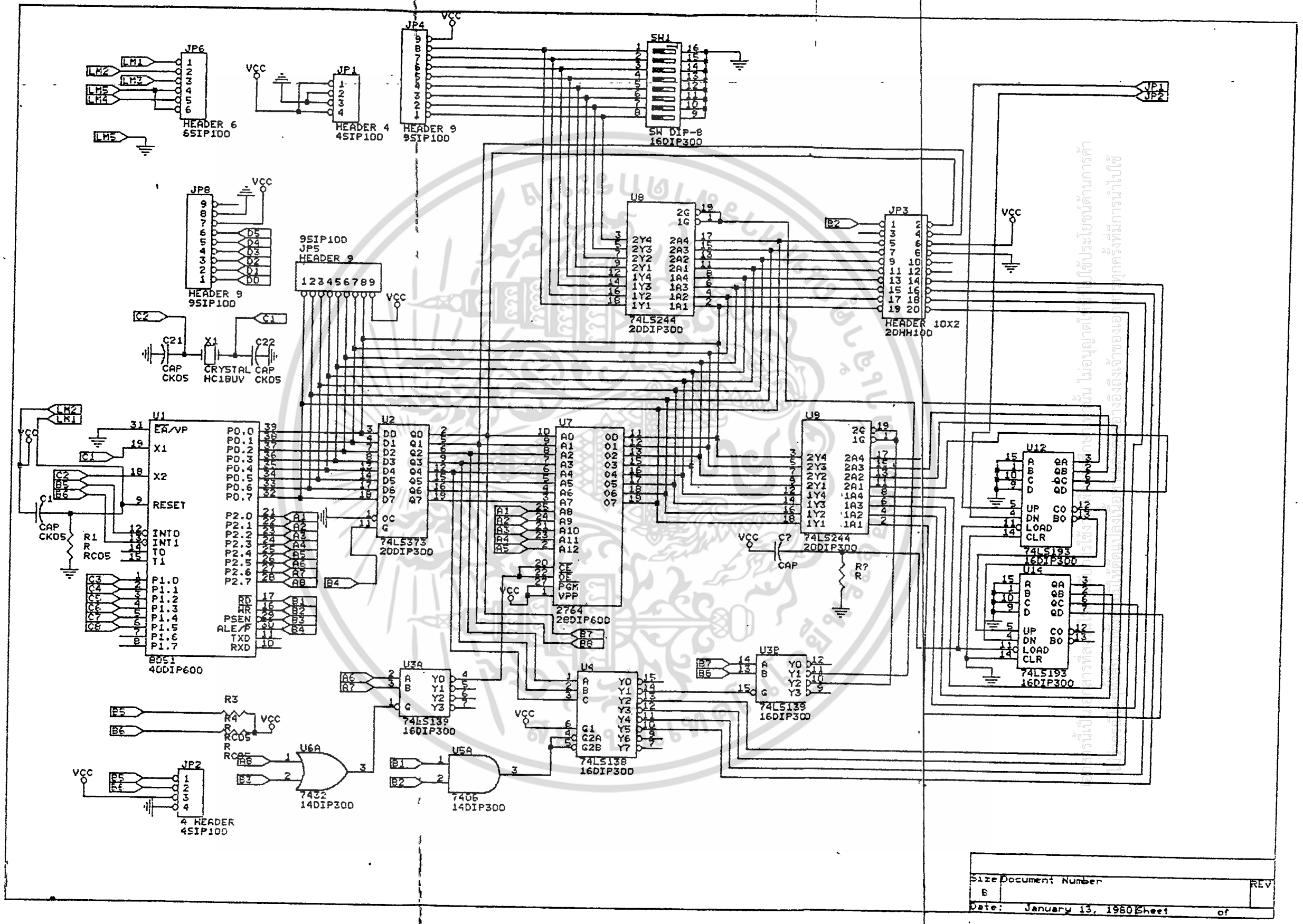
Transistor number				1	4	5	2	3	6	Hexadecimal
Bit number		7	6	5	4	3	2	1	0	
Step sequence	1	0	0	1	1	0	0	0	1	31
		1	1	0	1	0	1	0	1	D5
	2	0	0	1	0	0	0	1	1	23
		0	0	1	0	1	0	1	0	2A
	3	0	0	0	0	0	1	1	1	07
		1	1	0	1	0	1	0	1	D5
	4	0	0	0	0	1	1	1	0	0E
		0	0	1	0	1	0	1	0	2A
	5	0	0	0	1	1	1	0	0	1C
		1	1	0	1	0	1	0	1	D5
	6	0	0	1	1	1	1	0	0	38
		0	0	1	0	1	0	1	0	2A

รูปที่ 26 แสดงตารางและรูปคลื่นของการ Switching

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



Size	Document Number	REV
A		
Date: January 13, 1980	Sheet of	


JP6
LM1
LM2
LM3
LM5
LM4
HEADER 6
6SIP100
JP1
HEADER 4
4SIP100
JP4
HEADER 9
9SIP100
VCC
SW1
SW DIP-8
16DIP300
JP8
D5
D4
D3
D2
D1
D0
HEADER 9
9SIP100
9SIP100
JP5
HEADER 9
123456789
U8
74LS244
20DIP300
JP3
B2
HEADER 10X2
20HH100
C2
C1
C21
X1
C22
CAP
CK05
CRYSTAL
HC18UV
U1
8051
40DIP600
U2
74LS373
20DIP300
U7
2764
28DIP600
U9
74LS244
20DIP300
U12
74LS193
16DIP300
U14
74LS193
16DIP300
R1
R
RC05
C3
C4
C5
C6
C7
C8
B4
B7
B8
U3A
74LS139
16DIP300
U4
74LS138
16DIP300
U3P
74LS139
16DIP300
R3
R4
B5
B6
U6A
7432
14DIP300
U5A
7406
14DIP300
JP2
4 HEADER
4SIP100
C?
R?
Size
B
Document Number
REV
Date: January 13, 1980
Sheet
of

3.2 วงจรกำลังของพัลส์วิดท์มอดดูเลชั่น

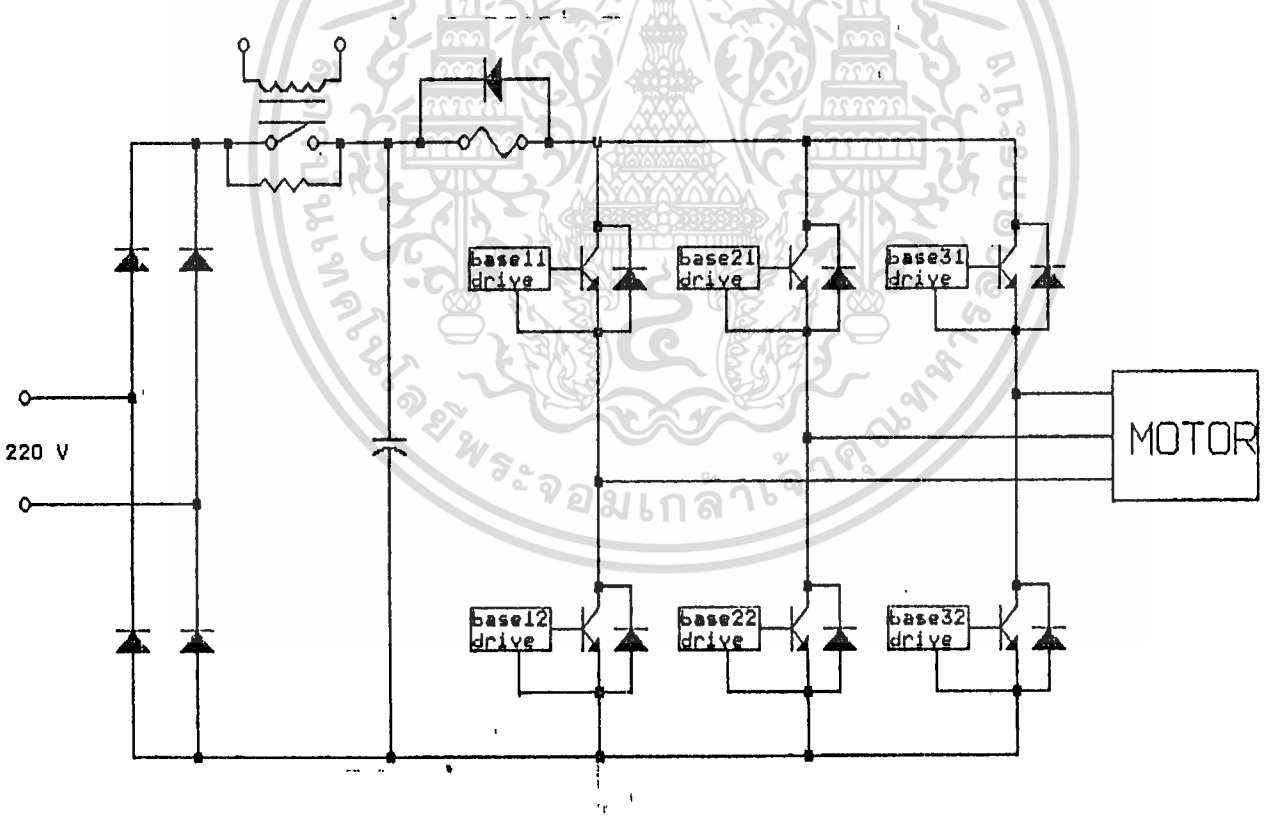
วงจรกำลังประกอบด้วยวงจรต่างๆ ดังนี้

-วงจรกำลังหลัก

-วงจรอินเวอร์เตอร์

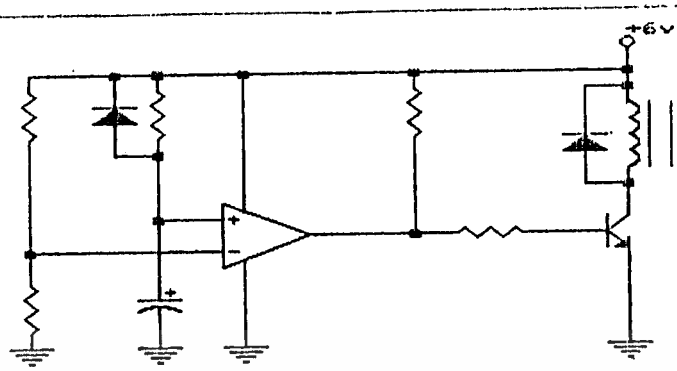
-วงจรจ่ายแรงดันไฟต่ำและหม้อแปลงไฟฟ้า

ส่วนของวงจรกำลังหลักประกอบด้วยวงจรเร็คติไฟเออร์ ทำหน้าที่แปลงไฟสลับให้เป็นไฟตรง จากนั้นผ่านวงจรกรองแรงดันไฟฟ้าและจ่ายให้แก่วงจรทรานซิสเตอร์อินเวอร์เตอร์ ดังแสดงในรูปที่ 27 โดยก่อนที่จะผ่านวงจรกรองแรงดันไฟฟ้าเราจะให้ผ่านตัวต้านทานก่อน เพื่อจำกัดกระแสเริ่มไหลของคอนเด็นเซ็อร์ โดยใช้วงจรหน่วงเวลาการทำงานของรีเลย์ได้แสดงไว้ดังในรูปที่ 28



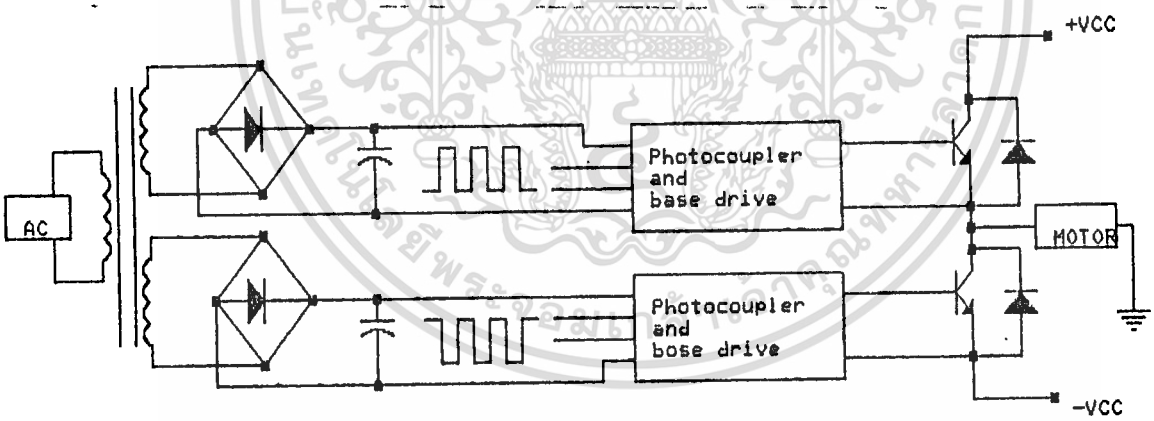
รูปที่ 27 วงจรกำลังหลัก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 28 วงจรหน่วงการทำงานของรีเลย์

