ภาควิชาครุศาสตร์วิศวกรรม

คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

ปริญญานิพนธ์ ไมโครคอมพิวเตอร์แบบแผ่นพิมพ์เดี่ยวเบอร์ 68000

THE 68000 SINGLE BOARD MICROCOMPUTER

นักศึกษา

1.นายคนิระ	วงศ์วีแก้ว	รหัสประจำตัว	39031401
2.นายฉัตรชัย	ไชยาภินันท์	รหัสประจำตัว	39031406
3.นายภักดี	พรอุดมเดช	รหัสประจำตัว	39031424
4.นายวิฑิต	ทักษิณาวาริน	รหัสประจำตัว	39031430

หลักสูตร ครุศาสตร์อุตสาหกรรมบัณฑิต **สาขาวิชา** อิเล็กทรอนิกส์และคอมพิวเตอร์

อาจารย์ผู้ควบคุมปริญญานิพนธ์

1.อาจารย์พงษ์เกียรติ เชษฐพิทักษ์สกุล

2.อาจารย์อำพล ทองระอา

3.อาจารย์ไพบูลย์ พวงวงศ์ตระกูล

คณะกรรมการสอบปริญญานิพนธ์		ลายมือชื่อ
อาจารย์อำพล	ทองระอา	.....
อาจารย์ไพบูลย์	พวงวงศ์ตระกูล	.....
อาจารย์กิติพงศ์	มะโน	.....
อาจารย์ปิยะ	จิตธรรมมาภิรมย์	.....
อาจารย์สุรพงษ์	สิริพงศ์ดี	.....

วันเดือนที่สอบ วันที่ 14 ธันวาคม 2540 **เวลา** 12.30 น. ถึง 14.30 น.

สถานที่สอบ ห้อง ค.310 คณะครุศาสตร์อุตสาหกรรม

ภาควิชารับรองแล้ว

.....

(ผู้ช่วยศาสตราจารย์ ดร.ธีรพล เทพหัสดิน ณ อยุธยา)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

วันที่.....เดือน.....พ.ศ.....

ปริญญานิพนธ์

ไมโครคอมพิวเตอร์แบบแผ่นพิมพ์เดี่ยวเบอร์ 68000

THE 68000 SINGLE BOARD MICROCOMPUTER

นายคนิระ วงศ์วีแก้ว

นายฉัตรชัย ไชยาภินันท์

นายภักดี พรอุดมเดช

นายวิฑิต ทักษิณาวาริน

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรครุศาสตร์อุตสาหกรรมบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์

ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์

เรื่อง ไมโครคอมพิวเตอร์แบบแผ่นพิมพ์เดี่ยวเบอร์ 68000

THE 68000 SINGLE BOARD MICROCOMPUTER

ผู้จัดทำ

1. นายคนิระ วงศ์วีแก้ว
2. นายฉัตรชัย ไชยาภินันท์
3. นายภักดี พรอุดมเดช
4. นายวิทิต ทักษิณาวาริน

อาจารย์ที่ปรึกษา

ลงนาม.....

(อาจารย์พงษ์เกียรติ เชษฐพิทักษ์สกุล)

ลงนาม.....

(อาจารย์อำพล ทองระอา)

ลงนาม.....

(อาจารย์ไพบูลย์ พวงวงศ์ตระกูล)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

ลงนาม.....

(ผศ.ดร.ธีระพล เทพหัสดิน ณ อยุธยา)

ปริญญานิพนธ์

เรื่อง ไมโครคอมพิวเตอร์แบบแผ่นพิมพ์เดี่ยวเบอร์ 68000

THE 68000 SINGLE BOARD MICROCOMPUTER

วัตถุประสงค์

1. เพื่อศึกษา 68000 ไมโครคอมพิวเตอร์แบบแผ่นพิมพ์เดี่ยว
2. เพื่อออกแบบ 68000 ไมโครคอมพิวเตอร์แบบแผ่นพิมพ์เดียว
3. เพื่อสร้าง Single Board ที่ใช้อุปกรณ์ภายในประเทศ
4. เพื่อออกแบบ Program Monitor ควบคุมการทำงานของ 68000
5. เพื่อสร้าง Program Monitor ควบคุมการทำงานของ 68000

ประโยชน์ที่คาดว่าจะได้รับ

1. เข้าใจการทำงานของไมโครโปรเซสเซอร์ 68000
2. สามารถออกแบบ 68000 ไมโครคอมพิวเตอร์แบบแผ่นพิมพ์เดี่ยว
3. สามารถสร้าง 68000 ไมโครคอมพิวเตอร์แบบแผ่นพิมพ์เดี่ยวที่สร้างขึ้นภายในประเทศ

ไมโครคอมพิวเตอร์แบบแผ่นพิมพ์เดี่ยวเบอร์ 68000

นายคนิระ วงศ์วีแก้ว
นายฉัตรชัย ไชยาภินันท์
นายภักดี พรอุดมเดช
นายวิฑิต ทักษิณาวาริน

อาจารย์ที่ปรึกษา
อาจารย์พงษ์เกียรติ เชษฐพิทักษ์สกุล
อาจารย์อำพล ทองระอา
อาจารย์ไพบูลย์ พวงวงศ์ตระกูล

ปีการศึกษา 2540

บทคัดย่อ

ปริญญานิพนธ์นี้นำเสนอการจัดสร้างโครงงานชุดฝึก 68000 ไมโครคอมพิวเตอร์แบบแผ่นพิมพ์เดี่ยว ด้วยการออกแบบระบบไมโครคอมพิวเตอร์พื้นฐานมีโครงสร้างหลักคือใช้ไมโครโปรเซสเซอร์เบอร์ 68000 ประกอบด้วยอุปกรณ์ต่อพ่วงรอบข้างคือ หน่วยความจำ ROM ขนาด 32 กิโลเวิร์ด, หน่วยความจำ RAM ขนาด 32 กิโลเวิร์ด, ติดต่อกับผู้ใช้ด้วยคีย์บอร์ด IBM PC XT/AT และจอแสดงผล LCD ขนาด 4 แถว

โครงงานที่สร้างนี้สามารถใช้สาธิต และศึกษาการออกแบบและใช้งานในด้านฮาร์ดแวร์และซอฟต์แวร์ของไมโครโปรเซสเซอร์ตระกูล 68000 ได้เป็นอย่างดี

THE 68000 SINGLE BOARD MICROCOMPUTER

MR.KHANIRA WONGWEKAEW
MR.CHATCHAI CHAIYAPINUN
MR.PAKDEE PORNOUDOMDAJ
MR.WITHIT THAKSINAWARIN

ADVISORS

MR.PONGKIAT CHEDPITAKSAKUL
MR.AMPHON THONGRA-AR
MR.PAIBOON PONGWONGTRAGULL

1997

ABSTRACT

This thesis presents the 68000 single board Microcomputer. The 68000 single board Micromputer includes of the 68000 Microprocessor, peripheral for Microcomputer system, ROM memory 32 Kword, RAM memory 32 Kword, Keyboad PC XT/AT and LCD display 4 line 16 columne

This thesis can be used to demonstrate and learn about design of hardware and software of Microprocessor

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ สำเร็จลุล่วงลงได้ด้วยดี เนื่องจากความอนุเคราะห์ของอาจารย์ที่ปรึกษาปริญญานิพนธ์ และอาจารย์ประจำภาควิชาครุศาสตร์วิศวกรรมทุกท่าน ที่ได้กรุณาให้คำปรึกษา,ข้อเสนอแนะพร้อมทั้งแนวทางแก้ไขปัญหาในการดำเนินงานรวมถึงเพื่อนๆ และน้องๆ ทุกคนที่คอยให้ความช่วยเหลือและกำลังใจตลอดเวลา ซึ่งโอกาสนี้ คณะผู้จัดทำรู้สึกซาบซึ้งในพระคุณของทุกๆ ท่านคณะผู้จัดทำขอขอบคุณเป็นอย่างยิ่ง มา ณ โอกาสนี้ด้วย

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญภาพ

สารบัญภาพ(ต่อ)

สารบัญภาพ(ต่อ)

สารบัญภาพ(ต่อ)

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปริญญานิพนธ์

ในปัจจุบันได้มีการใช้คอมพิวเตอร์มาใช้ในงานหลายๆ ด้านแทนมนุษย์และในแต่ละด้านมีความจำเป็นอย่างมากที่ต้องอาศัยผู้เชี่ยวชาญในด้านการใช้งานคอมพิวเตอร์กับงานด้านต่างๆ เพื่อให้เกิดผู้เชี่ยวชาญในการออกแบบคอมพิวเตอร์ฮาร์ดแวร์ และรวมถึงซอฟต์แวร์เฉพาะด้านมารวมกัน โครงงานนี้จึงจัดสร้างซิงเกิ้ลบอร์ดคอมพิวเตอร์ 68000 ซึ่งซิงเกิ้ลบอร์ดคอมพิวเตอร์ 68000 ที่จัดสร้างขึ้นมา สามารถใช้งานในด้านใดด้านหนึ่งของอุตสาหกรรมโดยเฉพาะหรือใช้เป็นอุปกรณ์ในการเรียนการสอนเพื่อสร้างให้ผู้เรียนเกิดความรู้, ทักษะและประสบการณ์ จะเห็นได้ว่าปัจจุบันซิงเกิ้ลบอร์ดคอมพิวเตอร์ที่ใช้ศึกษากันอยู่เป็นขนาด 8 บิต ซึ่งล้าสมัยไปแล้วแต่โครงงานนี้ใช้ซีพียูขนาด 16 บิต ซึ่งเป็นเทคโนโลยีและประสิทธิภาพที่สูงกว่า จึงเป็นการดีที่จะเรียนรู้ในเทคโนโลยีที่สูงขึ้นเพื่อให้เกิดผู้เชี่ยวชาญในการใช้งานคอมพิวเตอร์

ทางผู้จัดทำโครงงานได้เล็งเห็นถึงปัญหาในตรงจุดนี้และได้ให้ความสำคัญกับปัญหาดังกล่าวจึงมีแนวคิดที่จะผลิตซิงเกิ้ลบอร์ดคอมพิวเตอร์ เพื่อประยุกต์ใช้งานในด้านใดด้านหนึ่งของอุตสาหกรรมโดยเฉพาะ การประยุกต์ใช้งานไมโครโปรเซสเซอร์ไปควบคุมการทำงานในด้านต่างๆ ของงานอุตสาหกรรมหรือสามารถใช้เป็นอุปกรณ์ในการเรียนการสอนเพื่อทำให้การเรียนการสอนมีประสิทธิภาพยิ่งขึ้น ในปัจจุบันนี้ซิงเกิ้ลบอร์ดคอมพิวเตอร์ที่นำเข้าจากต่างประเทศ ทั้งหมดเป็นอุปกรณ์ที่มีราคาแพง ทางผู้จัดทำโครงงานจึงมีแนวคิดว่าเป็นการดีที่จะประหยัดเงินที่สูญเสียจากการซื้อสินค้าที่นำเข้าจากต่างประเทศ

ในโครงงานนี้เลือกใช้ซีพียูเบอร์ 68000 และอุปกรณ์เกี่ยวข้องการอินพุตหรือเอาต์พุตของบริษัทโมโตโรล่า ซีพียูเบอร์ 68000 ถูกผลิตขึ้นมาครั้งแรกในปี 1979 ใช้สถาปัตยกรรมภายในตัวซีพียูเป็นขนาด 32 บิต แต่สถาปัตยกรรมภายนอกมีขนาด 16 บิต ซีพียูตระกูล 680XX ถูกนำไปใช้งานที่รู้จักกันดีคือนำไปเป็นซีพียูของคอมพิวเตอร์แมคอินทอช

1.2 ขีดความสามารถของโครงงาน

ขีดความสามารถของโครงงาน มีรายละเอียดดังต่อไปนี้

ไมโครโปรเซสเซอร์	MC 68000
อุปกรณ์อินพุตและเอาต์พุต	-การติดต่อ RS 232 1 ตำแหน่ง
	-การติดต่อคีย์บอร์ด ชนิด IBM PC XT/AT
	-การติดต่อหน่วยแสดงผลชนิด LCD ขนาด 20 คอลัมน์ 4 แถว
ความถี่ของสัญญาณนาฬิกา	-10 เมกกะเฮิรตซ์
ขนาดของหน่วยความจำ	-หน่วยความจำชนิด RAM ขนาด 32 กิโลเวิร์ด
	-หน่วยความจำชนิด ROM ขนาด 32 กิโลเวิร์ด
โปรแกรมควบคุมการทำงาน	-ซอฟต์แวร์มอนิเตอร์ บนหน่วยความจำชนิด ROM 32 กิโลเวิร์ด
แหล่งจ่ายไฟ	-ขนาด +5 โวลต์ , +12 โวลต์ , -12 โวลต์

1.3 เนื้อหาโดยสังเขป

เนื้อหาภายในปริญญานิพนธ์ฉบับนี้แบ่งออกเป็นบทต่างๆ เพื่อความสะดวกต่อการศึกษาและทำความเข้าใจ ในแต่ละบทประกอบด้วยเนื้อหาที่สำคัญดังนี้

บทที่ 1 บทนำ กล่าวถึงความเป็นมาของซิงเกิ้ลบอร์ด คอมพิวเตอร์ 68000 ขีดความสามารถของซิงเกิ้ลบอร์ด คอมพิวเตอร์

บทที่ 2 ทฤษฎีและหลักการ กล่าวถึงเรื่องสถาปัตยกรรมของไมโครโปรเซสเซอร์ 68000 รูปแบบทางซอฟต์แวร์ของไมโครโปรเซสเซอร์ 68000 คำสั่งของไมโครโปรเซสเซอร์ 68000 อธิบายการใช้งานขาต่างๆ ของไมโครโปรเซสเซอร์ 68000 วงจรควบคุมการทำงานของไมโครโปรเซสเซอร์ 68000

บทที่ 3 การออกแบบและการทำงาน เทคนิควิธีการติดต่อไมโครโปรเซสเซอร์ 68000 กับหน่วยความจำและอุปกรณ์อินพุตและเอาต์พุตวงจรพื้นฐานในการควบคุมการทำงานของไมโครโปรเซสเซอร์ 68000

บทที่ 4 ผลการทดลองและทดสอบ การทดลองและทดสอบวงจรควบคุมการทำงานของไมโครโปรเซสเซอร์ 68000 โปรแกรมทดสอบอุปกรณ์อินพุตและเอาต์พุต

บทที่ 5 สรุป อภิปราย และข้อเสนอแนะ สรุปปัญหาที่พบในการจัดทำโครงงาน การแก้ปัญหา ประโยชน์ที่ได้รับจากการจัดทำโครงงานแนวทางการพัฒนาโครงงาน

ภาคผนวก ก โปรแกรมการทำงาน

ภาคผนวก ข ลายวงจรและแผ่นวงจรพิมพ์

ภาคผนวก ค คุณสมบัติของอุปกรณ์

บทที่ 2
ทฤษฎีและหลักการ

2.1 กล่าวนำ

เนื้อหาของปริญญานิพนธ์ในบทนี้จะเป็นทฤษฎีและหลักการที่นำมาใช้ประกอบการสร้างโครงงาน โดยประกอบด้วยรูปแบบทางซอฟต์แวร์ของไมโครโปรเซสเซอร์

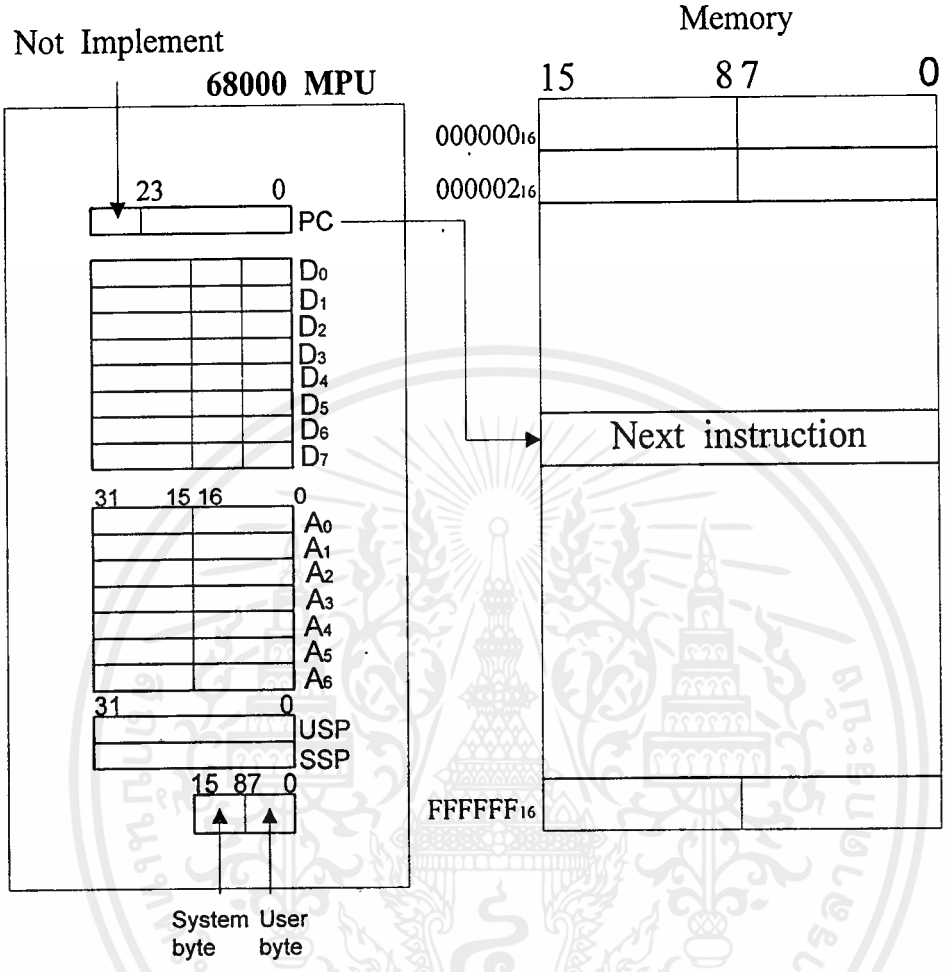
เราเริ่มต้นศึกษาสมาชิกในตระกูล 68000 มีขนาด 16 บิต ของบริษัทโมโตโลร่า หัวข้อที่จะอธิบายต่อไปมีดังนี้

2.2 โครงสร้างทางซอฟต์แวร์ 68000 ไมโครโปรเซสเซอร์

จุดมุ่งหมายของการพัฒนาโครงสร้างทางซอฟต์แวร์คือการที่จะทำให้โปรแกรมเมอร์เข้าใจในการทำงานของระบบไมโครโปรเซสเซอร์ จากการมองผ่านซอฟต์แวร์ เพื่อให้สามารถโปรแกรมไมโครโปรเซสเซอร์ได้อย่างมีประสิทธิภาพ โดยที่ไม่จำเป็นต้องรู้หมดทุกส่วนของฮาร์ดแวร์มีรายละเอียด ดังตัวอย่าง เราไม่จำเป็นที่จะรู้ฟังก์ชั่นของสัญญาณขาต่างๆ จำนวนมากจุดต่อทางไฟฟ้าหรือคุณลักษณะของการสวิตช์ชิ่ง

แล้วสิ่งใดคือสิ่งที่สำคัญที่ผู้เขียนโปรแกรมต้องรู้คือความหลากหลายของรีจิสเตอร์ภายใน และเข้าใจซึ่งจุดมุ่งหมายของมัน, ฟังก์ชั่น, ความจุและขอบเขตของมัน มากไปกว่าที่เป็นสิ่งที่จำเป็นคือการจัดการหน่วยความจำภายนอกและทำอย่างไรที่จะสามารถอ้างอิงถึงตำแหน่งจากคำสั่งและข้อมูล

รูปที่ 2.1 รูปแบบทางซอฟต์แวร์ของ 68000 ไมโครโปรเซสเซอร์ จะเห็นได้ว่า 68000 ไมโครโปรเซสเซอร์ มีจำนวน 8 ตัวของรีจิสเตอร์ข้อมูล (D0-D7), 7 ตัวของรีจิสเตอร์ตำแหน่ง (A0-A6), สแต็คพอยท์เตอร์ 2 ตัว (USP และ SSP), โปรแกรมเคาท์เตอร์และสถานะรีจิสเตอร์ สามารถอ้างอิงตำแหน่งในหน่วยความจำได้ถึง 16 เมกกะไบต์

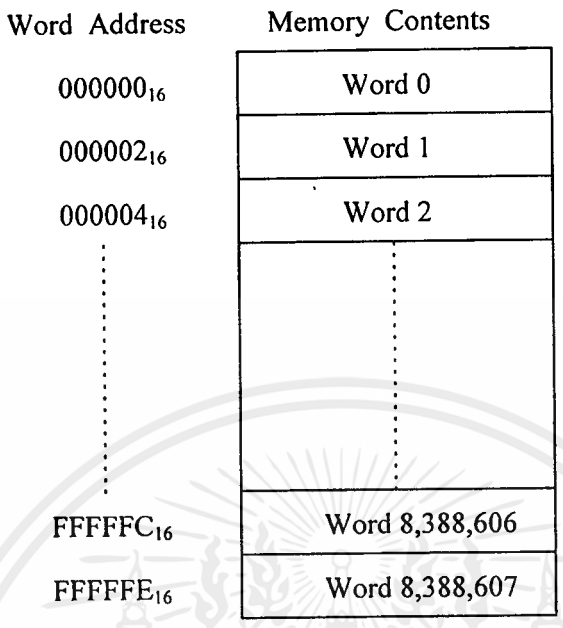


รูปที่ 2.1 รูปแบบทางซอฟต์แวร์ของ 68000 ไมโครโปรเซสเซอร์

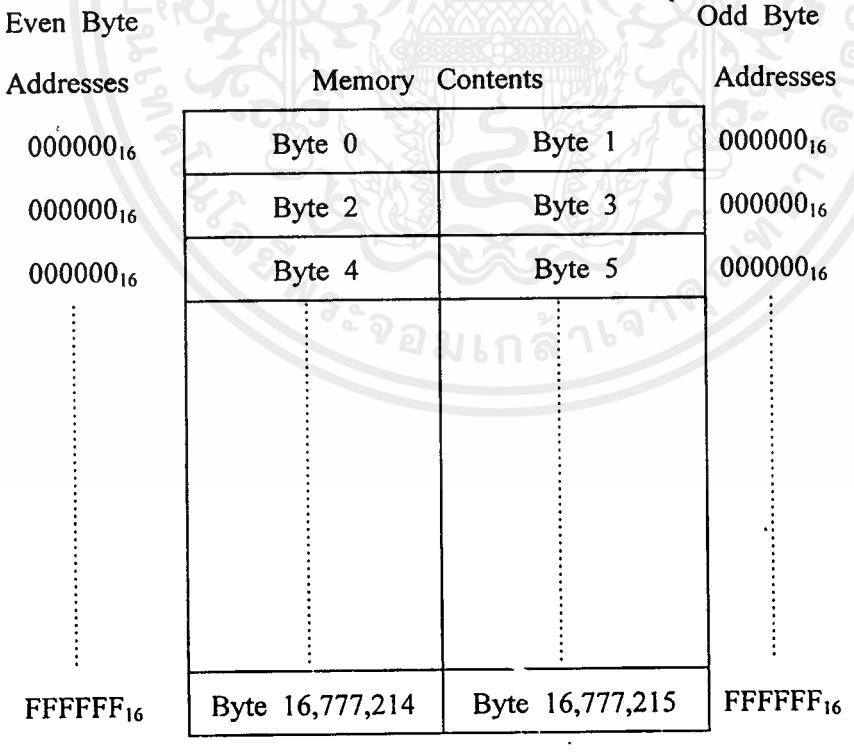
2.3 การจัดการกับหน่วยความจำ

รูปแบบทางซอฟต์แวร์ของ 68000 ไมโครโปรเซสเซอร์กับการอ้างอิงตำแหน่งในหน่วยความจำและการจัดการข้อมูลภายในหน่วยความจำดังแสดงในรูปที่ 2.2 แสดงถึง 68000 ไมโครโปรเซสเซอร์ สามารถสนับสนุนการอ้างตำแหน่งในหน่วยความจำได้ถึง 8,388,607 (8 เมกกะไบต์) เวิร์ด

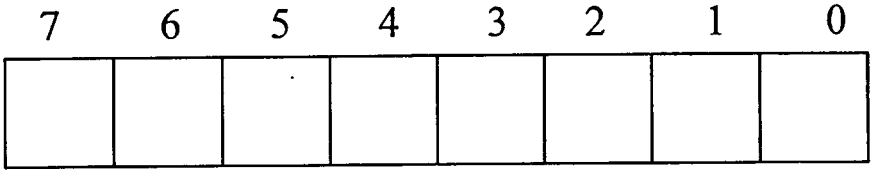
ในรูปที่ 2.2 จะเห็นว่ามีลำดับของเวิร์ดตำแหน่ง อยู่ในช่วง 000000_{16} ถึง $FFFFFF_{16}$ ส่วนในรูปที่ 2.3 นั้นจะเป็นการจัดการกับหน่วยความจำ ซึ่งเป็นการจัดในลักษณะของไบต์ มีลำดับ ของไบต์ตำแหน่ง สามารถอ้างอิงตำแหน่งได้ 16 เมกกะไบต์ (เพราะ 1 เวิร์ด มี 2 ไบต์)



รูปที่ 2.2 การอ้างอิงตำแหน่งในหน่วยความจำ

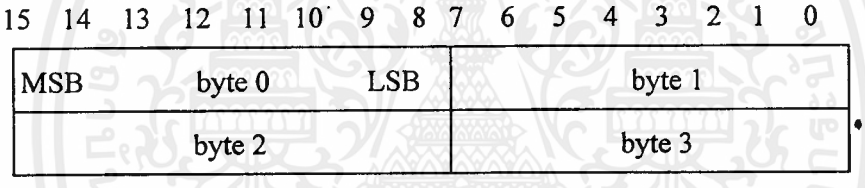


รูปที่ 2.3 การอ้างอิงตำแหน่งในหน่วยความจำแบบไบต์



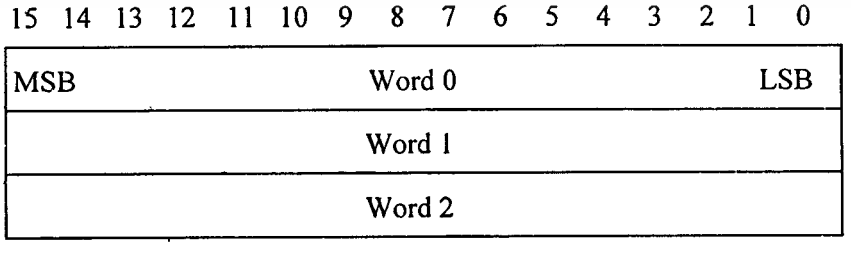
รูปที่ 2.4 รูปแบบการจัดเก็บข้อมูลแบบบิต

เกือบทั้งหมดของคำสั่งในชุดคำสั่งของ 68000 ไมโครโปรเซสเซอร์ มีความสามารถในการประมวลผลข้อมูลในรูปแบบของ ไบต์, เวิร์ดหรือลองเวิร์ด ในรูปที่ 2.4 จะเห็นภายในของข้อมูลชนิดไบต์, บิต "0" แสดงบิตนัยสำคัญต่ำและบิต "7" แสดงถึงบิตนัยสำคัญสูง



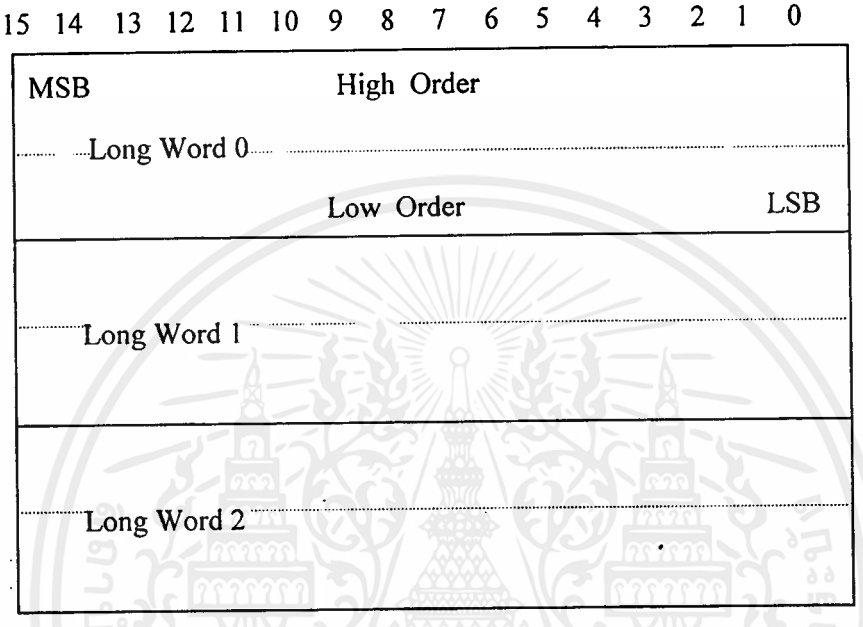
รูปที่ 2.5 รูปแบบการจัดเก็บข้อมูลแบบไบต์

รูปที่ 2.5 แสดงถึงข้อมูลขนาด 2 ไบต์ สามารถเก็บในแต่ละเวิร์ดตำแหน่ง โดยที่ตำแหน่งไบต์คู่ เช่น ไบต์ 0 และ ไบต์ 2 จะอยู่ในตำแหน่งของบิตที่มีนัยความสำคัญสูง ส่วนตำแหน่งไบต์คี่ เช่น ไบต์ 1 และไบต์ 3 จะอยู่ในตำแหน่งของบิตที่มีนัยความสำคัญต่ำ



รูปที่ 2.6 รูปแบบการจัดเก็บข้อมูลแบบเวิร์ด

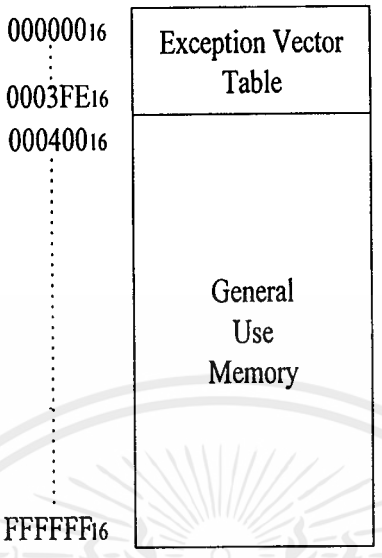
รูปที่ 2.6 และรูปที่ 2.7 แสดงถึงการเก็บข้อมูลแบบเวิร์ด ที่แต่เวิร์ดตำแหน่งและในข้อมูลแบบลองเวิร์ด ที่เก็บในลักษณะลำดับของ 3 ลองเวิร์ด .



รูปที่ 2.7 การจัดเก็บข้อมูลแบบหลายๆ ลองเวิร์ด

2.4 การบ่งชี้และการใช้หน่วยความจำ

จากที่ได้ศึกษาเบื้องต้นของโครงสร้างซอฟต์แวร์ของ 68000 ช่องว่างของการอ้างอิงตำแหน่งและการจัดการข้อมูล ต่อไปจะศึกษาในส่วนของช่วงตำแหน่งที่สามารถอ้างอิงถึงในการใช้งานทั่วไป 68000 สนับสนุนการอ้างอิงตำแหน่ง 16 เมกกะไบต์ ในรูปที่ 2.8 จะแสดงถึงตำแหน่งช่องว่างในหน่วยความจำ ชื่อว่า Dedicated Function ตำแหน่งนี้อยู่ในช่วง 000000_{16} ถึง $0003FE_{16}$ ในช่วงตำแหน่งนี้สำรองไว้เพื่อจะเก็บเวกเตอร์ ดังในรูปที่ 2.8 ตำแหน่งในช่วงนี้จะเก็บค่าของกรณีพิเศษของ 68000 เวกเตอร์เทเบิลแต่ละเวกเตอร์ขนาดของข้อมูล 4 บิตและขนาดของข้อมูลของแต่ละตำแหน่งนี้ สามารถเพิ่มขึ้นเป็น 2 เวิร์ดได้ ตัวอย่างของกรณีพิเศษของ 68000 คือ ฮาร์ดแวร์อินเตอร์รัพต์และส่วนที่เหลือในแผนผังหน่วยความจำ



รูปที่ 2.8 แผนผังหน่วยความจำ

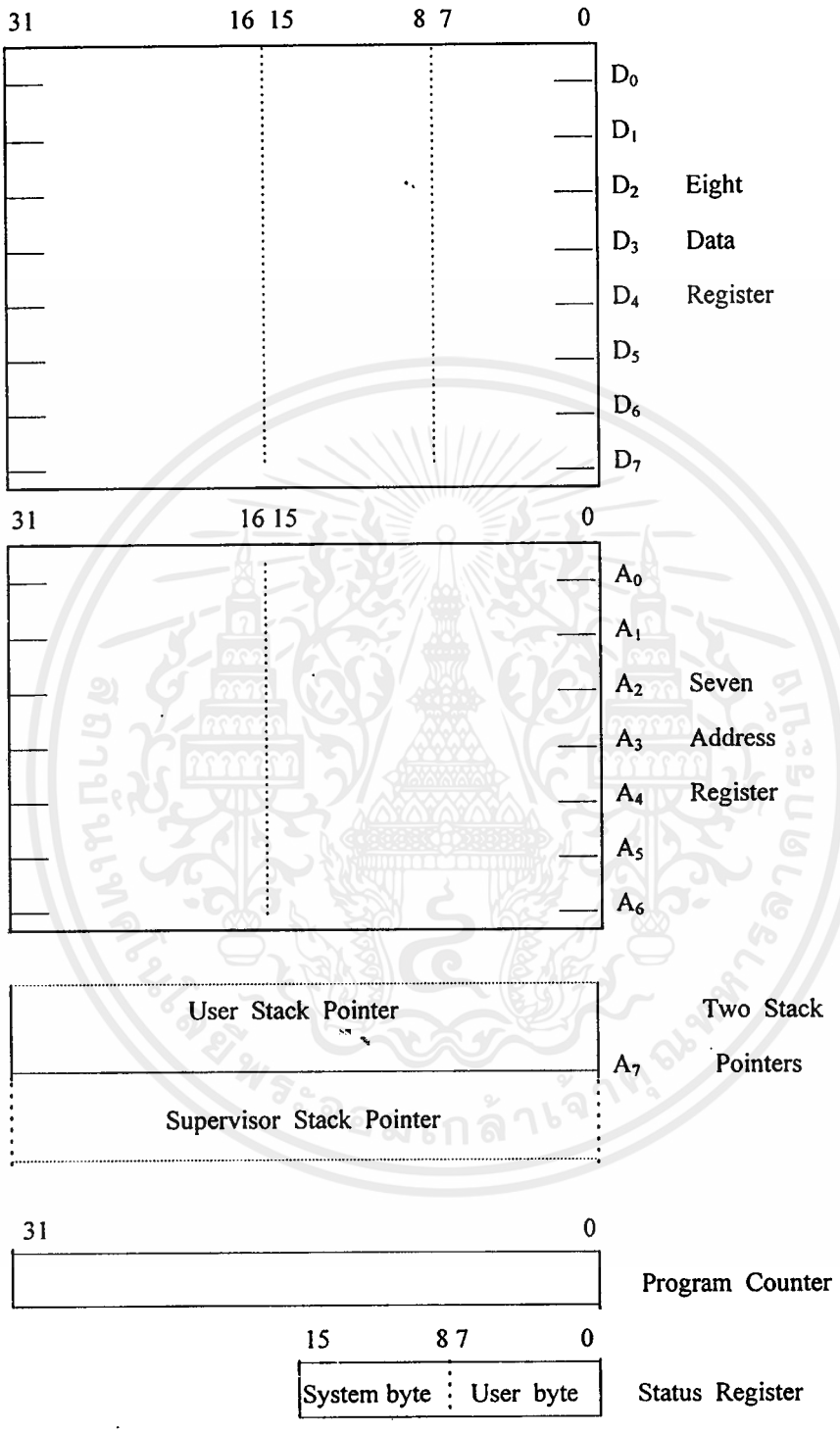
ในรูปที่ 2.8 เราจะเห็นในส่วนที่เหลือของ ตำแหน่งช่องว่างในหน่วยความจำส่วนนี้สำหรับการใช้ทั่วไป ซึ่งจะใช้เป็นส่วนที่เก็บคำสั่งของโปรแกรม, ตัวถูกดำเนินการข้อมูลหรือตำแหน่งที่อยู่ของข้อมูล (Address Information)

2.5 รีจิสเตอร์ภายในของ 68000 ไมโครโปรเซสเซอร์

ภายในของ 68000 ไมโครโปรเซสเซอร์ นั้นประกอบไปด้วย 18 ตัว ของรีจิสเตอร์ที่มีขนาด 32 บิตและ 1 ตัวของรีจิสเตอร์ที่มีขนาด 16 บิต ดังที่แสดงในรูปที่ 2.9 ซึ่งจะเห็นว่าสามารถย่อยส่วนของรีจิสเตอร์ให้ละเอียดขึ้นไปก็คือมีรีจิสเตอร์ข้อมูล 8 ตัว, ตำแหน่งรีจิสเตอร์ 7 ตัว, สแต็คพอยท์เตอร์ 2 ตัว, โปรแกรมเคาท์เตอร์ 1 ตัวและสถานะรีจิสเตอร์ 1 ตัว สถานะรีจิสเตอร์เป็นรีจิสเตอร์เพียงตัวเดียวที่มีขนาด 16 บิต