วงจรอินเวอร์เตอร์ประกอบด้วย / วงจรรวมในภาคกำลังกับภาคขับเบสเข้าด้วยกัน ดังแสดงที่ 29 ในรูปเป็นวงจรกำลังของอินเวอร์เตอร์ 1ϕ



รูปที่ 29 วงจรทรานซิสเตอร์อินเวอร์เตอร์

<u>วงจรขับเบส</u>

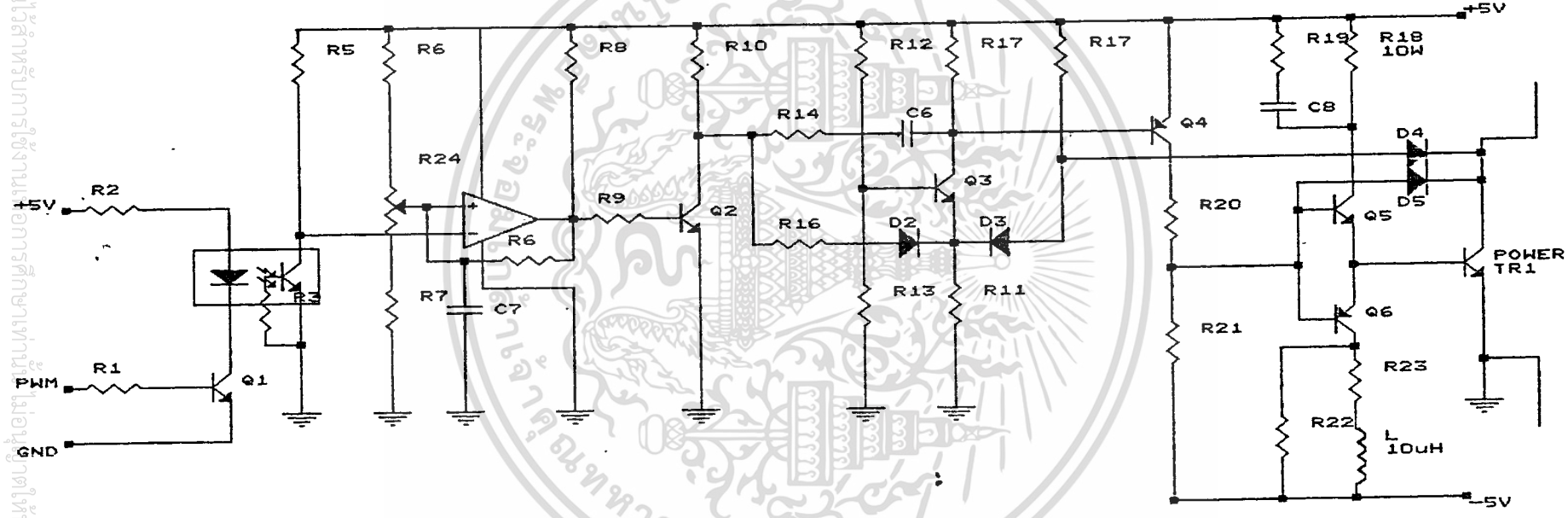
สิ่งที่ต้องพิจารณาถึงในการออกแบบวงจรขับเบสก็คือ อัตราการขยาย(h_{fe}) ของทรานซิสเตอร์กำลัง แรงดันอิ่มตัวของทรานซิสเตอร์(Vce set)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

และความเร็วในการสวิทช์ ในการไบอัสทรานซิสเตอร์นั้น ถ้ากระแสที่เราไบอัสน้อยเกินไป จะทำให้ทรานซิสเตอร์ทำงานในย่านแอคทีฟ ซึ่งอาจทำความเสียหายแก่ทรานซิสเตอร์กำลังได้ แต่ถ้ากระแสไบอัสมากเกินไป ก็อาจทำให้เกิดความสูญเสีย(loss) มากขึ้นเพราะช่วงหยุดนำกระแสของทรานซิสเตอร์นานขึ้น และอาจมีผลต่อความเร็วในการสวิทช์ด้วย ดังนั้นในกรณีการใช้งานที่ความถี่สูงๆเราจะต้องให้เวลาในการที่ทรานซิสเตอร์หยุดทำงานสั้นลง โดยการให้ไบอัสกลับ

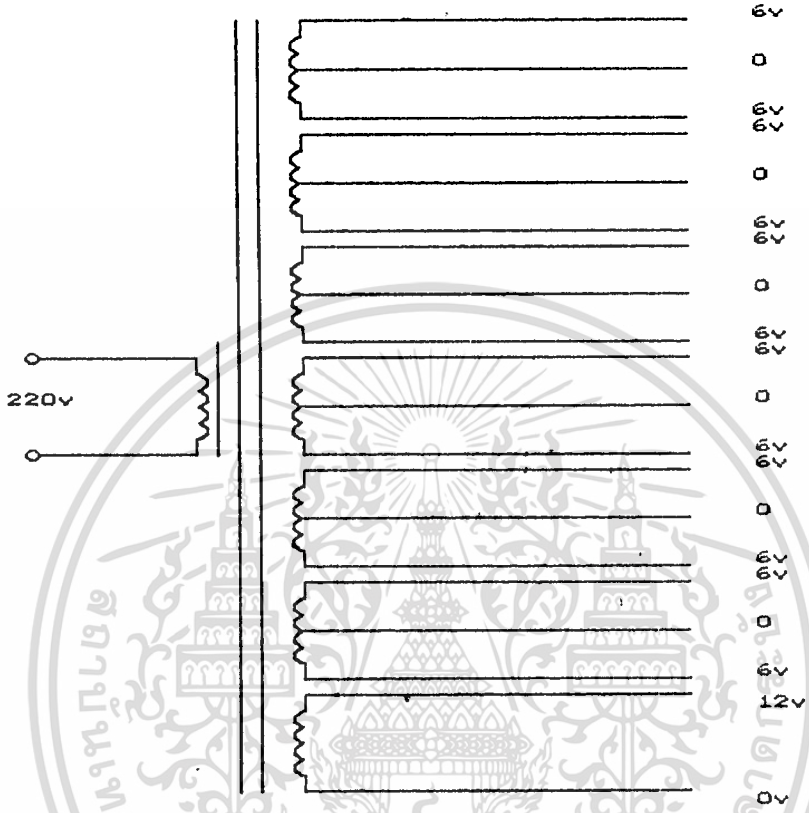
จากวงจร เรากำหนดให้ D_4 เป็นตัวตรวจสอบการทำงานของ Power Transistor และจะทำงานร่วมกับ D_3 และ R_{17} ซึ่งในการทำงานปกติ Vce ของ Transistor จะมีค่าน้อยมาก แรงเคลื่อนที่ขั้ว Anode ของ D_4 มากกว่า cathode เมื่อมีการเกิด over current จะทำให้ V_{ce} ของ transistor มีศักย์ไฟฟ้าสูงขึ้นและสูงมากกว่า V_b ของ Q_3 ทำให้กระแสจาก R_{17} เปลี่ยนทิศทางมาไหลผ่าน D_3 แทน ทำให้ V_c สูงกว่า V_b Q_3ไม่สามารถทำงานได้ Q_4 ก็ไม่สามารถทำงานได้เช่นกัน มีผลทำให้ขา base ของ Power Transistor เป็น LOW Power Transistor จะไม่ทำงาน การกำหนดกระแสขึ้นอยู่กับ R_{16} ซึ่งเป็นตัวกำหนดกระแสเบสให้แก่ Power Tr เมื่อ Power Tr ต้องการกระแสเบสมากกว่าที่กำหนด จะทำให้ Power Tr ทำงานในย่านแอคทีฟ Vce ของ Power Tr จะสูงกว่า V_b ของ Q_3 Power Tr ก็จะหยุดการทำงาน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



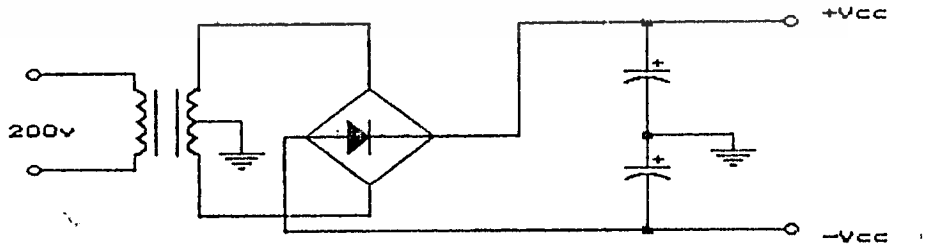
Size	Document Number	REV
A		
Date: October 2, 2049	Sheet of	

วงจรจ่ายแรงดันต่ำและหม้อแปลง



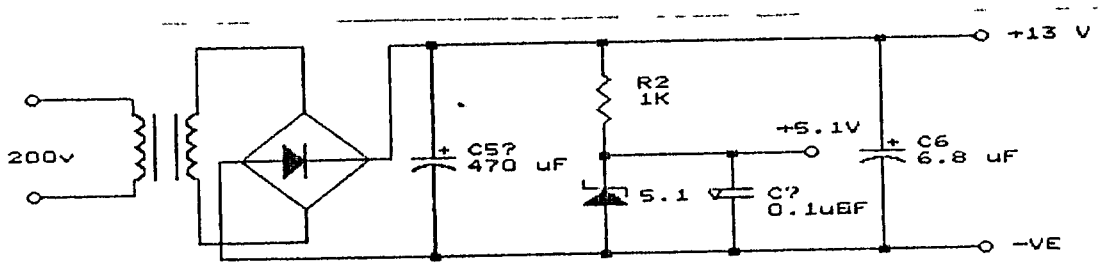
รูปที่ 30 แสดงวงจรหม้อแปลง

ในส่วนของวงจรควบคุม เราต้องสร้างแหล่งจ่ายไฟฟ้าระดับตรงที่มีระดับต่างๆกัน มี +6v,-6v ใช้สำหรับเลี้ยงวงจรขับเบสเพาเวอร์ทรานซิสเตอร์ +13v ใช้สำหรับเลี้ยงวงจรป้องกันอินเวอร์เตอร์ตามรูปที่ 31



รูปที่ 31 แสดงวงจรไฟเลี้ยงขับเบส

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 32 แสดงวงจรไฟเลี้ยงแก่วงจรป้องกัน

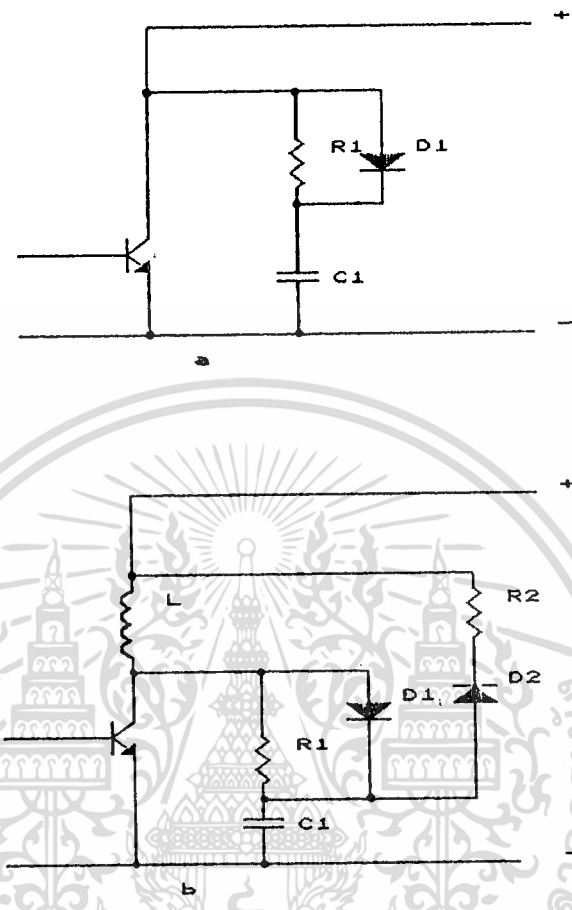
การใช้ทรานซิสเตอร์สำหรับวงจรขับมอเตอร์กระแสสลับ