2.5.1 รีจิสเตอร์ข้อมูล

รีจิสเตอร์ข้อมูลภายในไมโครโปรเซสเซอร์ 68000 มีจำนวน 8 ตัว ดังที่แสดงในรูปที่ 2.9 ประกอบไปด้วย D_0 ถึง D_7 ในแต่ละ รีจิสเตอร์ มีขนาด 32 บิต ซึ่งมีลาเบล "0" (บิตนัยสำคัญต่ำ) ถึง ลาเบล "31" (บิตนัยสำคัญสูง) ดังในรูปที่ 2.9



รูปที่ 2.9 รีจิสเตอร์ภายในของ 68000 ไมโครโปรเซสเซอร์

รีจิสเตอร์ข้อมูลใช้ในการเก็บข้อมูลชั่วคราว เพื่อรอการประมวล ดังตัวอย่าง เช่น มีการเก็บข้อมูลตัวถูกดำเนินการต้นทาง และตัวถูกดำเนินการปลายทางของคำสั่งการกระทำทางคณิตศาสตร์หรือคำสั่งการกระทำทางลอจิกในแต่ละรีจิสเตอร์สามารถเข้าถึงข้อมูลลักษณะไบต์ตัวถูกดำเนินการ, เวิร์ดตัวถูกดำเนินการ, ลองเวิร์ดตัวถูกดำเนินการข้อมูลแบบไบต์จะเก็บไว้ในบิตจำนวน 8 บิต นัยสำคัญต่ำในรีจิสเตอร์ข้อข้อมูลนั่นก็คือ B_0 ถึง B_7 ข้อมูลแบบเวิร์ด จะเก็บอยู่ใน 16 บิตล่างเสมอ นั่นก็คือ B_0 ถึง B_{15} และข้อมูลแบบลองเวิร์ด จะเก็บโดยใช้ทั้งหมดของรีจิสเตอร์ที่มีขนาด 32 บิต จะมีคำสั่งเพียงเล็กน้อยที่อนุญาตให้เข้าถึงการประมวลผลแบบบิตได้

ขนาดของข้อมูลที่ใช้ในระหว่างเอ็กซิคิวท์ของคำสั่ง เราสามารถกำหนดรายละเอียดหมายถึง กำหนดขนาดของข้อมูลลงไปในคำสั่งได้ เช่น คำสั่ง การเคลื่อนย้าย ข้อมูล (Move) แบบไบต์ ซึ่งใช้รีจิสเตอร์ D_0 เป็นตำแหน่งข้อมูลต้นทางและ D_7 เป็นตำแหน่งของข้อมูลปลายทางหลังจากการเอ็กซิคิวท์คำสั่ง บิต B_0 ถึง B_7 ของ D_0 ถูกทำซ้ำลงไปที่บิต B_0 ถึง B_7 ของรีจิสเตอร์ D_7 อีกทาง คำสั่งสามารถเซ็ตให้เป็นการประมวลผลแบบเวิร์ดได้ ในแบบนี้

การเอ็กซิคิวท์คำสั่ง ในลักษณะที่บิต B_0 ถึง B_{15} ของรีจิสเตอร์ D_0 จะถูกทำซ้ำลงไปที่บิต B_0 ถึง B_7 ของรีจิสเตอร์ D_7 68000 สามารถใช้รีจิสเตอร์ข้อมูลในฐานะรีจิสเตอร์อ้างอิงได้ในเงื่อนไขนี้ จะใช้ค่าในรีจิสเตอร์เป็นตำแหน่งออฟเซ็ต จะรวมไปถึงค่าของรีจิสเตอร์อื่นๆ เพื่อใช้เป็นการกำหนดตำแหน่งที่จะเข้าถึงในหน่วยความจำ

2.5.2 แอดเดรสรีจิสเตอร์

แอดเดรสรีจิสเตอร์ภายในไมโครโปรเซสเซอร์ 68000 มีจำนวน 7 ตัว ประกอบด้วย A_0 ถึง A_6 ดังที่แสดงในรูปที่ 2.9 แต่ละตัวมีขนาด 32 บิต เช่นเดียวกับรีจิสเตอร์ข้อมูลโดยที่รีจิสเตอร์เหล่านี้ ไม่ได้ใช้ประโยชน์ในการเก็บข้อมูลเพื่อรอการประมวลผล แต่มีประโยชน์คือเอาไว้เก็บตำแหน่งที่ต้องการอ้างอิงถึง เช่น ตำแหน่งที่อ้างอิงถึงตำแหน่งเหล่านี้ จะถูกใช้เพื่อเข้าถึงข้อมูลที่ต้นทางและข้อมูลที่ปลายทางของหน่วยความจำประโยชน์อย่างอื่น คือสามารถใช้แอดเดรสรีจิสเตอร์ในสถานะที่เป็นอินเด็กซ์รีจิสเตอร์

มีบางอย่างที่คล้ายกับรีจิสเตอร์ข้อมูลคือ แอดเดรสรีจิสเตอร์ที่ถูกใช้แบบจุดประสงค์ กล่าวคือ คำสั่งนั้นสามารถอ้างอิงถึงทุกๆ ตำแหน่ง เพื่อการเข้าถึงข้อมูลต้นทางหรือข้อมูลปลายทางในหน่วยความจำ

ค่าของตำแหน่งที่ถูกโหลดเข้าไปสู่แอดเดรสรีจิสเตอร์ สามารถกระทำได้โดยการผ่านการควบคุมทางซอฟต์แวร์ เมื่อแอดเดรสรีจิสเตอร์เป็นรีจิสเตอร์ต้นทางแอดเดรสรีจิสเตอร์สามารถเข้าถึงข้อมูลแบบลองเวิร์ดหรือเวิร์ดได้ ในทางกลับกันถ้าใช้แอดเดรสรีจิสเตอร์เป็นรีจิสเตอร์ปลายทาง ผลคือต้องใช้ในลักษณะ 32 บิต เสมอ

2.5.3 สแต็ค พอยท์เตอร์

มีรีจิสเตอร์ภายใน 2 ตัวที่ใช้ในการเก็บค่าตำแหน่งข้อมูลรีจิสเตอร์ 2 ตัวนี้คือ User Stack Pointer (USP) และ Supervisor Stack Pointer (SSP) Stack นี้ก็คือ เป็นส่วนพิเศษของหน่วยความจำย่อยสามารถใช้ประโยชน์ในการเก็บข้อมูลชั่วคราว ดังที่กล่าวมาแล้วว่า 68000 มีสแต็คพอยท์เตอร์ 2 ตัว เรียกว่า User Stack และ Supervisor Stack ตำแหน่งที่ USP (User Stack Pointer) ชี้อยู่นี้ เป็นตำแหน่งที่สามารถเข้าถึงข้อมูลใน User Stack ได้ หรืออาจกล่าวได้ว่าตำแหน่งบนสุดของสแต็คจะถูกชี้โดยข้อมูลในสแต็คพอยท์เตอร์เสมอ ส่วนของ SSP (Supervisor Stack Pointer) เช่นเดียวกัน ในที่นี้มีจำนวนสแต็ค 2 ตัว แต่ในการใช้งานจะมี 1 ตัวที่ทำงานได้ เมื่อตัวใดตัวหนึ่งทำงาน อีก 1 ตัวจะไม่ทำงาน โดยที่ User Stack Pointer จะทำงาน เมื่อ 68000 ปฏิบัติการในโหมด User Stack ในโหมดนี้ Supervisor Stack Pointer ไม่ทำงาน ตำแหน่งที่ตำแหน่งที่จะเก็บค่าลงใน User Stack Pointer จะถูกชี้ที่ตำแหน่งบนสุดของ User Stack ใน User Part ของหน่วยความจำ User Stack เป็นตำแหน่งที่เป็นการคืนค่าของแอดเดรสรีจิสเตอร์ข้อมูลหรือพารามิเตอร์อื่นๆ ซึ่งจะถูกนำมาเก็บ เมื่อซีพียูถูกขัดจังหวะหรือซีพียูทำคำสั่ง Call

68000 สามารถใช้งานได้อีกโหมดหนึ่ง เราเรียกว่า Supervisor Stack ในเงื่อนไขนี้ Supervisor Stack Pointer จะทำงานและ User Stack Pointer จะไม่ทำงาน Supervisor Stack Pointer จะใช้งานในจุดมุ่งหมายเดียวกับ User Stack แต่ยังสามารถถูกใช้งานในแบบ Supervisor Calls เช่น กรณีพิเศษของซอฟต์แวร์ (Exception Software), อินเตอร์รัพต์และกรณีพิเศษของการทำงานภายใน (Internal Exceptions)

2.5.4 โปรแกรมเคาน์เตอร์

ในระหว่างการกระทำพื้นฐาน 68000 จะทำการอ่าน (Fetches) คำสั่งมา 1 คำสั่งจากหน่วยความจำหลังจากนั้นจะทำการเอ็กซิคิวท์ ตำแหน่งที่เก็บไว้ในโปรแกรมเคาท์เตอร์ จะเป็นตำแหน่งของคำสั่งถัดไปที่จะถูกอ่านจากหน่วยความจำ หลังจากคำสั่งนั้นถูกอ่านแล้ว จึงถูกทำการถอดรหัสข้อมูลตัวถูกดำเนินการจะถูกอ่านมาจากที่ใดที่หนึ่งคือรีจิสเตอร์ภายใน

(Internal Register) หรือหน่วยความจำ การปฏิบัติการในลักษณะเฉพาะของคำสั่งจะดำเนินการกับข้อมูลโอเปอร์แรนด์และผลจะถูกเขียนในรีจิสเตอร์ภายในหรือตำแหน่งที่ถูกเก็บข้อมูลในหน่วยความจำ เมื่อดำเนินการตามขั้นตอนที่กล่าวมา 68000 ก็จะเตรียมพร้อมสำหรับกระทำคำสั่งต่อไป

ทุกเวลาที่มีอ่านคำสั่งจากหน่วยความจำเสร็จแล้ว ค่าที่เก็บใน PC จะเพิ่มขึ้นครั้งละ 2 ดังในรูปที่ 2.9 แสดงรูปโปรแกรมควบคุมซึ่งเป็นรีจิสเตอร์ขนาด 32 บิต อย่างไรก็ดี บิตด้านต่ำ 24 บิตเท่านั้น เป็นค่าจริงที่ถูกใช้ใน 68000 บิตด้านต่ำ 24 บิตนี้ ทำให้เกิด 16 เมกกะบิตของการอ้างอิงตำแหน่งในหน่วยความจำ แต่รหัสของคำสั่งถูกเก็บในแบบเวิร์ดเสมอเพราะฉะนั้นการอ้างอิงตำแหน่งจะต้องสามารถคิดพิจารณาในลักษณะ 8 เมกกะเวิร์ด ได้ด้วยซึ่งอยู่ในช่วง 00000_{16} ถึง $FFFFFF_{16}$ เงื่อนไขนี้ โปรแกรมะมีตำแหน่งอยู่ที่ใดที่หนึ่งก็ได้ในช่องว่างของหน่วยความจำขนาด 8 เมกกะเวิร์ด นี้

2.5.5 รีจิสเตอร์สถานะ

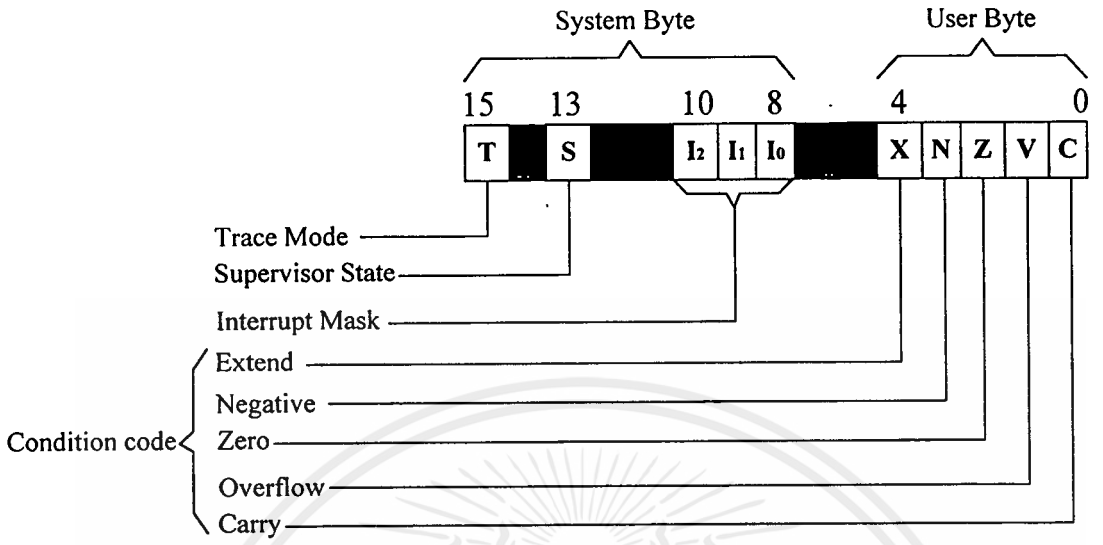
ในรูปที่ 2.10 แสดงรีจิสเตอร์สถานะของ 68000 ไมโครโปรเซสเซอร์ ดังรูปจะเห็นได้ว่า รีจิสเตอร์ถูกแบ่งย่อยออกเป็น 2 ส่วน เรียกว่า User Byte และซิสเต็มไบต์

ในรูปที่ 2.10 แสดงรีจิสเตอร์สถานะในรายละเอียดที่มากขึ้น ในที่นี้จะเห็นได้ว่าบิตที่เป็นผลใน User Byte จะถูกแสดงหรือชี้บ่ง ผลของประมวลผล หลังจากการเอ็กซิคิวท์ของคำสั่ง 5 บิต สถานะเงื่อนไข เรียกว่า แฟล็ก (Flag) ถูกใช้เป็นอุปกรณ์ในการแสดงผลมี ดังนี้

1.แฟล็กตัวทด (Carry Flag) อยู่ในตำแหน่งบิต 0 โดยที่แคร่แฟล็กจะเซ็ตเป็น 1 เมื่อผลของการบวกเกิดการทดเกิดขึ้นหรือทำคำสั่งลบแล้วการลบต้องมีการขอยืม คือ แฟล็กตัวทดจะเซ็ตเป็นศูนย์ เมื่อกระทำคำสั่งการเลื่อนหรือการหมุนข้อมูล โดยที่แคร่แฟล็กจะเป็นบิตที่เกินมาจากการหมุนและเลื่อนข้อมูลจากรีจิสเตอร์หรือหน่วยความจำ

2. แฟล็กค่าเกิน (Overflow Flag) อยู่ในตำแหน่งบิตที่ 1 ถ้าเกิดจะมีการทำทางคณิตศาสตร์ที่มีการคิดเครื่องหมายเกิดขึ้น แล้วผลลัพธ์ไม่ถูกต้องแฟล็กค่าเกินจะเซ็ต ถ้าไม่เช่นนี้จะรีเซ็ตหรือในขณะที่มีการกระทำคำสั่งทางการเลื่อนข้อมูล เพื่อผลทางคณิตศาสตร์ แฟล็กจะเซ็ตเมื่อมีผลของการเปลี่ยนแปลงเกิดขึ้นในบิตที่นัยสำคัญสูงสุด นอกเหนือจากนี้จะรีเซ็ต

3. แฟล็กศูนย์ (Zero Flog) เมื่อผลของการปฏิบัติคำสั่งแล้ว ทำให้ผลลัพธ์เป็นศูนย์ ในบิตที่ 2 ของ SR (Status Register) จะเซ็ตและรีเซ็ต ถ้าผลลัพธ์ไม่เป็นศูนย์



รูปที่ 2.10 รีจิสเตอร์สถานะ

4. แฟล็กค่าลบ (Negative Flog) อยู่ในตำแหน่งบิตที่ 3 ค่าที่เก็บไว้ในบิตที่ 3 นี้ คือ การทำซ้ำของบิตนัยสำคัญสูงของผลลัพธ์เมื่อขณะมีการทำคำสั่งทางคณิตศาสตร์, ทางลอจิก, เลื่อนข้อมูลหรือหมุนข้อมูล ในลักษณะอื่นเมื่อผลลัพธ์เป็นลบบิต N นี้จะเซ็ต เมื่อผลลัพธ์เป็นบวกจะรีเซ็ต

5. แฟล็กขยาย (Extend Flog) อยู่ในตำแหน่งบิตที่ 4 เมื่อขณะที่จะมีการกระทำทางคณิตศาสตร์, ทางลอจิก, เลื่อนข้อมูลหรือหมุนข้อมูล แฟล็กขยายจะได้รับผลของตัวทดสถานะโดยบิตตัวทดที่ได้รับมานั้นจะถูกใช้ในการกระทำการตรวจสอบความถูกต้อง

User Bit เหล่านี้ที่เป็นส่วนประกอบของรีจิสเตอร์สถานะ (Status Register) สามารถทำการทดสอบได้โดยผ่านทางซอฟต์แวร์ เพื่อพิจารณาหรือไม่พิจารณาต่อผลที่ปรากฏขึ้น ถ้าประเภทของผลที่ปรากฏขึ้นสามารถแสดงได้โดยเปลี่ยนแปลงแก้ไขในโปรแกรมได้จากการกำหนดค่าและซิสเต็มไบต์ของ SR ประกอบไปด้วย บิตที่ควบคุมการปฏิบัติการของ 68000 ไมโครโปรเซสเซอร์และประกอบไปด้วย Interrupt Mask บิตเหล่านี้ แสดงให้ดูในรูปที่ 2.10 ต่อไปจะมาดูรายละเอียด

1. Interrupt Mask (I_2 , I_1 , I_0) : บิตที่ 8 ถึง10 ของ SR เป็นบิตที่แสดงอินเตอร์รัพต์ของ 68000 ไมโครโปรเซสเซอร์ 3 บิต จะเป็นการกำหนดการอินเตอร์รัพต์ ซึ่งจะสามารถตอบรับการบริการอินเตอร์รัพต์หรือไม่สนใจ อุปกรณ์ที่ส่งการอินเตอร์รัพต์เข้ามาและมีความ

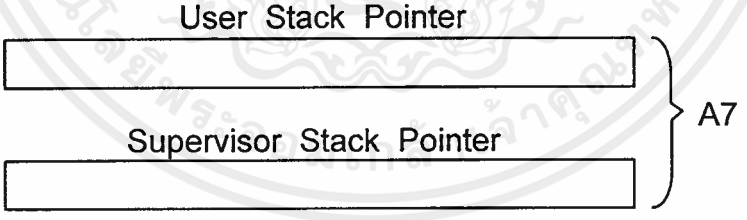
สำคัญสูงสุด แล้วบิต I_2, I_1, I_0 จะทำการตอบรับ และช่วงทำการอินเตอร์รัพต์เกิดขึ้นและยังไม่เสร็จสิ้น มีการส่งอินเตอร์รัพต์เข้ามาอีก จะไม่มีการตอบรับบริการอินเตอร์รัพต์เกิดขึ้นหรือไม่สนใจ ดังตัวอย่างถ้า I_2, I_1, I_0 เท่ากับ 011_2 แล้ว ผลของกลุ่มบิตดังกล่าวมีค่า 4 ถึง 7 จะเกิดการทำงาน หรือมีการบริการอินเตอร์รัพต์เกิดขึ้น แต่ถ้ามีค่าเป็น 1 ถึง 3 จะตรงกันข้าม

2. Supervisor (S) : บิตที่13 ของ SR เป็นบิตที่ใช้ประโยชน์ในการเลือกสถานะของการระหว่าง User และ Supervisor โดยที่ลอจิก "1" ที่แสดงในบิต 8 นี้จะ หมายถึง 68000 ไมโครโปรเซสเซอร์ปฏิบัติการในสถานะ Supervisory ในทางตรงกันข้าม ถ้าเป็นลอจิก "0" จะหมายถึง 68000 ไมโครโปรเซสเซอร์ปฏิบัติการในสถานะ User

3. Trace Mode (T) : สถานะของบิต T นี้ ถูกใช้ในการให้ทำงานหรือไม่ทำงานของ Trace Mode (Single Step) ในสถานะ Single-Step ทำงาน ค่าในบิตที่ 15 จะเซ็ต

2.6 ยูสเซอร์และซุปเปอร์ สแต็คพอยน์เตอร์

ชื่อว่า A7 ลักษณะเป็นรีจิสเตอร์คู่ขนาด 32 บิต ซึ่งเป็นลักษณะพิเศษของซีพียูเบอร์นี้ กล่าวคือ A7 ในความเป็นจริงนี้ จะมีอยู่ 2 รีจิสเตอร์ คือ A7 สำหรับเป็นผู้ใช้ (User Stack Pointer) และ A7 สำหรับเป็นของระบบ (Supervisor Stack Pointer) ตามรูปที่ 2.11

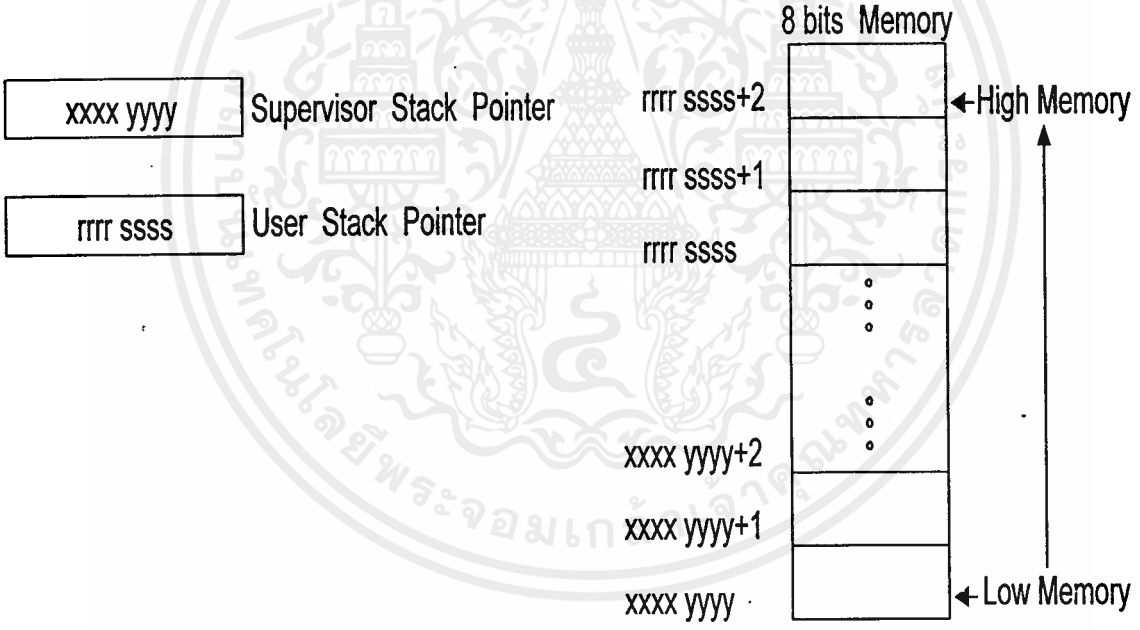


รูปที่ 2.11 สแต็คพอยท์เตอร์ทั้ง 2 ของซีพียู

ซึ่งแตกต่างจากระบบซีพียูเป็นพวกไมโครโปรเซสเซอร์ที่มักจะมีสแต็คพอยท์เตอร์เพียง รีจิสเตอร์เดียว ตัว 68000 จะมีการทำงานได้ใน 2 โหมดคือใน Supervisor Mode และ User Mode และในแต่ละโหมดจะมีสแต็คพอยท์เตอร์ของตัวเองตามรูปที่ 2.11 การเปลี่ยนโหมดกระทำได้โดยคำสั่งทางซอฟต์แวร์ ซึ่งทำให้ตัว 68000 เปลี่ยนการใช้สแต็คพอยท์เตอร์ให้ตรงกับโหมดนั้นโดยอัตโนมัติ แต่มีข้อพิเศษเพิ่มขึ้น เมื่อซีพียูทำงานใน User Mode จะไม่

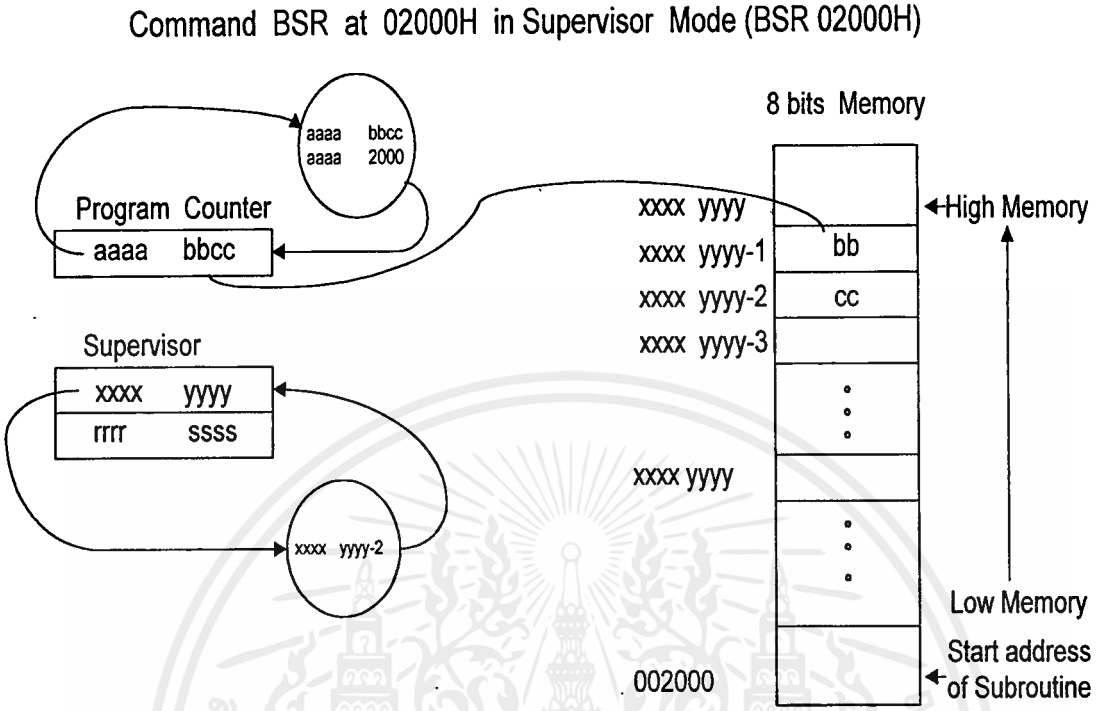
สามารถใช้คำสั่งบางคำสั่งที่มีในชุดคำสั่งได้ ซึ่งคำสั่งพวกนี้ เรียกว่า Privilege Instruction ซึ่งคำสั่งพวกนี้จะใช้ได้เมื่อซีพียูอยู่ใน Supervisor Mode เท่านั้น

ลักษณะการเก็บข้อมูลต่างๆ ลงในบริเวณสแต็คนั้นกระทำเช่นเดียวกับซีพียูทั่วๆ ไป กล่าวคือ ถ้าเราเก็บข้อมูลลงในสแต็ค หรือการทำงาน Push Stack ถ้าเป็นการ Push ค่าของโปรแกรมเคาน์เตอร์ ในกรณีเรียกใช้โปรแกรมย่อย สแต็คพอยท์เตอร์จะลดลง 4 ตำแหน่ง เพราะโปรแกรมเคาน์เตอร์ยาว 32 บิต จะใช้เนื้อที่ในการเก็บ 4 ไบต์ และในการเรียกค่าจากสแต็ค (Pop Stack) ก็เช่นกัน สแต็คพอยท์เตอร์จะเพิ่มขึ้น 4 ตำแหน่ง และ ค่า 4 ตำแหน่งนี้ ก็นำไปบรรจุในรีจิสเตอร์ขนาด 32 บิตที่ต้องการได้โดยลักษณะการเก็บข้อมูลดังที่กล่าวมาจะเป็นดังรูปที่ 2.12



รูปที่ 2.12 การอ้างหน่วยความจำของสแต็คพอยท์เตอร์

ถ้าเรา push ค่าของสแต็ค โดยการเรียกโปรแกรมย่อยลำดับ การทำงานตามรูปที่ 2.12 จะเห็นได้ว่า rrrr ssss ใน User Stack Pointer จะไม่มีการเปลี่ยนแปลง แต่ถ้า BSR (Branch to Subroutine) นี้ ถูกเอ็กซิคิวท์ใน User Mode แล้ว xxxx yyyy จะไม่เปลี่ยนแปลง แต่ rrrr ssss จะเปลี่ยนแปลงแทน



รูปที่ 2.13 การเปลี่ยนแปลงต่างๆ ภายในสแต็ค

ข้อสังเกตเล็กน้อยจะเห็นที่การเก็บค่าของโปรแกรมเคาน์เตอร์ในหน่วยความจำที่ Stack ชี้อยู่นั้น จะเก็บเพียง 16 บิตเท่านั้น (bbcc ในรูปที่ 2.13) ส่วนอีก 16 บิตบน (aaaa ในรูปที่ 2.13) จะไม่มีการเปลี่ยนแปลงค่าของโปรแกรมเคาน์เตอร์ช่วง 16 บิตบนเกิดขึ้นจะทำให้มีการผิดพลาดในการกลับมาที่ตำแหน่งที่จากไปได้

2.7 ชุดคำสั่งของ 68000

การใช้คำสั่งของ 68000 ผู้ใช้จะต้องรู้คำสั่งพื้นฐานต่างๆ เพื่อจะได้อำนวยความสะดวกในการนำไปใช้งาน โดยจะมีคำสั่งพื้นฐานในด้านต่างๆ ดังต่อไปนี้

2.7.1 คำสั่งในการโอนย้ายข้อมูล

คำสั่งพื้นฐานในการเคลื่อนย้ายข้อมูลมี Move (MOVE), Move Multiple (MOVEM), Load Effective Address (LEA), Exchange (EXG), Swap (SWAP) และ Clear (CLR)

1. คำสั่งในการโอนย้ายข้อมูล (Move Instruction-MOVE)

คำสั่งนี้มีความสามารถในการเคลื่อนย้ายข้อมูล คือ เคลื่อนย้ายข้อมูลจากรีจิสเตอร์กับรีจิสเตอร์ระหว่างรีจิสเตอร์กับหน่วยความจำดังในตารางที่ 2.1 จะเห็นได้ว่ามีอยู่ 8 รูปแบบที่มีความแตกต่างกันของคำสั่ง คือ ขนาดของข้อมูลที่ทำการประมวลผลและขนาดของข้อมูลที่เข้าถึง

รูปแบบแรกของ คำสั่ง MOVE คือ

MOVE EAs , EAd

จะอนุญาตให้เคลื่อนย้ายข้อมูลต้นทางที่ชี้ โดยตำแหน่งข้อมูลใช้งาน Eas ข้อมูลที่ถูกชี้โดยตำแหน่งข้อมูลใช้งาน Ead ข้อมูลต้นทางและปลายทางสามารถเก็บอยู่ในรีจิสเตอร์ ข้อมูล, แอดเดรสรีจิสเตอร์หรือเก็บอยู่ในหน่วยความจำ คำสั่งยังสามารถถูกใช้ประมวลผลข้อมูลในแบบไบต์, เวิร์ดหรือลองเวิร์ด

เมื่อใช้คำสั่งนี้ดำเนินการข้อมูลแบบเวิร์ดหรือลองเวิร์ดแล้ว ข้อมูลต้นทางสามารถกำหนดลักษณะเฉพาะโดยใช้ทุกการอ้างถึงตำแหน่งแบบต่างๆ (Addressing Mode) แต่ถ้าเป็นการดำเนินการข้อมูลแบบไบต์แล้ว การอ้างตำแหน่งแบบรีจิสเตอร์ไม่สามารถใช้ได้ เพราะว่าแอดเดรสรีจิสเตอร์สามารถเข้าถึงข้อมูลในลักษณะเวิร์ด, ลองเวิร์ด เท่านั้น แฟล็กแสดงสถานะ (Status Register) จะมีผลหลังจากเอ็กซิคิวท์คำสั่ง MOVE โดยบิตเงื่อนไขที่มีผล คือ บิตค่าลบ, บิตศูนย์, บิตค่าเกินและบิตตัวทด โดยที่บิตค่าลบและจะบิตศูนย์ เซ็ตหรือรีเซ็ตขึ้นอยู่กับผลลัพธ์ของคำสั่ง ดังตัวอย่างของคำสั่ง MOVE ที่ดำเนินการลักษณะเวิร์ด

MOVE . W D0,D1

ข้อมูลต้นทางที่อยู่ใน D_0 ซึ่งลักษณะการอ้างตำแหน่งแบบรีจิสเตอร์ข้อมูล ในที่นี้ ว่าข้อมูลที่อยู่ D_0 คือ 12345678_{16} ส่วน D_1 ซึ่งเป็นปลายทางก็ใช้ลักษณะการอ้างตำแหน่งแบบรีจิสเตอร์ข้อมูล เช่นกัน หลังจากทำการเอ็กซิคิวท์คำสั่งนี้แล้ว เวิร์ดนัยสำคัญต่ำ D_0 คือ 5678_{16} จะถูกทำซ้ำลงใน 16 บิตต่ำของ D_1 เมื่อผลลัพธ์ใน D_1 เป็นบวกและไม่เท่ากับศูนย์ บิตเงื่อนไขจะมีผลตามนี้คือ : N=0 , Z=0 , C=0 และ X ไม่มีผล

คำสั่งนี้จะยอมให้ส่วน Condition Code ของสถานะรีจิสเตอร์ถูกกำหนดลักษณะเป็นข้อมูลปลายทาง ข้อมูลที่ถูกชี้คือ RCC การอ้างถึงตำแหน่งข้อมูลทุกๆ แบบสามารถถูกใช้ในข้อมูลต้นทางคำสั่งนี้สามารถใช้โหลดข้อมูลในหน่วยความจำหรือรีจิสเตอร์ภายในไป

User Byte ของ SR (Status Register) เมื่อไรก็ตามที่ขนาดของข้อมูลต้นทางถูกกำหนดลักษณะแบบเวิร์ดมีเพียงแค่ 8 บิต นัยสำคัญต่ำเท่านั้น ที่ถูกใช้ในการเปลี่ยนแปลงรหัสเงื่อนไขใน SR

ตารางที่ 2.1 คำสั่งในการโอนย้ายข้อมูล

Mnemonic	Meaning	Type	Operand Size	Operation
MOVE	Move	MOVE Eas,Ead	8,16,32	(Eas) ⟶ EAd
		MOVE EA,CCR	8	(EA) ⟶ CCR
		MOVE EA,SR	16	(EA) ⟶ SR
		MOVE SR,EA	16	SR ⟶ EA
		MOVE USP,An	32	USP ⟶ An
		MOVE An,USP	32	An ⟶ USP
		MOVE EA,An	16,32	(EA) ⟶ An
		MOVEQ #XXX,Dn	8	#XXX ⟶ Dn
MOVEM	Move Multiple	MOVEM Reg_list,EA	16,32	Reg_list⟶EA
		MOVEM EA,Reg_list	16,32	(EA) ⟶Reg_list
LEA	Load effective Address	LEA EA,An	32	EA ⟶ An
EXG	Exchange	EXG Rx,Ry	32	Rx ⟷Ry
SWAP	Swap	SWAP Dn	16	Dn31:0⟷Dn15:0
CLR	Clear	CLR EA	8,16,32	0 ⟶EA

คำสั่งถัดไปของคำสั่ง MOVE มีไว้เพื่อจัดค่าเริ่มต้น แก่สถานะรีจิสเตอร์

MOVE EA,CCR

คำสั่งที่สามคือ

MOVE EA,SR

คำสั่งนี้ถูกใช้โหลดทั้ง 16 บิต ของ สถานะรีจิสเตอร์ เพราะฉะนั้นจะถูกโหลดลงทั้งคู่ของ System Byte และ User Byte ในเมื่อคำสั่งนี้จะเป็นการอับเดท ไบต์นัยสำคัญสูงใน SR จะถูกทำการเอ็กซิคิวท์เมื่อ 68000 อยู่ใน Supervisor State (เป็นคำสั่งสิทธิพิเศษ)

ตัวอย่างที่ 1

ผลลัพธ์อะไรจะเกิดขึ้นของการเอ็กซิคิวท์ตามลำดับของคำสั่ง

```
MOVE.W # 12,D0
MOVE    D0,SR
```

ให้คิดว่า 68000 ทำงานในสถานะ Supervisor State

เอ็กซิคิวท์ของคำสั่งแรกจะโหลดเวิร์ดบิตล่างของ D_0 ด้วยค่า 12_{10}

$12_{10} = 000C_{16} = 0000\ 0000\ 0000\ 1100_2$

หลังจากเอ็กซิคิวท์คำสั่งนี้แล้ว รหัสเงื่อนไขของ SR เป็นตามนี้

X = ไม่เปลี่ยนแปลง

N = 0

Z = 0

V = 0

C = 0

ในคำสั่งที่สองจะขึ้นอยู่กับสถานะการทำงานของ 68000 ในที่นี้ ให้คิดว่าทำงานในสถานะ Supervisor State, SR โหลดใส่ด้วย เวิร์ดล่าง ของ D_0 คือ $0000\ 0000\ 0000\ 1100_2$

$D0 = XXXXXXXXXXXXXXXX\ 0000\ 0000\ 0000\ 1100_2$

$SR = 0000\ 0000\ 0000\ 1100_2$

ส่วนผลของรหัสเงื่อนไขจะเป็นดังนี้

X = 0

N = 1

Z = 1

V = 0

C = 0

คำสั่งถัดไปของคำสั่ง MOVE ที่แสดงดังในตารางที่ 2.1 คือ

```
MOVE  SR , EA
```

คำสั่งนี้ข้อมูลต้นทางจะถูกเก็บใน SR เสมอ ส่วนข้อมูลปลายทางจะอยู่ในลักษณะตำแหน่งของข้อมูลใช้งาน เพราะฉะนั้นแล้วคำสั่งนี้จะยอมให้โปรแกรมเมอร์บันทึกคำสั่งที่ถูกเก็บในสถานะรีจิสเตอร์ลงใน แอดเดรสรีจิสเตอร์, รีจิสเตอร์ข้อมูลหรือในหน่วยความจำ