สวิทช์ชิ่งทรานซิสเตอร์ส่วนมากใช้กับระบบ พัลส์วิดท์มอดูเลชั่นอินเวอร์เตอร์ค่าพลังงานสูงสุดประมาณ 200 กิโลวัตต์ ถ้ามากกว่านี้นิยมใช้ จีทีโอจะดีกว่า ส่วนสวิทช์ชิ่งทรานซิสเตอร์จะเหมาะกับพวกที่เป็นแหล่งจ่ายกำเนิดแรงดันไฟฟ้าและจำเป็นจะต้องมีการต่อไดโอดย้อนกลับคร่อมไว้เพื่อให้กระแสสามารถไหลย้อนกลับได้ขณะเกิดการรีเจนเนอเรทีฟ

การป้องกันการสวิทช์ชิ่ง

เมื่อนำเอาทรานซิสเตอร์มาใช้ในวงจรขับมอเตอร์ โดยขณะมอเตอร์เริ่มทำงาน กระแสจะถูกเปลี่ยนจากการไหลผ่านไดโอดมาผ่านทรานซิสเตอร์ และจากคุณสมบัติของไดโอดทำให้เป็นปัญหาต่อทรานซิเตอร์ได้ โดยในขณะไดโอดหยุดทำงานกระแสย้อนกลับของไดโอด จะทำให้เกิดกระแสสูงไหลผ่านทรานซิสเตอร์ ถ้าไดโอดมีช่วงหยุดการทำงานช้า จะทำให้ทรานซิสเตอร์มีค่าเกินย่านความปลอดภัยของทรานซิสเตอร์ได้ โดยขณะเริ่มทำงานค่า V_{cc} ควรมีค่าลดลงอย่างรวดเร็วและกระแสคอลเล็คเตอร์เพิ่มขึ้นอย่างช้าๆ ขณะหยุดทำงานกระแสคอลเล็คเตอร์เพิ่มขึ้นอย่างช้าๆ ขณะหยุดทำงานกระแสคอลเล็คเตอร์ควรลดลงอย่างรวดเร็วและแรงดันไฟฟ้าควรเพิ่มขึ้นอย่างช้าๆ ดังนั้นจึงมีวงจรช่วยในการสวิทช์ ดังแสดงในรูปที่ 33

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 33 วงจรช่วยในการสวิทช์ชิ่งของทรานซิสเตอร์

จากรูปที่ 33(a) นั้นจะทำให้กระแสในขณะหยุดการทำงานของทรานซิสเตอร์มีค่าลดลงอย่างรวดเร็วโดยไหลผ่านไดโอด (D_1) ไปยังตัวเก็บประจุ (C_1) ในขณะที่แรงดันไฟฟ้าเพิ่ม ตัวต้านทาน (R_1) จำกัดการคายประจุของตัวเก็บประจุ ไปยังทรานซิสเตอร์ขณะเริ่มทำงาน

จากรูปที่ 33(b) ตัวอินดัคแตนซ์ (L) ช่วยลดอัตราการเพิ่มของกระแสในขณะที่ทรานซิสเตอร์เริ่มทำงาน และค่า R_2 และ D_2 จะป้องกันค่าแรงดันไฟฟ้าขณะเริ่มสูงอันเนื่องมาจาก snapping off ของกระแสใน D_1 และ L ในขณะหยุดทำงาน

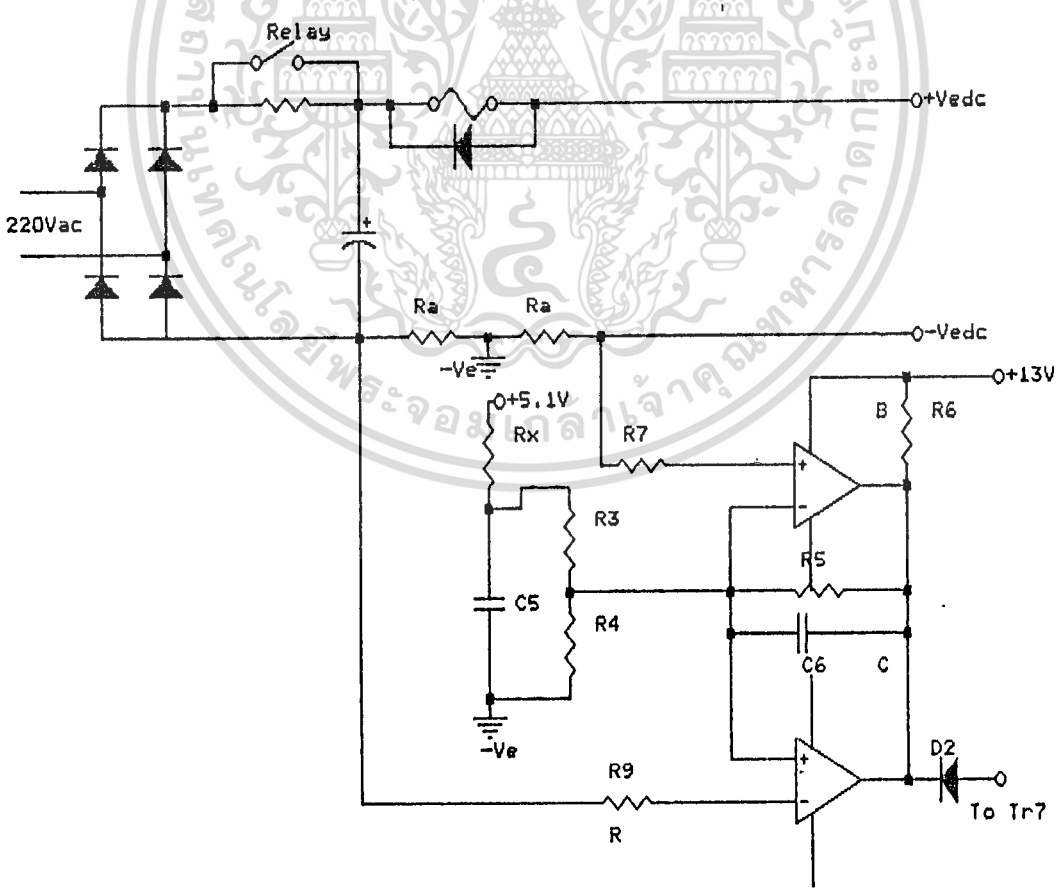
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การป้องกันความเสียหายแก่อินเวอร์เตอร์ (Fault Protection)

การป้องกันความเสียหายที่อาจจะเกิดขึ้นในขณะปฏิบัติงานซึ่งเป็นการป้องกันอินเวอร์เตอร์ก่อนที่จะเกิดความเสียหาย วงจรป้องกันการเสียหายต่างๆมีดังนี้

วงจรป้องกันกระแสเกิน

ขณะที่อินเวอร์เตอร์ทำงานปกตินั้นค่ากระแสที่ไหล ในวงจรกำลังจะถูก จำกัดโดยค่ากระแสที่เกิดจากตัวมอเตอร์ แต่ถ้าอินเวอร์เตอร์เกิดทำงานผิดปกติขึ้น มาจะทำให้วงจรจ่ายกระแสเพิ่มมากขึ้น ซึ่งจะทำให้ทรานซิสเตอร์เกิดการเสียหายได้จึงจำเป็นต้องมีวงจรป้องกันกระแสเกิน หลักการวัดกระแสนั้น ทำได้โดยการวัดค่าแรงดันไฟฟ้าตกคร่อมตัวต้านทานเทียบกับแรงดันอ้างอิง โดยการใช้ LM 339 เป็นตัวเปรียบเทียบ(comparator) ดังรูปที่ 34.



รูปที่ 34 วงจรป้องกันกระแสเกิน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในขณะสภาวะปกติเมื่อกระแสในวงจรหลักมีค่าต่ำกว่าที่กำหนด ก็จะให้แรงดันไฟฟ้าที่จุด X และ Y ในรูปมีค่าต่ำกว่าแรงดันไฟฟ้าที่จุด Z ก็จะได้แรงดันไฟฟ้าที่เอาท์พุทของตัวเปรียบเทียบ B และตัว C เป็น $+V_{sat}$ ซึ่งประมาณ $+V_{cc}$ หรือ +13v และเมื่อกระแสในวงจรหลักมีค่าเกินกว่าที่กำหนดก็จะทำให้แรงดันไฟฟ้าที่จุด X มีค่ามากกว่าแรงดันไฟฟ้าที่จุด Z ทำให้ค่าแรงดันไฟฟ้าที่เอาท์พุทของตัวเปรียบเทียบ B มีค่าเปลี่ยนแปลง ซึ่งจะเป็นสัญญาที่จะนำไปใช้ต่อไป

จากการกำหนดค่าของกระแสในวงจรหลักไม่เกิน 3 แอมป์ สามารถคำนวณหาค่าแรงดันไฟฟ้าอ้างอิงที่จุด Z ได้เท่ากับ $R*3 = V_{ref}$ ซึ่งคำนวณหาค่า R_x ได้จากกฎของ KCL ดังนี้

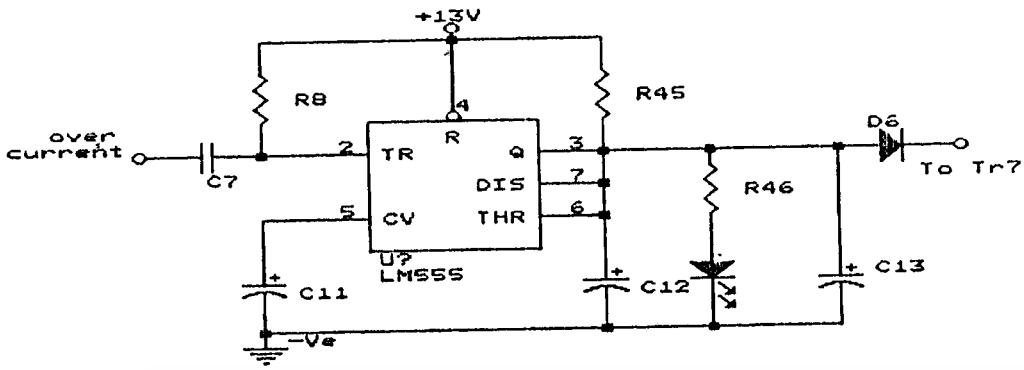
$$\frac{5.1-V_{ref}}{R_x + R_3} + \frac{+V_{sat} - V_{ref}}{R_5} = \frac{V_{ref}}{R_4}$$

ค่า R_x นี้เป็นตัวต้านทานที่สามารถปรับค่าได้ตามความต้องการที่จะให้กระแสในวงจรหลักมีค่าได้

สำหรับตัวเปรียบเทียบ C มีหลักการทำงานเหมือนตัว B เพียงแต่เพื่อต้องการป้องกันกระแสที่ไหลมาอีกทิศทางหนึ่งเท่านั้น

เมื่อเกิดกระแสไหลผ่านวงจรหลัก ก็จะทำให้แรงดันไฟฟ้าเอาท์พุทของตัวเปรียบเทียบเกิดการเปลี่ยนสถานะในขอบขาลง ซึ่งจะเป็นอินพุทให้กับวงจรโมโนสเตเบิล (Monostable) และจะให้สัญญานเอาท์พุท ออกมา หนึ่งลูกคลื่น ซึ่งมีช่วงเวลาเท่ากับ $t= 1.1\ R_{45}\ C_{12}$ ดังรูปที่ 35

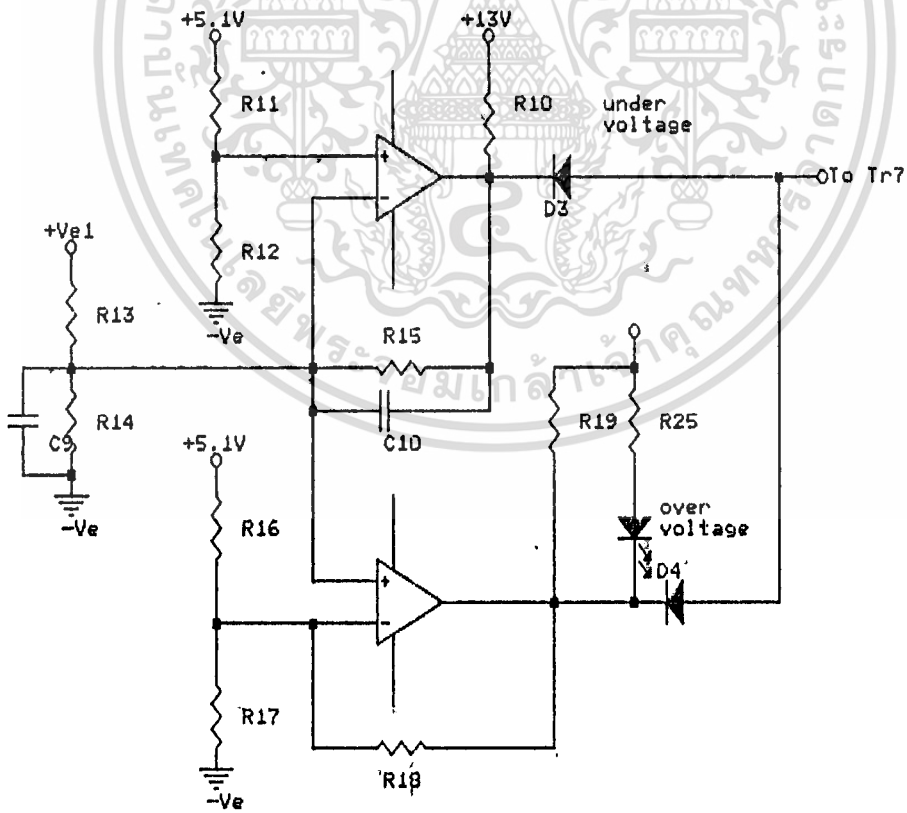
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 35 วงจรโมโนสเตเบิล