ดังตัวอย่างนี้

MOVE SR , D7

ในคำสั่งนี้เป็นการทำซ้ำข้อมูลที่บรรจุใน SR ไปที่รีจิสเตอร์ข้อมูล D_7 รหัสเงื่อนไขจะไม่มีผลในการเอ็กซิคิวท์คำสั่งนี้ ในเมื่อคำสั่งนี้ไม่ได้ทำการเปลี่ยนแปลงข้อมูลที่บรรจุ อยู่ใน 68000 สามารถเอ็กซิคิวท์ เมื่ออยู่ในสถานะใดสถานหนึ่ง User State หรือ Supervisor State

คำสั่ง MOVE User Stack Pointer แสดงในตารางที่ 2.1 คือ

MOVE USP , An

และ

MOVE An , USP

คำสั่งเคลื่อนย้ายข้อมูลระหว่าง User Stack Pointer (USP) รีจิสเตอร์กับตำแหน่งรีจิสเตอร์ สำหรับเหตุนี้ คำสั่งนี้ ใช้สำหรับทำการอ่านหรือเปลี่ยนแปลง USP ตามลำดับ เนื่องจาก USP เป็นรีจิสเตอร์ขนาด 32 บิต เพราะฉะนั้นข้อมูลต้นทางและข้อมูลปลายทางจะต้องมีขนาดลองเวิร์ดเสมอ คำสั่งทั้งสองที่แสดงให้เห็นเป็นคำสั่งประเภทที่มีประสิทธิภาพพิเศษและจำต้องเอ็กซิคิวท์ เมื่อ 68000 ทำงานในสถานะ Supervisor State เท่านั้น

คำสั่ง MOVE อีกคำสั่งที่มีประสิทธิภาพในการโหลดข้อมูลให้แอดเดรสรีจิสเตอร์จากแอดเดรสรีจิสเตอร์หรือรีจิสเตอร์ข้อมูลในหน่วยความจำที่แสดงในตารางที่ 2.1

MOVE EA , An

คำสั่งนี้จะอนุญาตให้ข้อมูลที่มีขนาด 16 บิตหรือ 32 บิต อย่างใดอย่างหนึ่ง ถ้าข้อมูลต้นทางถูกกำหนดลักษณะเป็นแบบเวิร์ด บิตเครื่องหมายของเวิร์ด จะถูกทำซ้ำไปจนครบตั้งแต่บิตที่16-31 เพื่อให้เป็นข้อมูลลักษณะแบบลองเวิร์ด ก่อนถูกโอนย้ายไปยังปลายทาง

ข้อมูลต้นทางสามารถกำหนดลักษณะโดยการอ้างถึงตำแหน่งทุกๆ แบบของ 68000 ดังตัวอย่างของคำสั่ง

MOVE.L (A0) , A6

คำสั่งนี้ใช้การอ้างถึงตำแหน่งแอดเดรสรีจิสเตอร์แบบผ่าน ผลของการเอ็กซิคิวท์คำสั่งเอาข้อมูลชนิดลองเวิร์ด ที่เก็บไว้ในหน่วยความจำ ที่ถูกชี้โดยแอดเดรสรีจิสเตอร์ทำการโหลดลงใน แอดเดรสรีจิสเตอร์ A_6 และรหัสเงื่อนไขไม่มีผลต่อการเอ็กซิคิวท์คำสั่ง

ลำดับสุดท้ายของคำสั่ง MOVE ที่แสดงไว้ในตารางที่ 2.1 คือ

MOVEQ # XXX , Dn

คำสั่งนี้คือการโอนย้ายอย่างเร่งด่วน (Move Quick) ซึ่งถูกใช้ในการโหลดข้อมูลลงรีจิสเตอร์ข้อมูลด้วยไบต์ข้อมูลแบบ Immediate ดังตัวอย่างที่แสดงต่อไป

MOVEQ # 4 , D1

ข้อมูลแบบทันทีทันใด (Immediate Operand) ซึ่งก็คือเลข 4 ในฐานสิบ ซึ่งจะถูกเข้ารหัสเพื่อกระทำการแบบเวิร์ดเมื่อคำสั่งนี้ถูกเอ็กซิคิวท์แล้ว ข้อมูลแบบทันทีทันใดจะถูกโหลดลงสู่รีจิสเตอร์ข้อมูล D_1 อย่างไรก็ดีก่อนที่ข้อมูลจะถูกโหลดบิตเครื่องหมายจะถูกทำซ้ำ ให้ครบบิตที่เหลือจนครบ 32 บิตก่อน เพราะฉะนั้นค่าที่เก็บใน D_1 คือ 00000004_{16}

2. Move Multiple Register Instruction - MOVEM

คำสั่งนี้จะมีประสิทธิภาพสำหรับทำการบันทึกข้อมูลที่จะเก็บไว้ ภายในรีจิสเตอร์ลงในหน่วยความจำหรือทำการคืนค่าดังกล่าวจากหน่วยความจำ เช่น ใช้คำสั่งนี้ในการกำหนดกลุ่มข้อมูลเริ่มต้นของรีจิสเตอร์จากภายในหน่วยความจำการกระทำนี้ สามารถใช้ชุดคำสั่งของ MOVE หรือใช้เพียงแค่ MOVEM เพียงคำสั่งเดียว

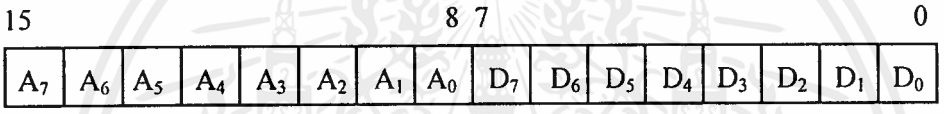
การกระทำการแบบอื่นของคำสั่งนี้ที่มีประโยชน์มาก คือ เมื่อใช้ทำงานในลักษณะโปรแกรมย่อย (Subroutine) ดังตัวอย่าง ถ้าเกิดการเรียกโปรแกรมย่อยเกิดขึ้นแล้ว ข้อมูลที่เกิดในรีจิสเตอร์จะถูกบันทึกลงในหน่วยความจำ หลักจากที่เอ็กซิคิวท์คำสั่งข้อมูลที่เก็บในรีจิสเตอร์จะถูกคืนค่าอีกครั้งจากหน่วยความจำ เมื่อโปรแกรมควบคุมกลับสู่โปรแกรมหลักข้อมูลที่อยู่ในรีจิสเตอร์เป็นข้อมูลเดิมที่เก็บไว้ก่อนที่จะเรียกโปรแกรมย่อยออกมาโดยที่จะเลือกอย่างใดอย่างหนึ่งระหว่างบันทึกหรือการคืนค่า จะดำเนินการได้โดยใช้คำสั่ง MOVEM มี 2 แบบฟอร์ม

แบบฟอร์มแรกคือ

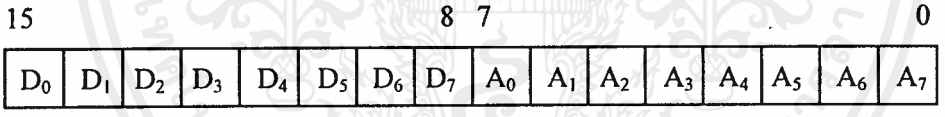
MOVEM Reg-list , EA

คำสั่งนี้ใช้ในการบันทึกข้อมูลที่เก็บในรีจิสเตอร์ในลักษณะ Register List เป็นข้อมูลที่เก็บในลักษณะต่อเนื่องภายในรีจิสเตอร์ทำการเก็บลงในหน่วยความจำโดยจะบันทึกโดยตำแหน่งที่เริ่มเก็บในหน่วยความจำจะมาจากข้อมูลปลายทาง

Register List นี้ จะเป็นได้ทั้งข้อมูลและแอดเดรสรีจิสเตอร์โดยที่ List ของรีจิสเตอร์จะถูกเข้ารหัสเข้าเก็บในเวิร์ดที่สองของคำสั่งโดยที่เวิร์ดนี้ มีชื่อเรียกว่า Register List Mask ซึ่งแสดงในรูปที่ 2.14 ซึ่งแต่ละบิต ดังกล่าวจะสอดคล้องกับรีจิสเตอร์ภายในของ 68000 โดยที่ถ้าเราต้องเซ็ตบิตเป็น "1" จะเป็นการแสดงว่ารีจิสเตอร์ที่สอดคล้องกันใน List รวมอยู่ด้านใน List และถ้าเป็น "0" จะไม่รวมอยู่ด้วยใน List



รูปที่ 2.14 แบบข้อมูลของรีจิสเตอร์ที่เก็บในลักษณะต่อเนื่องภายในรีจิสเตอร์แบบเพิ่มค่า



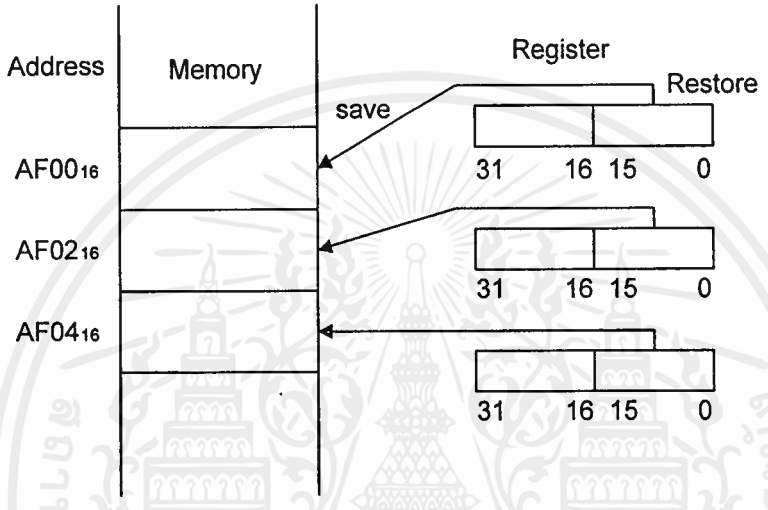
รูปที่ 2.15 แบบข้อมูลของรีจิสเตอร์ที่เก็บในลักษณะต่อเนื่องภายในรีจิสเตอร์แบบลดค่า

รีจิสเตอร์ข้อมูล D_0 และ D_7 ให้จะสอดคล้องกับบิต 0 ถึง 7 ของ Mask ตามลำดับตำแหน่ง รีจิสเตอร์ A_0 ถึง A_7 จะสอดคล้องกับบิต 8 ถึง 15 ตามลำดับ รีจิสเตอร์ที่สอดคล้องกับบิตแรกถูกบันทึกแล้วบิตต่อๆ มาที่สอดคล้องกับตำแหน่งของรีจิสเตอร์ก็จะตามลำดับเรื่อยไปจนถึงบิตสุดท้าย

คำสั่งนี้เขียนขึ้นเพื่อดำเนินการในลักษณะเวิร์ดหรือลองเวิร์ด ในการเคลื่อนย้ายข้อมูลในแบบเวิร์ดจะเปลืองเนื้อที่ในหน่วยความจำ 1 เวิร์ด ในรูปที่ 2.16 แสดงเหตุการณ์ที่เกิดขึ้นในขณะที่ประมวลผล 2 คำสั่งดังกล่าว

รูปแบบที่สองของคำสั่ง MOVEM ที่แสดงในตารางที่ 2.1 จะอนุญาตให้รีจิสเตอร์ภายในของ 68000 เป็นตัวกำหนดค่าเริ่มต้นหรือเป็นการคืนค่าจากหน่วยความจำ ซึ่งจะเขียนได้ดังนี้

MOVEM EA , Reg-list



รูปที่ 2.16 การดำเนินการเก็บและคืนค่าข้อมูลระหว่างรีจิสเตอร์กับหน่วยความจำด้วยคำสั่ง MOVEM

ตัวอย่างที่ 2

MOVEM.W D0 / D1 / A5 , $AF00

คำสั่งนี้เป็นการบันทึก 8 บิตล่างในข้อมูลแบบเวิร์ดของรีจิสเตอร์ D_0, D_1 และA_5 ลงในหน่วยความจำที่เวิร์ดตำแหน่ง $AF00_{16}$, $AF02_{16}$ และ $AF04_{16}$ ตามลำดับ สำหรับการคืนค่ารีจิสเตอร์ คำสั่งจะเขียนได้ดังนี้

MOVEM.W $AF00 , D0 / D1 / A5

3. คำสั่งในการโหลดตำแหน่งข้อมูลใช้งาน (Load Effective Address Instruction)

LEA EA , An

ผลของการเอ็กซิคิวท์คำสั่งนี้ ไม่ใช่ทำการโหลด ข้อมูลปลายทางด้วยข้อมูลที่เก็บอยู่ในต้นทาง แต่เป็นการคิดตำแหน่งของข้อมูลใช้งานที่ใช้เป็นตำแหน่งฐานบวกกับค่าที่จะไปรวมด้วย แล้วนำไปเก็บในตำแหน่งปลายทาง

ตัวอย่างที่ 3 อธิบายเหตุการณ์ที่เกิดขึ้นของคำสั่ง

LEA $6(A_1, D_0), A_2$

ให้คิดว่า $A_1 = 00004000_{16}$

และ

$D_0 = 000012AB_{16}$

คำสั่งนี้จะใช้การอ้างตำแหน่งแบบผ่านรีจิสเตอร์ (Address Register Indirect) ด้วยการอ้างตำแหน่งแบบดัชนี (Index Addressing) สำหรับข้อมูลต้นทาง ส่วนปลายทางจะเป็นแอดเดรสรีจิสเตอร์ A_2 หลังจากการเอ็กซิคิวท์ ของคำสั่งนี้ A_2 จะถูกโหลดด้วยตำแหน่งข้อมูลใช้งาน (Effective Address)

$$A_2 = A_1 + D_0 + 6_{10}$$

$$A_2 = 00004000_{16} + 000012AB_{16} + 6_{16}$$

$$= 000052B1_{16}$$

4. คำสั่งในการแลกเปลี่ยนข้อมูล (Exchange Instruction - EXG)

EXG Rx , Ry

ในที่นี้ Rx และ Ry อยู่ในลักษณะตามใจชอบในการเลือกระหว่างข้อมูลหรือตำแหน่งรีจิสเตอร์ ตัวอย่างของคำสั่งนี้คือ

EXG D0 , A3

คำสั่งนี้ทำการโหลดรีจิสเตอร์ข้อมูล D_0 ด้วยข้อมูลที่เก็บอยู่ในแอดเดรสรีจิสเตอร์ A_3 และ A_3 ก็จะถูกโหลดข้อมูลก่อนที่ D_0 จะถูกเปลี่ยน เช่น ถ้า D_0 เก็บข้อมูล $FFFFFFFF_{16}$ และ A_3 จะเก็บข้อมูล 00000000_{16} ผลจากการเอ็กซิคิวท์คำสั่งนี้ D_0 จะเก็บข้อมูล 00000000_{16} และ A_3 จะเก็บข้อมูล $FFFFFFFF_{16}$ การโอนถ่ายข้อมูลจะทำการใช้ทั้งหมด 32 บิต เสมอ และ Condition Code จะไม่มีผลต่อคำสั่งนี้

5. คำสั่ง SWAP

คำสั่ง SWAP มีความคล้ายคลึงกับคำสั่งในการแรกเปลี่ยนข้อมูล โดยที่คำสั่งนี้มีความสามารถในการเปลี่ยนค่า 2 ค่า คำสั่งนี้จะใช้แลกเปลี่ยนเวิร์ดข้อมูล 8 บิตบนกับ 8 บิตล่างในรีจิสเตอร์ข้อมูล รูปแบบทั่วไปของคำสั่ง SWAP ที่แสดงในตารางที่ 2.1 คือ

SWAP Dn

ตัวอย่างคือ

SWAP Dn

เมื่อทำการเอ็กซิคิวท์คำสั่งนี้แล้วข้อมูลที่เก็บใน 8 บิตล่างข้อมูล16 บิต ของ D_0 จะทำการแลกเปลี่ยนกับ 8 บิตบนของข้อมูล 16 บิต ถ้าข้อมูลที่บรรจุใน D_0 คือ $FFFF0000_{16}$ หลังจากเอ็กซิคิวท์คำสั่งนี้แล้วผลใน D_0 จะเป็นค่า $0000FFFF_{16}$ ผลของค่า 32 บิต ใน D_0 หลังจากทำคำสั่ง Swap แล้ว จะมีผลในการเซ็ตหรือการรีเซ็ตของรหัสเงื่อนไข

6. คำสั่งเคลียร์ (Clear Instruction - CLR)

คำสั่ง CLR จะเป็นการกำหนดค่าเริ่มต้นข้อมูลที่เก็บอยู่ในรีจิสเตอร์ภายในหรือที่เก็บข้อมูลในหน่วยความจำให้เป็นศูนย์ รูปแบบทั่วไปของคำสั่งนี้คือ

CLR EA

สามารถปฏิบัติการในระดับไบต์, เวิร์ด, ลองเวิร์ด สามารถที่จะใช้กับการอ้างถึงข้อมูลทุกๆ แบบ เสียแต่การอ้างถึงตำแหน่งแอดเดรสรีจิสเตอร์ เช่น ในการเคลียร์ 8 บิตนัยสำคัญต่ำของ D_0 ทำได้ดังคำสั่งนี้

CLR.B D0

2.7.2 คำสั่งการกระทำทางคณิตศาสตร์

เซ็ตของคำสั่งนี้ จะเป็นการจัดให้ 68000 ดำเนินการกระทำการทางคณิตศาสตร์แบบเลขฐานสอง เช่น การบวก, ลบ, คูณและหาร คำสั่งเหล่านี้สามารถประมวลผลในแบบคิดเครื่องหมายและไม่คิดเครื่องหมาย อีกทั้งข้อมูลที่ใช้ในการประมวลผลจะทำได้ในลักษณะไบต์, เวิร์ดหรือลองเวิร์ด กลุ่มของคำสั่งนี้แสดงในตารางที่ 2.2

รหัสเงื่อนไขใน RS Register จะทำการเซ็ตหรือการรีเซ็ตขึ้นอยู่กับการกระทำของคำสั่งทางคณิตศาสตร์ สำหรับสั่งการ ADD, SUR และ NEG 8 บิตรหัสเงื่อนไขจะมีผลตามนี้

N จะเซ็ต เมื่อผลลัพธ์เป็นค่าลบ ถ้าไม่เช่นนั้นจะ เคลียร์

Z จะเซ็ต เมื่อผลลัพธ์เป็นศูนย์ ถ้าไม่เช่นนั้นจะ เคลียร์

V จะเซ็ต เมื่อปรากฏแฟล็กค่าเกิน ถ้าไม่เช่นนั้นจะ เคลียร์

X และ C จะเซ็ต การบวกเกิดมีการยืมเกิดขึ้น ถ้าไม่เช่นนั้นจะ เคลียร์

สำหรับคำสั่ง MUL, DIV และ EXT บิตเงื่อนไข V และ C จะเคลียร์เสมอ, X ไม่มีผล, N และ Z จะเซ็ตหรือเคลียร์ขึ้นอยู่กับผลลัพธ์ที่ได้

ตารางที่ 2.2 คำสั่งประมวลผลทางคณิตศาสตร์

Mnemonic	Meaning	Type	Operand Size	Operation
ADD	Add	ADD EA , Dn	8,16,32	(EA) + Dn ⟶ Dn
		ADD Dn , EA	8,16,32	Dn + (EA) ⟶ EA
		ADDI #XXX , EA	8,16,32	CCR ⟶ SR
		ADDQ #XXX , EA	8,16,32	SR ⟶ EA
		ADDX Dy , Dx	8,16,32	USP ⟶ An
		ADDX ^(Ay) ,^(Ax)	8,16,32	An ⟶ USP
		AXXA EA , An	16,32	(EA) ⟶ An
SUB	Subtract	SUB EA , Dn	8,16,32	Dn - (EA) ⟶ Dn
		SUB Dn , EA	8,16,32	(EA) - Dn ⟶ EA
		SUBI #XXX , EA	8,16,32	(EA) - #XXX ⟶ EA
		SUBQ #XXX , EA	8,16,32	(EA) - #XXX ⟶ EA
		SUBX Dy , Dx	8,16,32	Dx - Dy - X ⟶ Dx
		SUBX ^(Ay) , ^(Ax)	8,16,32	^(Ax)-^(Ay)-X⟶ (Ax)
		SUBA EA , An	16,32	An - (EA) ⟶ An
NEG	Negate	NEG EA	8,16,32	0 - (EA) ⟶ EA
		NEGX EA	8,16,32	0 - (EA) - X ⟶ EA
MUL	Multiply	MULS EA , Dn	16	(EA) * Dn ⟶ Dn
		MULS EA , Dn	16	(EA) * Dn ⟶ Dn
DIV	Divide	DUVS EA , Dn	32 ÷16	Dn ÷(EA) ⟶ Dn
		DIVU EA , Dn	32 ÷16	Dn ÷(EA) ⟶ Dn
EXT	Extend Sign	EXT.W Dn	8 ⟶16	Dn byte ⟶ Dn word
		EXT.L Dn	16 ⟶32	Dn word ⟶ Dn long word

2. กลุ่มคำสั่งกระทำการลบ (Subtraction Instruction–SUB, SUBI, SUBQ, SUBX และ SUBA)

คำสั่งในการลบประกอบด้วย 5 รูปแบบเบื้องต้น ในแต่ละรูปแบบของคำสั่งแบบลบจะมีคำอธิบายเหมือนกันทุกอย่างกับคำสั่งการบวก ดังในตารางที่ 2.2 ที่แสดงให้เห็น ต่อไปจะเป็นรายละเอียดของแต่ละคำสั่งต่อไป

รูปแบบที่ 1 รูปทั่วไปของคำสั่งการลบ (SUB) ของ 68000 คือ

SUB EA , Dn

หรือ

SUB Dn , EA

รูปแบบแรกอนุญาตให้ข้อมูลที่เก็บภายในรีจิสเตอร์หรือตำแหน่งในหน่วยความจำที่จะทำการลบกับข้อมูลที่เก็บในรีจิสเตอร์ข้อมูลการทำงานสามารถเขียนเป็นสูตรได้คือ

$$Dn - (EA) \longrightarrow Dn$$

ดังตัวอย่างของคำสั่งนี้

SUB D0 , D1

จะดำเนินการลบระหว่างรีจิสเตอร์กับรีจิสเตอร์ผลลัพธ์ของ $D_1 - D_0$ จะเก็บไว้ใน D_1

รูปแบบที่ 2 ของคำสั่งลบจะดำเนินการที่ตรงกันข้ามกับคำสั่งแรก ก็คือ ข้อมูลต้นทางจะเป็นตำแหน่งข้อมูลใช้งาน ในรีจิสเตอร์ข้อมูลหรือตำแหน่งในหน่วยความจำหรือในแอดเดรสรีจิสเตอร์

รูปแบบที่ 3 และ 4 ของคำสั่งการลบที่แสดงไว้ในตารางที่ 2.2 คือ Subtract Immediate (SUBI) และ Subtract Quick (SUBQ) จะอนุญาตให้การเข้าถึงข้อมูลแบบทันทีทันใดใน Program Memory ณระทำการลบกับข้อมูลปลายทางที่ถูกชี้โดยตำแหน่งข้อมูลใช้งาน ข้อมูลปลายทางนี้จะเก็บในรีจิสเตอร์ข้อมูลหรือที่ตำแหน่งในหน่วยความจำการกระทำของคำสั่งนี้จะเหมือนกับคำสั่งการบวก เว้นเสียแต่ผลลัพธ์ที่ได้จะเป็นการลบระหว่างข้อมูลต้นทางและข้อมูลปลายทางแทน

ดังตัวอย่าง

SUBI.W # $ 1234,DO

ในคำสั่งนี้ ค่า 1234_{16} จะเป็นตัวลบจากข้อมูลที่เก็บใน D_0 ในที่นี้ให้คิดว่า D_0 เก็บค่า $0000FFFF_{16}$ และผลลัพธ์ที่ได้เก็บใน D_0

$$FFFF_{16} - 1234_{16} = 0000EDCB_{16}$$

รูปแบบที่ 5 ของคำสั่งการลบ คือ Extend Subtract (SUBX) เหมือนกับคำสั่ง ADDX ก็คือการกระทำคำสั่งจะเป็นการรวม Bit Extend (X) ของ SR ในการลบด้วย ดังตัวอย่าง

SUBX Dy , Dx

การดำเนินการของคำสั่งคือ

$$Dx - Dy - X \longrightarrow Dx$$

ดังตัวอย่างถ้า D_1 และ D_2 เก็บค่า 76543210_{16} และ $0000EDCB_{16}$ ตามลำดับและบิตเพิ่มขยายเก็บค่า 1 ผลลัพธ์ของการเอ็กซิคิวท์คำสั่งนี้ คือ

SUBX.W D1 , D2

ถ้า D_0 เก็บค่า $12_{10} = 00010010_2$, D_1 เก็บค่า $37_{10} = 00110111_2$ และบิต X เคลียร์เอ็กซิคิวท์ของคำสั่งนี้ จะได้ผลบวกเป็น

$$D_2 = ABCD_{16} - 3210_{16} - 1_{16}$$
$$= 000079BC_{16}$$

รูปแบบที่ 6 ของคำสั่งการลบ คือ การกระทำการลบค่าในแอดเดรสรีจิสเตอร์(SUBA) ดังแสดงในตารางที่ 2.2 มีประโยชน์ที่ใช้ในการปรับปรุงค่าแอดเดรสรีจิสเตอร์ ใน A_0 ถึง A_6 ดังตัวอย่าง ในการลบข้อมูลที่เก็บในรีจิสเตอร์ข้อมูล D_7 จากแอดเดรสรีจิสเตอร์ A_5 ด้วยคำสั่ง

SUBA D7 , A5

3. กลุ่มคำสั่งกระทำการติดลบ (Negate Instruction - NEG และ NEGX)

รูปแบบอื่นๆ ของคำสั่งการกระทำทางคณิตศาสตร์ (Arithmetic Instruction) คือคำสั่ง Negate คำสั่งนี้มี 2 รูปแบบ ดังแสดงในตารางที่ 2.2 คำสั่ง Negate มีความคล้ายคลึงกับคำสั่งการลบ มีลักษณะเฉพาะ คือ การกำหนดให้ข้อมูลตัวตั้งเป็นศูนย์เสมอ เพราะฉะนั้นการลบที่มีตัวตั้งเป็นศูนย์ผลลัพธ์ที่ได้จะเป็นลบ

คำสั่ง Negate (NEG) เบื้องต้น คือ

NEG EA

ตัวอย่างของคำสั่งคือ

EG.W D0

ถ้าข้อมูลที่เก็บใน D_0 คือ $00FF_{16}$ ผลลัพธ์ของการเอ็กซิคิวท์คำสั่งนี้คือ $FF01_{16}$

Negate With Extend (NEGX) มีความแตกต่างเล็กน้อยจากคำสั่ง NEG คือจะทำการลบทั้งข้อมูลและ Bit Extend (X) จากศูนย์ นั่นคือ

$$0 - (EA) - X \longrightarrow EA$$

ทั้งสองคำสั่งนี้จะเขียนเพื่อกระทำการกับข้อมูลทั้งแบบไบต์, เวิร์ด และลองเวิร์ด

4. กลุ่มคำสั่งกระทำการคูณ (Multiplication Instruction - MULS และ MULU)

68000 จัดหาคำสั่งที่ดำเนินการคณิตศาสตร์ทางการคูณ (Multiplication Arithmetic) ที่ได้ผลลัพธ์ที่คิดเครื่องหมายและไม่คิดเครื่องหมาย ดังแสดงในตารางที่ 2.2 ถ้าเป็นการคูณแบบคิดเครื่องหมาย คือ

```
MULS    EA , Dn
```

และการคูณแบบไม่คิดเครื่องหมายคือ

```
MULU    EA , Dn
```

คำสั่ง MULS และ MULU จะเป็นคำสั่งที่กระทำการกับข้อมูลในระดับ 16 บิต โดยที่ข้อมูลต้นทางจะเป็นตำแหน่งข้อมูลใช้งาน ซึ่งสามารถใช้การอ้างถึงตำแหน่งข้อมูลทุกรูปแบบ และข้อมูลปลายทางจะต้องใช้การอ้างตำแหน่งแบบรีจิสเตอร์ข้อมูลเสมอ เมื่อกระทำคำสั่ง MULS จะเป็นการกระทำแบบคิดเครื่องหมายกับทั้งคู่ของของข้อมูลต้นทางและข้อมูลปลายทาง ถ้าต้องการผลลัพธ์แบบไม่คิดเครื่องหมายจะใช้คำสั่ง MULS โดยที่ผลลัพธ์จะเป็นข้อมูล 32 บิตและมีตำแหน่งที่เก็บในรีจิสเตอร์ข้อมูล

ดังตัวอย่างที่ต้องการผลลัพธ์การคูณแบบไม่คิดเครื่องหมาย ระหว่าง D_1 กับ D_0

```
MULU    D0 , D1
```

เมื่อคำสั่งนี้ถูกเอ็กซิคิวท์แล้ว จะได้ผลลัพธ์ข้อมูลแบบลองเวิร์ด เก็บใน D_1

เกือบทั้งหมดของคำสั่งการกระทำทางคณิตศาสตร์ จะมีผลต่อการเปลี่ยนแปลงของบิตSR โดยขึ้นอยู่กับผลของผลลัพธ์ด้วย โดยที่บิตเงื่อนไข คือบิตศูนย์ (Z) และบิตค่าลบ (N) จะมีผลต่อผลลัพธ์ที่ได้และบิตตัวทด (C) และบิตค่าเกิน (V) จะเคลียร์อยู่เสมอ

5. กลุ่มคำสั่งกระทำการหาร (Division Instruction - DIVS และ DIVU)

มีความคล้ายคลึงกับคำสั่งการคูณของ 68000 ซึ่งการหารแบบคิดเครื่องหมาย (DIVS) และไม่คิดเครื่องหมาย (DIVU) เขียนให้อยู่ในรูปแบบทั่วไปคือ

```
DIVS    EA , Dn
```

และ

DIVU EA , Dn

ข้อมูลปลายทางจะเป็นตัวหาร ข้อมูลที่เก็บในรีจิสเตอร์ข้อมูลและข้อมูลต้นทางที่เป็นตัวตั้งจะสามารถใช้การอ้างถึงตำแหน่งข้อมูลได้ทุกแบบของ 68000

การเอ็กซิคิวท์ของแต่ละคำสั่งจะมีตัวตั้งเป็นข้อมูลแบบ 32 บิต จะถูกตัวหาร ซึ่งเป็นข้อมูลแบบ 16 บิต ผลลัพธ์ที่ได้จากการหารจะเป็นข้อมูลขนาด 16 บิต โดยผลลัพธ์ที่ได้จะเก็บใน 8 บิตล่างของข้อมูลแบบเวิร์ดของรีจิสเตอร์ข้อมูลหรือถูกเปลี่ยนไปเก็บใน 8 บิต บนของข้อมูลเวิร์ด ในรีจิสเตอร์ข้อมูลตัวเดียวกันและในการหารแบบคิดเครื่องหมาย แล้วเครื่องหมายของผลลัพธ์จะเหมือนกับจำนวนตัวตั้งที่ถูกหารเสมอ

รหัสเงื่อนไขที่มีผลต่อคำสั่งการหารคือแฟล็กศูนย์และแฟล็กค่าลบ จะทำการเซ็ตหรือรีเซ็ตขึ้นอยู่กับค่าผลลัพธ์ของการหารแฟล็กตัวทดจะเคลียร์อยู่เสมอ แฟล็กศูนย์จะเซ็ต เมื่อข้อมูลปลายทางไม่มีการเปลี่ยนแปลง

6. คำสั่งในการเพิ่มขยายบิตเครื่องหมาย (Sign Extend Instruction - EXT)

68000 มีคำสั่ง Sign Extend (EXT) สำหรับการเพิ่มขยายบิตเครื่องหมายของไบต์ข้อมูลหรือเวิร์ดข้อมูล รูปแบบทั่วไปของคำสั่งนี้ คือ

EXT Dn

ข้อมูลสามารถเก็บในรีจิสเตอร์ข้อมูล เมื่อคำสั่ง EXT ถูกเอ็กซิคิวท์บิตเครื่องหมายของข้อมูลจะถูกทำซ้ำที่บิตนัยสำคัญสูงของรีจิสเตอร์

ดังตัวอย่าง เมื่อ D_1 เก็บค่าของข้อมูลเป็นลักษณะเวิร์ด เมื่อกระทำคำสั่งนี้แล้ว D_1 จะถูกเพิ่มขยายบิตเป็นข้อมูลในลักษณะลองเวิร์ด คำสั่งที่ว่านี้คือ

EXT.L D1

ในเงื่อนไขนี้ค่าในบิตที่ 15 จะถูกทำซ้ำต่อไปจากบิต 16 จนถึง 31 ของ D_1

การเพิ่มขยายบิตเครื่องหมายความต้องการใช้ เมื่อข้อมูลมีลักษณะความยาวไม่เท่ากัน ซึ่งเป็นปัญหาที่ยุ่งยากมากในการทำคำสั่งทางคณิตศาสตร์ เช่น ถ้าเราต้องการใช้คำสั่งในการบวก ซึ่งคำสั่งบวกนี้ต้องการข้อมูลในการประมวลผลเป็นขนาดข้อมูลชนิดเวิร์ด แต่ถ้าขนาดของข้อมูลเป็นชนิดไบต์ ก็ต้องมีการเพิ่มขยายบิตให้เป็นขนาดข้อมูลแบบเวิร์ด

ตัวอย่างที่ 4

สมมุติว่าให้รีจิสเตอร์ข้อมูล D_0, D_1 และ D_2 เก็บข้อมูลที่มีขนาดไบต์, เวิร์ดและลองเวิร์ด ในรูปแบบ 2'S Complement ตามลำดับ และเขียนลำดับที่จะกระทำการได้ตามนี้

D0 + D1 - D2 ⟶ D0

ก่อนที่จะกระทำคำสั่งบวกหรือลบใดๆ เราจำต้องเพิ่มขยายบิตแต่ละค่าให้เป็นข้อมูลในลักษณะลองเวิร์ดก่อนในที่นี้ D_0 เป็นข้อมูลลักษณะไบต์ก่อนที่จะทำเป็นลองเวิร์ด ต้องผ่านกระบวนการทำเป็นเวิร์ดก่อน ดังตัวอย่างคำสั่งนี้

```
EXT.W    D0
EXT.L    D0
```

ซึ่งมีความคล้ายคลึงกันกับข้อมูลชนิดเวิร์ดใน D_1 ทำให้เป็นลองเวิร์ด ใช้คำสั่ง

```
EXT.L    D1
```

เมื่อข้อมูลใน D_2 เป็นข้อมูลในลักษณะลองเวิร์ด ไม่จำเป็นในการเพิ่มขยายบิตอีก

ในการกระทำคำสั่งทางคณิตศาสตร์ ดังเช่น ต้องการบวกข้อมูลที่เก็บใน D_0 และ D_1 จากนั้นนำไปลบกับข้อมูลที่เก็บใน D_2 ลำดับของคำสั่งเขียนได้ดังนี้

```
EXT.W    D0
EXT.L    D0
EXT.L    D1
ADD.L    D1 , D0
SUB.L    D2 , D0
```

2.7.3 คำสั่งการกระทำทางคณิตศาสตร์ของเลขฐานสิบ

คำสั่งการกระทำทางคณิตศาสตร์เป็นการประมวลผลข้อมูลที่เป็นตัวเลขฐานสอง เมื่อข้อมูลเป็นเลข BCD จึงต้องมีส่วนของ การดำเนินการ ที่กระทำการแปลงรหัส BCD เป็นรหัสฐานสอง และ รหัสฐานสองเป็นรหัส BCD เมื่อต้องการกระทำการข้อมูลเลข BCD

ไมโครโปรเซสเซอร์ 68000 มีความสามารถในการกระทำคำสั่งทางคณิตศาสตร์ เช่น บวก, ลบ และ Negate โดยตรงจากข้อมูลเลข BCD ได้เลย คำสั่งกระทำทางคณิตศาสตร์ด้วยข้อมูล BCD คือ ABCD, SBCD และ NBCD จัดหาให้ สำหรับจุดมุ่งหมายนี้ ซึ่งประสิทธิภาพและมีหลักการที่ใช้ง่ายสำหรับเป็นเครื่องมือในการกระทำทางคณิตศาสตร์ต่อข้อมูลเลข BCD

1. คำสั่งในการบวกเลขฐานสิบและการเพิ่มขยายบิต (Add Decimal With Extend Instruction - ABCD)

เริ่มต้นด้วยคำสั่ง Add Binary-Coded Decimal (ABCD) ซึ่งแสดงไว้ในตารางที่ 2.3 แบบแรกที่จะกล่าวถึงคือ

ABCD Dy , Dx

การอ้างถึงตำแหน่งแบบรีจิสเตอร์ข้อมูล สำหรับข้อมูลต้นทางและข้อมูลปลายทาง เพราะฉะนั้นแล้วข้อมูลต้นทางและข้อมูลปลายทางต้องเก็บอยู่ภายในรีจิสเตอร์ข้อมูลของ 68000 ส่วนรูปแบบของคำสั่งนี้คือ

ABCD -(Ay) , -(Ax)

ใช้การ Precrement ในการอ้างถึงตำแหน่งแบบผ่านแอดเดรสรีจิสเตอร์

ในการเอ็กซิคิวท์ของคำสั่ง ABCD จะเป็นการบวก ข้อมูลต้นทางและข้อมูลปลายทาง พร้อมกับ เพิ่มขยายบิต (X) ของ SR และผลบวกที่ได้จะเก็บไว้ที่ตำแหน่งข้อมูลปลายทาง

ตารางที่ 2.3 คำสั่งกระทำการคณิตศาสตร์ของเลขฐานสองเป็นเลขฐานสิบ

Mnemonic	Meaning	Type	Operand Size	Operation
ABCD	Add BCD numbers	ABCD Dy , Dx	8	Dy+Dx+X ⟶ Dx
		ABCD -(Ay) , -(Ax)	8	-(Ay)+-(Ax)+X ⟶(Ax)
MOVEM	Subtract BCD numbers	SBCD Dy , Dx	8	Dy-Dx-X ⟶ Dx
		SBCD -(Ay) ,-(Ax)	8	-(Ay)--(Ax)-X ⟶(Ax)
NBCD	Negate BCD numbers	NBCD EA	8	0 - (EA) - X ⟶ EA

คำสั่ง ABCD เป็นการดำเนินการบวกเลขฐานสิบ ฉะนั้นต้องใช้ข้อมูลเลขฐานสิบแทนที่เลขฐานสอง ข้อมูลเลขฐานสิบนี้จะเก็บในรหัสแบบ BCD และผลบวกที่ได้จากคำสั่งนี้ก็จะเป็นเลขฐานสิบด้วย และก็เก็บในรหัสแบบ BCD เช่นกัน ขนาดของข้อมูลต้องเป็นลักษณะไบต์เสมอ ดังเช่นตัวอย่าง

ABCD D0 , D1

ถ้า D_0 เก็บค่า 12_{10} = 00010010_2 , D_1 เก็บค่า 37_{10} = 00110111_2 และบิต X เคลียร์ เอ็กซิคิวท์ของคำสั่งนี้จะได้ผลบวกเป็น

$$D0 + D1 + X = 12_{10} + 37_{10} + 0_{10}$$

$$= 49_{10}$$

เมื่อกระทำคำสั่งนี้แล้ว D_0 ยังคงเก็บค่าของ 12_{10} แต่ข้อมูลที่ D_1 เก็บเปลี่ยนเป็น 49_{10} ส่วนบิตยังคง เคลียร์เพราะไม่มีการยืมเกิดขึ้น

ในคำสั่งนี้บิตเงื่อนไข Z, Nและ C จะมีผลโดยขึ้นอยู่กับผลลัพธ์ที่ได้จากการบวกบิต C และ X จะ set อยู่เสมอ

2. คำสั่งกระทำการลบเลขฐานสิบและการเพิ่มขยายบิต (Subtract Decimal with Extend Instruction - SBCD)

คำสั่งกระทำการลบเลขฐานสิบและการเพิ่มขยายบิต(SBCD) จะทำงานคล้ายกับคำสั่ง ABCD แต่คำสั่งนี้จะเป็นการกระทำทางคณิตศาสตร์ในการลบ ดังที่แสดงให้ดูในตารางที่ 2.3 คำสั่งนี้มี 2 รูปแบบ คือ

SBCD Dy , Dx

และ

SBCD -(Ay) , -(Ax)

สำหรับตัวอย่างคำสั่ง ในที่นี้คือ

SBCD -(A0) , -(A1)

เมื่อคำสั่งนี้ถูกเอ็กซิคิวท์โดยที่ความกว้างของข้อมูลเป็นแบบไบต์ โดยที่ข้อมูลที่เก็บในข้อมูลต้นทางและบิตของ SR จะถูกลบออกจากข้อมูลปลายทางและผลจากการลบที่ได้จะเก็บไว้ที่ข้อปลายทาง

ตัวอย่างนี้แสดงให้ดูการอ้างตำแหน่งแบบผ่านแอดเดรสรีจิสเตอร์ การ Predecrement เพราะฉะนั้นข้อมูลที่เก็บในแอดเดรสรีจิสเตอร์ A_0 และ A_1 ลำดับแรกจะถูกลดค่าลงไปใน 1 ตัวอย่าง ถ้าข้อมูลที่เก็บเป็น $0000110F_{16}$ และ $0000120F_{16}$ ตามลำดับ เมื่อมีการลดค่าลงด้วย 1 ทำให้ $A_0 = 0000110E_{16}$ และ $A_1 = 0000120E_{16}$ ตัวเลขนี้ จะเป็นตำแหน่งที่ทำการแอดเดรสในหน่วยความจำ สมมุติที่ตำแหน่ง $00120E_{16}$ ในหน่วยความจำเก็บค่า 37_{10} และที่ตำแหน่ง $00110E_{16}$ เก็บค่า 12_{10} และที่บิต x = 1 ผลลัพธ์ที่ได้ของคำสั่งนี้คือ

$$(00120E_{16}) - (00110E_{16}) - X = 37_{10} - 12_{10} - 1_{10}$$

$$= 24_{10}$$

ผลลัพธ์ที่ได้จะเก็บที่ตำแหน่ง $(00120E_{16})$ ในหน่วยความจำและ เงื่อนไข Z , X และ C จะเคลียร์

3. Negate Decimal Instruction – NBCD

คำสั่งสุดท้ายในการกระทำทางคณิตศาสตร์ของเลขฐานสิบที่แสดงไว้ในตารางที่ 2.3 คือ Negate Binary – Coded Decimal (NBCD)

NBCD EA

NBCD เป็นผลมาจากคำสั่ง SBCD จะทำการลบจำนวนๆ หนึ่งด้วยศูนย์เสมอ เขียนขั้นตอนการทำงานได้เป็น

$0 - (EA) \longrightarrow EA$

ข้อมูลที่ถูกชี้โดยตำแหน่งข้อมูลใช้งาน (EA) จะสามารถที่จะใช้การอ้างตำแหน่งข้อมูลที่เปลี่ยนแปลง เว้นเสียแต่การอ้างตำแหน่งข้อมูลแบบแอดเดรสรีจิสเตอร์จะไม่สามารถใช้ได้

ดังตัวอย่างจะเป็นการเข้าถึงข้อมูลโดย การอ้างตำแหน่งข้อมูลแบบผ่านแอดเดรสรีจิสเตอร์ โดยทำการเพิ่มค่าก่อน

NBCD (A5) + EA

บิตเงื่อนไข ที่มีผลต่อคำสั่ง NBCD จะมีผลการเปลี่ยนแปลงเหมือนคำสั่ง SBCD

2.7.4 คำสั่งประมวลผลทางตรรกศาสตร์

เครื่องมือในการกระทำทางลอจิก AND, OR, exclusive–OR และ NOT ที่กล่าวมานี้เป็นเซ็ตของคำสั่งที่ 68000 จัดหาให้สำหรับกลุ่มคำสั่งทางลอจิกกลุ่มคำสั่งนี้ แสดงให้ดูในตารางที่ 2.4 ซึ่งประกอบด้วยรูปแบบที่แตกต่างกัน, ขนาดของข้อมูลและกระทำการทางลอจิก จะทำการการเอ็กซิคิวท์ของเซ็ต คำสั่งทางลอจิก บิตเงื่อนไข N และ Z จะมีผลต่อผลลัพธ์ที่ได้, V และ C จะเคลียร์, บิต X ไม่มีผลใดๆ

1. AND Instruction – AND และ ANDI

จากคำสั่งที่แสดงในในตารางที่ 2.4 ซึ่งมี 4 รูปแบบของคำสั่ง AND จะเป็นการนำข้อมูลที่เก็บในข้อมูลรีจิสเตอร์ ทำการคูณเข้ากับข้อมูลที่ตำแหน่งข้อมูลใช้งาน รูปแบบแรกของคำสั่ง

AND EA , Dn

คำสั่งดังกล่าวข้อมูลต้นทางสามารถใช้การอ้างถึงตำแหน่งข้อมูลสำหรับ การให้ค่าข้อมูลเพราะฉะนั้นข้อมูลต้นทางสามารถใช้การอ้างถึงตำแหน่งข้อมูลทุกแบบ เว้นแต่การอ้างตำแหน่งแบบแอดเดรสรีจิสเตอร์ แต่สำหรับข้อมูลปลายทางการอ้างตำแหน่งแบบรีจิสเตอร์ข้อมูลเท่านั้น ซึ่งก็คือ 1 ใน 8 ของรีจิสเตอร์ข้อมูลที่มีอยู่ภายใน 68000

ตารางที่ 2.4 คำสั่งประมวลผลทางตรรกศาสตร์

Mnemonic	Meaning	Type	Operand Size	Operation
AND	Logical AND	AND EA , Dn	8,16,32	(EA)* Dn ⟶ Dn
		AND Dn , EA	8,16,32	Dn * (EA) ⟶ EA
		ANDI #XXX,EA	8,16,32	#XXX * (EA) ⟶ EA
		ANDI #XXX,CCR	8	#XXX*CCR ⟶ CCR
		ANDI #XXX,SR	16	#XXX * SR ⟶ SR
OR	Logical OR	OR EA , Dn	8,16,32	(EA) + Dn ⟶ Dn
		OR Dn , EA	8,16,32	Dn + (EA) ⟶ EA
		ORI #XXX,EA	8,16,32	#XXX + (EA) ⟶ EA
		ORI #XXX,CCR	8	#XXX+CCR ⟶ CCR
		ORI #XXX,SR	16	#XXX + SR ⟶ SR
EOR	Logical exclusive – OR	EOR Dn , EA	8,16,32	Dn ⊕ (EA) ⟶ EA
		EORI #XXX,EA	8,16,32	#XXX ⊕ (EA) ⟶ EA
		EORI #XXX,CCR	8	#XXX ⊕ CCR ⟶ CCR
		EORI #XXX,SR	16	#XXX ⊕ SR ⟶ SR
NOT	Logical NOT	NOT EA	8,16,32	$(\overline{EA})$ ⟶ EA

สำหรับตัวอย่างของคำสั่ง ซึ่งใช้การอ้างถึงตำแหน่งแบบรีจิสเตอร์ข้อมูล ทั้งข้อมูลต้นทางและข้อมูลปลายทาง คือ

AND.B D0 , D1

เอ็กซิคิวท์ของคำสั่งนี้ จะดำเนินการ AND ข้อมูลในลักษณะไบต์ของ D_0 และ D_1 จะเก็บผลที่ได้ไว้ในรีจิสเตอร์ D_1 ตัวอย่าง ถ้า D_1 เก็บข้อมูล $0000ABCD_{16}$ และ D_0 เก็บข้อมูล $00000F0F_{16}$ และกระทำการ AND กันระหว่าง 8 บิตนัยสำคัญต่ำ

$$CD_{16} * 0F_{16} = 11001101_2 * 00001111_2$$

$$= 00001101_2$$

เพราะฉะนั้นข้อมูลที่เก็บใน D_1 มีผลลัพธ์เป็น $0000ABCD_{16}$ และผลของบิตเงื่อนไขของ SR มีดังนี้ C และ V จะทำการเคลียร์อยู่เสมอ, Z และ N จะเซ็ตหรือรีเซ็ตขึ้นอยู่กับผลลัพธ์ที่เกิดการคูณกัน

รูปแบบที่ 2

AND Dn , EA

จะทำการนำข้อมูลที่เก็บในรีจิสเตอร์ข้อมูลทำการ AND กับข้อมูลปลายทาง ที่ถูกชี้โดยตำแหน่งข้อมูลใช้งาน (EA) ในคำสั่งนี้ตำแหน่งของข้อมูลปลายทาง สามารถอ้างถึงได้โดยใช้การอ้างตำแหน่งในหน่วยความจำที่ปรับเปลี่ยนแก้ไขให้เกิดความเหมาะสมได้

รูปแบบที่ 3 ของกลุ่มคำสั่งการ AND คือ AND Immediate คำสั่งนี้ ข้อมูลต้นทางจะเป็นการอ้างข้อมูลแบบทันทีทันใด ซึ่งเขียนอยู่ในรูป #XXX จะทำการ AND กับข้อมูลที่เก็บตำแหน่งข้อมูลปลายทาง ข้อมูลแบบทันทีทันใดนี้ มีตำแหน่งที่เก็บในหน่วยความจำรูปแบบแรกของคำสั่ง ANDI คือ

ANDI #XXX , EA

จะทำการ AND ข้อมูลแบบทันทีทันใดกับข้อมูลที่เก็บในตำแหน่งปลายทาง ซึ่งถูกชี้โดยตำแหน่งข้อมูลใช้งาน (EA) ข้อมูลในตำแหน่งปลายทางนี้สามารถเก็บในรีจิสเตอร์ข้อมูล, แอดเดรสรีจิสเตอร์หรือตำแหน่งในหน่วยความจำ

ตัวอย่างของคำสั่งนี้คือ

ANDI.B #7 , D1

เอ็กซิคิวท์ของคำสั่งนี้ จะทำการนำเลขฐานสองจาก เลข 7 ในเลขฐานสิบ ทำการคูณกับข้อมูลที่เก็บใน D_1 ในที่นี้ให้สมมุติว่า D_1 เก็บค่า $FFFFFFFF_{16}$ เอ็กซิคิวท์ คำสั่งนี้จะทำให้

$$D_1 = FFFFFFFF_{16} * 7_{16}$$

$$= FFFFFFF7_{16}$$

รูปแบบที่สองของคำสั่ง ANDI คือ

ANDI #XXX , SR

และ

ANDI #XXX , CCR

ใช้สำหรับการ AND ข้อมูลที่เก็บในสถานะรีจิสเตอร์และรหัสรีจิสเตอร์ ซึ่งเป็นส่วนประกอบของ SR กับข้อมูลแบบทันทีทันใด (Immediate Data) ตามลำดับ รูปแบบแรกจะเป็น

การดำเนินการแบบสิทธิพิเศษ (Privileged) ซึ่งจะสามารถกระทำการเอ็กซิคิวท์ เมื่อ 68000 อยู่ในสถานะ Supervisor State เท่านั้น

2. OR Instruction – OR และ ORI

คำสั่งการ OR จะมีรูปแบบอยู่ 5 รูปแบบ ดังแสดงในตารางที่ 2.4 คำสั่ง OR พื้นฐานจะกระทำการ OR ระหว่างข้อมูลที่เก็บในรีจิสเตอร์ข้อมูลกับข้อมูลที่เก็บในรีจิสเตอร์ข้อมูลหรือแอดเดรสรีจิสเตอร์หรือในหน่วยความจำ ซึ่งกำหนดลักษณะเฉพาะโดยใช้การอ้างถึงตำแหน่งข้อมูล ดังตัวอย่างของคำสั่ง

OR.B (A0) , D0

คำสั่งนี้จะทำการ OR ในลักษณะไบต์ของข้อมูลตำแหน่งที่ A_0 ชี้อยู่กับข้อมูลที่เก็บใน D_0 และผลที่ได้จะเก็บใน D_0 สามารถเขียนการดำเนินการได้ คือ

$$(EA) + D0 \longrightarrow D0$$

ในที่นี้ให้สมมุติว่าข้อมูลที่ต้องการถูกชี้อยู่โดยตำแหน่งใน A_0 คือ $AAAAAAAA_{16}$ และข้อมูลที่เก็บใน D_0คือ 55555555_{16} ผลลัพธ์ที่ได้จากการ เอ็กซิคิวท์ คำสั่งนี้คือ

$$D0 = AAAAAAAA_{16} + 55555555_{16}$$
$$= FFFFFFFF_{16}$$

รูปแบบคำสั่ง OR แบบทันทีทันใด (OR Immediate) จะยอมให้ข้อมูลทันทีทันใดทำการ OR กับข้อมูลในหน่วยความจำ, รีจิสเตอร์ข้อมูลหรือสถานะรีจิสเตอร์ ตัวอย่างของคำสั่งนี้คือ

ORI #$FF00 , SR

เอ็กซิคิวท์ของคำสั่งนี้ คือ บิตทั้งหมดของไบต์สูงของ SR จะถูกเซ็ตเป็น 1 โดยจะปราศจากการเปลี่ยนแปลงไบต์ต่ำ เมื่อสถานะรีจิสเตอร์ส่วนของไบต์สูงมีการเปลี่ยนแปลงการทำงานจะสามารถดำเนินการเมื่ออยู่ในสถานะ Supervisor State เท่านั้น

3. Exclusive – OR Instruction – EOR และ EOI

จากในตารางที่ 2.4 จะเห็นว่ามีคำสั่งที่จัดหาให้สำหรับการ Exclusive–OR (EOR) จะดำเนินการทางลอจิก Exclusive–OR ของข้อมูลที่เก็บในตำแหน่งต้นทางกับข้อมูลที่เก็บในตำแหน่งปลายทาง

ตัวอย่างแรกของคำสั่งนี้คือ

EOR.L D0 , A0

เมื่อทำการเอ็กซิคิวท์คำสั่งการทำงานจะดำเนินงานดังนี้

$$D0 \oplus A0 \longrightarrow A0$$

ตัวอย่างอื่นคือ

EOR #$OF , CCR

เมื่อทำการเอ็กซิคิวท์คำสั่งการทำงานจะดำเนินงานดังนี้

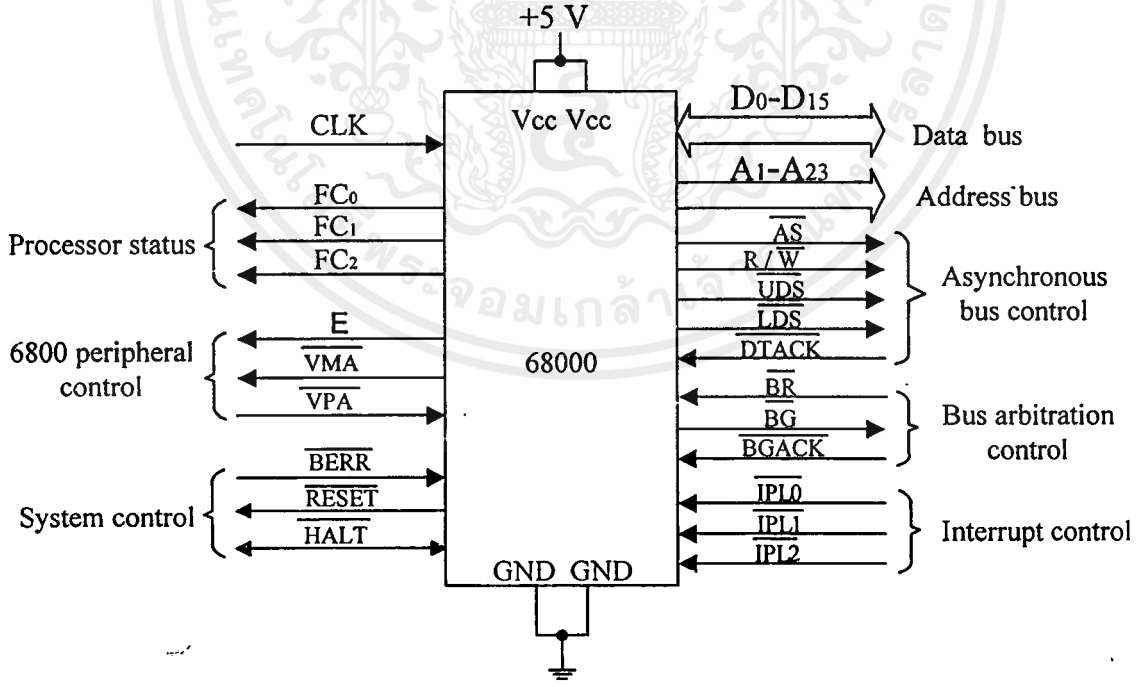
$$\$OF \oplus CCR \longrightarrow CCR$$

4. NOT Instruction – NOT

คำสั่ง NOT จะมีความแตกต่างจากคำสั่งการ AND, OR และ EOR ในคำสั่งนี้มีเพียงขนาดข้อมูลที่สามารถกำหนดลักษณะได้ ดังที่ได้แสดงในตารางที่ 2.4

2.8 รายละเอียดทางฮาร์ดแวร์ของ 68000

2.8.1 ลักษณะจำเพาะของซีพียู



รูปที่ 2.17 ขาสัญญาณอินพุตและเอาต์พุตของซีพียู 68000

68000 เป็นไมโครโปรเซสเซอร์ขนาด 16 บิต ซึ่งใช้การติดต่อกับอุปกรณ์ภายนอกแบบ 16 บิต ความได้เปรียบในการอ้างข้อมูลของแบบ 16 บิตดีกว่า 8 บิต สามารถแลกเปลี่ยนข้อมูลได้เป็น 2 เท่าในเวลา 1 ครั้ง ซึ่งมีผลต่อการลดเวลาในการเข้าถึงข้อมูลในการเอ็กซิคิวท์โปรแกรม 68000 ตำแหน่งบัส 24 เส้น สามารถอ้างหน่วยความจำได้ 16 เมกกะไบต์ หรือ 8 เมกกะไบต์ ขาสัญญาณคอนโทรลสามารถจัดอุปกรณ์ภายนอกเข้ามาใช้งานบัสของ 68000 และมีขาอินเตอร์รัพต์ 3 เส้น มีลำดับขั้นความสำคัญ 7 ระดับของอุปกรณ์ภายนอกที่เข้ามาอินเตอร์รัพต์ ขาสัญญาณ์คอลโทรลเอาต์พุต 3 เส้น ใช้สำหรับการเข้าถึงรหัสได้เป็น 8 สถานะภายใน ซีพียูและประกอบด้วยขาสัญญาณคอนโทรล ซึ่งทำงานร่วมกันจะสามารถให้ซีพียู 68000 สามารถอินเตอร์เฟสกับอุปกรณ์ข้างเคียง 68000 (สำหรับผู้ใช้ที่ต้องการพัฒนาฮาร์ดแวร์ให้ดีขึ้น) 68000 มีความสามารถในการทำงานของสัญญาณนาฬิกา 4 เมกกะเฮิรตซ์ ถึง 16 เมกกะเฮิรตซ์ ดังในรูปที่ 2.17

2.8.2 รายละเอียดขาของ 68000

1. Vcc , GND และ CLK

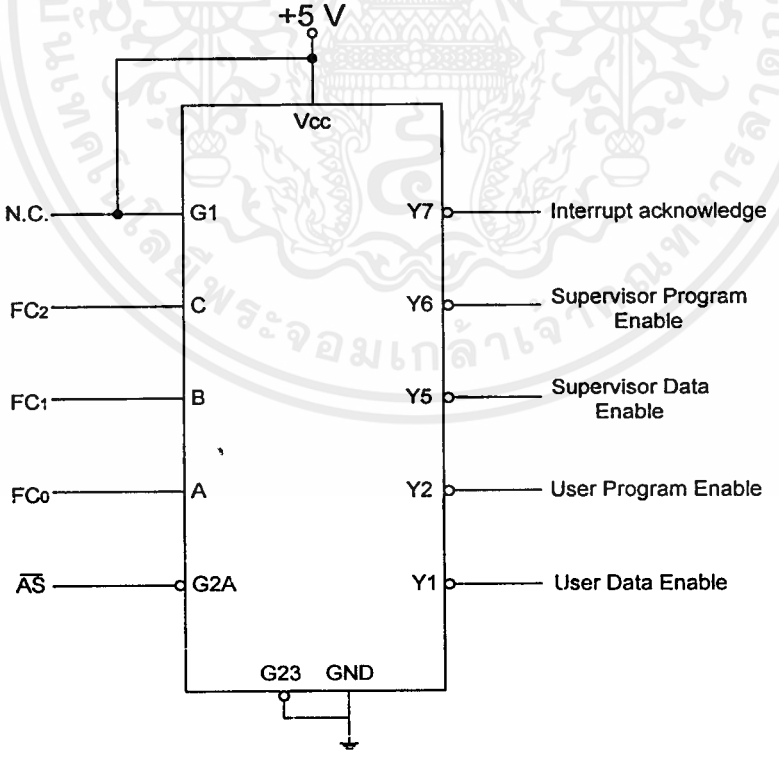
ขาการทำงานและเป็นสัญญาณนาฬิกา 68000 ทำงานด้วยแรงดันไฟ 5 โวลต์สามารถจ่ายแรงดันเกินหรือขาด ± 5 % และใช้กำลังไฟ 1.5 วัตต์ (ที่สัญญาณนาฬิกา 8 เมกกะเฮิรตซ์) และสำหรับขา CLK ต้องการช่วงเวลาไต่ขึ้นและช่วงเวลาไต่ลง 10 นาโนวินาที และต้องการไอซีประเภท TTL ที่สามารถทำงานเข้ากันได้ในรูปคลื่นสัญญาณที่ 50 % ดิวตี้ไซเคิล

2. FC_0 , FC_1 และ FC_2

กลุ่มนี้เป็นสัญญาณที่ใช้เป็นเอาต์พุต ซึ่งถูกถอดรหัสบอกสถานะการทำงานของซีพียู ซึ่ง FC_0, FC_1 และ FC_2 จะเป็น function code ของเอาต์พุต ซึ่งจะใช้ชี้บอกสถานะการทำงานภายในของ 68000 (ประเภทของวัฏจักรในการเอ็กซิคิวท์) Output Function Code จะทำงานเมื่อมีสัญญาณ $\overline{AS}$ ทำงานด้วยดังตารางที่ 2.5 แสดงการทำงานในการเข้ารหัสของขาเอาต์พุต Function Code ทั้ง 3 นี้ สามารถแบ่งแยกสถานะการทำงาน เป็น User Data, User Program และ Interrupt Acknowledge ในระบบ Output Function Code จะถูกใช้สำหรับการห้ามมิให้มีการเข้าถึงหน่วยความจำจากวงจรถอดรหัสตำแหน่งหน่วยความจำ ในกรณีที่ถูกใช้เมื่อมีความจำเป็นเมื่อผู้ใช้ต้องการเก็บหน่วยความจำช่วงนั้นสำหรับ supervisor mode

ตารางที่ 2.5 กลุ่มของสัญญาณ FC_0, FC_1, FC_2 แสดงสถานะการทำงานของ 68000

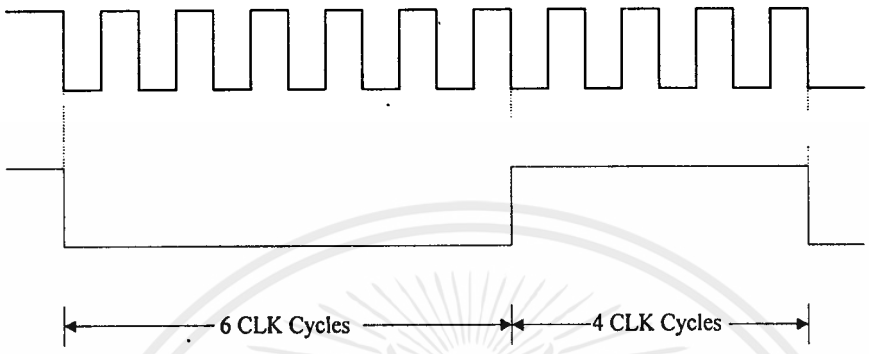
FC_2	FC_1	FC_0	Cycle Type
0	0	0	Reversed
0	0	1	User data
0	1	0	User program
0	1	1	Reversed
1	0	0	Reversed
1	0	1	Supervisor data
1	1	0	Supervisor program
1	1	1	Interrupt acknowledge



รูปที่ 2.18 การใช้ IC เบอร์ 74LS138 ในการถอดรหัสสัญญาณ Output Function Code

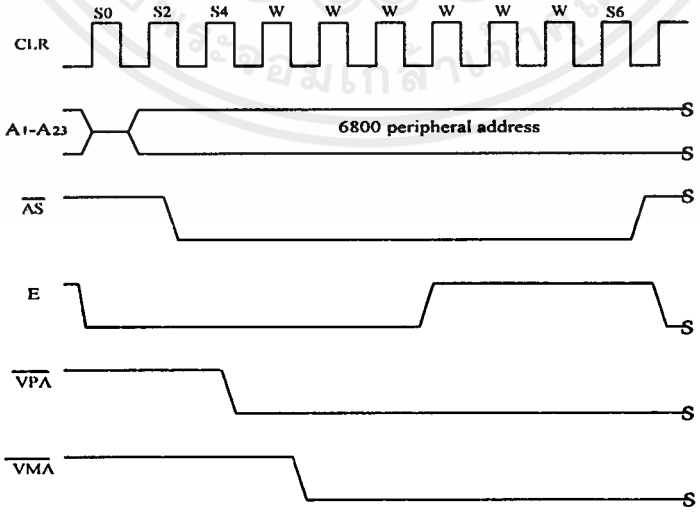
วิธีการง่ายๆ ที่จะถอดรหัสสถานะทั้ง 8 คือ ใช้ IC เบอร์ 74LS138

3. E , $\overline{\text{VMA}}$, and $\overline{\text{VPA}}$



รูปที่ 2.19 ความสัมพันธ์ระหว่างสัญญาณ CLK กับสัญญาณ E

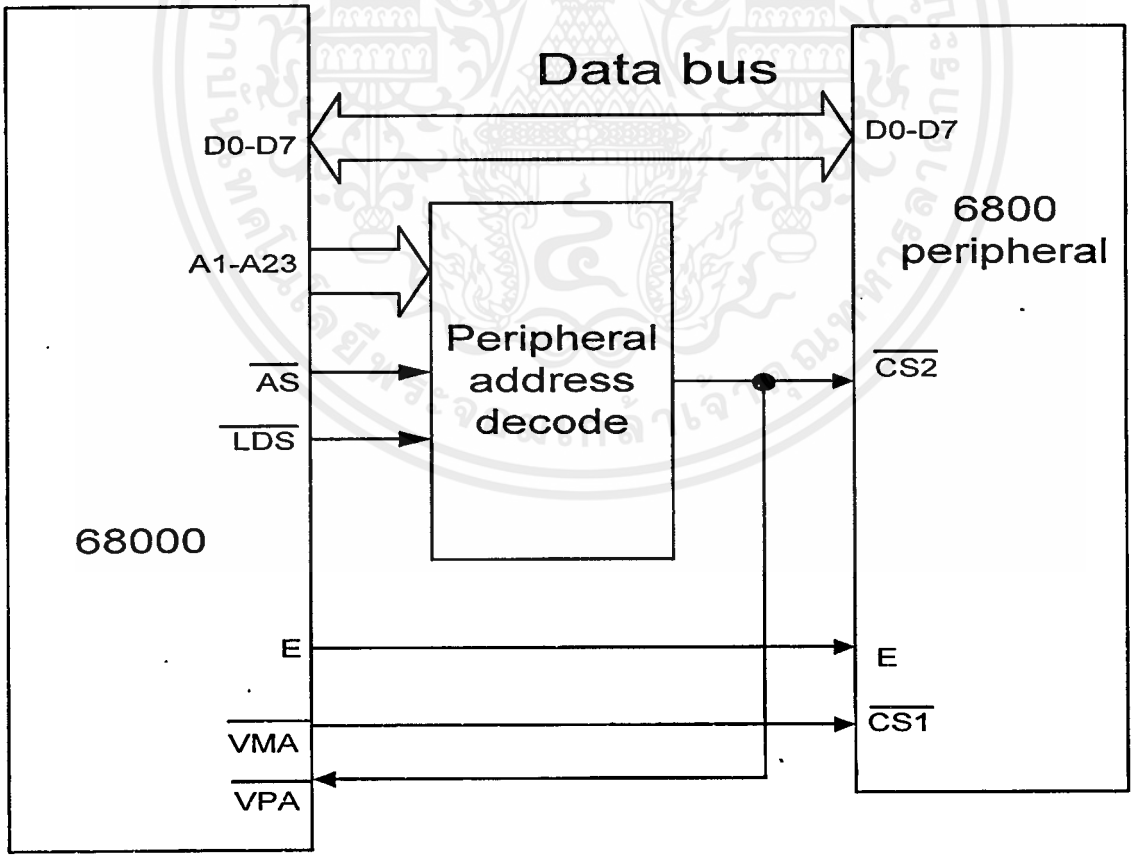
กลุ่มของสัญญาณนี้จัดหาให้ 68000 มีความสามารถในการควบคุมอุปกรณ์ต่อพ่วงอยู่ภายนอก 6800 ประกอบด้วย E Clock, $\overline{\text{VMA}}$ (Valid Memory Address) และ $\overline{\text{VPA}}$ (Valid Peripheral Address) มีเพียงขา $\overline{\text{VPA}}$ เป็นขาอินพุต ขา E ถูกใช้ในการให้กำเนิดสัญญาณนาฬิกาที่เหมาะสมสำหรับอุปกรณ์ต่อพ่วงภายนอก 6800 โดยการส่งสัญญาณนาฬิกาภายในซีพียู 68000 จะถูกหารลง 10 เท่าโดยมีดิวตี้ไซเคิล 40 % รูปที่ 2.19 ความสัมพันธ์ดังกล่าว



รูปที่ 2.20 สัญญาณนาฬิกาของอุปกรณ์ภายนอก 6800

$\overline{VPA}$ เป็นขาอินพุตถูกใช้ในการดำเนินการ 68000 ติดต่อกับอุปกรณ์ภายนอก 6800 โดยข้อมูลถูกโอนถ่ายแบบเข้าจังหวะกับสัญญาณนาฬิกา E Clock การเข้าจังหวะจะถูกบ่งชี้โดยขา $\overline{VMA}$ เป็นเอาต์พุต ซึ่งจะถูกส่งออกไปเป็นสถานะศูนย์ เมื่อดำเนินการเข้าจังหวะกับขา E clock โดยลำดับขั้นการทำงานของอุปกรณ์ต่อพ่วง 6800 จะเป็นตามนี้

68000 ส่งสัญญาณเอาต์พุต ซึ่งเป็น ตำแหน่งของ 6800 โดยใช้ขา A_1 ถึง A_{23} สามารถตอบสนองจะกลับมาโดยการส่งสัญญาณศูนย์มายังขา $\overline{VPA}$ ซึ่งจะเป็นอินพุตของ 68000 เมื่อมีการดำเนินการที่เป็นการเข้าจังหวะกับ E Clock และมีขาสัญญาณ $\overline{VMA}$ ซึ่งเป็นเอาต์พุต โดยจะมีการโอนถ่ายข้อมูลเสร็จสิ้นในรูปที่ 2.20 แสดงลำดับของสัญญาณนาฬิกาและในรูปที่ 2.21 ผังงานของการติดต่อสำหรับการใช้ 6800 เป็นการอินเตอร์เฟสโดยความยาวของ Bus Cycle ขึ้นอยู่กับ 68000 ในการเข้าจังหวะกับ E Clock ดังรูปที่ 2.20 เงื่อนไขที่ดีของสัญญาณนาฬิกาสำหรับการเข้าถึงอุปกรณ์ต่อพ่วงในการที่ดีควรให้มีสัญญาณยาวกว่าW States (Wait States)

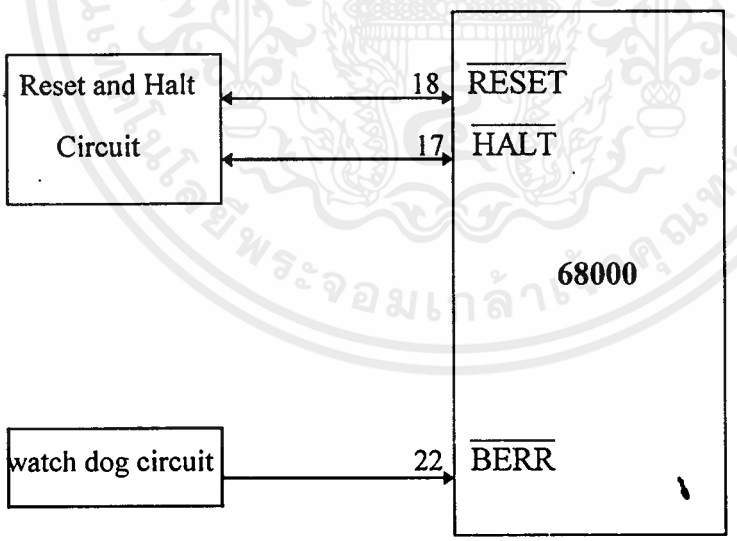


รูปที่ 2.21 การอินเตอร์เฟสกับอุปกรณ์ภายนอก 6800

4. $\overline{\text{RESET}}$, $\overline{\text{HALT}}$ และ $\overline{\text{BERR}}$

กลุ่มของสัญญาณจะถูกใช้ในการควบคุมระบบ $\overline{\text{BERR}}$ (Bus Error), $\overline{\text{RESET}}$ และ $\overline{\text{HALT}}$ ดังแสดงในรูปที่2.22 สัญญาณ $\overline{\text{BERR}}$ เป็นสัญญาณอินพุตถูกใช้ในรูปแบบดำเนินการ เมื่อวัฏจักรของการเอ็กซิคิวท์มีปัญหาเกิดขึ้นดำเนินการอ้างตำแหน่งในหน่วยความจำผิด หรือเกิดBus Error เกิดขึ้น ในรูปแบบนี้ 68000 เกิดการผิดพลาดเกิดขึ้นในระหว่างการเอ็กซิคิวท์ 68000 จะเลือกการกระทำที่เหมาะสมโดยจะเลือกการกระทำระหว่างการยกเว้นการผิดพลาดหรือการ Rerunning Bus Cycle ถ้าไม่เกิดสัญญาณ $\overline{\text{HALT}}$ ระหว่างการเกิดขึ้นของ Bus Error 68000 จะกระทำการยกเว้นการเกิด Bus Error แต่ถ้าเกิดสัญญาณ $\overline{\text{HALT}}$ ขึ้นก่อนหรือในเวลาเดียวกันกับสัญญาณ $\overline{\text{BERR}}$ 68000 จะทำการ Rerun ที่ Bus Cycle ดังกล่าวใหม่ โดยการทำให้ที่ Cycle สุดท้ายของบัสข้อมูลและบัสตำแหน่งเป็นสถานะ High Impedance ในขณะดังกล่าวนี้