วงจรป้องแรงดันสูงและแรงดันต่ำ

วงจรป้องกันแรงดันไฟฟ้านี้ จะใช้ตัวคอมพาราเตอร์จำนวนสองตัวเปรียบเทียบระหว่าง แรงดันไฟฟ้าอ้างอิงกับแรงดันไฟฟ้าที่วัดจากวงจรหลักแสดงดังรูปที่ 34 ซึ่งเป็นลักษณะของวงจรเปรียบเทียบแบบวินโดว์(window circuit)



รูปที่ 36 วงจรป้องกันไฟสูงเกิน-ต่ำเกิน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ค่าแรงดันไฟฟ้าที่จุด Q ขณะตัวคอมพาราเตอร์ตัว A มีค่าเอาท์พุทเป็น $+V_{sat}$ หาได้จาก

$$(V_Q - 5.1)/R_{16} + V_Q - (+V_{sat}))/R_{16} = 0$$

จากค่า V_Q ที่หาได้ทำการเทียบค่า $+V_{s1}$ ได้จากสมการดังนี้

$$[(V_Q/R_{14}) - (13 - V_Q)/_{15}] * R_{13} = +V_{s1} - V_Q$$

ค่าแรงดันไฟฟ้าที่จุด Q ขณะคอมพาราเตอร์ A มีค่าเอาท์พุทเป็น 0v หาได้จาก

$$5.1 * (R_{17}/R_{18})/(R_{17}/R_{18})+R_{16} = V_Q$$

ดังนั้นเมื่อเทียบค่า $+V_{s2}$ ได้

$$[(V_Q/R_{14}) - (13-V_Q)/R_{15}]*R_{13} = +V_{s2} - V_Q$$

ดังนั้นวงจรป้องกันแรงดันไฟฟ้าสูงเกิน เมื่อค่า $+V_s$ มีค่าเกิน $+V_{s1}$ จะทำให้คอมพาราเตอร์ตัว A ให้เอาทพุทออกมาเป็น 0v ทำให้ไดโอดเปลงแสงทำงาน แสดงผลของการเกิดแรงดันไฟฟ้าเกิน ไดโอดเปล่งแสงจะทำงานจนค่า $+V_{s1}$ มีค่าลดลงจนต่ำกว่า $+V_{s2}$ จะหยุดทำงาน ซึ่งจะมีฮีทเตอร์รีซิส (Hyteresis) ระหว่าง $+V_{s1}$ ถึง $+V_{s2}$ สัญญานเอาท์พุทของคอมพาราเตอร์เมื่อเกิดแรงดันไฟฟ้าสูงเกินมอเตอร์จะต้องหยุดการทำงาน

ค่าแรงดันไฟฟ้าที่เกินนี้คิดเป็นเปอร์เซ็นต์ได้ดังนี้ เมื่อคิดให้มอเตอร์ทำงานปกติ ที่ 220ac ดังนั้นค่าแรงดันไฟฟ้าที่เกินเท่ากับ

$(311-(V_s*100/311) = X\ \%$

สำหรับส่วนวงจรป้องกันแรงดันต่ำเกิน ค่าแรงดันไฟฟ้าที่จุด P มีค่าเท่ากับ $5.1*R_{12}/(R_{11}+R_{12}) = V_p$ ซึ่งมีค่าเท่ากับ $+V_s$ ดังนั้น

$$[(V_p/R_{14} - (13-V_p)/R_{15}]*R_{13} = +V_{s3} - V_p$$

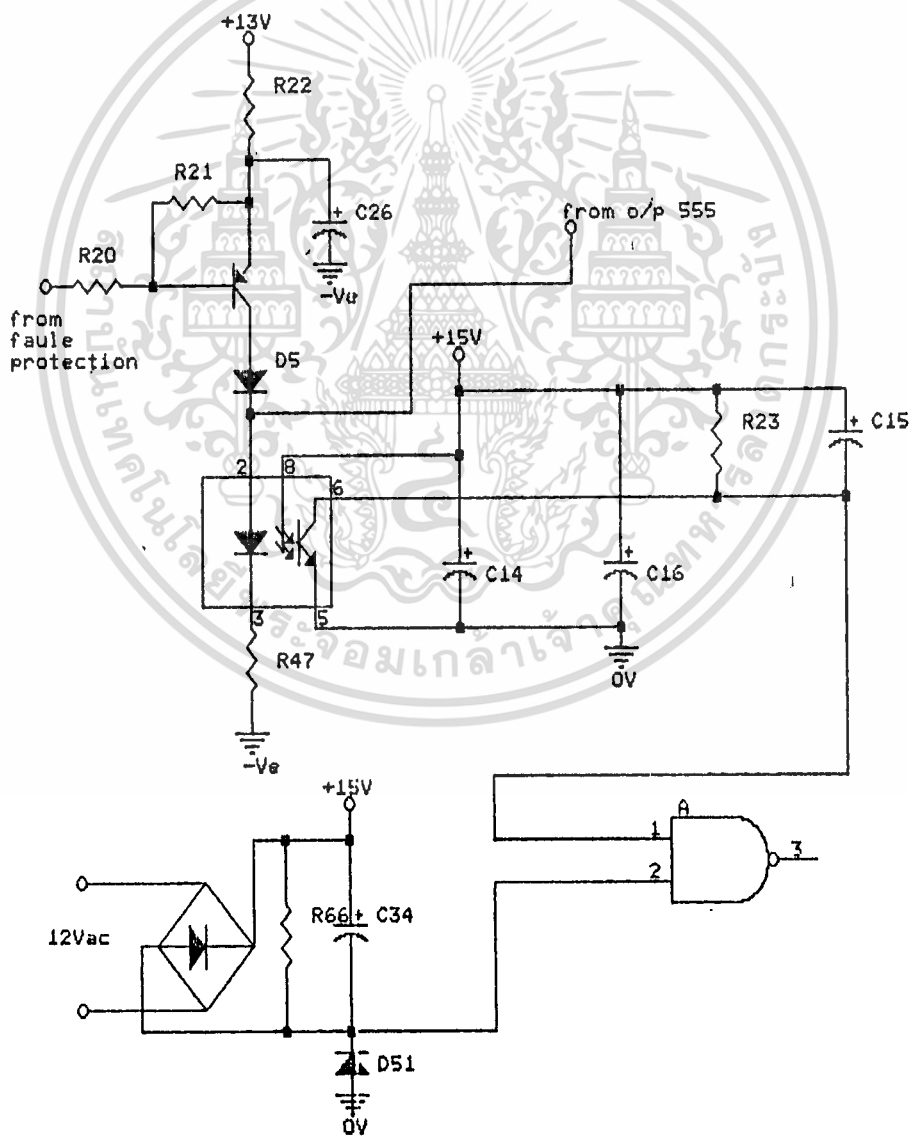
ซึ่งหมายความว่า เมื่อแรงดันไฟฟ้า $+V_s$ มีค่าต่ำกว่า $+V_{s3}$ มอเตอร์จะหยิดการทำงานที่ $(311-(+V_{s3})*100/311 = Y\%$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

วงจรแยกกราวด์เพื่อความปลอดภัย

เอาท์พุทของคอมพาราเตอร์ ที่ได้จากวงจรป้องกันกระแสเกิน วงจรป้องกันแรงดันไฟนี้จะนำมาขับทรานซิสเตอร์ เพื่อให้ออปโตคัปเปิลทำงานอีกที โดยทำหน้าที่แยกกราวด์ระหว่างตัวตรวจจับความผิดปกติซึ่งมีกราวด์ร่วมกับวงจรกำลัง และควบคุมการทำงานของมอเตอร์

เมื่อเกิดความผิดปกติขึ้นขา 3 ของแนนเกทจะเป็น 1 ส่งไปยังส่วนรับค่าสัญญาณผิดปกติ สำหรับขา 2 นั้นทำการต่อกับส่วนตรวจจับกระแสเกินที่มอเตอร์ซึ่งทำเผื่อไว้ โดยปกติขานี้มีค่าเป็น 1 และเมื่อเกิดความผิดปกติขึ้นที่กระแสในมอเตอร์จะทำให้ขา 2 เป็น 0



รูปที่ 37 วงจรแยกกราวด์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 4
ผลการทดลอง

ขั้นตอนการทดลอง

-ทำการต่อวงจรทางด้านเพาเวอร์ เข้ากับชุดสร้างสัญญานพัลซ์วิดท์มอดดูเลชั่น ที่ได้จากไมโครโปรเซสเซอร์(Mcs-51) จากนั้นค่อยๆ ปรับแรงเคลื่อนไฟฟ้าให้แก่วงจรเพาเวอร์ เพื่อความปลอดภัยแก่เพาเวอร์ทรานซิสเตอร์

-จากนั้นค่อยๆ ปรับความถี่ให้เปลี่ยนแปลงไปเรื่อยๆสังเกตุการหมุนของมอเตอร์

-ทำการวัดแรงเคลื่อนไฟฟ้าเฟส R-S และ S-T

-ทำการวัดกระแสที่เฟส S ที่ความถี่ 25 Hz และ 50 Hz

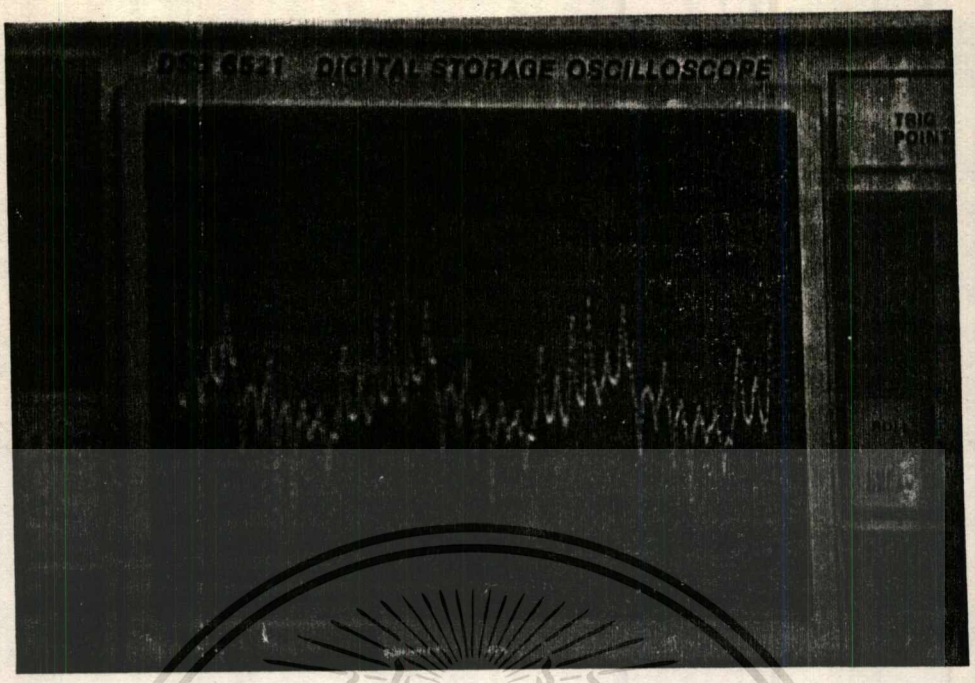
เนื่องจากในการวัดกระแสเราไม่มี current prope เราจำเป็นต้องใช้รีซิสเตอร์ต่ออนุกรมกับเอาท์พุท เพื่อที่จะดูการเปลี่ยนแปลงของแรงเคลื่อนที่เฟสนั้นๆ