68000 จะไม่มีการกระทำใดๆ ทั้งสิ้น เมื่อสัญญาณ $\overline{\text{HALT}}$ ไม่ active แล้ว (เป็น High) 68000 จะไม่มีการดำเนินการก่อนหน้านั้น (ในที่นี้จะใช้ข้อมูลเดิม, ตำแหน่งเดิม)



รูปที่ 2.22 ขา $\overline{\text{RESET}}$, $\overline{\text{HALT}}$ และ $\overline{\text{BERR}}$

ขา $\overline{\text{RESET}}$ และขา $\overline{\text{HALT}}$ เป็นขาสัญญาณแบบสองทิศทาง (Bidirectional) ซึ่งก็สามารถกระทำการได้ทั้งอินพุตและเอาต์พุตในเกิดการทำงานขึ้น สัญญาณ $\overline{\text{RESET}}$ และ $\overline{\text{HALT}}$ เกิดขึ้น

พร้อมกันหลังจากเกิดการทำงานโดยจะเกิดอย่างน้อยที่สุด 100 มิลลิวินาที ของสัญญาณพัลซ์ ทำให้ 68000 โหลดข้อมูลโปรแกรมนับจากเวกเตอร์ 0 (ตำแหน่ง 000000) และโปรแกรมนับจากเวกเตอร์ 1 (ตำแหน่ง 000004) ที่รีจิสเตอร์ทุกตัวไม่มีการเปลี่ยนแปลง เว้นเสียแต่สเตตัสรีจิสเตอร์ ถูกปรับให้บ่งชี้เป็นระดับ 7 ของสถานะอินเตอร์รัพต์

ขา $\overline{\text{RESET}}$ และขา $\overline{\text{HALT}}$ จะถูกใช้อีกครั้งเมื่อเป็นการดำเนินการรีเซ็ต โดยต้องเกิดสัญญาณทั้งคู่ เป็นเวลาอย่างน้อย 10 ไซเกิลสัญญาณนาฬิกา

ชุดคำสั่ง 68000 จะมีคำสั่งรีเซ็ตอยู่ด้วย และเมื่อคำสั่งนี้ถูกเอ็กซิคิวท์จะทำให้ขา $\overline{\text{RESET}}$ ของ 68000 เป็นเอาต์พุตทำให้เกิดสัญญาณระดับลอจิกต่ำจำนวน 124 ไซเคิลสัญญาณนาฬิกา ซึ่งจะมีผลต่อการรีเซ็ตอุปกรณ์ภายนอกที่ต่ออยู่กับขา $\overline{\text{RESET}}$

รูปที่ 2.23 สัญญาณทำให้เกิดการทำงาน, $\overline{\text{RESET}}$ และ $\overline{\text{HALT}}$

ขา $\overline{\text{HALT}}$ เมื่อได้รับสัญญาณระดับลอจิกต่ำจากอุปกรณ์ภายนอกทำให้ 68000 จะทำการ Halted Bus Cycle เมื่อเกิดการ Halted เกิดขึ้นขาสัญญาณที่มีคุณสมบัติ Tristate จะทำตัวเองเป็นสถานะ High Impedance และขาสัญญาณควบคุมทุกเส้นจะไม่ทำงานชุดคำสั่งของ 68000 สามารถทำให้เกิดการ Halt ได้โดยอาศัยคำสั่ง STOP เมื่อกระทำคำสั่งดังกล่าว 68000 จะเกิดสัญญาณ 0 ที่ขา $\overline{\text{HALT}}$ สิ่งที่ต้องระวังคือผู้ออกแบบวงจรต้องมีความแน่ใจในการใช้ขาสัญญาณ $\overline{\text{RESET}}$ และ $\overline{\text{HALT}}$ เพื่อการทำงานที่ถูกต้องและเหมาะสมในสถานะใดสถานะหนึ่งระหว่างอินพุตและเอาต์พุต ดังในรูปที่ 2.23 แสดงประเภทของการทำให้เกิดการทำงานและความ

สัมพันธ์กับสัญญาณนาฬิกา ซึ่งบ่งชี้ว่าการทำงานไม่เกิดขึ้น ถ้าไม่มีการเริ่มต้น อย่างน้อยที่สุด 100 มิลลิวินาที หลังจากที่ Vcc คงที่ที่ 5 โวลต์

5. $\overline{IPL_0}$, $\overline{IPL_1}$ และ $\overline{IPL_2}$

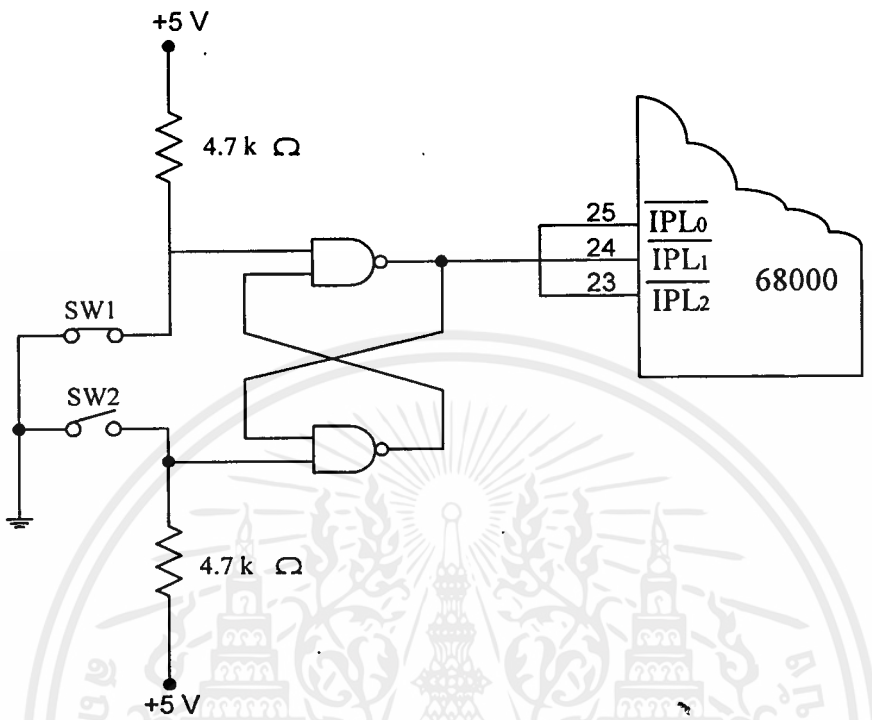
กลุ่มนี้ใช้สำหรับการควบคุมการอินเตอร์รัพต์สัญญาณอินพุต ทั้ง 3 ขาประกอบด้วย $\overline{IPL_0}$, $\overline{IPL_1}$ และ $\overline{IPL_2}$ จะถูกใช้สอดโดยวงจรภายนอก เพื่อการรอรับสัญญาณอินเตอร์รัพต์โดย สัญญาณดังกล่าวจะอยู่ในรูปของการเข้ารหัส ซึ่งบ่งบอกลำดับความสำคัญ โดยที่ระดับ 7 จะเป็นระดับที่มีความสำคัญสูงสุดและระดับศูนย์ เป็นการบอกว่าไม่มีการอินเตอร์รัพต์เกิดขึ้น (ทั้ง 3 เส้นของสัญญาณอินพุตจะเป็น ระดับลอจิกสูง)

ในตารางที่ 2.6 แสดงระดับสถานะของลอจิกที่ต้องการในแต่ละ 3 สัญญาณอินพุต

ตารางที่ 2.6 การเข้ารหัสระดับสัญญาณอินเตอร์รัพต์

Interrupt			Interrupt level
FC_2	FC_1	FC_0	
0	0	0	(lowest , none)
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	(Highest , nonmaskable)

สำหรับการได้รับการอินเตอร์รัพต์ $\overline{IPL_2}$ จะเป็นสัญญาณที่เป็นนัยสำคัญสูงสุด โดยที่ระดับของตัวเลข (0-7) และเลขไบนารีที่เป็นค่าจริงเป็นที่ต้องการของ $\overline{IPL_0}$ ถึง $\overline{IPL_2}$ จะถูกกลับสัญญาณเวกเตอร์อินเตอร์รัพต์และออโต้อินเตอร์รัพต์ กำหนดจาก $\overline{IPL_0}$ ถึง $\overline{IPL_2}$ ดังแสดงในรูปที่ 2.24 แสดงให้เห็นว่าสามารถใช้ปุ่มกดกำเนิดการอินเตอร์รัพต์ทั้ง 7 ระดับ



รูปที่ 2.24 การกำเนิดสัญญาณ interrupt ทั้ง 7 ระดับจากการใช้ push button

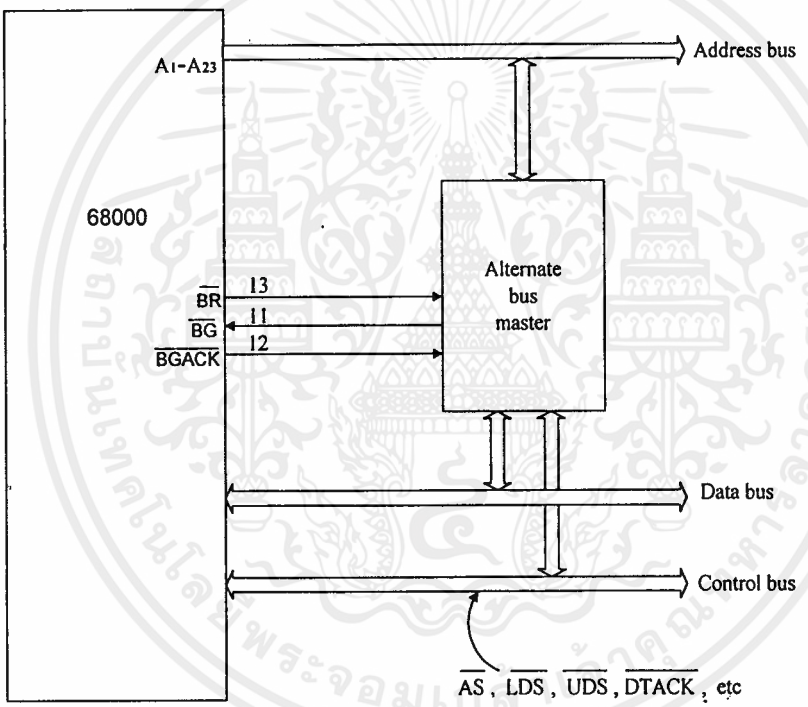
6. $\overline{BR}$, $\overline{BG}$ และ $\overline{BGACK}$

กลุ่มนี้ใช้เพื่อเป็นทำการควบคุมการทำงานของบัส ขาสัญญาณแต่ละเส้นสามารถทำให้ 68000 อยู่ในสภาวะรอคอย ในขณะที่กำลังติดต่อกับอุปกรณ์ภายนอก สำหรับวิธีการที่จะเข้าครอบครองบัสระบบของซีพียูนี้ เราเรียกว่า ขบวนการ DMA หรืออีกวิธีการคือมีซีพียูมากกว่า 1 ตัวภายในระบบ

โปรโตคอลการควบคุมการทำงานของบัสมีดังนี้ เมื่อมีการร้องขอใช้บัสจากอุปกรณ์ภายนอกซึ่งเรียกการร้องขอใช้บัสนี้ว่า Bus Master เมื่อมีการร้องขอขอใช้บัสจากอุปกรณ์ภายนอก จะส่งทำงานสัญญาณ $\overline{BR}$ (Bus Request) ซึ่งสัญญาณ $\overline{BR}$ นี้ เป็นอินพุตของ 68000 จะตอบสนองต่อขบวนการ Bus Master โดยจะมีการทำงานสัญญาณ $\overline{BG}$ กลับไปซึ่งสัญญาณเป็นเอาต์พุตของ 68000 โดยที่ 68000 รับสัญญาณ $\overline{BR}$ แล้วก่อนจะตอบสนองต่อการทำงานออกไปต้องรอจนสิ้นสุดบัสไซเคิลนั้นๆ เสียก่อน เมื่ออุปกรณ์ภายนอกได้รับสัญญาณ $\overline{BG}$

อุปกรณ์ภายนอกจะขับสัญญาณ $\overline{BGACK}$ กลับมาซึ่งมีอยู่เกิดขึ้นก่อนการขับสัญญาณ $\overline{BGACK}$ อยู่ 4 เงื่อนไข ดังนี้

1. $\overline{BG}$ จะต้องทำงาน
2. $\overline{AS}$ จะต้องไม่ทำงานเพื่อแสดงว่าในขณะนั้น ซีพียู ไม่ได้ใช้บัสระบบอยู่
3. $\overline{DTACK}$ จะต้องไม่ทำงานเพื่อแสดงว่าไม่มีอุปกรณ์ภายนอกใช้บัสระบบอยู่
4. $\overline{BGACK}$ จะต้องไม่ทำงานเพื่อให้เกิดความมั่นใจ Bus Master อื่นๆ ยังไม่พร้อมที่จะเข้ามาใช้บัสของระบบ



รูปที่ 2.25 ผังงานการควบคุมการทำงานของบัส

ในการควบคุมบัสของระบบ เมื่ออุปกรณ์ภายนอกได้รับสัญญาณ $\overline{BG}$ แล้ว อุปกรณ์ภายนอกต้องทำการทำงานสัญญาณ $\overline{BGACK}$ และทำการไม่ทำงานสัญญาณ $\overline{BR}$ และอุปกรณ์ต้องทำการทำงานสัญญาณ $\overline{BGACK}$ เป็นเวลานานตราบเท่าที่อุปกรณ์นั้นยังขอใช้บัสระบบอยู่ส่วนที่ต้องไม่ทำงานสัญญาณ $\overline{BR}$ นั้น เพื่อป้องกันการตีความความผิดพลาดของซีพียูคือถ้าสัญญาณ $\overline{BR}$ ทำงานนานกว่า $\overline{BGACK}$ ทำให้ซีพียูตีความหมายเป็น การขอใช้บัสซ้อนได้ เมื่ออุปกรณ์ภายนอกดำเนินงานของตัวเองเสร็จสิ้นแล้ว และต้องการคืน Bus Master ของระบบ

ให้แก่ 68000 ทำได้โดยการไม่ทำงานสัญญาณ $\overline{BGACK}$ ดังรูปที่ 2.25 แสดงการติดต่อสำหรับการทำขบวนการควบคุมการทำงานของบัส

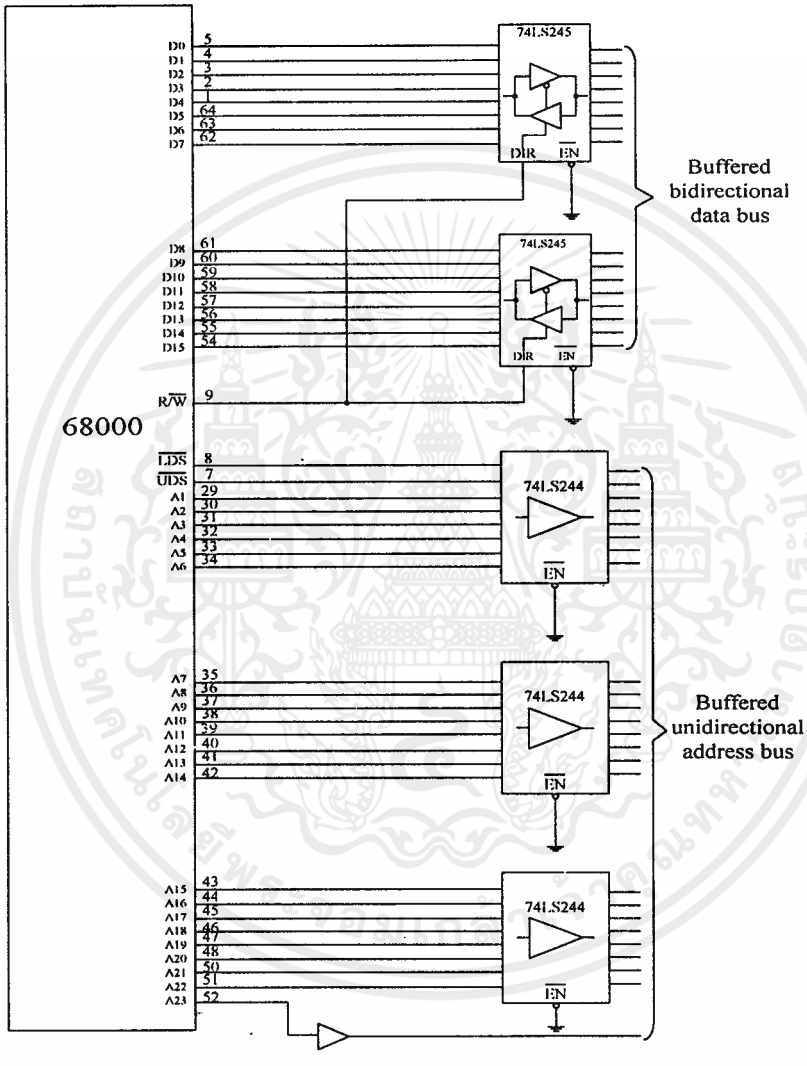
7. $\overline{AS}$, R / $\overline{W}$, $\overline{UDS}$, $\overline{LDS}$ และ $\overline{DTACK}$

ในกลุ่มนี้จะเป็นบัสควบคุมแบบอะซิงโครนัสประกอบไปด้วย 5 สัญญาณ ที่มีความสำคัญเป็นการทำให้เกิดความเหมาะสมในการดำเนินงานของฮาร์ดแวร์ภายนอก โดยที่ $\overline{AS}$, R / $\overline{W}$, $\overline{UDS}$และ $\overline{LDS}$ เป็นขาสัญญาณเอาต์พุตและ $\overline{DTACK}$ เป็นขาสัญญาณอินพุต ขาสัญญาณ $\overline{AS}$ ถูกใช้บ่งชี้ว่าขณะนี้ค่าตำแหน่งของหน่วยความจำ มีอยู่บนตำแหน่งบัสแล้ว ขาสัญญาณ R / $\overline{W}$ จะเป็นขาที่บ่งบอกว่าขณะนี้ไซเคิลการทำงานเป็นการอ่านหรือเขียนเกิดขึ้น $\overline{UDS}$ (Upper Data Strobe) และ $\overline{LDS}$ (Lower Data Strobe) ขาสัญญาณทั้ง 2 ขานี้ถูกใช้เป็นประตูของ 8 บิตข้อมูลที่ 68000จะติดต่อด้วย $\overline{UDS}$ ควบคุมข้อมูลในบิต 8 ถึงบิต15 ของบัสข้อมูล และ $\overline{LDS}$ ควบคุมข้อมูลในบิต 0 ถึงบิต 7 ถ้าต้องการที่จะโอนถ่ายข้อมูล ขนาด 8 บิต เราก็ต้องการเพียงการทำงานของขาใดขาหนึ่งระหว่าง $\overline{UDS}$ และ $\overline{LDS}$ แต่ถ้าโอนถ่ายข้อมูลขนาด 16 บิต แล้วทั้งขา $\overline{UDS}$และ $\overline{LDS}$ ต้องทำงาน และขาสัญญาณสุดท้ายคือขา $\overline{DTACK}$ ซึ่งขานี้ จะถูกทำให้เกิดสัญญาณจากวงจรภายนอก เพื่อการดำเนินการโอนถ่ายข้อมูลแบบอะซิงโครนัส (เมื่อดำเนินการถ่ายโอนข้อมูลแบบอะซิงโครนัส จะใช้ขาE, $\overline{VMA}$ และ $\overline{VPA}$) เมื่อสัญญาณ $\overline{DTACK}$ ถูกทำให้เกิดจากอุปกรณ์ฮาร์ดแวร์ภายนอกแล้ว 68000 ก็จะยอมรับว่าการโอนถ่ายข้อมูลเสร็จสิ้นสมบูรณ์ เช่นข้อมูลถูกเคลื่อนย้ายลงบนบัสข้อมูลจะยังคงเก็บค่าข้อมูลอยู่ภายในบัส สำหรับการเขียน จนกระทั่งเกิดสัญญาณ $\overline{DTACK}$ จากฮาร์ดแวร์ภายนอกแล้ว ข้อมูลเกิดอยู่ในบัสจะถูกแทนที่โดยข้อมูลอื่นได้ ถ้ายังไม่เกิดสัญญาณ $\overline{DTACK}$ แสดงว่าการเขียนยังไม่ถูกเขียนลงบนฮาร์ดแวร์ภายนอก ซึ่งเป็นหน้าที่ของอุปกรณ์ภายนอกในการกำเนิดสัญญาณ $\overline{DTACK}$ ในเวลาที่เหมาะสม เมื่อ 68000 ดำเนินการแบบ Full Speed แล้วจังหวะเวลาของสัญญาณ $\overline{DTACK}$ จะมีความสำคัญมาก

8. A_1 ถึง A_{23} , D_0 ถึง D_{15}

กลุ่มสุดท้ายที่จะกล่าวถึงคือประกอบด้วยบัสตำแหน่งและบัสข้อมูล ในกลุ่มของตำแหน่งบัส จะมีขา $\overline{UDS}$และขา $\overline{LDS}$ รวมอยู่ด้วย ซึ่งจะทำให้เกิดการอ้างตำแหน่งข้อมูลได้มากกว่า16 ล้านไบต์ของหน่วยความจำ ความสามารถในการเข้าถึงตำแหน่งหน่วยความจำโดยตรงทุกตำแหน่ง ทำให้โปรแกรมเมอร์มีสภาพแวดล้อมในการโปรแกรมมิ่งอ่อนตัวมาก A_1ถึง A_{23} เป็นบัสแบบทิศทางเดียว (Unidirectional) หรือเป็นเอาต์พุตเท่านั้น จะถูกใช้อยู่เสมอ

การติดต่อกับตำแหน่งในหน่วยความจำ เว้นเสียแต่ในขณะเกิดไซเคิลของการอินเตอร์รัพต์เกิดขึ้น ในเงื่อนไขนี้ A_1, A_2, และ A_3 จะเก็บค่าระดับของการเข้าถึงการอินเตอร์รัพต์และบัสตำแหน่งอื่นๆ จะเป็นสภาวะ เป็น 1

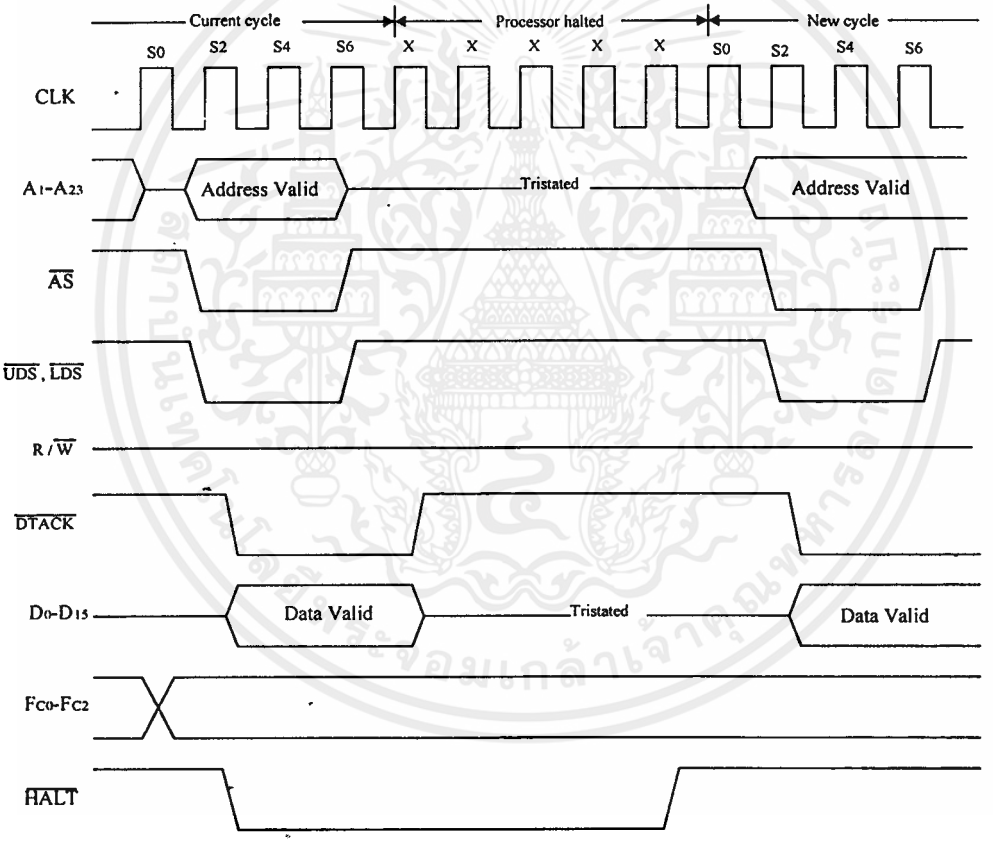


รูปที่ 2.26 การบัฟเฟอร์บัสตำแหน่งและบัสข้อมูล

D_0 ถึง D_{15} คือ บัสข้อมูลเป็นบัสแบบ 2 ทิศทาง ที่ถูกใช้ในการโอนถ่ายข้อมูลระหว่าง 68000 กับอุปกรณ์ภายนอกโดยที่ D_0ถึง D_7มีหน้าที่ในขณะที่เกิด Interrupt Acknowledge คือ ทำหน้าที่ส่งผ่าน Interrupt Vector ให้กับ 68000 และในการติดต่อ 68000 กับหน่วยความจำ เช่น EPROM, RAM ก็ต้องการใช้บัส แต่บัสเหล่านี้ก็มีขีดจำกัดในการขับอุปกรณ์ภายนอก

ซึ่งกระแสที่ขับออกมามีหน่วยเป็นมิลลิแอมป์ เหตุผลนี้จึงมีความจำเป็นต้องใช้วงจรบัฟเฟอร์พิเศษเพิ่มขึ้นมาระหว่างบัสตำแหน่งและบัสข้อมูลกับฮาร์ดแวร์ภายนอกบัฟเฟอร์จะทำให้มีความสามารถในการช่วยขับอุปกรณ์ที่มากขึ้น ดังในรูปที่ 2.26 แสดงตัวอย่างบัสตำแหน่งและบัสข้อมูลที่ต่อกับบัฟเฟอร์

แผนภาพสัญญาณนาฬิการะบบสัญญาณนาฬิกาเป็นมาตรฐานที่สำคัญนำมาใช้ในการออกแบบระบบ 68000 ในส่วนนี้จะกล่าวถึงสัญญาณที่สำคัญ 2 สัญญาณ ที่มีรายละเอียด เป็นขาสัญญาณที่ใช้ในการควบคุมโปรเซสเซอร์



รูปที่ 2.27 สถานะสัญญาณนาฬิกา HALT

สถานะสัญญาณนาฬิกา HALT จากรูปที่ 2.27 แสดงผังเวลาเมื่อ 68000 อยู่ในสถานะ halt mode จากผังเวลาที่แสดงอยู่ในไซเคิลของการอ่านข้อมูล จากสแต็ค S0 (CLK เป็น "1") ถึงสแต็ค 8 (CLK เป็น "0" รวมกันเป็น 8 สแต็คที่จำเป็นต้องมีเพื่อให้ได้ความสมบูรณ์ของ

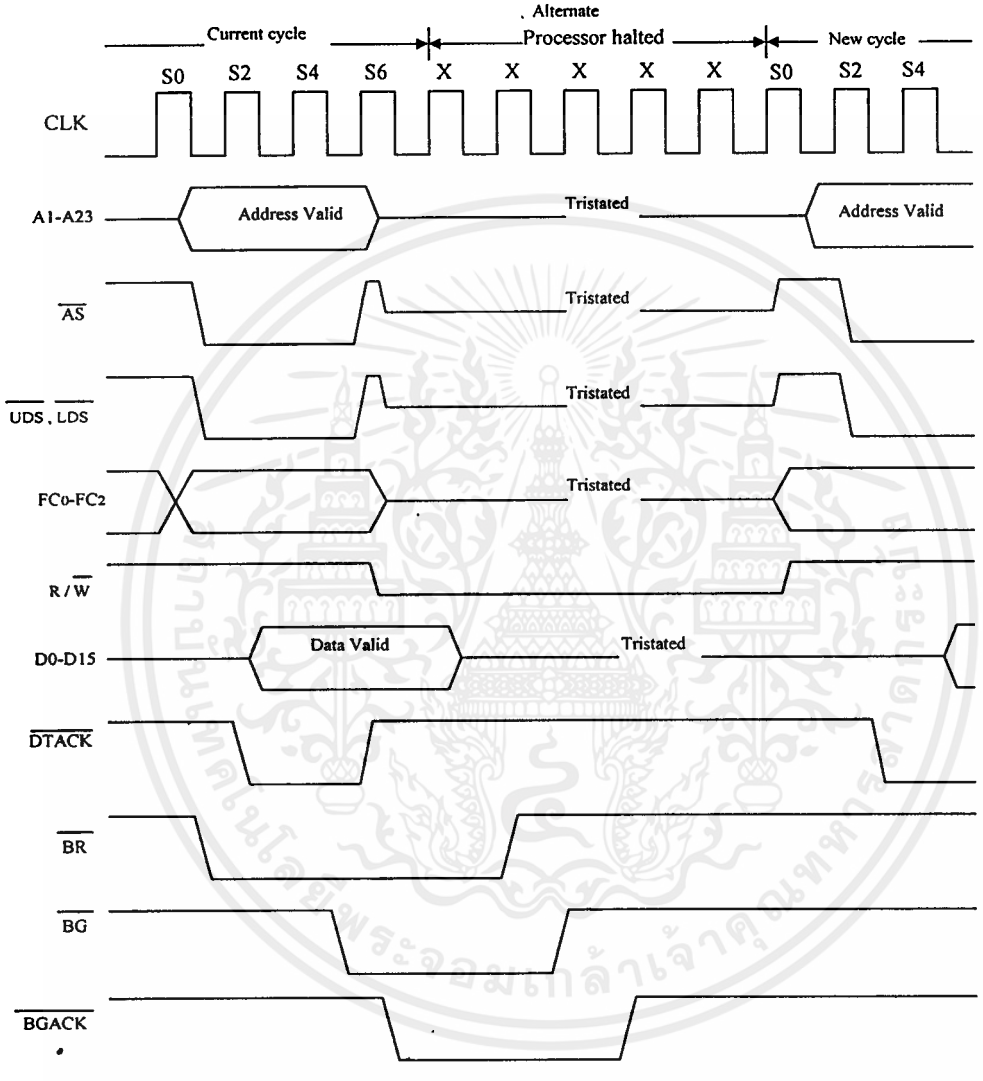
ไซเคิลแต่ละสแต็คอาจเรียกได้ว่าเป็น"ไซเคิลสแต็ค" หรือ"สแต็คไทม์" และแต่ละสแต็คก็เป็นครึ่งไซเคิลของ 1 สัญญาณนาฬิกา ดังตัวอย่าง ที่ความถี่ 10 เมกกะเฮิรตซ์ 68000 มีแต่ละสแต็คเท่ากับ 70 นาโนวินาที แต่ละสแต็คจะต้องมีจำนวนเวลาที่ต้องมารวมกัน เพื่อมาตรฐานในการดำเนินการ คำสั่งต่างๆ และทุกคำสั่งก็ต้องการส่วนประกอบของแต่ละสแต็คนี้ เพราะในการเอ็กซิคิวท์สั่งขึ้นอยู่กับความถี่ของสัญญาณนาฬิกา ดังในรูปขณะที่อยู่ในช่วง 8 สแต็ค เห็นได้ว่ามีการเปลี่ยนแปลงที่บัสตำแหน่ง (S_1 ถึง S_7) ตำแหน่งบัสมีค่าของตำแหน่งลงในบัสตำแหน่งส่วนที่สัญญาณอื่นๆ เช่น $\overline{AS}$, $\overline{DTACK}$, D_0 ถึง D_1 ความสัมพันธ์กับการประมวลผลของสแต็คไทม์นี้

เมื่อการประมวลผลของไซเคิลนี้จบสิ้น สัญญาณ $\overline{AS}$, $\overline{UDS}$ และ $\overline{LDS}$ ไม่ทำงานหรือเป็น 1 และจะสังเกตได้ว่าสัญญาณ $\overline{HALT}$ ก็มีการเกิดขึ้นในไซเคิลนี้ (S_2 ถึง S_7) และสังเกตเห็นว่าเมื่อสิ้นสุดสแต็คที่ 7 แล้ว ไมโครโปรเซสเซอร์ จะไม่เริ่มต้นบัสไซเคิลใหม่ โดยสัญญาณ $\overline{HALT}$ นี้ จะเข้าไปแทนที่การจากไปของสัญญาณควบคุมที่ 3 ($\overline{AS}$, $\overline{UDS}$และ $\overline{LDS}$) ที่ไม่ทำงานแล้ว เห็นได้ว่าแอดเดรสบัสและบัสข้อมูลจะอยู่ในสถานะ High Impedance โดยที่ในขณะ 68000 จะอยู่ในสถานะการรอคอยและช่วงเวลาของการรอก่อนที่จะมี การเริ่มต้นของไซเคิลของคำสั่งใหม่เกิดขึ้น ก็จะขึ้นอยู่กับว่าระยะเวลาที่อุปกรณ์ภายนอกส่งสัญญาณ $\overline{HALT}$ จะเข้ามาให้ 68000 จะใช้เวลาเพียงไม่กี่สเตทหลังจากอุปกรณ์ภายนอกไม่ทำงานสัญญาณ $\overline{HALT}$ แล้วในการเริ่มต้นของเอ็กซิคิวท์คำสั่งต่อไป เว้นเสียแต่มีการเกิดขึ้นของสัญญาณ $\overline{RESET}$ หรือการอินเตอร์รัพต์ในขณะ $\overline{HALT}$ สเตท เพราะทั้งสองเงื่อนไขดังกล่าวทำให้ $\overline{HALT}$ สเตทจบสิ้นลง

9. สัญญาณนาฬิกาของการควบคุมบัส

ในส่วนนี้จะอธิบายถึงการจัดการสัญญาณนาฬิกาที่มีความยุ่งยาก เมื่อเกิดเหตุการณ์ที่อุปกรณ์ภายนอกต้องการที่จะเข้ามายึดครองบัสระบบของ 68000 สัญญาณนาฬิกา ที่แสดงให้ดูดังในรูปที่ 2.28 ถึงเหตุการณ์ที่เกิดขึ้น เมื่อ 68000 ได้รับสัญญาณร้องขอการใช้บัส($\overline{BR}$) จากอุปกรณ์ภายนอกในขณะที่มีการเอ็กซิคิวท์คำสั่งเกิดขึ้น และ 68000 จะรอจนจบสิ้น ไซเคิลของคำสั่งนั้นเสียก่อนจึงตอบสัญญาณการส่งสัญญาณ Bus Grant ($\overline{BG}$) ซึ่งเป็นสัญญาณตอบรับการใช้บัสออกไป เมื่อ 68000 ส่งสัญญาณ $\overline{BG}$ ออกมาขาควบคุมสัญญาณที่ควบคุมของ 68000 ($\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, R / $\overline{W}$ และ FC_0-FC_2) และรวมถึงบัสตำแหน่งและบัสข้อมูลทั้งหมดนี้เข้าสู่

สภาวะอิมพีแดนซ์สูง เมื่อเข้าสู่สภาวะอิมพีแดนซ์สูงแล้วก็จะอนุญาตให้อุปกรณ์ภายนอกที่ทำงานร่วมกับ 68000 เข้าครอบครองบัสได้



รูปที่ 2.28 สัญญาณนาฬิกาของการควบคุมบัส

ซึ่งในขณะเวลานี้ 68000 จะไม่ตอบสนองต่อกระบวนการทุกอย่างบนบัสไซเคิล เมื่อสัญญาณ $\overline{\text{BGACK}}$ ยังทำงานอยู่ เมื่อขบวนการที่อุปกรณ์ภายนอกเข้าครอบครองบัสจบสัญญาณ $\overline{\text{BGACK}}$ ถูกทำให้ไม่ทำงานจากอุปกรณ์ภายนอก 68000 กลับเข้าสู่การเริ่มต้น

เอ็กซิคิวท์ในคำสั่งถัดไป เป็นสิ่งที่เป็นข้อได้เปรียบในการประยุกต์ใช้งานกับหน่วยความจำแบบไดนามิก RAMS ซึ่งจะต้องการความต่อเนื่องในการรีเฟรช เมื่อยังคงข้อมูลไว้อยู่

ไดนามิก RAMS ต้องการการรีเฟรชในขณะเวลาที่โปรเซสเซอร์ยังไม่ต้องการเข้าถึงข้อมูลในขณะนั้น เราจึงสามารถออกแบบวงจรรีเฟรช ซึ่งใช้ประโยชน์จาก Bus Arbitration ของ 68000 ในการ Refresh หน่วยความจำแบบไดนามิก RAMS สังเกตเมื่ออุปกรณ์ภายนอกได้รับสัญญาณ $\overline{BG}$ ต้องทำการทำงานสัญญาณ $\overline{BGACK}$ และไม่ทำงานสัญญาณ $\overline{BR}$ แล้วอุปกรณ์จะต้องยังคงทำงานสัญญาณ $\overline{BGACK}$ เป็นเวลานานอุปกรณ์หน่วยนั้น ยังคงใช้บัสของระบบส่วนที่ต้องการไม่ทำงานสัญญาณ $\overline{BR}$ นั้นก็เพื่อป้องกันการตีความผิดของซีพียู คือถ้าสัญญาณ $\overline{BR}$ ทำงานนานกว่า $\overline{BGACK}$ จะทำให้ซีพียูตีความหมายว่าเป็น การขอใช้บัสซ้อน