รูปที่ 36 สัญญานพัลซ์ที่ออกจากไมโครโปรเซสเซอร์

ด้านบน input Tr1 ด้านล่าง input Tr2

Voltage per devition = 5V Time per devition = 10 ms

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



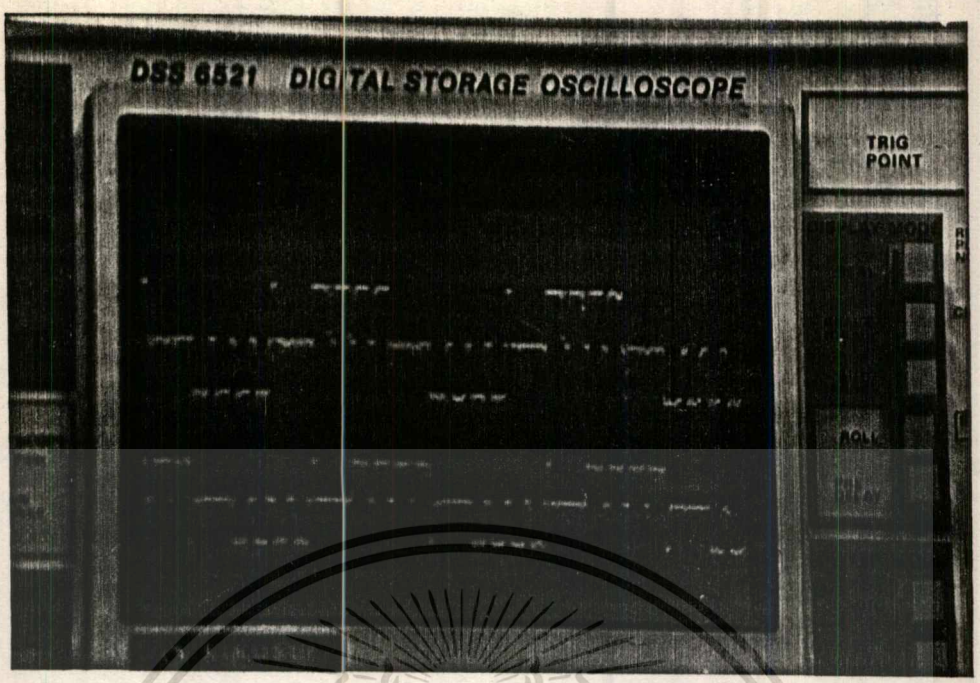
รูปที่ 37 กระแสเฟส S ที่ 25 Hz

Voltage per devition = 1V Time per devition = 10 ms

รูปที่ 38 กระแสเฟส S ที่ 50 Hz

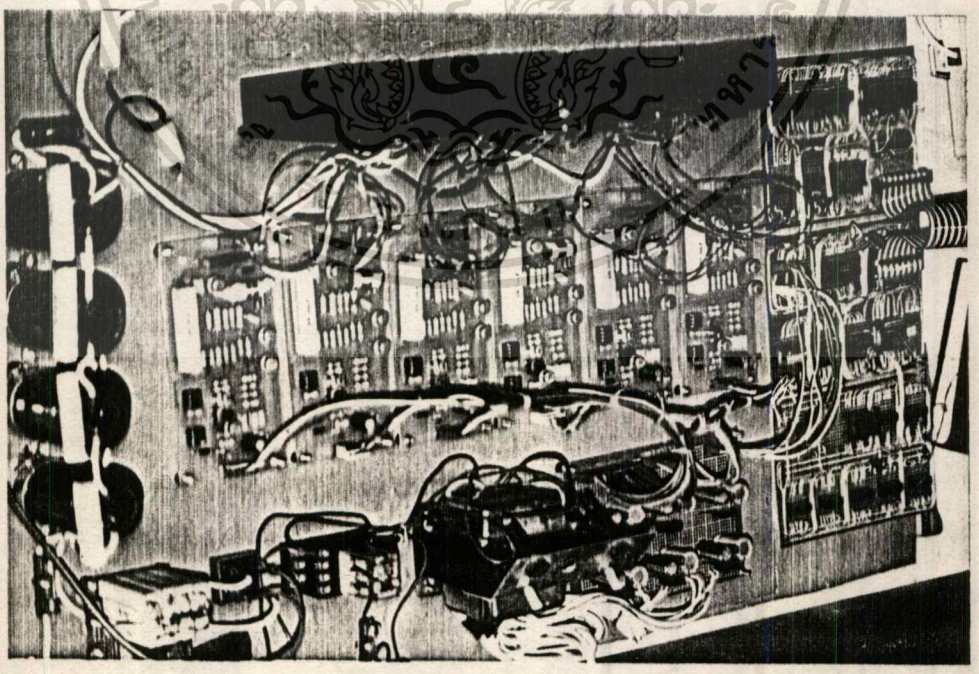
Voltage per devition = 1V Time per devition = 10 ms

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 39 สัญญานแรงเคลื่อนเอาท์พุท
ด้านบนเป็นเฟส R-S ด้านล่างเป็นเฟส S-T

Voltage per devition = 10V Time per devition = 10 ms



รูปที่ 40 ส่วนประกอบของโครงงานนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 5
บทสรุปและวิจารณ์

ในการสร้างวงจรพัลซ์วิดท์มอดดูเลชั่นอินเวอร์เตอร์นั้น ส่วนสำคัญจะอยู่ที่สัญญาณที่จะนำมาขับเบสของทรานซิสเตอร์ ช่วงเวลาในการทำงานและหยุดการทำงานของทรานซิสเตอร์จะต้องเร็ว เพื่อลดค่า $d_{i/dt}$ และ $d_{v/dt}$ ในขณะเริ่มทำงานและหยุดการทำงานของทรานซิสเตอร์ ซึ่งขึ้นอยู่กับวงจรขับเบสและวงจรสนับเบอร์ เนื่องจากโครงงานนี้ต้องใช้ความละเอียดในการทำงานสูง ทำให้โครงงานนี้ไม่สมบรูณ์เท่าที่ควร การปรับความถี่ให้กับมอเตอร์นี้ ตอนนี้สามารถปรับได้ตั้งแต่ 12 Hz ถึง 55 Hz ซึ่งจุดนี้ขึ้นอยู่กับโปรแกรม ที่จะต้องมีการปรับปรุงอีกต่อไป

ในการออกแบบชุดขับเพาเวอร์ทรานซิสเตอร์นั้นในช่วงแรกๆ นั้นจะมีปัญหาตรงช่วงเวลาเริ่มนำกระแสและ เวลาหยุดนำกระแสทำให้เพาเวอร์ทรานซิสเตอร์เกิดการเสียหายเป็นจำนวนมาก ต่อมาได้มีการปรับปรุงให้ดีขึ้นโดยการใช้ไดโอดป้องกันการอิ่มตัวเกินควรเพื่อลด storage time และใช้คอนเด็นเซอร์ C6, C8 ช่วยในการเร่งกระแสเบสในการเริ่มนำกระแสเพื่อเป็นการลดช่วงเวลาเริ่มการทำงาน และใช้ขดลวดเหนี่ยวนำสำหรับดึงกระแสออกจากขาเบสของทรานซิสเตอร์เป็นการลดช่วงเวลาหยุดการทำงานของทรานซิสเตอร์ วงจรขับเบสชุดนี้ใช้สำหรับขับมอเตอร์เหนี่ยวนำ 3ϕ 1 แรงม้า และสามารถที่จะนำไปใช้กับมอเตอร์ที่มีขนาดแรงม้าสูงๆได้ โดยการปรับปรุงชุดขับเบสและ เปลี่ยนขนาดเพาเวอร์ทรานซิสเตอร์ให้มีขนาดสูงขึ้น การปรับปรุงวงจรขับเบสชุดนี้จะต้องคำนึงถึงกระแสเบสของเพาเวอร์ทรานซิสเตอร์เป็นหลัก เพราะชุดขับเบสชุดนี้มีวงจรจำกัดกระแสเอาท์พุทซึ่งจะเป็นการป้องกันเพาเวอร์ทรานซิสเตอร์ได้

ส่วนโครงงานชิ้นนี้ มีการพัฒนาต่อให้ดีขึ้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ภาคผนวก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MCS®-51 INSTRUCTION SET

Table 10. 8051 Instruction Set Summary

Interrupt Response Time: Refer to Hardware Description Chapter.

Instructions that Affect Flag Settings(1)

Instruction	Flag C	Flag OV	Flag AC	Instruction	Flag C	Flag OV	Flag AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	0	X		ANL C,/bit	X		
DIV	0	X		ORL C,bit	X		
DA	X			ORL C,bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

(1)Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

Note on instruction set and addressing modes:

Rn — Register R7–R0 of the currently selected Register Bank.

direct — 8-bit internal data location's address. This could be an Internal Data RAM location (0–127) or a SFR [i.e., I/O port, control register, status register, etc. (128–255)].

@Ri — 8-bit internal data RAM location (0–255) addressed indirectly through register R1 or R0.

#data — 8-bit constant included in instruction.

#data 16 — 16-bit constant included in instruction.

addr 16 — 16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K-byte Program Memory address space.

addr 11 — 11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.

rel — Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is −128 to +127 bytes relative to first byte of the following instruction.

bit — Direct Addressed bit in Internal Data RAM or Special Function Register.

Mnemonic		Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS				
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Add direct byte to Accumulator	2	12
ADD	A,@Ri	Add indirect RAM to Accumulator	1	12
ADD	A,#data	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with Carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with Carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC	A,#data	Add immediate data to Acc with Carry	2	12
SUBB	A,Rn	Subtract Register from Acc with borrow	1	12
SUBB	A,direct	Subtract direct byte from Acc with borrow	2	12
SUBB	A,@Ri	Subtract indirect RAM from ACC with borrow	1	12
SUBB	A,#data	Subtract immediate data from Acc with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12
INC	direct	Increment direct byte	2	12
INC	@Ri	Increment direct RAM	1	12
DEC	A	Decrement Accumulator	1	12
DEC	Rn	Decrement Register	1	12
DEC	direct	Decrement direct byte	2	12
DEC	@Ri	Decrement Indirect RAM	1	12

All mnemonics copyrighted ©Intel Corporation 1980

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS (Continued)				
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A & B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12
LOGICAL OPERATIONS				
ANL	A,Rn	AND Register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND Indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL	A,@Ri	Exclusive-OR Indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12
LOGICAL OPERATIONS (Continued)				
RL	A	Rotate Accumulator Left	1	12
RLC	A	Rotate Accumulator Left through the Carry	1	12
RR	A	Rotate Accumulator Right	1	12
RRC	A	Rotate Accumulator Right through the Carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12
DATA TRANSFER				
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12
MOV	A,#data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	Rn,#data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,#data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12

All mnemonics copyrighted ©Intel Corporation 1980

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
DATA TRANSFER (Continued)				
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri, # data	Move Immediate data to Indirect RAM	2	12
MOV	DPTR, # data16	Load Data Pointer with a 16-bit constant	3	24
MOVC	A,@A + DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVC	A,@A + PC	Move Code byte relative to PC to Acc	1	24
MOVX	A,@Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVX	A,@DPTR	Move External RAM (16-bit addr) to Acc	1	24
MOVX	@Ri,A	Move Acc to External RAM (8-bit addr)	1	24
MOVX	@DPTR,A	Move Acc to External RAM (16-bit addr)	1	24
PUSH	direct	Push direct byte onto stack	2	24
POP	direct	Pop direct byte from stack	2	24
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	12
XCHD	A,@Ri	Exchange low-order Digit Indirect RAM with Acc	1	12

Mnemonic		Description	Byte	Oscillator Period
BOOLEAN VARIABLE MANIPULATION				
CLR	C	Clear Carry	1	12
CLR	bit	Clear direct bit	2	12
SETB	C	Set Carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement Carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to CARRY	2	24
ANL	C,/bit	AND complement of direct bit to Carry	2	24
ORL	C,bit	OR direct bit to Carry	2	24
ORL	C,/bit	OR complement of direct bit to Carry	2	24
MOV	C,bit	Move direct bit to Carry	2	12
MOV	bit,C	Move Carry to direct bit	2	24
JC	rel	Jump if Carry is set	2	24
JNC	rel	Jump if Carry not set	2	24
JB	bit,rel	Jump if direct Bit is set	3	24
JNB	bit,rel	Jump if direct Bit is Not set	3	24
JBC	bit,rel	Jump if direct Bit is set & clear bit	3	24
PROGRAM BRANCHING				
ACALL	addr11	Absolute Subroutine Call	2	24
LCALL	addr16	Long Subroutine Call	3	24
RET		Return from Subroutine	1	24
RETI		Return from Interrupt	1	24
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24