10. การออกแบบหน่วยความจำ

ในที่นี้จะอธิบายถึงการทำงานของขาสัญญาณควบคุมของ 68000 ($\overline{AS}$, $\overline{DTACK}$, $\overline{UDS}$, $\overline{LDS}$, และ $R/\overline{W}$) ที่ถูกใช้เป็นหลักในการอ้างถึงหน่วยความจำในการอ่านและเขียนข้อมูลในหน่วยความจำ รวมถึงการติดต่อกับหน่วยความจำแบบสเตติกและไดนามิกด้วย

ตารางที่ 2.7 UDS และ LDS function

$\overline{UDS}$	$\overline{LDS}$	$R/\overline{W}$	$D_8 - D_{15}$	$D_0 - D_7$
High	High	–	No Valid data	No Valid data
0	0	High	Valid data bit 8–15	Valid data bit 0–7
High	0	High	No Valid data	Valid data bit 0–7
0	High	High	Valid data bit 8–15	No Valid data
0	0	0	Valid data bit 8–15	Valid data bit 0–7
High	0	0	Valid data bit 0–7	Valid data bit 0–7
0	High	0	Valid data bit 8–15	Valid data bit 8–15

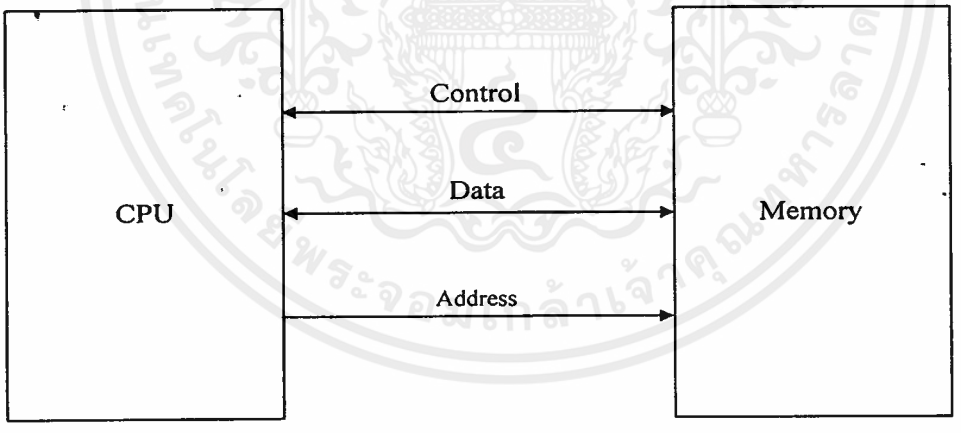
การทำงานของอุปกรณ์ภายนอกมีความต้องการจะขอใช้บัสของ 68000 ในขณะที่ซีพียูยึดครองอยู่นี้ เรียกว่า ขบวนการ DMA

11. แอดเดรสบัสและบัสข้อมูลของ 68000

68000 ไมโครโปรเซสเซอร์ขนาด 16 บิต มีความสามารถในการอ่านและเขียนข้อมูลขนาด 8 บิต หรือ 16 บิตในหนึ่งครั้ง แอดเดรสบัสขนาด 24 บิต สามารถอ้างหน่วยความจำภายนอกได้ 16 MB ซึ่งจะใช้เพียง A_1-A_{23} ซึ่งเป็นจำนวน 23 เส้น ส่วนที่เหลือคือ A_0 ที่อยู่ภายในซีพียูจะเป็นสายสัญญาณควบคุม คือ $\overline{UDS}$ และ $\overline{LDS}$ ซึ่งเป็นการโอนถ่ายข้อมูลไม่ว่าจะเป็นแบบไบต์หรือแบบเวิร์ดจะสามารถควบคุมได้ที่ $\overline{UDS}$ และ $\overline{LDS}$ จะเห็นได้ว่า $\overline{UDS}$ และ $\overline{LDS}$ จะเป็นการบ่งบอกถึงลักษณะการโอนถ่ายข้อมูลที่เกิดขึ้นบนบัสข้อมูลตารางที่ 2.7 เป็นรายละเอียดของการฟังก์ชั่นการทำงานของสัญญาณ $\overline{UDS}$ และ $\overline{LDS}$

12. บัสบัฟเฟอร์

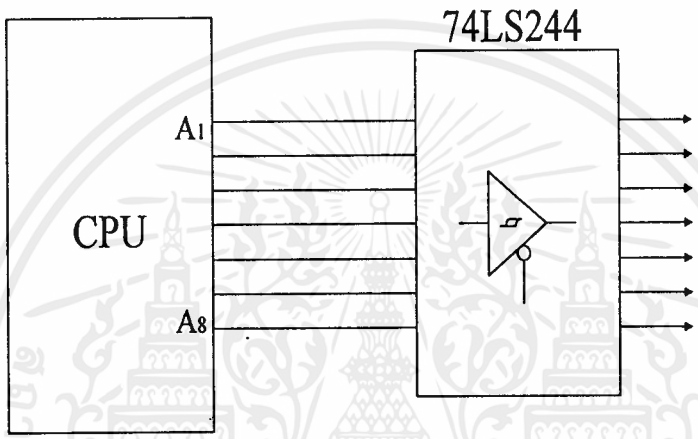
หน่วยความจำพื้นฐานที่ทำงานกับไมโครโปรเซสเซอร์จะเป็น RAM หรือ EPROM ซึ่งจะติดต่อเข้ากับบัสของไมโครโปรเซสเซอร์โดยตรงในการจะอ่านหรือเขียนข้อมูลจากหน่วยความจำ ดังแสดงในรูปที่ 2.29 ซึ่งเรียกว่า บัสควบคุม, บัสข้อมูลและแอดเดรสบัส ซึ่งแสดงให้เห็นถึงความสัมพันธ์ของซีพียู, บัสและระบบหน่วยความจำ



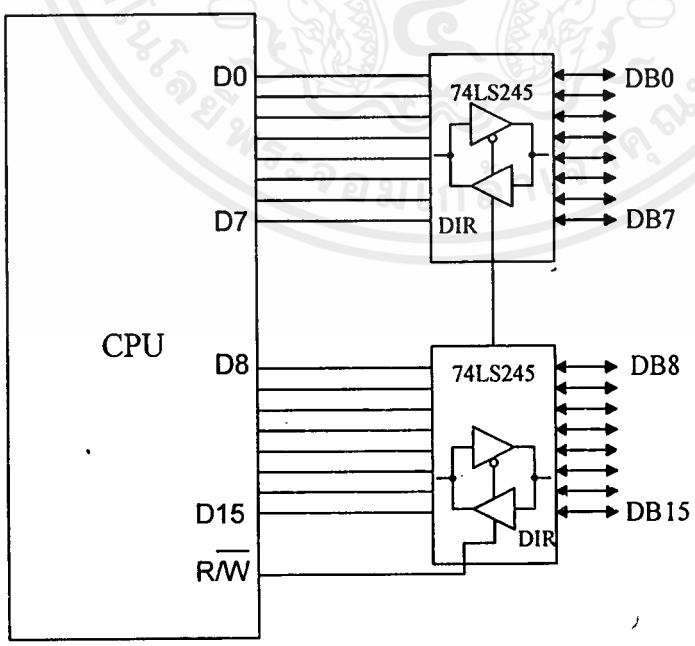
รูปที่ 2.29 โครงสร้างของบัสในหน่วยความจำ

ถ้าหากว่าซีพียูเกิดขับโหลดที่ต่อด้วยหลายๆ ตัว เป็นสาเหตุให้เกิดโอเวอร์โหลดขึ้น จึงมีความจำเป็นที่จะต้องพิจารณาค่า Fan Out ด้วย แต่ถ้าหากว่าต้องมีความจำเป็นในการเชื่อมบัสตำแหน่งและบัสข้อมูลกับหน่วยความจำจนเกิดการโอเวอร์โหลดเกิดขึ้น การแก้ปัญหาจะใช้การบัฟเฟอร์ระหว่างซีพียูกับหน่วยความจำ

ในรูปที่ 2.30 แสดงการต่อสายสัญญาณแอดเดรสบัสเข้ากับบัฟเฟอร์ 74LS244 ซึ่งมีอินพุตและเอาต์พุตขนาด 8 บิต โดยที่แอดเดรสบัสของ 68000 จะมีความสามารถในการขับกระแส Sink เท่ากับ 3.2 มิลลิแอมป์ แต่เมื่อนำบัฟเฟอร์ 74LS244 มาเชื่อมต่อระหว่างซีพียูกับโหลด ทำให้ประสิทธิภาพของกระแสซิงค์เพิ่มขึ้นเป็น 24 มิลลิแอมป์ เมื่อเชื่อมต่อบัฟเฟอร์ 74LS244 แล้วก็จะทำให้ CPU สามารถขับอุปกรณ์หน่วยความจำได้มากขึ้น



รูปที่ 2.30 การต่อสายสัญญาณแอดเดรสกับบัฟเฟอร์ 74LS244



รูปที่ 2.31 การบัฟเฟอร์แบบ 2 ทิศทาง โดยใช้ IC เบอร์ 74LS245

ส่วนในการบัฟเฟอร์ที่บัสข้อมูลนั้นเนื่องจากว่าบัสข้อมูลเป็นบัสแบบ 2 ทิศทาง จำเป็นต้องใช้การบัฟเฟอร์ทั้ง 2 ทิศทาง ในรูปที่ 2.31 แสดงการบัฟเฟอร์แบบ 2 ทิศทาง โดยใช้ไอซีเบอร์ 74LS245 ข้อมูลที่จะส่งผ่านบัฟเฟอร์ สามารถควบคุมได้จากสัญญาณ DIR ของตัวบัฟเฟอร์ ซึ่งจะบอกว่าการส่งผ่านข้อมูลเป็นในแบบซ้ายไปขวาหรือเป็นเอาต์พุตของซีพียู และจากขวาไปซ้ายหรือเป็นอินพุตของซีพียู ซึ่งในการควบคุมทิศทางของ 74LS245 ก็จะอาศัยขาสัญญาณ R / $\overline{W}$ ของ 68000 ซึ่งจะเป็นตัวบ่งบอกบนบัสของ 68000

13. การเข้าถึงหน่วยความจำ (Accessing Memory)

รูปแบบของการดำเนินการที่จะพิจารณาคือ

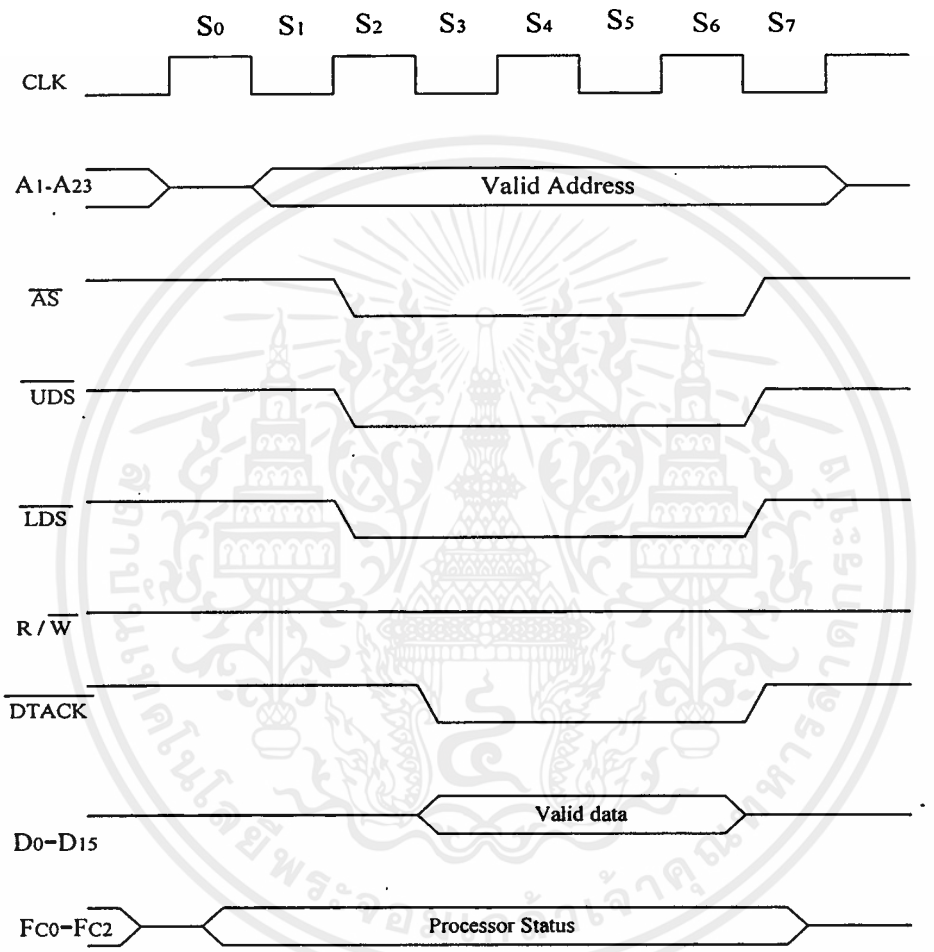
– อ่านข้อมูลจากหน่วยความจำ (Read data form memory)

– เขียนข้อมูลลงในหน่วยความจำ (Write data form memory)

– ไม่มีการเข้าถึงหน่วยความจำ (Do not access memory)

สองเงื่อนไขแรกเป็นการโอนถ่ายข้อมูลระหว่าง 68000 กับหน่วยความจำ เงื่อนไขที่สาม จะปรากฏขึ้น เมื่อ 68000 ดำเนินการในหน้าที่อื่นหรือเกิดการเอ็กซิคิวท์คำสั่งขึ้นและไม่มีความต้องการที่จะติดต่อกับหน่วยความจำ ซึ่งถ้าเกิดสัญญาณที่ขาสัญญาณของ 68000 เป็นอย่างใดก็แสดงถึงความต้องการของ 68000 ว่าต้องการติดต่อกับหน่วยความจำหรือไม่ เช่น ที่ขา $\overline{AS}$ ซึ่งเป็นเอาต์พุตถ้าหากเป็นศูนย์แสดงว่า 68000 ต้องการเข้าถึงหน่วยความจำ และถ้าเป็น 1 แสดงว่า 68000 กำลังทำงานอื่นอยู่ เมื่อไรที่อุปกรณ์หน่วยความจำเห็นสัญญาณ $\overline{AS}$ เป็น 0 ก็จะเป็นการอธิบายได้ว่า 68000 กำลังส่งค่าตำแหน่งในหน่วยความจำลงบนบัสตำแหน่ง และถ้าตำแหน่งในหน่วยความจำที่ส่งออกไป เป็นช่วงตำแหน่งในหน่วยความจำที่ถูกอนุญาตให้ติดต่ออุปกรณ์หน่วยความจำจะตอบสนองต่อ 68000 เพื่อดำเนินการโอนย้ายข้อมูลด้วยการส่ง $\overline{DTACK}$ กลับมาเป็น 0 สัญญาณ $\overline{DTACK}$ ถูกใช้ในการเข้าจังหวะระหว่าง 68000 กับอุปกรณ์หน่วยความจำ หลังจากการ 68000 ส่งค่าตำแหน่งหน่วยความจำลงบนบัสตำแหน่งพร้อมกับส่งสัญญาณ $\overline{AS}$ แล้ว 68000 จะทำการรอจนกระทั่งเกิดสัญญาณ $\overline{DTACK}$ กลับมาเป็น 0 จึงทำให้เป็นความรับผิดชอบของอุปกรณ์หน่วยความจำต้องทำงานสัญญาณ $\overline{DTACK}$ กลับมาในการดำเนินการปกติ 68000 จะใช้เวลาในการโอนย้ายข้อมูลสั้นมาก เมื่อได้รับสัญญาณ $\overline{DTACK}$ แล้ว

รูปที่ 2.32 แสดงสัญญาณนาฬิกาของวัฏจักรของการอ่านข้อมูลอย่างน้อยที่สุดของวัฏจักรการอ่านข้อมูลจะมีจำนวน 8 สเตท คือ จะเริ่มจากสเตท S_0 ไปจนถึงสเตท S_7 สองสเตทเท่ากับ 1 ไซเคิล



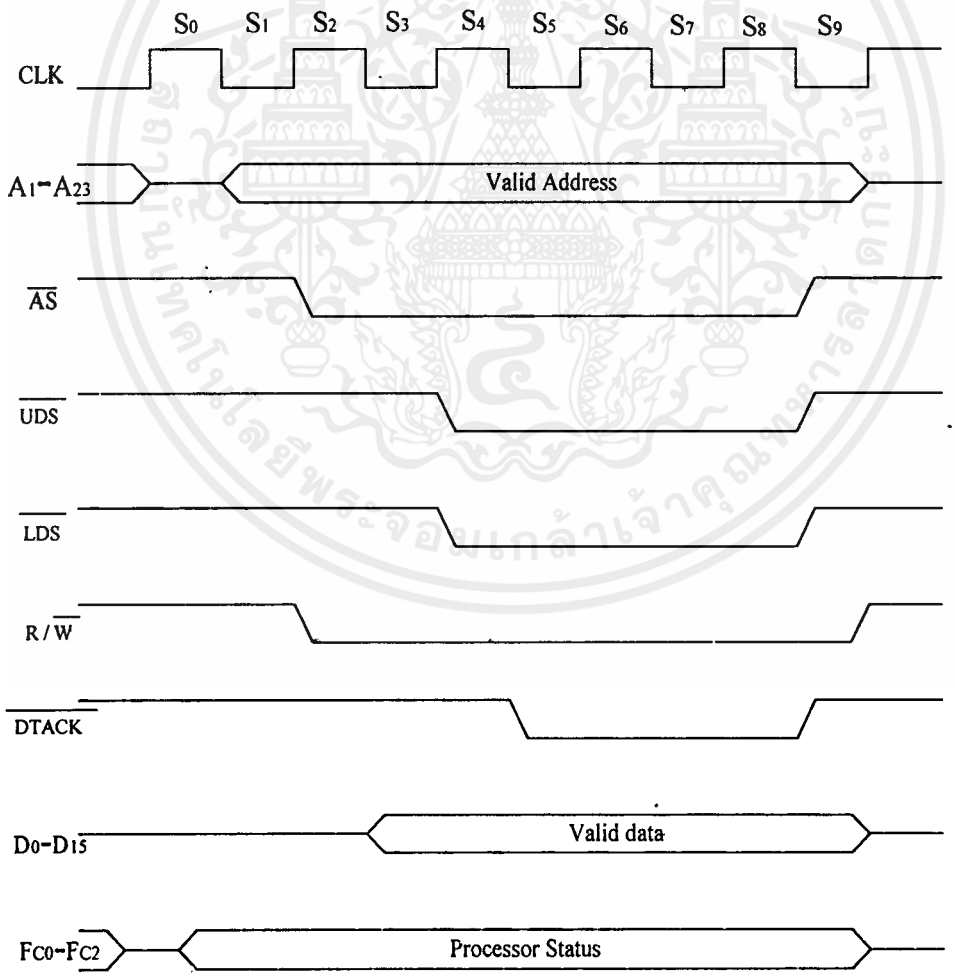
รูปที่ 2.32 ผังงานเวลาของการอ่านหน่วยความจำ (Memory read cycle timing)

ที่สเตท S_0 เป็นการเริ่มต้นของวัฏจักรการอ่านบัสตำแหน่งและ Output Function Code จะอยู่ในสภาวะอิมพีแดนซ์สูงและที่ขาสัญญาณ R / $\overline{W}$ จะถูก set เป็น "1"

ที่สเตท S_1 ฟังก์ชั่นรหัสจะส่งข้อมูลลงมาบนบัส (บ่งบอกถึงประเภทในการดำเนินงานของบัสไซเคิล) และบัสตำแหน่งออกจากสภาวะ High-impedance ตำแหน่งของหน่วยความจำที่จะติดต่อด้วยอยู่บนบัสแทน

ที่สเตท S_0 สัญญาณ $\overline{AS}$, $\overline{UDS}$ และ $\overline{LDS}$ จะทำงาน (เมื่อ $\overline{UDS}$ และ $\overline{LDS}$ ทำงานพร้อมแสดงว่าการอ่านข้อมูลแบบเวิร์ด) และพร้อมกันนี้ อุปกรณ์ฮาร์ดแวร์นอกจะทำการถอดรหัสหน่วยความจำที่ต้องการติดต่อด้วย

ในช่วง S_3 ถึง S_4 เกิดการส่งข้อมูลมาบนบัสข้อมูลและทำงานสัญญาณ $\overline{DTACK}$ ซึ่งสัญญาณ $\overline{DTACK}$ ถูกส่งมาจากอุปกรณ์ฮาร์ดแวร์ภายนอก เพื่อบอกซีพียูให้ดำเนินการต่อไป โดยทั่วไปซีพียูเริ่มต้นมองหาสัญญาณ $\overline{DTACK}$ เมื่อถึง S_5 ถ้าถึง S_5 แล้ว อุปกรณ์ฮาร์ดแวร์ภายนอกก็ยังไม่ส่ง $\overline{DTACK}$ กลับมาซีพียูจะทำการรอเกิดขึ้น ถ้าเป็น RAM หรือ EPROM ชนิดมีความไวในการทำงานจะเกิดสัญญาณ $\overline{DTACK}$ ได้ ประมาณ S_5 แต่ถ้าเป็นชนิดซีพียูจำเป็นต้องสร้างสัญญาณรอ 1 ไซเคิลหรือมากกว่าการเกิดสัญญาณ



รูปที่ 2.33 ผังงานเวลาของการเขียนหน่วยความจำ(Memory write cycle timing)

ในรูปที่ 2.33 แสดงการดำเนินการในวัฏจักรของการอ่านข้อมูล ซึ่งวัฏจักรนี้ต้องการ 10 สเตท ในการทำงาน (S_0 ถึง S_9) ระหว่างสเตท S_0 เอาต์พุตของฟังก์ชั่นและบัสตำแหน่งเป็นอิมพีแดนซ์สูง

ที่สเตท S_1 บัสตำแหน่งและฟังก์ชั่นโค๊ดและทำงาน

ที่สเตท S_2 สัญญาณ $\overline{AS}$ เป็น 0 และสัญญาณ $R/\overline{W}$ ก็เป็น 0 ด้วย (บ่งบอกว่าการเขียนข้อมูลลงหน่วยความจำ)

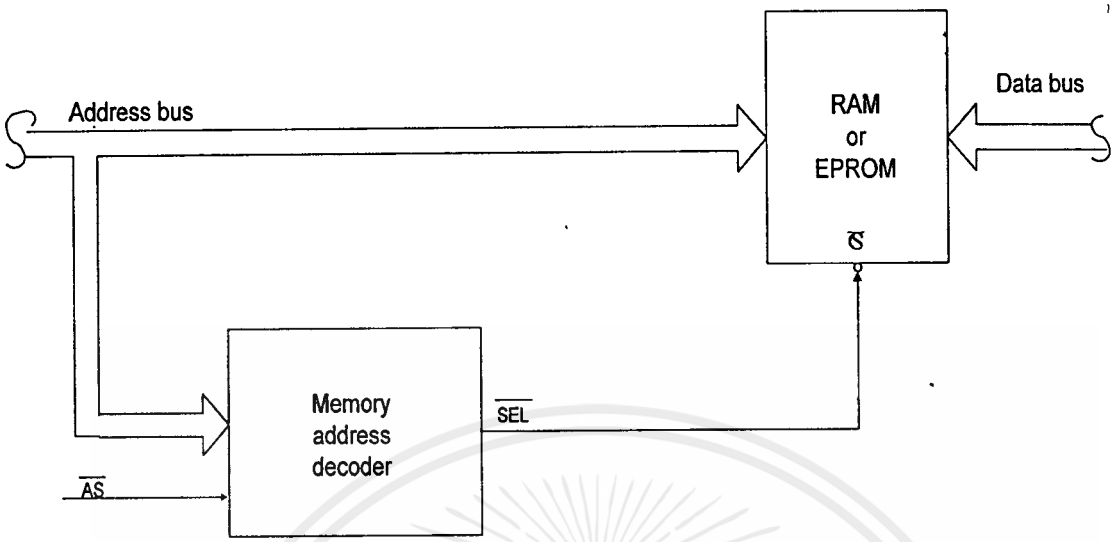
ถ้าสัญญาณ $\overline{DTACK}$ เกิดขึ้นก่อน สเตท S_7 แล้ววัฏจักรการทำงานนี้จะดำเนินต่อไป ถ้าสัญญาณ $\overline{DTACK}$ ไม่เกิดขึ้นซีพียูจะทำการสร้างสัญญาณการรอในสเตท S_7 ถึง S_8 หรือจนกระทั่งเกิดสัญญาณ $\overline{DTACK}$ ขึ้น จากรูปที่ 2.33 จะเห็นว่า และ กลับเป็น 1 ที่ช่วงเริ่มต้นของสเตท S_9 จากรูปที่ 2.33 สัญญาณ $\overline{AS}$, $\overline{UDS}$ และ $\overline{LDS}$ กลับมาเป็น 1 ที่ช่วงเริ่มต้นของสเตท S_9 ซึ่งเป็นสเตทสุดท้ายของวัฏจักรการเขียน

ในทั้งสองเงื่อนไขทั้งการอ่านและเขียนข้อมูลนั้น ในความเป็นจริงเวลาในการกลับเป็นสัญญาณตรงกันข้าม (Negation) ของ $\overline{AS}$ และ $\overline{DTACK}$ จะสามารถแปรเปลี่ยนได้ ขึ้นอยู่กับการออกแบบอุปกรณ์ฮาร์ดแวร์ภายนอก (จากรูปจะเห็นเป็นการ Negation พร้อมกัน) แต่ก็ไม่อนุญาตให้เกิน 1 ไซเคิล

2.9 การออกแบบการถอดรหัสตำแหน่งในหน่วยความจำ

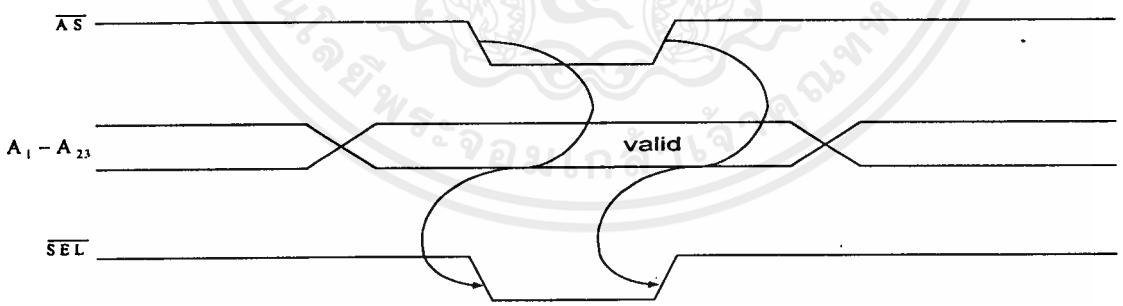
การทำงานของการถอดรหัสตำแหน่งในหน่วยความจำคือ การสถานะของบัสตำแหน่งพร้อมกับการทำงานของชิพหน่วยความจำที่ถูกทำให้ใช้งานในที่นี้ความหมายของชิพหน่วยความจำ คือ RAM หรือ EPROM ที่ผู้ออกแบบต้องการใช้ในระบบคอมพิวเตอร์ ดังนั้นก่อนที่ผู้ออกแบบจะเริ่มต้นการออกแบบจำเป็นต้องตัดสินใจ ต่อขนาดหน่วยความจำที่ต้องการ เช่น กำหนด 8 กิโลเวิร์ด ของหน่วยความจำ EPROM เพียงพอดังนั้นแล้วผู้ออกแบบรู้ว่าต้องใช้เส้นตำแหน่งจำนวน 13 เส้นที่ต้องการในการอ้างค่าตำแหน่งในหน่วยความจำที่ครอบคลุมทุกตำแหน่งทั้งหมดใน EPROM ใช้บัสข้อมูลจำนวนเท่าใดกับหน่วยความจำขนาด 32 กิโลเวิร์ด คำตอบคือ 15 เส้น เพราะว่า 2 กำลัง 15 เท่ากับ 32768

ขั้นตอนแรกที่ผู้ออกแบบการถอดรหัสตำแหน่งในหน่วยความจำคือจำนวนเส้นตำแหน่งที่ต้องใช้ต่อกับหน่วยความจำ ตำแหน่งที่เหลือจะถูกใช้ในการถอดรหัสตำแหน่ง



รูปที่ 2.34 ผังงานของการถอดรหัสตำแหน่งในหน่วยความจำ

ตามรูปที่ 2.34 แสดงผังงานของการถอดรหัสตำแหน่งในหน่วยความจำซึ่งต่อกับชิพหน่วยความจำและรูปที่ 2.35 แสดงผังงานเวลาของการทำงานตำแหน่งบัสและสัญญาณ $\overline{AS}$ การถอดรหัสตำแหน่งในหน่วยความจำจะรอให้มีการเกิดของตำแหน่งในหน่วยความจำบนตำแหน่งและสัญญาณ $\overline{AS}$ เป็น 0 ก่อนแล้วจึงให้มีการเกิดขึ้นของสัญญาณ $\overline{SEL}$ เป็น 0



รูปที่ 2.35 ผังเวลาของการถอดรหัสตำแหน่งในหน่วยความจำ

เมื่อสัญญาณ $\overline{SEL}$ เป็น 0 แล้วจะเป็นเหตุให้ $\overline{CS}$ ซึ่งเป็นอินพุตของหน่วยความจำถูกทำให้ Enables หน่วยความจำจะต่อกับบัสข้อมูลของซีพียู เมื่อตำแหน่งหน่วยความจำบนตำแหน่งบัสเปลี่ยนไปจากเดิมหรือกำเนิดสัญญาณ $\overline{AS}$ เป็นสถานะสูงเกิดขึ้นแล้ว จะทำให้

เอาต์พุตของการถอดรหัสจะเป็น High ก็จะเป็นสาเหตุทำให้ไม่เกิดการทำงานขึ้นที่ชิพหน่วยความจำและจะทำให้บัฟเฟอร์ภายในของชิพหน่วยความจำเป็นสถานะอิมพีแดนซ์สูง ดังนั้นจึงทำให้ RAM หรือ EPROM มี สถานะเป็นไม่มีการติดต่อกับบัสข้อมูล

2.9.1 วงจรกำเนิดสัญญาณ $\overline{\text{DTACK}}$ (Generating $\overline{\text{DTACK}}$)

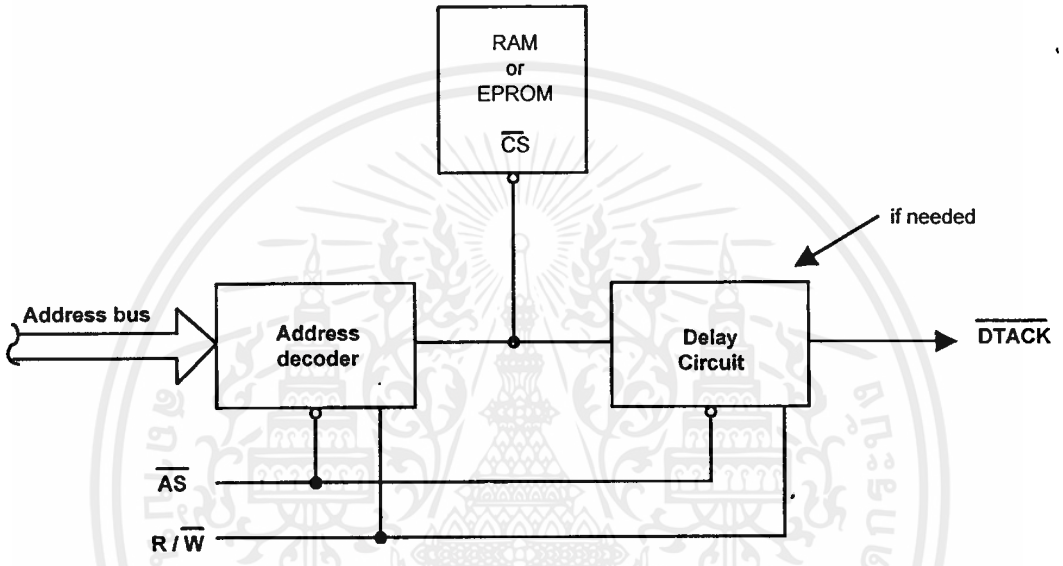
$\overline{\text{DTACK}}$ เป็นสัญญาณที่เป็นอินพุตของซีพียู 68000 ว่ามีอ่านหรือเขียนข้อมูลในหน่วยความจำเกิดขึ้นในหัวข้อนี้ เพื่อให้เกิดความเข้าใจถึงการทำงานในการถอดรหัสตำแหน่งในหน่วยความจำ เงื่อนไขแรกของการทำงานคือมีการถอดรหัสตำแหน่งในหน่วยความจำโดยอาศัยข้อมูลจากตำแหน่งบัส เมื่อทำการ Active หน่วยความจำ RAM หรือ EPROM โดยที่ตำแหน่งที่ทำการถอดรหัสต้องอยู่ในช่วงระยะของการถอดรหัสที่ได้กำหนดไว้ เงื่อนไขต่อมาของการถอดรหัส ในช่วงนี้ซีพียูจะสร้างสถานะการรอจนกระทั่งหน่วยความจำมีเวลาเพียงพอในการกระทำตามคำสั่งให้เสร็จสมบูรณ์ ถ้าเป็นหน่วยความจำประเภท RAM ต้องการเวลาประมาณ 100 นาโนวินาที ในการทำงานภายหลังจากทำให้เกิดการทำงานแล้ว ซึ่งในระหว่างนี้เป็นความต้องการของวงจรภายในหน่วยความจำในการเข้ารหัสตำแหน่งที่สำรองไว้และทำการเริ่มต้นสภาวะการทำงานของบัฟเฟอร์ภายใน วัฏจักรในการอ่านและเขียนข้อมูลที่เกิดขึ้นนี้ใช้เวลาภายใน 100 นาโนวินาที ในการเข้าถึงข้อมูลปัญหาอื่นที่อาจเกิดขึ้นเช่นมีข้อมูลสูญหายเกิดขึ้น

ในรูปที่ 2.36 ผังงานของวงจรกำเนิดสัญญาณ $\overline{\text{DTACK}}$ เห็นได้ว่าเอาต์พุตของตำแหน่งถอดรหัสจะต่อกับอุปกรณ์หน่วยความจำและวงจรหน่วงเวลา โดยที่วงจรกำเนิดการหน่วงเวลาจะถูกสร้างขึ้นถ้าเกิดมีความต้องการเกิดขึ้นและถือว่าการหน่วงเวลาที่เกิดขึ้นเป็นสัญญาณ $\overline{\text{DTACK}}$ และการหน่วงเวลาที่เกิดขึ้นต้องเป็นเวลาที่เหมาะสมด้วยและจะเห็นว่าสัญญาณ $\overline{\text{AS}}$ ก็จะถูกต่อเข้ากับวงจรหน่วงเวลาด้วยเพื่อเป็นการรับรอง ได้ว่าเมื่อ $\overline{\text{AS}}$ เป็น High แล้ว $\overline{\text{DTACK}}$ ก็เป็น 1 สัญญาณ $\text{R}/\overline{\text{W}}$ ถูกต่อเข้ากับวงจรหน่วงเวลา เพราะว่าวัฏจักรของการเขียนนั้นใช้เวลาที่ไม่เท่ากันและการหน่วงเวลาที่เกิดขึ้นนี้เราสามารถปรับแต่ง เพื่อให้เกิดความถูกต้องทั้งนี้ การปรับแต่งนี้ขึ้นอยู่กับประเภทไซเคิลการทำงานของบัส

ถ้ามีวงจรหน่วงเวลา เกิดขึ้นแล้วเราจำเป็นต้องทราบสิ่งต่าง ๆ ดังต่อไปนี้

- สเตทเวลาของการประมวลผล
- เวลาที่ใช้ในการถอดรหัสตำแหน่ง
- เวลาที่ใช้ในการเข้าถึงหน่วยความจำ

สเตทเวลาของการประมวลผลคือจำนวนผลรวมของสัญญาณนาฬิกาทั้งที่เป็น LOW หรือ High ดังนั้นแล้วครึ่งของคาบเวลา (Period) ของสัญญาณนาฬิกา คือ สเตทเวลา คาบเวลาเป็นส่วนกลับของความถี่ 63000 ทำงานที่ความถี่ 8 เมกกะเฮิรตซ์ จึงมีคาบเวลาเท่ากับ $\frac{1}{8\text{MHz}}$ = 125 นาโนวินาที ดังนั้นแล้วสเตทเวลาจึงเท่ากับ 62.5 นาโนวินาที



รูปที่ 2.36 ผังงานของวงจรกำเนิดสัญญาณ $\overline{DTACK}$

เวลาที่ใช้ในการถอดรหัสตำแหน่งจะเท่ากับผลรวมของเวลาที่ใช้ในการทำงานของแต่ละเกท ที่ใช้ในการถอดรหัส ถ้าวงจรถอดรหัสประกอบไปด้วยน็อตเกทจำนวนเล็กน้อย ซึ่งจะทำงานร่วมกับแนนด์เกทชนิดหลายอินพุต รวมของเวลาที่ใช้ในการทำงาน 35 นาโนวินาที ถ้าวงจรที่ใช้ในการถอดรหัสมากไปกว่านี้ เวลาที่ใช้ในการทำงานก็จะเพิ่มขึ้นด้วย