All mnemonics copyrighted ©Intel Corporation 1980

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
PROGRAM BRANCHING (Continued)				
JMP	@A + DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is Zero	2	24
JNZ	rel	Jump if Accumulator is Not Zero	2	24
CJNE	A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24
CJNE	A, #data,rel	Compare immediate to Acc and Jump if Not Equal	3	24
CJNE	Rn, #data,rel	Compare immediate to register and Jump if Not Equal	3	24
CJNE	@Ri, #data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ	Rn,rel	Decrement register and Jump if Not Zero	2	24
DJNZ	direct,rel	Decrement direct byte and Jump if Not Zero	3	24
NOP		No Operation	1	12

All mnemonics copyrighted ©Intel Corporation 1980

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 11. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr, code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr, code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A, #data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr, code addr
31	2	ACALL	code addr
32	1	RETI	
33	1	RLC	A
34	2	ADDC	A, #data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr, #data
44	2	ORL	A, #data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr, #data
54	2	ANL	A, #data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A,@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr,A
63	3	XRL	data addr, #data
64	2	XRL	A, #data
65	2	XRL	A,data addr

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 11. Instruction Opcodes in Hexadecimal Order (Continued)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr, data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0,data addr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data,code addr
B7	3	CJNE	@R1,#data,code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Table 11. Instruction Opcodes in Hexadecimal Order (Continued)

Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

INSTRUCTION DEFINITIONS

ACALL addr11

Function: Absolute Call

Description: ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the Stack Pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, opcode bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

Example: Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345 H. After executing the instruction,

```
ACALL  SUBRTN
```

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

Bytes: 2

Cycles: 2

Encoding:

a10 a9 a8 1	0 0 0 1	a7 a6 a5 a4	a3 a2 a1 a0

Operation: ACALL

$(PC) \leftarrow (PC) + 2$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{7\text{-}0})$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{15\text{-}8})$
$(PC_{10\text{-}0}) \leftarrow$ page address

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ADD A,<src-byte>

Function: Add

Description: ADD adds the byte variable indicated to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occured.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B). The instruction,

```
ADD   A,R0
```

will leave 6DH (01101101B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADD A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	1 r r r

Operation: ADD
(A) ← (A) + (Rn)

ADD A,direct

Bytes: 2

Cycles: 1

Encoding:

0 0 1 0	0 1 0 1	direct address

Operation: ADD
(A) ← (A) + (direct)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ADD A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	0 1 1 i

Operation: ADD
(A) ← (A) + ((Ri))

ADD A,#data

Bytes: 2

Cycles: 1

Encoding:

0 0 1 0	0 1 0 0	immediate data

Operation: ADD
(A) ← (A) + #data

ADDC A,<src-byte>

Function: Add with Carry

Description: ADDC simultaneously adds the byte variable indicated, the carry flag and the Accumulator contents, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occured.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) with the carry flag set. The instruction,

ADDC A,R0

will leave 6EH (01101110B) in the Accumulator with AC cleared and both the Carry flag and OV set to 1.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ADDC A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	1 r r r

Operation: ADDC
$(A) \leftarrow (A) + (C) + (R_n)$

ADDC A,direct

Bytes: 2

Cycles: 1

Encoding:

0 0 1 1	0 1 0 1	direct address

Operation: ADDC
$(A) \leftarrow (A) + (C) + (direct)$

ADDC A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	0 1 1 i

Operation: ADDC
$(A) \leftarrow (A) + (C) + ((R_i))$

ADDC A,#data

Bytes: 2

Cycles: 1

Encoding:

0 0 1 1	0 1 0 0	immediate data

Operation: ADDC
$(A) \leftarrow (A) + (C) + \#data$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

AJMP addr11

Function: Absolute Jump

Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (*after* incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.

Example: The label "JMPADR" is at program memory location 0123H. The instruction,

```
AJMP   JMPADR
```

is at location 0345H and will load the PC with 0123H.

Bytes: 2

Cycles: 2

Encoding:

a10 a9 a8 0	0 0 0 1	a7 a6 a5 a4	a3 a2 a1 a0

Operation: AJMP

$(PC) \leftarrow (PC) + 2$

$(PC_{10\text{-}0}) \leftarrow$ page address

ANL <dest-byte>,<src-byte>

Function: Logical-AND for byte variables

Description: ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: If the Accumulator holds 0C3H (11000011B) and register 0 holds 55H (01010101B) then the instruction,

```
ANL   A,R0
```

will leave 41H (01000001B) in the Accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the Accumulator at run-time. The instruction,

```
ANL   P1,#01110011B
```

will clear bits 7, 3, and 2 of output port 1.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ANL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 0 1	1 r r r

Operation: ANL
(A) ← (A) ∧ (Rn)

ANL A,direct

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 1 0 1	direct address

Operation: ANL
(A) ← (A) ∧ (direct)

ANL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 1 0 1	0 1 1 i

Operation: ANL
(A) ← (A) ∧ ((Ri))

ANL A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 1 0 0	immediate data

Operation: ANL
(A) ← (A) ∧ #data

ANL direct,A

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 0 1 0	direct address

Operation: ANL
(direct) ← (direct) ∧ (A)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ANL direct,#data

Bytes: 3

Cycles: 2

Encoding:

0 1 0 1	0 0 1 1	direct address	immediate data

Operation: ANL
(direct) ← (direct) ∧ #data

ANL C,<src-bit>

Function: Logical-AND for bit variables

Description: If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected.* No other flags are affected.

Only direct addressing is allowed for the source operand.

Example: Set the carry flag if, and only if, P1.0 = 1, ACC. 7 = 1, and OV = 0:

```
MOV   C,P1.0     ;LOAD CARRY WITH INPUT PIN STATE
ANL   C,ACC.7    ;AND CARRY WITH ACCUM. BIT 7
ANL   C,/OV      ;AND WITH INVERSE OF OVERFLOW FLAG
```

ANL C,bit

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 1 0	bit address

Operation: ANL
(C) ← (C) ∧ (bit)

ANL C,/bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 1	0 0 0 0	bit address

Operation: ANL
(C) ← (C) ∧ ¬(bit)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

CJNE <dest-byte>,<src-byte>, rel

Function: Compare and Jump if Not Equal.

Description: CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

The first two operands allow four addressing mode combinations: the Accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The Accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```
        CJNE    R7,#60H, NOT__EQ
;       ...     .....           ; R7 = 60H.
NOT__EQ: JC     REQ__LOW        ; IF R7 < 60H.
;       ...     .....           ; R7 > 60H.
```

sets the carry flag and branches to the instruction at label NOT__EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to Port 1 is also 34H, then the instruction,

```
WAIT:  CJNE   A,P1,WAIT
```

clears the carry flag and continues with the next instruction in sequence, since the Accumulator does equal the data read from P1. (If some other value was being input on P1, the program will loop at this point until the P1 data changes to 34H.)

CJNE A,direct,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 0 1	direct address	rel. address

Operation:

```
(PC) ← (PC) + 3
IF (A) <> (direct)
THEN
        (PC) ← (PC) + relative offset

IF (A) < (direct)
THEN
        (C) ← 1
ELSE
        (C) ← 0
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

CJNE A,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 0 0	immediate data	rel. address

Operation:

```
(PC) ← (PC) + 3
IF (A) <> data
THEN
        (PC) ← (PC) + relative offset

IF (A) < data
THEN
        (C) ← 1
ELSE
        (C) ← 0
```

CJNE Rn,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	1 r r r	immediate data	rel. address

Operation:

```
(PC) ← (PC) + 3
IF (Rn) <> data
THEN
        (PC) ← (PC) + relative offset

IF (Rn) < data
THEN
        (C) ← 1
ELSE
        (C) ← 0
```

CJNE @Ri,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 1 i	immediate data	rel. address

Operation:

```
(PC) ← (PC) + 3
IF ((Ri)) <> data
THEN
        (PC) ← (PC) + relative offset

IF ((Ri)) < data
THEN
        (C) ← 1
ELSE
        (C) ← 0
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

CLR A

Function: Clear Accumulator

Description: The Accumulator is cleared (all bits set on zero). No flags are affected.

Example: The Accumulator contains 5CH (01011100B). The instruction,

CLR A

will leave the Accumulator set to 00H (00000000B).

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	0 1 0 0

Operation:

CLR
(A) ← 0

CLR bit

Function: Clear bit

Description: The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

Example: Port 1 has previously been written with 5DH (01011101B). The instruction,

CLR P1.2

will leave the port set to 59H (01011001B).

CLR C

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 0 1 1

Operation:

CLR
(C) ← 0

CLR bit

Bytes: 2

Cycles: 1

Encoding:

1 1 0 0	0 0 1 0	bit address

Operation:

CLR
(bit) ← 0

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

CPL A

Function: Complement Accumulator

Description: Each bit of the Accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to a zero and vice-versa. No flags are affected.

Example: The Accumulator contains 5CH (01011100B). The instruction,

```
CPL A
```

will leave the Accumulator set to 0A3H (10100011B).

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	0 1 0 0

Operation:

```
CPL
(A) ← ¬(A)
```

CPL bit

Function: Complement bit

Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.

Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

Example: Port 1 has previously been written with 5BH (01011101B). The instruction sequence,

```
CPL P1.1
CPL P1.2
```

will leave the port set to 5BH (01011011B).

CPL C

Bytes: 1

Cycles: 1

Encoding:

1 0 1 1	0 0 1 1

Operation:

```
CPL
(C) ← ¬(C)
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

CPL bit

Bytes: 2

Cycles: 1

Encoding:

1 0 1 1	0 0 1 0	bit address

Operation: CPL
(bit) ← ¬(bit)

DA A

Function: Decimal-adjust Accumulator for Addition

Description: DA A adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If Accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the Accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the Accumulator, depending on initial Accumulator and PSW conditions.

Note: DA A *cannot* simply convert a hexadecimal number in the Accumulator to BCD notation, nor does DA A apply to decimal subtraction.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Example: The Accumulator holds the value 56H (01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set. The instruction sequence.

```
ADDC  A,R3
DA    A
```

will first perform a standard twos-complement binary addition, resulting in the value 0BEH (10111110) in the Accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the Accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56, 67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the Accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,

```
ADD   A,#99H
DA    A
```

will leave the carry set and 29H in the Accumulator, since 30 + 99 = 129. The low-order byte of the sum can be interpreted to mean 30 − 1 = 29.

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 1 0 0

Operation: DA

```
-contents of Accumulator are BCD
IF    [[(A3-0) > 9] V [(AC) = 1]]
      THEN(A3-0) ← (A3-0) + 6
           AND

IF    [[(A7-4) > 9] V [(C) = 1]]
      THEN (A7-4) ← (A7-4) + 6
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DEC byte

Function: Decrement

Description: The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

```
DEC  @R0
DEC  R0
DEC  @R0
```

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

DEC A

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 1 0 0

Operation: DEC
(A) ← (A) − 1

DEC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	1 r r r

Operation: DEC
(Rn) ← (Rn) − 1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DEC direct

Bytes: 2

Cycles: 1

Encoding:

0 0 0 1	0 1 0 1	direct address

Operation: DEC
(direct) ← (direct) − 1

DEC @Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 1 1 i

Operation: DEC
((Ri)) ← ((Ri)) − 1

DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B. The Accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

Exception: if B had originally contained 00H, the values returned in the Accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example: The Accumulator contains 251 (0FBH or 11111011B) and B contains 18 (12H or 00010010B). The instruction,

DIV AB

will leave 13 in the Accumulator (0DH or 00001101B) and the value 17 (11H or 00010001B) in B, since 251 = (13 X 18) + 17. Carry and OV will both be cleared.

Bytes: 1

Cycles: 4

Encoding:

1 0 0 0	0 1 0 0

Operation: DIV
$(A)_{15-8}$
$(B)_{7-0} \leftarrow (A)/(B)$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DJNZ <byte>,<rel-addr>

Function: Decrement and Jump if Not Zero

Description: DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

```
DJNZ   40H,LABEL_1
DJNZ   50H,LABEL_2
DJNZ   60H,LABEL_3
```

will cause a jump to the instruction at label LABEL_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was *not* taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

```
          MOV     R2,#8
TOGGLE:   CPL     P1.7
          DJNZ    R2,TOGGLE
```

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output Port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

DJNZ Rn,rel

Bytes: 2

Cycles: 2

Encoding:

1 1 0 1	1 r r r	rel. address

Operation: DJNZ