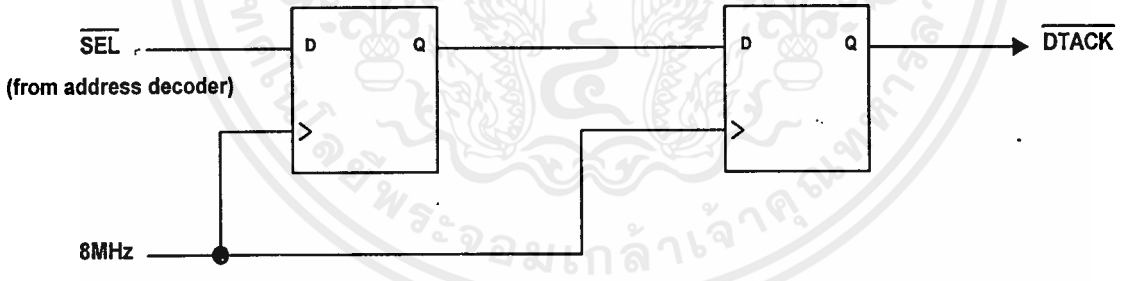
หน่วยความจำ EPROM ใช้ในการเข้าถึงข้อมูลอยู่ในช่วง 150 ถึง 200 นาโนวินาที ส่วนหน่วยความจำประเภทสเตติก RAM เวลาในการเข้าถึงข้อมูลน้อยกว่า 100 นาโนวินาที และหน่วยความจำประเภทไดนามิก RAM เวลาในการเข้าถึงข้อมูลประมาณ 70 นาโนวินาที

ถ้าการทำงานอยู่ในวัฏจักรของการอ่านข้อมูลแล้ว สัญญาณ $\overline{AS}$ จะยังไม่เกิดจนกว่าจะขึ้นสเตทที่ 5 ข้อมูลจะถูกอ่านเข้าซีพียูเมื่อสิ้นสเตทที่ 6 ในที่นี้มีความหมายว่าวงจรถอดรหัสและอุปกรณ์หน่วยความจำมีความสมบูรณ์ของการทำงานภายใน 5 สเตทเวลา ซึ่งแต่ละสเตท

เท่ากับ 62.5 นาโนวินาที ดังนั้นแล้วในการทำงานของการอ่านข้อมูลของระบบหน่วยความจำ จึงใช้เวลาเท่ากับ 312.5 นาโนวินาที เวลาที่ใช้ในการถอดรหัส 35 นาโนวินาที และถ้าใช้ EPROM ชนิดความไวสูงแล้วใช้เวลาประมาณ 150 นาโนวินาที เราต้องการเวลาเพียง 185 นาโนวินาที สำหรับการดำเนินการอ่านข้อมูลให้สมบูรณ์หลังจากการเริ่มต้นของสเตทที่ 2 และสัญญาณ $\overline{\text{DTACK}}$ จะกำเนิดขึ้นในช่วงเวลาสเตทที่ 3

แต่ถ้าเราใช้หน่วยความจำ EPROM รุ่นเก่าเวลาที่ใช้ในการเข้าถึงข้อมูลจะเป็น 350 นาโนวินาที ดังนั้นเวลาที่ใช้ในการถอดรหัสรวมกับเวลาที่ใช้กับตัว EPROM จะไม่เท่ากับ 185 นาโนวินาที ส่วนเวลาในการทำงานของการอ่านข้อมูลของระบบหน่วยความจำต้องมากกว่า 312.5 นาโนวินาที ซึ่งในเงื่อนไขนี้วงจรหน่วงเวลาที่ใช้ในการกำเนิดสัญญาณจะกำเนิดขึ้น ความจำต้องมากกว่า 312.5 นาโนวินาที ซึ่งในเงื่อนไขนี้ วงจรหน่วงเวลาที่ใช้ในการกำเนิดสัญญาณ $\overline{\text{DTACK}}$ จะกำเนิดขึ้นในระหว่างสเตทที่5 เมื่อเป็นเช่นนี้ 68000 จึงจำต้องสร้าง CLK เพิ่มขึ้นมาอีก 2 สเตท ในวัฏจักรของการอ่านข้อมูล

ทำอย่างไรจึงสร้างวงจรหน่วงเวลาเมื่อกำเนิดสัญญาณ $\overline{\text{DTACK}}$ คือเราใช้วงจรชิฟรีจิสเตอร์ ดังในรูปที่ 2.37 วงจร 2 บิตชิฟรีจิสเตอร์

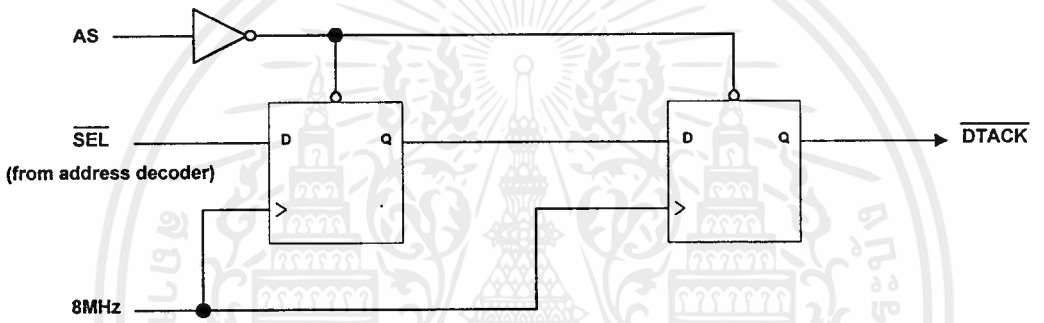


รูปที่ 2.37 การหน่วงเวลาเมื่อกำเนิดสัญญาณ $\overline{\text{DTACK}}$ โดยใช้วงจร 2 บิตชิฟรีจิสเตอร์

ป้อนอินพุตของ D ฟลิปฟลอป ที่ขา D_1 สัญญาณ $\overline{\text{SEL}}$ เอาต์พุตของ D ฟลิปฟลอป ตัวที่ 1 คือขาQ_1 เป็นอินพุตของ D ฟลิปฟลอป ตัวที่ 2 คือขาD_2 และเอาต์พุตของ D ฟลิปฟลอป ตัวที่2 คือขา Q_2 ก็จะเป็นสัญญาณ $\overline{\text{DTACK}}$ ด้วยถ้าสัญญาณ $\overline{\text{SEL}}$ เป็น 0 แล้วจากคุณสมบัติของ D ฟลิปฟลอป นั้น $\overline{\text{DTACK}}$ ก็จะเป็น 0 เมื่อสัญญาณ Clock ผ่านไป 2 ลูก ถ้าทำงานที่ความถี่ 8 เมกกะเฮิรตซ์ ถ้าสัญญาณ Clock แต่ละลูกเท่ากับ 125 นาโนวินาที เพราะฉะนั้น $\overline{\text{DTACK}}$ จะเป็นศูนย์ได้ เมื่อ $\overline{\text{SEL}}$ จะเป็นศูนย์ได้เมื่อศูนย์และเวลาผ่านไป 250 นาโนวินาที (สัญญาณ

นาฬิกา 2 ลูก) ซึ่งจะเป็นสาเหตุให้ $\overline{\text{DTACK}}$ เกิดขึ้นหลังสเตทที่5 จึงมีความจำเป็นต้องขยายบัสไซเคิลต่อไป

การทำงานของสัญญาณ $\overline{\text{DTACK}}$ ต้องทำงานร่วมกับสัญญาณ $\overline{\text{AS}}$ เป็น High แล้ว $\overline{\text{DTACK}}$ ก็ต้องเป็น High และสัญญาณ $\overline{\text{AS}}$ ก็ควบคุมการถอดรหัสตำแหน่งและเอาต์พุตของวงจรถอดรหัสตำแหน่ง คือสัญญาณ $\overline{\text{SEL}}$ และสัญญาณ $\overline{\text{SEL}}$ จะเป็น High เมื่อสัญญาณ $\overline{\text{AS}}$ เป็น High แต่ $\overline{\text{DTACK}}$ ไม่เป็น High ถ้าปราศจากสัญญาณ $\overline{\text{SEL}}$ และเวลาผ่านไป 250 นาโนวินาที (คุณสมบัติ D ฟลิปฟลอป) แต่ถ้าเราต้องการให้ $\overline{\text{DTACK}}$ เป็น High พร้อมกันเมื่อ $\overline{\text{AS}}$ เป็น High แล้ว จำเป็นต้องปรับปรุงวงจรชิฟรีจิสเตอร์



รูปที่ 2.38 วงจรหน่วงเวลาที่ใช้ในการกำเนิดสัญญาณ $\overline{\text{DTACK}}$ ที่ผ่านการปรับปรุงให้ดีขึ้น

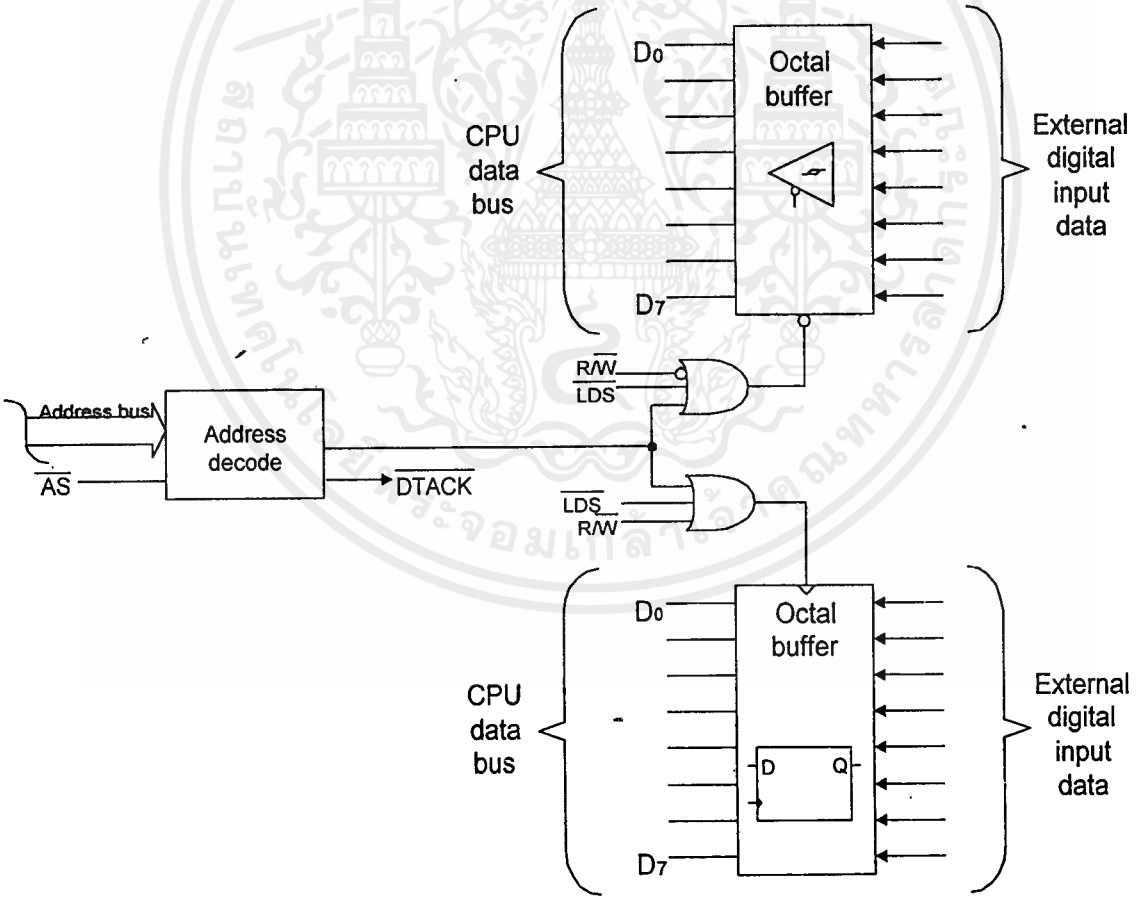
ในรูปที่ 2.38 จะเห็นว่าเราต่อสัญญาณ $\overline{\text{AS}}$ เข้ากับขา Preset ของ D ฟลิปฟลอป โดยผ่านน็อตเกท เมื่อใดที่ $\overline{\text{AS}}$ เป็น High แล้ว ที่อินพุตของ Preset จะเป็น 0 จากคุณสมบัติของ D ฟลิปฟลอป จะเป็นการ set ที่เอาต์พุตของ D ฟลิปฟลอป โดยตรงดังนั้น $\overline{\text{DTACK}}$ ก็จะเป็น High ด้วย แต่เมื่อ $\overline{\text{AS}}$ เป็น 0 ของ $\overline{\text{AS}}$ บอกว่าเป็นการเริ่มต้นติดต่อกับหน่วยความจำ ก็จะทำให้อินพุตของ Preset เป็น High ที่สภาวะนี้จึงอาศัยแค่ 2 อินพุตคือ D อินพุตและสัญญาณนาฬิกาอินพุตในการทำงานของDฟลิปฟลอปดังนั้น $\overline{\text{DTACK}}$ จะเป็น 1 เมื่อ $\overline{\text{SEL}}$ เป็น 1 แล้วและเวลาผ่านไป 250 นาโนวินาที และเมื่อ $\overline{\text{AS}}$ กลับเป็น High (เป็นบ่งบอกความสมบูรณ์ในการติดต่อกับหน่วยความจำ) ขา Preset ของ D ฟลิปฟลอป เป็น Low เป็นการ set ที่เอาต์พุตของ D ฟลิปฟลอป โดยตรงดังนั้น $\overline{\text{DTACK}}$ จึงเป็น High ด้วย

จากที่อธิบายไปข้างต้น ดังนั้นแล้วเราจึงได้วงจรหน่วงเวลาในการกำเนิดสัญญาณ $\overline{\text{DTACK}}$ ให้เกิดขึ้นในเวลาที่เหมาะสมในขณะไซเคิลของการติดต่อในหน่วยความจำ

2.10 การออกแบบอินพุตและเอาต์พุตของระบบ (I/O System Design)

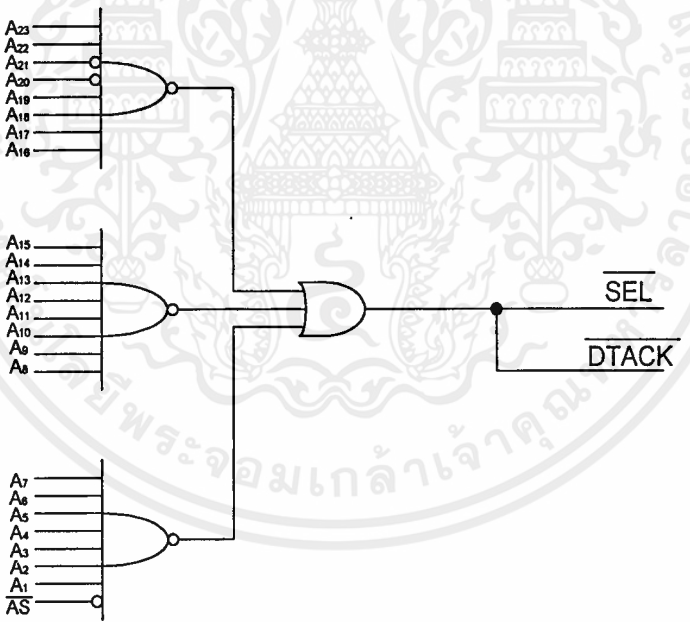
ก่อนหน้านี้เราได้เรียนรู้การออกแบบระบบหน่วยความจำที่จะนำมาใช้กับ 68000 ไป ระบบอินพุตและเอาต์พุตมีการออกแบบที่คล้ายคลึงกันอุปกรณ์ต่อพ่วงอินพุตและเอาต์พุต ต้องอินเตอร์เฟสกับ 68000 ซึ่งใช้หลักการถอดรหัสเหมือนกับระบบหน่วยความจำ ตำแหน่งในการถอดรหัสตำแหน่งอินพุตและเอาต์พุต จะถูกเก็บในรีจิสเตอร์เฉพาะงาน เมื่อ 68000 ต้องการอ่านหรือเขียนในตำแหน่งใดตำแหน่งหนึ่งในตารางอินพุตและเอาต์พุตของระบบ ส่วนเทคนิคการดำเนินการของตารางอินพุตและเอาต์พุตของระบบ เป็นการแนะนำเบื้องต้น โดยใช้อุปกรณ์ต่อพ่วง 6800

2.10.1 หลักการวางผังตำแหน่งของอุปกรณ์อินพุตและเอาต์พุต



รูปที่ 2.39 วงจรติดต่อกับหน่วยอินพุตและเอาต์พุต

68000 มีความสามารถในการดำเนินการวางผังตำแหน่งอุปกรณ์อินพุตและเอาต์พุตของข้อมูลในแบบไบต์และเวิร์ดทั้งสองแบบต้องอาศัยการถอดรหัสตำแหน่งในหน่วยความจำการใช้วงจรที่เหมาะสม ดังตัวอย่างการวางผังของตำแหน่งของเอาต์พุตโดยอาศัยการถอดรหัสตำแหน่งในหน่วยความจำจากการถอดรหัสทำให้เกิด Clock Pulse ให้กับ IC Octal Latch ซึ่งจะเป็นการ Latch ข้อมูลของเอาต์พุตที่ต้องการ ส่วนการวางผังของตำแหน่งของอินพุตนั้นจำเป็นต้องใช้ IC Tristate Octal Buffer ในการเคลื่อนย้ายข้อมูลจากภายนอกเข้าสู่ซีพียูโดยทั้งสองกรณีต้องมีการ Active ของดาต้าบัส ในรูปที่ 2.39 แสดงให้เห็นถึงวงจรวางผังของตำแหน่งอินพุตและเอาต์พุตแบบ 8 บิต ในการถอดรหัสตำแหน่งของการกระทำกับข้อมูลทั้งเป็นอินพุตและเอาต์พุต ในรูปที่ 2.39 จะเห็นว่าขาสัญญาณ $R/\overline{W}$ ถูกใช้ในการควบคุมการดำเนินการติดต่อกับพอร์ต ซึ่งการออกแบบตำแหน่งถอดรหัสแสดงให้เห็นดังในรูปที่ 2.40 ใช้หลักการ Partial–Address Decoding



รูปที่ 2.40 การถอดรหัสแอดเดรสติดต่อกับอุปกรณ์อินพุตและเอาต์พุตที่ตำแหน่ง CFFFFE

ตำแหน่งในหน่วยความจำของการถอดรหัสตำแหน่งในรูปที่ 2.40 คือ CFFFFE ในการอ่านและเขียนของการ Memory–Mapped สามารถกระทำได้ด้วยวิธีการหลายวิธี วิธีการหนึ่งในการเข้าตำแหน่งโดยตรงสามารถกระทำได้โดยใช้คำสั่ง ดังเช่น

```
MOVE.B    $CFFFFE , D0
```

หรือ

```
MOVE.B    D0 , $CFFFFE
```

ในวิธีการอื่น เราอาจใช้การอ้างตำแหน่งในหน่วยความจำแบบแอดเดรสรีจิสเตอร์อินไดเร็คแอดเดรสซึ่งโหมด ซึ่งตำแหน่งของอินพุตและเอาต์พุตต้องถูกทำการโหลดเข้ามาในตำแหน่งรีจิสเตอร์ก่อนที่จะมีการเข้าถึงอินพุตและเอาต์พุต ดังตัวอย่าง เราใช้คำสั่งวนรอบในการอ้างในหน่วยความจำแบบบแอดเดรส รีจิสเตอร์อินไดเร็คแอดเดรสซึ่งโหมด เป็นการอ่านข้อมูลเข้ามาและเขียนข้อมูลออกไป

```
        MOVEA.L  #$CFFFFF , A0
ECHO :  MOVE.B   (A0) , D0
        MOVE.B   Do , (A0)
        BRA      ECHO
```

พอร์ตอินพุตและเอาต์พุตขนาด 16 บิตนี้ ออกแบบมาโดยต้องการไอซีแลท์ขนาด 8 บิต 2 ตัวและไอซีบัฟเฟอร์ ขนาด 8 บิต 2 ตัว ตัวแรกของของไอซีแลทซ์และบัฟเฟอร์ ถูกใช้ติดต่อกับบัสข้อมูล 8 เส้น ในการถอดรหัสตำแหน่งใช้ขาสัญญาณ $\overline{UDS}$ และ $\overline{LDS}$ ในการควบคุมการทำงานของไอซีแลทซ์และบัฟเฟอร์

บทที่ 3
การสร้างและการออกแบบ

3.1 กล่าวนำ

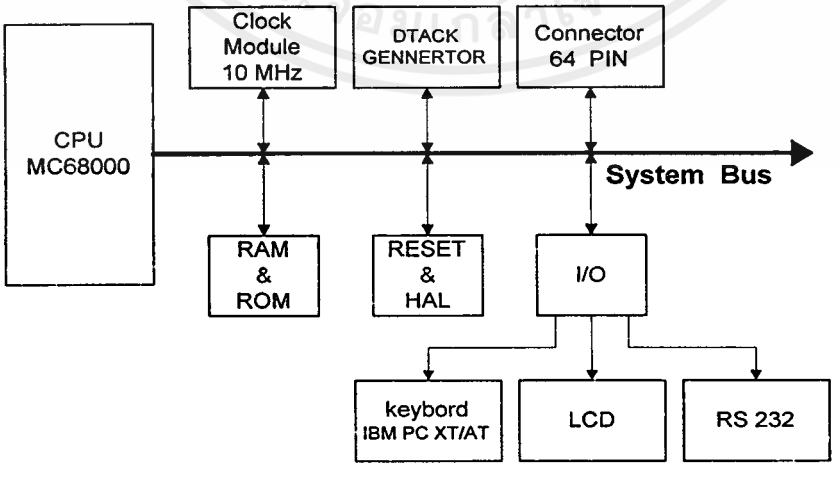
ในบทที่ 2 ได้กล่าวถึงสถาปัตยกรรมของ 68000 ไมโครคอมพิวเตอร์ รวมไปถึงเทคนิค วิธีการติดต่อกับหน่วยความจำและการติดต่อกับอุปกรณ์อินพุตและเอาต์พุต ในการสร้างและการออกแบบ 68000 ไมโครคอมพิวเตอร์แบบแผ่นพิมพ์เดี่ยวนี้ได้ ต้องนำเอาความรู้เบื้องต้นที่กล่าวมา ซึ่งการออกแบบสามารถแบ่งได้เป็น 3 ด้าน ดังต่อไปนี้

1. ขอบเขตและความสามารถของโครงงาน
2. การออกแบบฮาร์ดแวร์
3. การออกแบบซอฟแวร์มอนิเตอร์

จากการจัดแบ่งทั้ง 3 ด้านสามารถอธิบายหลักการออกแบบและวิธีการสร้างในแต่ละด้านได้ดังนี้

3.2 การออกแบบฮาร์ดแวร์

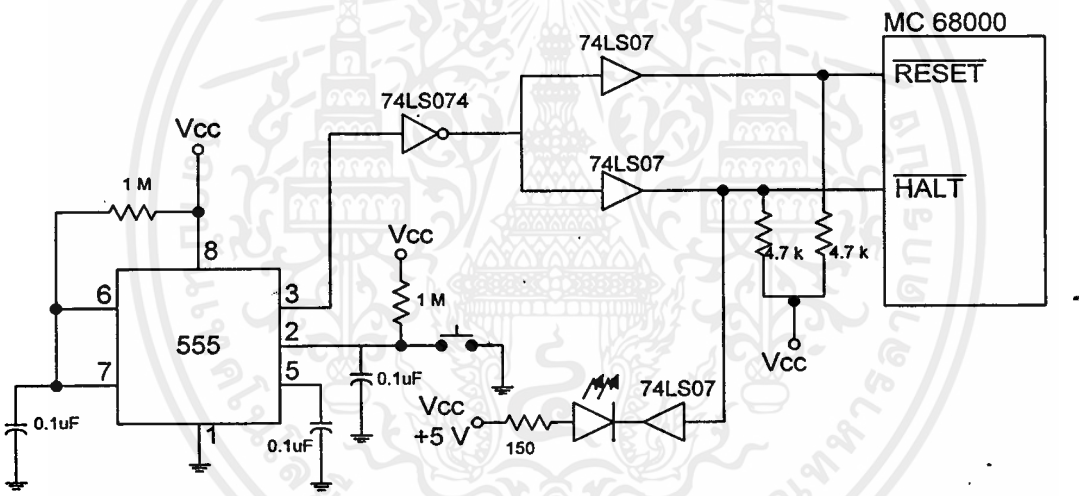
จากขอบเขตและความสามารถของโครงงานสามารถเขียนเป็นโครงสร้างเพื่อนำไปออกแบบฮาร์ดแวร์ได้ดังนี้



รูปที่ 3.1 โครงสร้างในการออกแบบฮาร์ดแวร์

3.2.1 การออกแบบวงจร $\overline{\text{RESET}}$ และ $\overline{\text{HALT}}$

เนื่องจากการเริ่มต้นการทำงานของซีพียู 68000 นั้นจะเข้าสู่ระบบการทำงานของ $\overline{\text{RESET}}$ ก่อนจากการเปิดแหล่งจ่ายไฟให้แก่ซีพียู 68000 วงจร $\overline{\text{RESET}}$ และ $\overline{\text{HALT}}$ มีความสำคัญมากในการออกแบบเป็นอันดับแรก จากการออกแบบได้ทำการออกแบบวงจร $\overline{\text{RESET}}$ และ $\overline{\text{HALT}}$ นั้น สามารถจะหน่วงเวลาให้เป็นลอจิก "0" ที่ขาสัญญาณของ $\overline{\text{RESET}}$ และ $\overline{\text{HALT}}$ นานกว่า 100 มิลลิวินาที ซีพียู 68000 จึงเข้าสู่สภาวะ $\overline{\text{RESET}}$ จากรูปที่ 3.2 แสดงวงจร $\overline{\text{RESET}}$ และ $\overline{\text{HALT}}$ ที่ใช้ไอซี 555 ในการกำเนิดสัญญาณ $\overline{\text{RESET}}$ และ $\overline{\text{HALT}}$ ขณะเปิดเครื่องและทุกครั้งที่มีการดำเนินการ $\overline{\text{RESET}}$ โดยการกดสวิตซ์ $\overline{\text{RESET}}$ และมีลอจิกเกท เบอร์ 7407 เป็นบัฟเฟอร์ของทั้งสองสัญญาณ



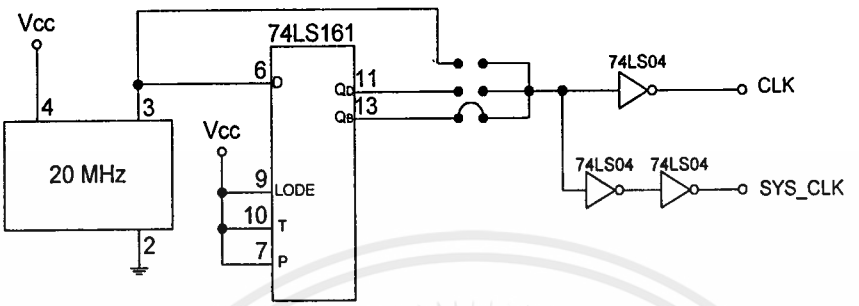
รูปที่ 3.2 วงจร Reset และ Halt

3.2.2 วงจรกำเนิดสัญญาณนาฬิกา

วงจรสัญญาณนาฬิกาเป็นวงจรที่มีความสำคัญมาก หากไม่มีวงจรสัญญาณนาฬิกานั้น ไมโครโปรเซสเซอร์ก็ไม่สามารถประมวลผลหรือระทำงานได้ วงจรสัญญาณนาฬิกาที่ต้องการจะออกแบบนั้น จะต้องมีความถี่สัญญาณนาฬิกา 10 เมกกะเฮิรตซ์ และมีช่วงเวลาของลอจิก 0 และลอจิก 1 เป็นอัตราส่วนที่เท่าๆกัน

จากข้อจำกัดข้างต้นทางกลุ่มได้ทำการออกแบบโดยใช้การกำเนิดสัญญาณนาฬิกาจากคริสตัลโมดูลความถี่ 20 เมกกะเฮิรตซ์ นำความถี่ที่ได้มาทำการหารความถี่ลง 2 เท่าและ 4 เท่า

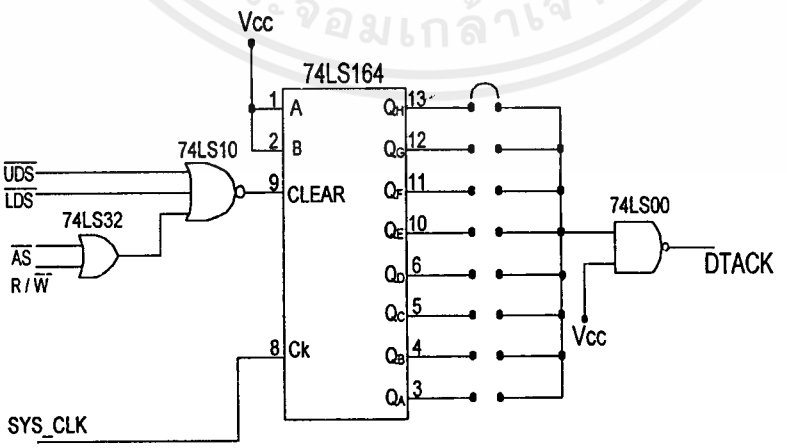
สามารถทำให้สัญญาณความถี่นาฬิกา 10 เมกกะเฮิรตซ์และ 5 เฮิรตซ์ โดยใช้วงจรหารความถี่จากวงจรนับขนาด 4 บิต ซึ่งอยู่ภายใน IC 74LS161 ซึ่งสามารถเขียนวงจรได้ดังในรูปที่ 3.3



รูปที่ 3.3 วงจรกำเนิดสัญญาณนาฬิกา

3.2.3 วงจร WAIT หรือ DTACK GENERATOR

จากบทที่ 2 นั้นเราทราบว่าหากไมโครโปรเซสเซอร์สามารถประมวลผลเสร็จภายในสัญญาณนาฬิกาสเตทที่ 7 หมายถึงไมโครโปรเซสเซอร์ไม่ต้องทำงานในสภาวะ wait หรืออาจกล่าวได้ว่าอุปกรณ์ที่เชื่อมต่อกับไมโครโปรเซสเซอร์มีความเร็วในการประมวลผลใน การเข้าถึงข้อมูลหรือรับส่งภายในสัญญาณนาฬิกาสภาวะปกติ แต่ถ้าหากความเร็วในการเข้าถึงข้อมูลของอุปกรณ์เชื่อมต่ออินพุตหรือเอาต์พุตไม่สามารถกระทำได้เสร็จทันภายในสเตทที่ 7 จึงต้องมีความจำเป็นในการสร้างวงจร DTACK GENERATOR หรือวงจรกำเนิดสัญญาณ wait

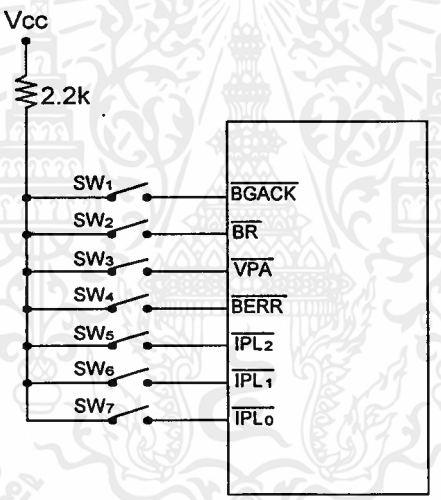


รูปที่ 3.4 วงจรกำเนิดสัญญาณ DTACK GENERATOR

จากที่กล่าวมาข้างต้นจึงได้จัดสร้างวงจรกำเนิดสัญญาณหน่วงเวลา โดยใช้ขาสัญญาณกลุ่มของสัญญาณ Asynchronous bus control ซึ่งสัญญาณ $\overline{AS}$, R / $\overline{W}$, $\overline{UDS}$, $\overline{LDS}$, $\overline{DTACK}$ ออกแบบร่วมกัน เนื่องจากกลุ่มสัญญาณมีความสัมพันธ์ในการรับส่งข้อมูลตรง ซึ่งสามารถออกแบบได้ดังรูปที่ 3.4

3.2.4 การต่อสัญญาณควบคุมอื่นๆ

เนื่องจากการออกแบบในหัวข้อที่ผ่านมาในหัวข้อที่ 3.3.1 ถึง 3.3.4 ยังไม่สามารถดำเนินการให้ซีพียู 68000 ทำงานได้ในสภาวะเริ่มต้น เนื่องจากสัญญาณอินพุตที่เหลือไว้คือ $\overline{BGACK}$, $\overline{BR}$, $\overline{VPA}$, $\overline{BERR}$, $\overline{IPL_0}$, $\overline{IPL_1}$, $\overline{IPL_2}$ ขาสัญญาณทั้งหมดที่กล่าวมา สามารถต่อลอจิก "1" ไว้โดยใช้ดิฟสวิตซ์ดังรูปที่ 3.5



รูปที่ 3.5 การต่อสัญญาณควบคุมอื่นๆของ 68000

จากรูปที่3.5 สามารถอธิบายสภาวะของ SW1-SW7 ที่มีผลต่อการทำงานของซีพียู 68000 ดังนี้

SW1,SW2 สภาวะ on จะทำให้ 68000 ไม่สามารถตอบรับการขอใช้บัสได้ หากต้องการให้ 68000 ตอบรับการขอใช้บัสเปิดวงจร SW1,SW2

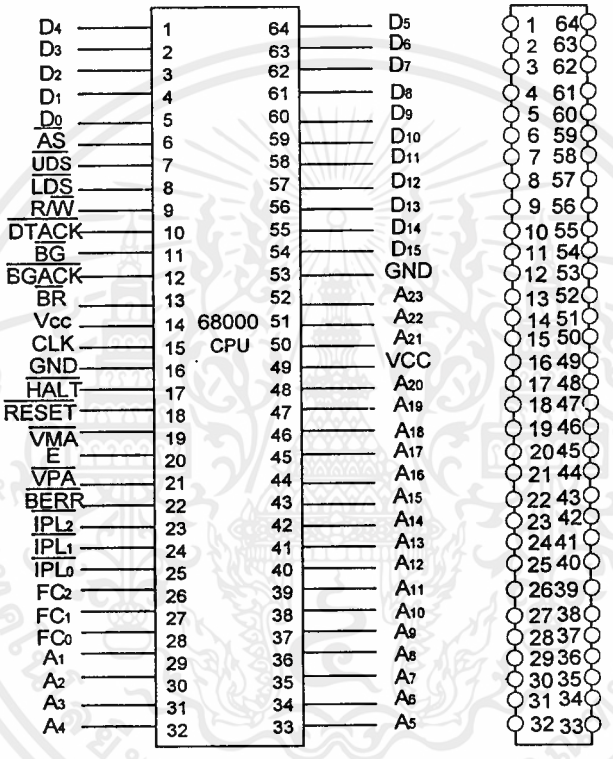
SW3 สภาวะ on จะทำให้ 68000 ไม่สามารถติดต่อกับอุปกรณ์เชื่อมต่อประเภทที่ขึ้นอยู่กับสัญญาณนาฬิกา เช่น RS232

SW4 สภาวะ ON จะทำให้ 68000 ไม่สามารถรับรู้การผิดพลาดของบัส

SW5,SW6,SW7 สภาวะ ON จะทำให้ 68000 ไม่ตอบรับการขัดจังหวะจากอุปกรณ์ภายนอก แม้ว่าจะมี การขอขัดจังหวะจากอุปกรณ์ภายนอก

หากต้องการออกแบบในส่วนใดส่วนหนึ่งที่ต้องใช้สัญญาณในกลุ่มสัญญาณกล่าวมาให้เปิดสวิตซ์และต่อสัญญาณทั้งหมดนี้ผ่านคอนเน็คเตอร์ซึ่งจะกล่าวในหัวข้อต่อไป

3.3.5 การเชื่อมต่อคอนเน็คเตอร์กับซีพียู 68000



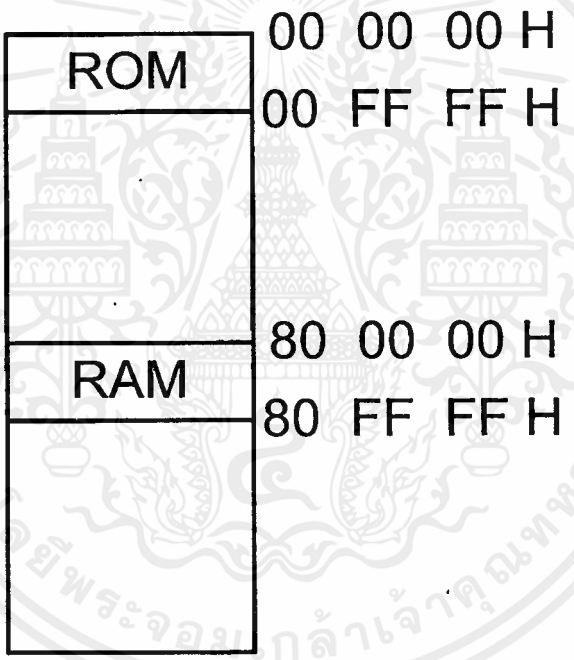
รูปที่ 3.6 การเชื่อมต่อ 68000 กับคอนเน็คเตอร์

เนื่องจาก 68000 จะมีขาสัญญาณทั้งหมด 64 ขา เพราะฉะนั้นจะต้องทำการออกแบบคอนเน็คเตอร์ให้มีจำนวนขาเท่ากับขาสัญญาณของ 68000 โดยมีรายละเอียดดังรูปที่ 3.6

3.2.6 การติดต่อกับหน่วยความจำ

จากหัวข้อที่ผ่านมาเราสามารถออกวงจรที่จำเป็นพื้นฐานไปแล้วสิ่งหนึ่งที่เราไม่สามารถมองข้ามไปได้คือ การเชื่อมต่อกับหน่วยความจำซึ่งการเชื่อมต่อกับหน่วยความจำในโครงงานนี้จะใช้หน่วยความจำ ROM และ RAM