```
(PC) ← (PC) + 2
(Rn) ← (Rn) − 1
IF  (Rn) > 0 or (Rn) < 0
    THEN
        (PC) ← (PC) + rel
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

DJNZ direct,rel

Bytes: 3

Cycles: 2

Encoding:

1 1 0 1	0 1 0 1	direct address	rel. address

Operation:

```
DJNZ
(PC) ← (PC) + 2
(direct) ← (direct) − 1
IF (direct) > 0 or (direct) < 0
   THEN
        (PC) ← (PC) + rel
```

INC <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H. No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Register 0 contains 7EH (01111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

```
INC  @R0
INC  R0
INC  @R0
```

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

INC A

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 0 0

Operation:

```
INC
(A) ← (A) + 1
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

INC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	1 r r r

Operation: INC
(Rn) ← (Rn) + 1

INC direct

Bytes: 2

Cycles: 1

Encoding:

0 0 0 0	0 1 0 1	direct address

Operation: INC
(direct) ← (direct) + 1

INC @Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 1 i

Operation: INC
((Ri)) ← ((Ri)) + 1

INC DPTR

Function: Increment Data Pointer

Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2^{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order byte (DPH). No flags are affected.

This is the only 16-bit register which can be incremented.

Example: Registers DPH and DPL contain 12H and 0FEH, respectively. The instruction sequence,

```
INC   DPTR
INC   DPTR
INC   DPTR
```

will change DPH and DPL to 13H and 01H.

Bytes: 1

Cycles: 2

Encoding:

1 0 1 0	0 0 1 1

Operation: INC
(DPTR) ← (DPTR) + 1

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

JB bit,rel

Function: Jump if Bit set

Description: If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56 (01010110B). The instruction sequence,

```
JB   P1.2,LABEL1
JB   ACC.2,LABEL2
```

will cause program execution to branch to the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding:

0 0 1 0	0 0 0 0	bit address	rel. address

Operation:

```
JB
(PC) ← (PC) + 3
IF  (bit) = 1
    THEN
        (PC) ← (PC) + rel
```

JBC bit,rel

Function: Jump if Bit is set and Clear bit

Description: If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *The bit will not be cleared if it is already a zero.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

Example: The Accumulator holds 56H (01010110B). The instruction sequence,

```
JBC   ACC.3,LABEL1
JBC   ACC.2,LABEL2
```

will cause program execution to continue at the instruction identified by the label LABEL2, with the Accumulator modified to 52H (01010010B).

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Bytes: 3

Cycles: 2

Encoding:

0 0 0 1	0 0 0 0	bit address	rel. address

Operation:

```
JBC
(PC) ← (PC) + 3
IF  (bit) = 1
    THEN
        (bit) ← 0
        (PC) ← (PC) + rel
```

JC rel

Function: Jump if Carry is set

Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Example: The carry flag is cleared. The instruction sequence,

```
JC    LABEL1
CPL   C
JC    LABEL2
```

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 0 0	0 0 0 0	rel. address

Operation:

```
JC
(PC) ← (PC) + 2
IF  (C) = 1
    THEN
        (PC) ← (PC) + rel
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

JMP @A+DPTR

Function: Jump indirect

Description: Add the eight-bit unsigned contents of the Accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^{16}): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the Accumulator nor the Data Pointer is altered. No flags are affected.

Example: An even number from 0 to 6 is in the Accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP_TBL:

```
        MOV     DPTR,#JMP_TBL
        JMP     @A+DPTR
JMP_TBL: AJMP   LABEL0
        AJMP    LABEL1
        AJMP    LABEL2
        AJMP    LABEL3
```

If the Accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

Bytes: 1

Cycles: 2

Encoding:

0 1 1 1	0 0 1 1

Operation: JMP
(PC) ← (A) + (DPTR)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

JNB bit,rel

Function: Jump if Bit Not set

Description: If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56H (01010110B). The instruction sequence,

```
JNB  P1.3,LABEL1
JNB  ACC.3,LABEL2
```

will cause program execution to continue at the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding:

0 0 1 1	0 0 0 0	bit address	rel. address

Operation:

```
JNB
(PC) ← (PC) + 3
IF  (bit) = 0
    THEN (PC) ← (PC) + rel.
```

JNC rel

Function: Jump if Carry not set

Description: If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

Example: The carry flag is set. The instruction sequence,

```
JNC  LABEL1
CPL  C
JNC  LABEL2
```

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 0 1	0 0 0 0	rel. address

Operation:

```
JNC
(PC) ← (PC) + 2
IF  (C) = 0
    THEN  (PC) ← (PC) + rel
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

JNZ rel

Function: Jump if Accumulator Not Zero

Description: If any bit of the Accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally holds 00H. The instruction sequence,

```
JNZ   LABEL1
INC   A
JNZ   LABEL2
```

will set the Accumulator to 01H and continue at label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 1	0 0 0 0	rel. address

Operation:

```
JNZ
(PC) ← (PC) + 2
IF    (A) ≠ 0
      THEN  (PC) ← (PC) + rel
```

JZ rel

Function: Jump if Accumulator Zero

Description: If all bits of the Accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally contains 01H. The instruction sequence,

```
JZ    LABEL1
DEC   A
JZ    LABEL2
```

will change the Accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 0	0 0 0 0	rel. address

Operation:

```
JZ
(PC) ← (PC) + 2
IF    (A) = 0
      THEN  (PC) ← (PC) + rel
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

LCALL addr16

Function: Long call

Description: LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the Stack Pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.

Example: Initially the Stack Pointer equals 07H. The label "SUBRTN" is assigned to program memory location 1234H. After executing the instruction,

```
LCALL   SUBRTN
```

at location 0123H, the Stack Pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1234H.

Bytes: 3

Cycles: 2

Encoding:

0 0 0 1	0 0 1 0	addr15-addr8	addr7-addr0

Operation: LCALL
(PC) ← (PC) + 3
(SP) ← (SP) + 1
((SP)) ← (PC_{7-0})
(SP) ← (SP) + 1
((SP)) ← (PC_{15-8})
(PC) ← $addr_{15-0}$

LJMP addr16

Function: Long Jump

Description: LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

Example: The label "JMPADR" is assigned to the instruction at program memory location 1234H. The instruction,

```
LJMP   JMPADR
```

at location 0123H will load the program counter with 1234H.

Bytes: 3

Cycles: 2

Encoding:

0 0 0 0	0 0 1 0	addr15-addr8	addr7-addr0

Operation: LJMP
(PC) ← $addr_{15-0}$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MOV <dest-byte>,<src-byte>

Function: Move byte variable

Description: The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

Example: Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

```
MOV   R0,#30H   ;R0 <= 30H
MOV   A,@R0     ;A <= 40H
MOV   R1,A      ;R1 <= 40H
MOV   B,@R1     ;B <= 10H
MOV   @R1,P1    ;RAM (40H) <= 0CAH
MOV   P2,P1     ;P2 #0CAH
```

leaves the value 30H in register 0, 40H in both the Accumulator and register 1, 10H in register B, and 0CAH (11001010B) both in RAM location 40H and output on port 2.

MOV A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	1 r r r

Operation: MOV
(A) ← (Rn)

MOV A,direct

Bytes: 2

Cycles: 1

Encoding:

1 1 1 0	0 1 0 1	direct address

Operation: MOV
(A) ← (direct)

MOV A,ACC is not a valid instruction.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MOV A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	0 1 1 i

Operation: MOV
(A) ← ((Ri))

MOV A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	0 1 0 0	immediate data

Operation: MOV
(A) ← #data

MOV Rn,A

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	1 r r r

Operation: MOV
(Rn) ← (A)

MOV Rn,direct

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	1 r r r	direct addr.

Operation: MOV
(Rn) ← (direct)

MOV Rn,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	1 r r r	immediate data

Operation: MOV
(Rn) ← #data

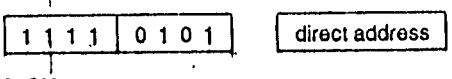
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MOV direct,A

Bytes: 2

Cycles: 1

Encoding:



Operation: MOV
(direct) ← (A)

MOV direct,Rn

Bytes: 2

Cycles: 2

Encoding:

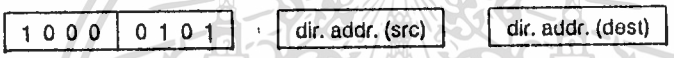
Operation: MOV
(direct) ← (Rn)

MOV direct,direct

Bytes: 3

Cycles: 2

Encoding:



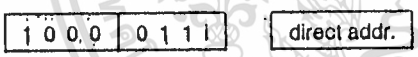
Operation: MOV
(direct) ← (direct)

MOV direct,@Ri

Bytes: 2

Cycles: 2

Encoding:



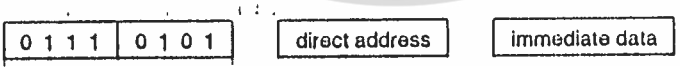
Operation: MOV
(direct) ← ((Ri))

MOV direct,#data

Bytes: 3

Cycles: 2

Encoding:



Operation: MOV
(direct) ← #data

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

MOV @Ri,A

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	0 1 1 i

Operation: MOV
((Ri)) ← (A)

MOV @Ri,direct

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	0 1 1 i	direct addr.

Operation: MOV
((Ri)) ← (direct)

MOV @Ri,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 1	0 1 1 i	immediate data

Operation: MOV
((RI)) ← #data

MOV <dest-bit>,<src-bit>

Function: Move bit data

Description: The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

Example: The carry flag is originally set. The data present at input Port 3 is 11000101B. The data previously written to output Port 1 is 35H (00110101B).

```
MOV  P1.3,C
MOV  C,P3.3
MOV  P1.2,C
```

will leave the carry cleared and change Port 1 to 39H (00111001B).

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MOV C,bit

Bytes: 2

Cycles: 1

Encoding: | 1 0 1 0 | 0 0 1 0 | bit address |

Operation: MOV
(C) ← (bit)

MOV bit,C

Bytes: 2

Cycles: 2

Encoding: | 1 0 0 1 | 0 0 1 0 | bit address |

Operation: MOV
(bit) ← (C)

MOV DPTR, #data16

Function: Load Data Pointer with a 16-bit constant

Description: The Data Pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

This is the only instruction which moves 16 bits of data at once.

Example: The instruction,

MOV DPTR, #1234H

will load the value 1234H into the Data Pointer: DPH will hold 12H and DPL will hold 34H.

Bytes: 3

Cycles: 2

Encoding: | 1 0 0 1 | 0 0 0 0 | immed. data15-8 | immed. data7-0 |

Operation: MOV
(DPTR) ← #data$_{15-0}$
DPH ☐ DPL ← #data$_{15-8}$ ☐ #data$_{7-0}$

MOVC A,@A+<base-reg>

Function: Move Code byte

Description: The MOVC instructions load the Accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit Accumulator contents and the contents of a sixteen-bit base register, which may be either the Data Pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Example: A value between 0 and 3 is in the Accumulator. The following instructions will translate the value in the Accumulator to one of four values defined by the DB (define byte) directive.

```
REL_PC: INC   A
        MOVC  A,@A+PC
        RET
        DB    66H
        DB    77H
        DB    88H
        DB    99H
```

If the subroutine is called with the Accumulator equal to 01H, it will return with 77H in the Accumulator. The INC A before the MOVC instruction is needed to "get around" the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the Accumulator instead.

MOVC A,@A+DPTR

Bytes: 1

Cycles: 2

Encoding:

1 0 0 1	0 0 1 1

Operation: MOVC
(A) ← ((A) + (DPTR))

MOVC A,@A + PC

Bytes: 1

Cycles: 2

Encoding:

1 0 0 0	0 0 1 1

Operation: MOVC
(PC) ← (PC) + 1
(A) ← ((A) + (PC))

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MOVX <dest-byte>,<src-byte>

Function: Move External

Description: The MOVX instructions transfer data between the Accumulator and a byte of external data memory, hence the "X" appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the Data Pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the Data Pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

Example: An external 256 byte RAM using multiplexed address/data lines (e.g., an Intel 8155 RAM/I/O/Timer) is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

```
MOVX  A,@R1
MOVX  @R0,A
```

copies the value 56H into both the Accumulator and external RAM location 12H.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MOVX A,@Ri

Bytes: 1

Cycles: 2

Encoding:

1 1 1 0	0 0 1 i

Operation: MOVX
(A) ← ((Ri))

MOVX A,@DPTR

Bytes: 1

Cycles: 2

Encoding:

1 1 1 0	0 0 0 0

Operation: MOVX
(A) ← ((DPTR))

MOVX @Ri,A

Bytes: 1

Cycles: 2

Encoding:

1 1 1 1	0 0 1 i

Operation: MOVX
((Ri)) ← (A)

MOVX @DPTR,A

Bytes: 1

Cycles: 2

Encoding:

1 1 1 1	0 0 0 0

Operation: MOVX
(DPTR) ← (A)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

MUL AB

Function: Multiply

Description: MUL AB multiplies the unsigned eight-bit integers in the Accumulator and register B. The low-order byte of the sixteen-bit product is left in the Accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

Example: Originally the Accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

```
MUL AB
```

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the Accumulator is cleared. The overflow flag is set, carry is cleared.

Bytes: 1

Cycles: 4

Encoding:

1 0 1 0	0 1 0 0

Operation: MUL

$(A)_{7-0} \leftarrow (A) \times (B)$

$(B)_{15-8}$

NOP

Function: No Operation

Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Example: It is desired to produce a low-going output pulse on bit 7 of Port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence,

```
CLR   P2.7
NOP
NOP
NOP
NOP
SETB  P2.7
```

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 0 0 0

Operation: NOP

$(PC) \leftarrow (PC) + 1$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ORL <dest-byte> <src-byte>

Function: Logical-OR for byte variables

Description: ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: If the Accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

```
ORL  A,R0
```

will leave the Accumulator holding the value 0D7H (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the Accumulator at run-time. The instruction,

```
ORL  P1,#00110010B
```

will set bits 5, 4, and 1 of output Port 1.

ORL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 0 0	1 r r r

Operation: ORL
(A) ← (A) V (Rn)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ORL A,direct

Bytes: 2

Cycles: 1

Encoding:

0 1 0 0	0 1 0 1	direct address

Operation: ORL
(A) ← (A) ∨ (direct)

ORL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 1 0 0	0 1 1 i

Operation: ORL
(A) ← (A) ∨ ((Ri))

ORL A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 0 0	0 1 0 0	immediate data

Operation: ORL
(A) ← (A) ∨ #data

ORL direct,A

Bytes: 2

Cycles: 1

Encoding:

0 1 0 0	0 0 1 0	direct address

Operation: ORL
(direct) ← (direct) ∨ (A)

ORL direct,#data

Bytes: 3

Cycles: 2

Encoding:

0 1 0 0	0 0 1 1	direct addr.	immediate data

Operation: ORL
(direct) ← (direct) ∨ #data

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ORL C,<src-bit>

Function: Logical-OR for bit variables

Description: Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise . A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

Example: Set the carry flag if and only if P1.0 = 1, ACC. 7 = 1, or OV = 0:

```
MOV  C,P1.0     ;LOAD CARRY WITH INPUT PIN P10
ORL  C,ACC.7    ;OR CARRY WITH THE ACC. BIT 7
ORL  C,/OV      ;OR CARRY WITH THE INVERSE OF OV.
```

ORL C,bit

Bytes: 2

Cycles: 2

Encoding:

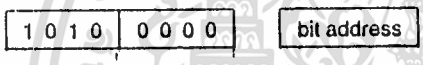
Operation:
ORL
(C) ← (C) ∨ (bit)

ORL C,/bit

Bytes: 2

Cycles: 2

Encoding:



Operation:
ORL
(C) ← (C) ∨ $\overline{(bit)}$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

POP direct

Function: Pop from stack.

Description: The contents of the internal RAM location addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

Example: The Stack Pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

```
POP   DPH
POP   DPL
```

will leave the Stack Pointer equal to the value 30H and the Data Pointer set to 0123H. At this point the instruction,

```
POP   SP
```

will leave the Stack Pointer set to 20H. Note that in this special case the Stack Pointer was decremented to 2FH before being loaded with the value popped (20H).

Bytes: 2

Cycles: 2

Encoding:

1 1 0 1	0 0 0 0	direct address

Operation: POP
(direct) ← ((SP))
(SP) ← (SP) − 1

PUSH direct

Function: Push onto stack

Description: The Stack Pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the Stack Pointer. Otherwise no flags are affected.

Example: On entering an interrupt routine the Stack Pointer contains 09H. The Data Pointer holds the value 0123H. The instruction sequence,

```
PUSH   DPL
PUSH   DPH
```

will leave the Stack Pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

Bytes: 2

Cycles: 2

Encoding:

1 1 0 0	0 0 0 0	direct address

Operation: PUSH
(SP) ← (SP) + 1
((SP)) ← (direct)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

RET

Function: Return from subroutine

Description: RET pops the high- and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

Example: The Stack Pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

```
RET
```

will leave the Stack Pointer equal to the value 09H. Program execution will continue at location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0 0 1 0	0 0 1 0

Operation:

```
RET
(PC15-8) ← ((SP))
(SP) ← (SP) − 1
(PC7-0) ← ((SP))
(SP) ← (SP) − 1
```

RETI

Function: Return from interrupt

Description: RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is left decremented by two. No other registers are affected; the PSW is *not* automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

Example: The Stack Pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

```
RETI
```

will leave the Stack Pointer equal to 09H and return program execution to location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0 0 1 1	0 0 1 0

Operation:

```
RETI
(PC15-8) ← ((SP))
(SP) ← (SP) − 1
(PC7-0) ← ((SP))
(SP) ← (SP) − 1
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

RL A

Function: Rotate Accumulator Left

Description: The eight bits in the Accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

```
RL   A
```

leaves the Accumulator holding the value 8BH (10001011B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0 0 1 0	0 0 1 1

Operation: RL

$(A_{n+1}) \leftarrow (An)$ $n = 0 - 6$

$(A0) \leftarrow (A7)$

RLC A

Function: Rotate Accumulator Left through the Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

```
RLC   A
```

leaves the Accumulator holding the value 8BH (10001010B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	0 0 1 1

Operation: RLC

$(An + 1) \leftarrow (An)$ $n = 0 - 6$

$(A0) \leftarrow (C)$

$(C) \leftarrow (A7)$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

RR A

Function: Rotate Accumulator Right

Description: The eight bits in the Accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

```
RR A
```

leaves the Accumulator holding the value 0E2H (11100010B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 0 1 1

Operation: RR

$(An) \leftarrow (A_n + 1) \quad n = 0 - 6$

$(A7) \leftarrow (A0)$

RRC A

Function: Rotate Accumulator Right through Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B), the carry is zero. The instruction,

```
RRC A
```

leaves the Accumulator holding the value 62 (01100010B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 0 1 1

Operation: RRC

$(An) \leftarrow (An + 1) \quad n = 0 - 6$

$(A7) \leftarrow (C)$

$(C) \leftarrow (A0)$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

SETB <bit>

Function: Set Bit

Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.

Example: The carry flag is cleared. Output Port 1 has been written with the value 34H (00110100B). The instructions,

```
SETB  C
SETB  P1.0
```

will leave the carry flag set to 1 and change the data output on Port 1 to 35H (00110101B).

SETB C

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 0 1 1

Operation:
```
SETB
(C) ← 1
```

SETB bit

Bytes: 2

Cycles: 1

Encoding:

1 1 0 1	0 0 1 0	bit address

Operation:
```
SETB
(bit) ← 1
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

SJMP rel

Function: Short Jump.

Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it.

Example: The label "RELADR" is assigned to an instruction at program memory location 0123H. The instruction,

SJMP RELADR

will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.

(*Note:* Under the above conditions the instruction following SJMP will be at 0102H. Therefore, the displacement byte of the instruction will be the relative offset (0123H-0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be a one-instruction infinite loop.)

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 0 0	rel. address

Operation: SJMP
(PC) ← (PC) + 2
(PC) ← (PC) + rel

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

SUBB A,<src-byte>

Function: Subtract with borrow

Description: SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving the result in the Accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set *before* executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the source operand.) AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

SUBB A,R2

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction.

SUBB A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 0 0 1	1 r r r

Operation: SUBB

(A) ← (A) − (C) − (Rn)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

SUBB A,direct

Bytes: 2

Cycles: 1

Encoding:

1 0 0 1	0 1 0 1	direct address

Operation: SUBB
(A) ← (A) − (C) − (direct)

SUBB A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1 0 0 1	0 1 1 i

Operation: SUBB
(A) ← (A) − (C) − ((Ri))

SUBB A,#data

Bytes: 2

Cycles: 1

Encoding:

1 0 0 1	0 1 0 0	immediate data

Operation: SUBB
(A) ← (A) − (C) − #data

SWAP A

Function: Swap nibbles within the Accumulator

Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the Accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

SWAP A

leaves the Accumulator holding the value 5CH (01011100B).

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 1 0 0

Operation: SWAP
$(A_{3-0}) \rightleftarrows (A_{7-4})$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

XCH A,<byte>

Function: Exchange Accumulator with byte variable

Description: XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

Example: R0 contains the address 20H. The Accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

```
XCH  A,@R0
```

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

XCH A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	1 r r r

Operation:
XCH
(A) ⇄ (Rn)

XCH A,direct

Bytes: 2

Cycles: 1

Encoding:

1 1 0 0	0 1 0 1	direct address

Operation:
XCH
(A) ⇄ (direct)

XCH A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 1 1 i

Operation:
XCH
(A) ⇄ ((Ri))

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

XCHD A,@RI

Function: Exchange Digit

Description: XCHD exchanges the low-order nibble of the Accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.

Example: R0 contains the address 20H. The Accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD A,@R0

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the Accumulator.

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 1 1 i

Operation: XCHD
$(A_{3-0}) \rightleftarrows ((Ri_{3-0}))$

XRL <dest-byte>,<src-byte>

Function: Logical Exclusive-OR for byte variables

Description: XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

(*Note:* When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.)

Example: If the Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

XRL A,R0

will leave the Accumulator holding the value 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the Accumulator at run-time. The instruction,

XRL P1,#00110001B

will complement bits 5, 4, and 0 of output Port 1.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

XRL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 1 1 0	1 r r r

Operation: XRL
(A) ← (A) ∀ (Rn)

XRL A,direct

Bytes: 2

Cycles: 1

Encoding:

0 1 1 0	0 1 0 1	direct address

Operation: XRL
(A) ← (A) ∀ (direct)

XRL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 1 1 0	0 1 1 i

Operation: XRL
(A) ← (A) ∀ ((Ri))

XRL A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 1 0	0 1 0 0	immediate data

Operation: XRL
(A) ← (A) ∀ #data

XRL direct,A

Bytes: 2

Cycles: 1

Encoding:

0 1 1 0	0 0 1 0	direct address

Operation: XRL
(direct) ← (direct) ∀ (A)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า

XRL direct, #data

Bytes: 3

Cycles: 2

Encoding:

0 1 1 0	0 0 1 1	direct address	immediate data

Operation: XRL
(direct) ← (direct) ∀ #data

ศูนย์วิทยทรัพยากร จุฬาลงกรณ์มหาวิทยาลัย
ธนานันท์โนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กิตติกรรมประกาศ

โครงการนี้สำเร็จลงได้ด้วยดี โดยต้องขอขอบคุณ อาจารย์วิทยา ทิพย์สุวรรณพร ที่คอยแนะนำให้คำปรึกษา ตลอดจนให้ความช่วยเหลือในด้านต่างๆ และขอขอบคุณ อาจารย์ สิงห์ทอง พัฒนเศษฐานนธ์ ที่ได้แนะนำโครงงานนี้ รวมทั้งอาจารย์ทุกท่านที่ได้ให้การช่วยเหลือด้วยดีตลอดมา และที่จะลืมเสียมิได้ก็ คือ สโตส์ที่ได้ให้ความสะดวกเกี่ยวกับเครื่องมือและอุปกรณ์ต่างๆ สุดท้าย ต้องขอขอบคุณเพื่อนๆทุกๆคนที่คอยให้กำลังใจและให้กำลังทรัพย์ในการทำโครงงานครั้งนี้ให้สำเร็จลุล่วงด้วยดีตลอดมา

ทนง รัตนพิทักษ์

สรรเพรช ทองปาน

2533

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

หนังสืออ้างอิง

1. B.W Williams ,"Power Electronics Devices,Drivers and Applications" ,Macmillian,1978

2. Takashi Kenjo ,"Power Electronics for the Microprocessor Age" ,OXFORD NEW YORK TOKYO., 1990

3. David Finney ,"Variable Frequency AC Motor Drive System" ,Perter Peregrins Ltd,1988

4. K.S Rajashekara,Joesph Vithyathil,"Protection and Switching Aid Networks for transisters Bridge Inverters", IEEE Trans.Industrial Electronics, vol, IE-33, No.2, 1986

5. W. Shepher and L.N. Hulley, "Power Electronics and Motor Control", Cambridge University, 1987

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้