ในขั้นแรกนั้นเราต้องกำหนดตำแหน่งของ ROM และ RAM ภายใต้ความสามารถของการอ้างตำแหน่งของ 68000 ที่สามารถอ้างตำแหน่งในหน่วยความจำได้ไม่เกิน 16 เมกกะไบต์ เนื่องจาก 68000 นั้นจะประมวลผลเริ่มแรกคือในสภาวะ RESET และจะนำค่าตำแหน่งชี้ของสแต็คและโปรแกรมนับจาก 8 ไบต์ แรกจากหน่วยความจำโดยอยู่ในตำแหน่ง 0000-0003 เป็นค่าของหน่วยความจำสแต็กและ 0004-0007 เป็นค่าตำแหน่ง หน่วยความจำของโปรแกรมนับที่ต้องการให้ประมวลผลเริ่มต้นเป็นคำสั่งแรก เพราะฉะนั้นการออกแบบหน่วยความจำที่ตำแหน่ง 0000 เป็นต้นไปถึง 0008 เพื่อเก็บตำแหน่งของสแต็คและโปรแกรมเคาท์เตอร์จึงสามารถออกแบบหน่วยความจำทั้ง ROM และ RAM ได้ดังรูปที่ 3.7



รูปที่ 3.7 การออกแบบหน่วยความจำ

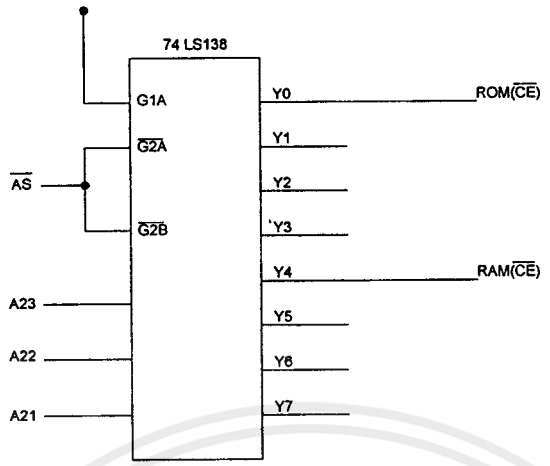
จากรูปที่ 3.7 จากตำแหน่งของ RAM และ ROM เราสามารถนำแอดเดรสในไบต์ที่ 3 คือบิตที่ 20-23 จะนำมาทำการถอดรหัส โดยใช้ไอซีถอดรหัส 74LS138 มาทำการแบ่งหน่วยความจำได้ทั้งหมด 8 ส่วนโดยแต่ละส่วนจะมีขนาดของหน่วยความจำ 2 เมกกะไบต์ดังตารางที่ 3.1

ตารางที่ 3.1 การ Memory Map หน่วยความจำ

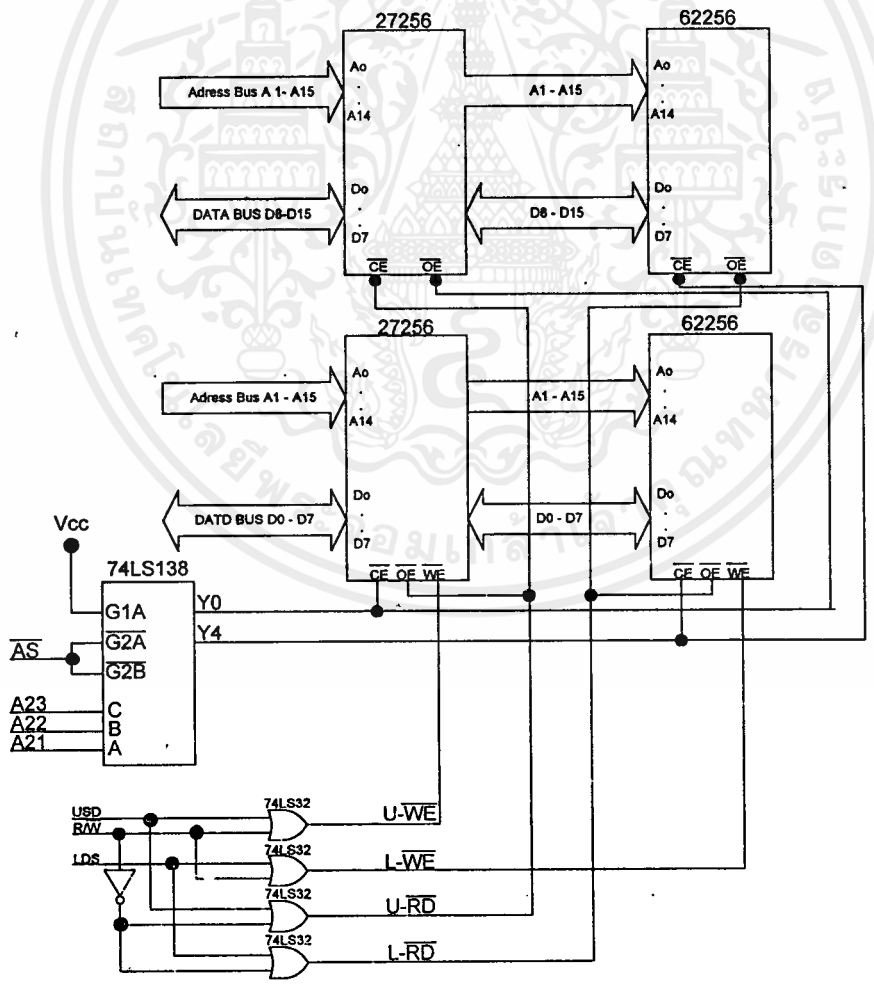
A_{23}	A_{22}	A_{21}	A_{20}	ตำแหน่ง	ชนิด
0	0	0	0	00 00 00 H	ROM
0	0	0	1	1F FF FF H	
0	0	1	0	20 00 00 H	
0	0	1	1	3F FF FF H	
0	1	0	0	40 00 00 H	
0	1	0	1	5F FF FF H	
0	1	1	0	60 00 00 H	
0	1	1	1	7F FF FF H	
1	0	0	0	80 00 00 H	RAM
1	0	0	1	9F FF FF H	
1	0	1	0	A0 00 00 H	
1	0	1	1	9F FF FF H	
1	1	0	0	C0 00 00 H	
1	1	0	1	DF FF FF H	
1	1	1	0	E0 00 00 H	
1	1	1	1	FF FF FF H	

จากตารางที่ 3.1 สามารถเขียนวงจรด้วย 74LS138 ดังวงจรในรูปที่ 3.8

จากรูปที่ 3.8 ขา $\overline{AS}$ ที่เชื่อมต่อกับ 74LS138 นั้นเมื่อขาสัญญาณ $\overline{AS}$ เป็นศูนย์จะเป็นการบ่งชี้ว่าค่าของตำแหน่งแอดเดรสที่ต้องการติดต่ออยู่ในแอดเดรสบัสเรียบร้อยแล้วโดยต่อ $\overline{AS}$ เข้ากับขาสัญญาณ $\overline{G2A}$ และ $\overline{G2B}$ เนื่องจากสภาวะลอจิก "0" ของ $\overline{AS}$ นั้นบอกให้ทราบว่า 68000 ต้องการติดต่อหน่วยความจำขนาดและมีกลุ่มของขาสัญญาณที่ใช้ร่วมกันในการติดต่อหน่วยความจำคือ $\overline{AS}, \overline{LDS}, \overline{UDS}, R/\overline{W}$ ซึ่งได้กล่าวถึงรายละเอียดในการทำงานดังนั้นแล้วจึงนำขาสัญญาณเหล่านี้มาออกแบบวงจรไดดังรูปที่ 3.9



รูปที่ 3.8 วงจรถอดรหัสตำแหน่งในหน่วยความจำโดยใช้ IC 74LS138



รูปที่ 3.9 การถอดรหัสตำแหน่งในหน่วยความจำ

จากรูปที่ 3.9 สามารถอธิบายหน้าที่ขาสัญญาณ U_$\overline{WE}$, L_$\overline{WE}$, U_$\overline{RD}$, L_$\overline{RD}$ ได้ดังนี้

U_$\overline{WE}$ ทำหน้าที่ กำหนดการเขียนหน่วยความจำในไบต์สูง

L_$\overline{WE}$ ทำหน้าที่ กำหนดการเขียนหน่วยความจำในไบต์ต่ำ

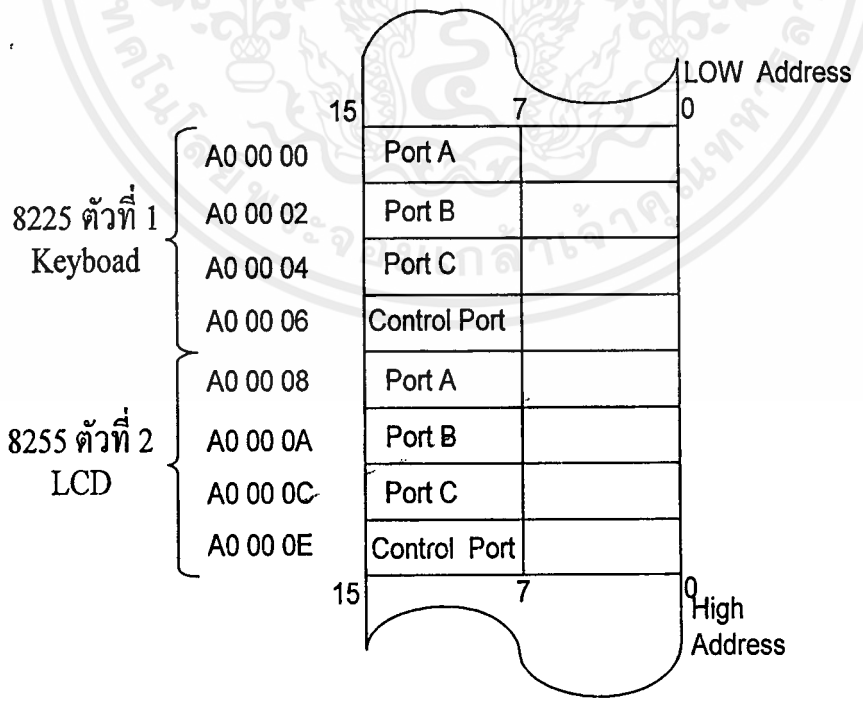
U_$\overline{RD}$ ทำหน้าที่ กำหนดการอ่านหน่วยความจำในไบต์สูง

L_$\overline{RD}$ ทำหน้าที่ กำหนดการอ่านหน่วยความจำในไบต์ต่ำ

3.2.7 การติดต่ออุปกรณ์อินพุตและเอาต์พุต

จากหัวข้อในการออกแบบนั้นได้กล่าวถึงอุปกรณ์ในการเชื่อมต่ออินพุตและเอาต์พุต โดยมีหน่วยรับข้อมูลเข้าเป็นคีย์บอร์ดชนิด IBM PC XT/AT และหน่วยแสดงผลแบบ LCD

จากข้อจำกัดข้างต้นทางกลุ่มได้ออกแบบอุปกรณ์เชื่อมต่อโดยใช้ IC 8255 ในการเป็นอุปกรณ์ตัวกลางในการเชื่อมต่อ โดยได้กำหนดตำแหน่งของ IC8255 เป็น 2 หน่วยโดย IC8255 ตัวแรกทำการควบคุมการเชื่อมต่อกับคีย์บอร์ดและ IC8255 ตัวที่สองทำหน้าที่เชื่อมต่อกับ LCD โดยทั้งสองพอร์ตจะกระทำผ่าน CONNECTOR ขนาด 40 PIN โดยสามารถกำหนดตำแหน่งของพอร์ต 8255 ทั้งสองได้ดังแสดงในรูปที่ 3.10

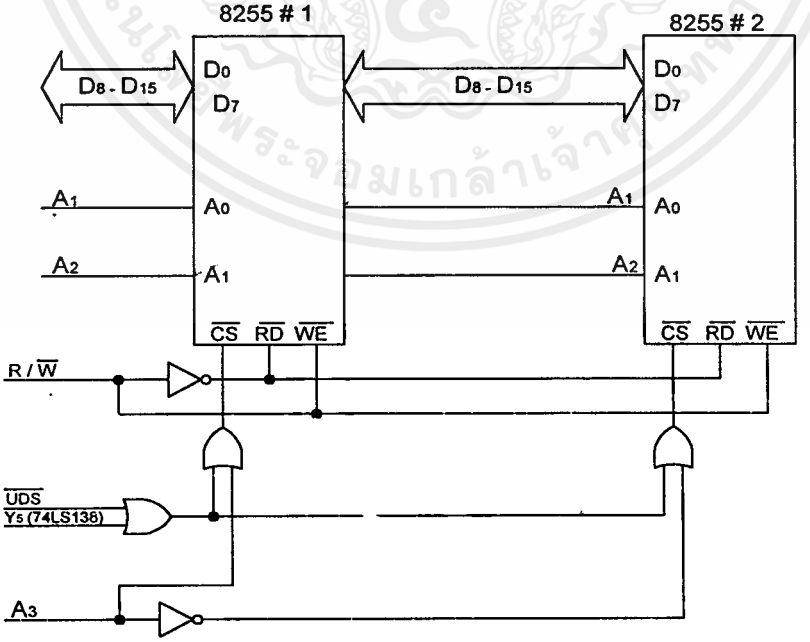


รูปที่ 3.10 การกำหนดตำแหน่งของ 8255 จำนวน 2 ตัว

จากรูปที่ 3.10 เราสามารถนำข้อมูลในแอดเดรสบัสในไบท์ต่ำสุดมาทำการถอดรหัสร่วมกับสัญญาณเอาต์พุตของ IC 74LS138 ในการแบ่งหน่วยความจำออกเป็น 8 ส่วนจากหน่วยความจำทั้งหมดในที่นี้นำสัญญาณ Y5 จาก IC 74LS138 มาถอดรหัสร่วมกับสัญญาณ A1-A3 ได้ดังตารางที่ 3.2

ตารางที่ 3.2 การถอดรหัสตำแหน่งของอุปกรณ์จาก A_1-A_3

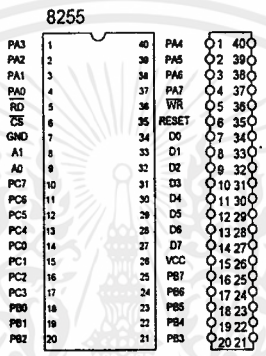
A_3	A_2	A_1	ตำแหน่ง	พอร์ต	อุปกรณ์
0	0	0	00 00 00 H	Port A	Keyboard
0	0	1	00 00 02 H	Port B	IBM PC
0	1	0	00 00 04 H	Port C	XTหรือAT
0	1	1	00 00 06 H	Control Port	
1	0	0	00 00 08 H	Port A	
1	0	1	00 00 0A H	Port B	LCD
1	1	0	00 00 0C H	Port C	
1	1	1	00 00 0F H	Control Port	



รูปที่ 3.11 การเชื่อมต่อ 8255 จำนวน 2 ตัว

จากตารางที่ 3.2 เราสามารถออกแบบวงจรโดยสังเกตได้ว่าตำแหน่งของ 8255 ตัวที่ 1 และตัวที่ 2 สามารถแยกออกจากกันโดยอาศัยบิตของ A3 และการเข้าถึงข้อมูลจะเป็นการเข้าถึงข้อมูลในไบต์สูงเพราะฉะนั้นเราจำเป็นต้องนำขาสัญญาณ $\overline{ULS}$ มาออกแบบร่วมด้วย สามารถเขียนวงจรได้ดังรูปที่ 3.11

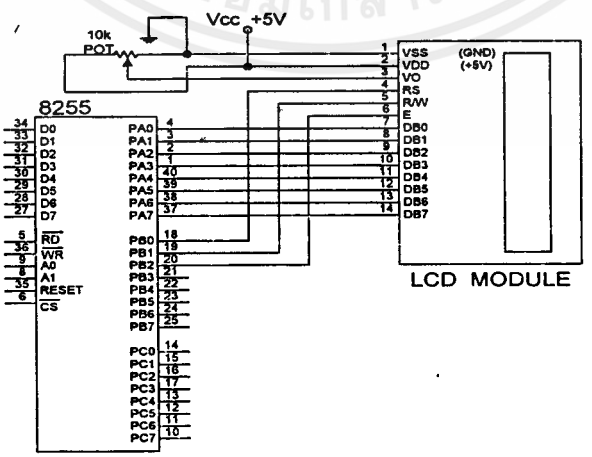
จากรูปที่ 3.11 นั้นสามารถแสดงส่วนขยายการเชื่อมต่อกับคอนเน็คเตอร์ได้ ดังรูปที่ 3.12



รูปที่ 3.12 การเชื่อมต่อ 8255 กับ Connector

3.2.8 การเชื่อมต่อพอร์ต 8255#1 กับภาคแสดงผลแบบผลึกเหลว

ในโครงงานนี้เลือกใช้พอร์ตในการเชื่อมต่อระหว่าง 68000 กับภาคแสดงผลแบบผลึกเหลวโดยใช้ IC เบอร์ 8255 ดังแสดงในรูปที่ 3.13

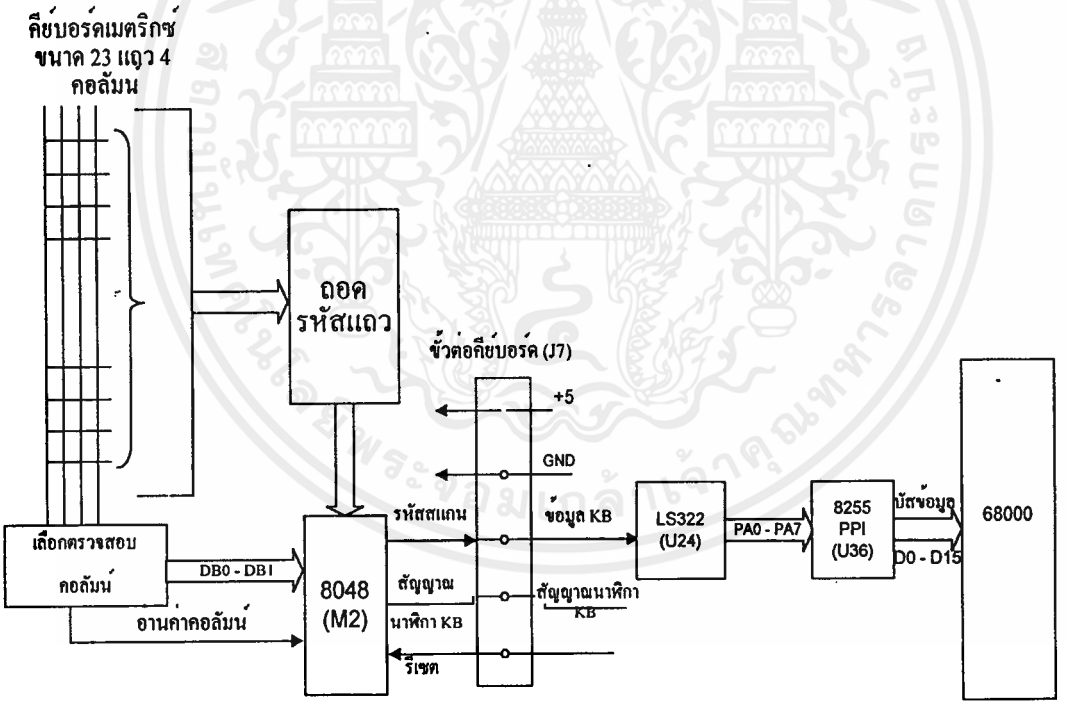


รูปที่ 3.13 การเชื่อมต่อพอร์ต 8255 กับ LCD

จากวงจรเป็นการต่อพอร์ต 8255 ให้เข้าใช้กับภาคแสดงผลแบบผลึกเหลวโดยเราจะจำลองสัญญาณต่างๆขึ้นมาโดยพอร์ต A นั้นเราให้เป็นคาต้าพอร์ตและพอร์ต B นั้นเราให้เป็นสัญญาณควบคุมไปใช้ เมื่อเราเริ่มเปิดไฟป้อนให้ HD44780 นั้นก็จะทำการรีเซ็ตตัวเองจะใช้เวลาประมาณ 10 มิลลิวินาที หลังจากไฟ VDD ถึง 4.5 โวลท์แล้วจะ เซ็ตตัวเอง

3.2.9 การเชื่อมต่อพอร์ต 8255#2 กับ คีย์บอร์ด

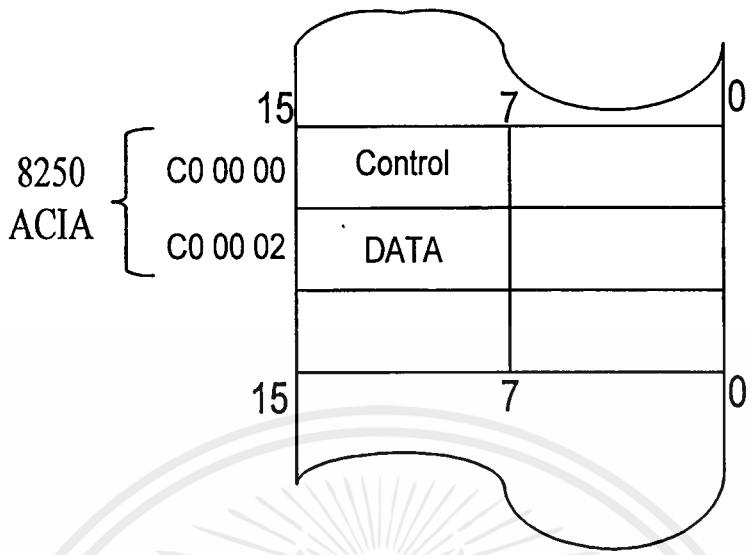
ในการเชื่อมต่อกับคีย์บอร์ดนั้น โครงสร้างทางฮาร์ดแวร์มีการทำงานในลักษณะที่ซับซ้อนพอควร กล่าวคือโครงสร้างของอินพุตจากคีย์บอร์ดนี้ ทำงานร่วมกับอินเตอร์รัพต์ โดยโครงสร้างของคีย์บอร์ดที่ต่อกับเมนบอร์ดจะใช้สายเชื่อมโยง 5 เส้นจากนั้นจึงส่งข้อมูลไปที่พอร์ต 8255 ซึ่งพอร์ต 8255 นี้จะทำการติดต่อกับซีพียูโดยมีรายละเอียดของผังงานโครงสร้างการเชื่อมโยงดังรูปที่ 3.14



รูปที่ 3.14 โครงสร้างการเชื่อมโยงคีย์บอร์ดกับ 68000

3.2.10 การเชื่อมต่อซีพียู 68000 กับ RS232

เนื่องจากการออกแบบวงจรที่ต้องการติดต่อกับ RS232 หรือพอร์ตอนุกรมนั้นได้ทำการออกแบบโดยกำหนดตำแหน่งของพอร์ตอนุกรมนี้ไว้ดังรูปที่ 3.15



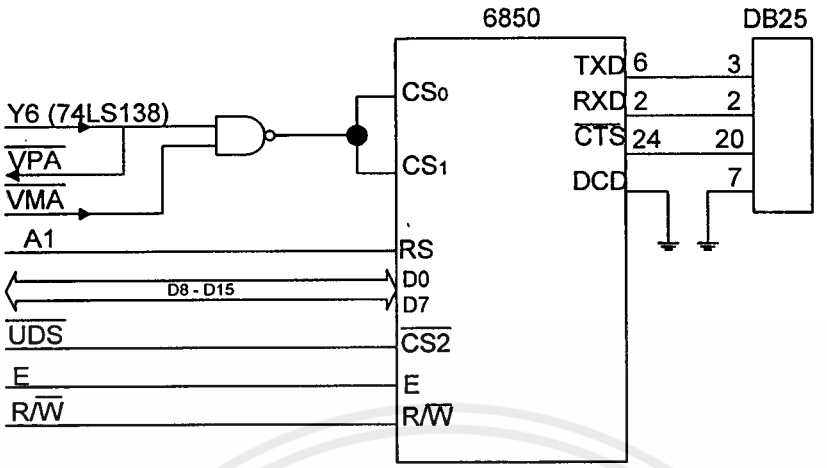
รูปที่ 3.15 ตำแหน่งของ 8250 ACIA

จากรูปที่ 3.15 เลือกใช้ IC 8250 ACIA เป็นตัวติดต่อในมาตรฐานของพอร์ตอนุกรม โดยเลือกตำแหน่งของหน่วยความจำที่ C0 00 00 เป็น Control Port และ C0 00 02 เป็น Data Port นอกจากการกำหนดตำแหน่งข้างต้นแล้วในขั้นต่อไปจะต้องทำการถอดรหัสตำแหน่งของหน่วยความจำที่กำหนดไว้ โดยสามารถถอดรหัสตำแหน่งได้ดังตารางที่ 3.3

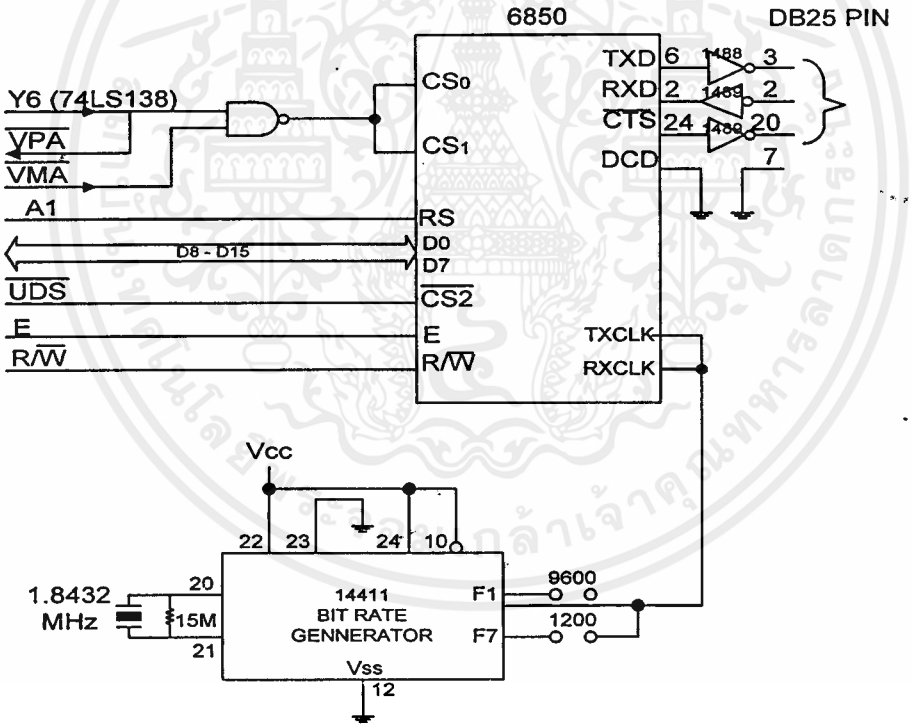
ตารางที่ 3.3 การถอดรหัสตำแหน่งของ 8250

A3	A2	A1	ตำแหน่ง	พอร์ต
0	0	0	00 00 00	คอนโทรล พอร์ต 8250
0	0	1	00 00 02	ดาต้าพอร์ต 8250

จากตารางที่ 3.3 นั้นการถอดรหัสแอดเดรสนั้นจะต้องกระทำร่วมกับขาสัญญาณอื่นด้วยคือ $\overline{UDS}$ และ R/$\overline{W}$ นอกจากนั้นการเชื่อมต่อพอร์ตอนุกรมนั้นต้องติดต่อกับสัญญาณ $\overline{VPA}$ และ $\overline{VMA}$ ด้วยโดยสามารถเขียนวงจรได้ดังรูปที่ 3.16 โดยได้นำเอาเอาต์พุต Y6 ของ 74LS138 มาถอดรหัสร่วมด้วย



รูปที่ 3.16 การถอดรหัสการเชื่อมต่อ 68000 กับ 6850



รูปที่ 3.17 การเชื่อมต่อซีพียู 68000 กับ 6850

จากรูปที่ 3.17 การเชื่อมต่อ 6850 กับสายนำสัญญาณนั้นจะเชื่อมต่อกับ DB 25 PIN นอกจากวงจรถอดรหัสแล้วและการต่อสัญญาณควบคุมให้กับ 6850 แล้วนั้น 6850 ยังต้องการสัญญาณนาฬิกาในการทำงานด้วยซึ่งได้ออกแบบการเชื่อมต่อสัญญาณนาฬิกาให้กับ 6850

โดยใช้ IC 14411 BIT RATE GENERATOR ซึ่งสามารถเลือกความถี่ในการติดต่อคือ 9600 และ 1200 ดังรูปที่ 3.17

นอกจากเราสามารถเชื่อมต่ออุปกรณ์ทางฮาร์ดแวร์ของ 6850 แล้ว 6850 เป็น IC ที่ต้องมีการเขียนโปรแกรมควบคุมการทำงานด้วย

3.3 การออกแบบโปรแกรมมอนิเตอร์

3.3.1 ทฤษฎีและหลักการออกแบบโปรแกรมควบคุมการทำงานของซิงเกิ้ลบอร์ด

จากหัวข้อที่ผ่านมาเราทราบแล้วว่าขั้นตอนในการออกแบบฮาร์ดแวร์นั้นทำให้เราสามารถทราบวิธีการเชื่อมต่อ อุปกรณ์ร่วมไปถึงตำแหน่งหน่วยความจำ การเขียนโปรแกรมควบคุมนับได้ว่ามีความสำคัญมาก เพราะระบบของไมโครคอมพิวเตอร์จะสามารถทำงานได้นั้นจะต้องประกอบไปด้วยส่วนทางด้านฮาร์ดแวร์และซอฟร์แวร์ ซึ่งทางกลุ่มผู้จัดทำได้ทำการออกแบบ โปรแกรมควบคุม โดยได้แบ่งเป็นด้านต่างๆ ดังนี้ คือ

1. ขีดความสามารถของโปรแกรมควบคุม
2. การออกแบบหน่วยความจำในการเขียนโปรแกรมมอนิเตอร์
3. การกำหนดความสามารถของโปรแกรมควบคุมกับอุปกรณ์เชื่อมต่อ

3.3.2 ขีดความสามารถของโปรแกรมควบคุม

1.โปรแกรมควบคุมสามารถรับข้อมูลทางคีย์บอร์ด และแสดงผลทางที่ภาคแสดงผลแบบผลึกเหลว หรือ LED บนซิงเกิ้ลบอร์ดได้

2.สามารถนำข้อมูลลงสู่หน่วยความจำได้

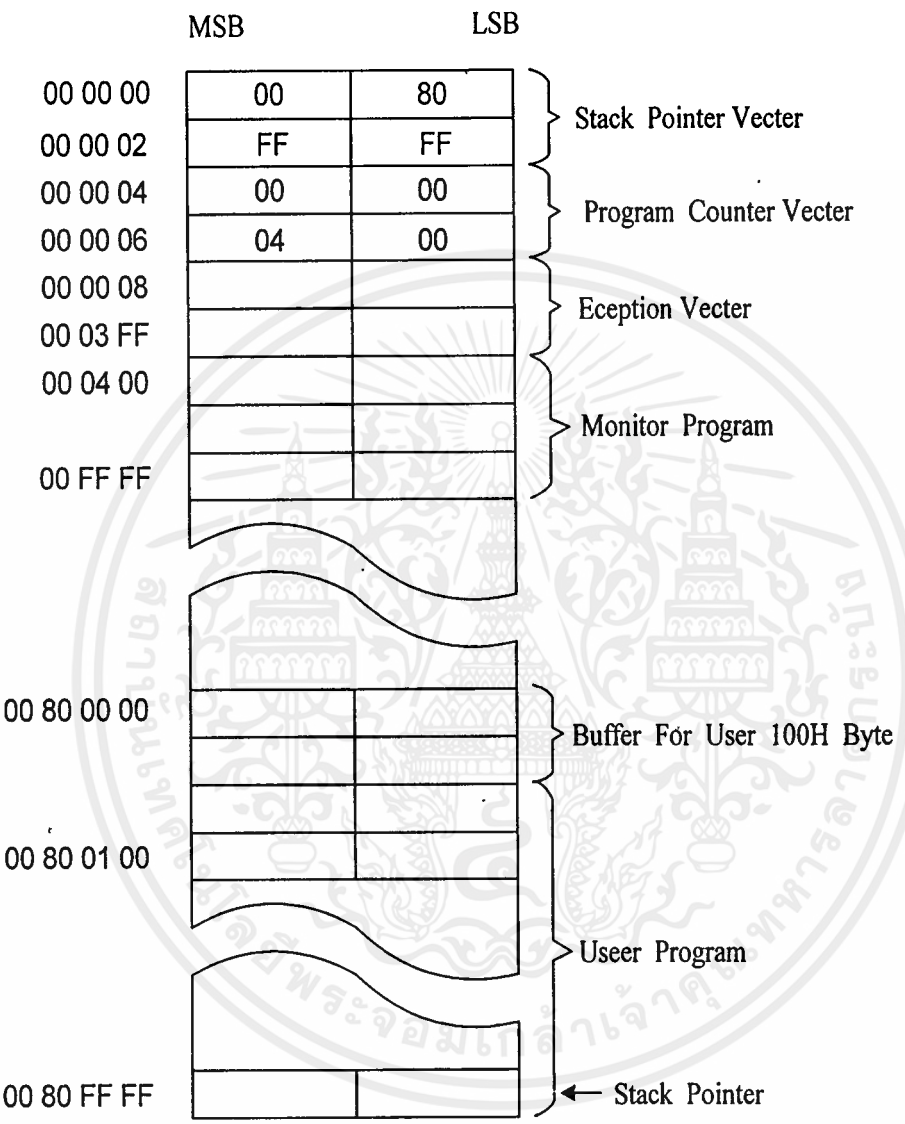
3.สามารถประมวลผลจากการโปรแกรมของผู้ใช้ได้

3.3.3 การออกแบบหน่วยความจำโดยการเขียนโปรแกรมมอนิเตอร์

จากบทที่ 3 หน่วยความจำที่เราสามารถอ้างถึงได้นั้นแบ่งได้เป็น หน่วยความจำชนิด ROM และหน่วยความจำชนิด RAM จะมีโปรแกรมควบคุมการทำงานจะอยู่ในหน่วยความจำชนิด ROM และการเขียนโปรแกรมของผู้ใช้นั้นจะถูกเขียนลงสู่หน่วยความจำชนิด RAM

เนื่องจากการออกแบบหน่วยความจำของโปรแกรมควบคุมต้องอยู่ใน ROM เพราะว่าในสภาวะเริ่มต้นการทำงานนั้น 68000 จะอ่านตำแหน่งหรือเวคเตอร์ จำนวน 2 เวคเตอร์ คือใน 68000 แนะนำข้อมูลใน 4 ไบต์แรกของหน่วยความจำเพื่อถึงตำแหน่งของสแตค

พอยต์เตอร์ และ 4 ไบต์ต่อมาก็จะเป็นตำแหน่งของการเริ่มต้นการทำงาน โดยสามารถแสดงได้ดังรูปที่ 3.18



รูปที่ 3.18 การออกแบบหน่วยความจำสำหรับโปรแกรมควบคุม

จากรูปที่3.18 สามารถอธิบายได้ดังนี้ คือ

1. ตำแหน่ง 00 00 00 - 00 00 03 เก็บค่าตำแหน่งของสแตคจากรูปสแตคพอยต์เตอร์ จะมีค่าคือ USP = 00 80 FF FF โหมดการทำงานของ User Mode และ SSP = 00 80 FF FF โหมด Supervisor Mode ซึ่ง USP และ SSP คือ รีจิสเตอร์ A7 นั่นเอง

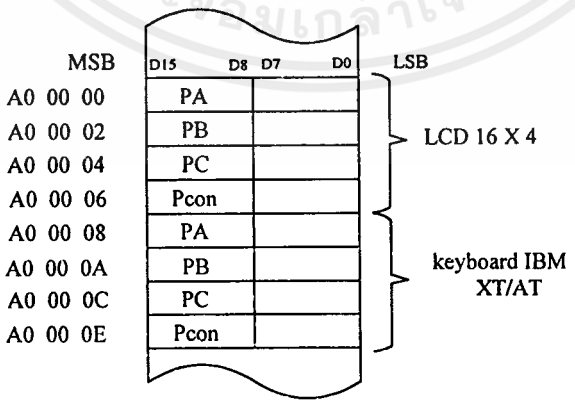
2. ตำแหน่ง 00 00 04 - 00 00 07 จะเก็บตำแหน่งของโปรแกรมเอาต์พุต จากรูปที่ 3.18 โปรแกรมเคาน์เตอร์จะเริ่มทำงานที่ตำแหน่ง 04 00 H

3. ตำแหน่ง 00 00 08 - 00 03 FF ใช้เป็นค่าของเวคเตอร์ในการประมวลผลแบบ Eception ซึ่งได้กล่าวมาแล้วว่าการ Eception จะเกิดขึ้นได้อย่างไรในบทที่ 3 รวมไปถึงตำแหน่งของเวคเตอร์ในการทำงานแบบขัดจังหวะด้วย

4. ตำแหน่ง 00 04 00 - 00 FF FF เป็นหน่วยความจำของ ROM นำโปรแกรมควบคุมลงไปในช่วงนี้ ซึ่งอยู่ในช่วงของ ROM 32 กิโลเวิร์ด

5. ตำแหน่ง 00 80 00 00 - 00 80 01 00 เป็นตำแหน่งของที่พักของข้อมูลในการเขียนโปรแกรมควบคุม และโปรแกรมของผู้ใช้ ซึ่งมีค่า 100H ไบต์

6. ตำแหน่ง 00 80 01 00 - 00 80 FF FF เป็นตำแหน่งของข้อมูลที่ผู้ใช้เขียนลงบนหน่วยความจำ ซึ่งสามารถที่จะเขียนข้อมูลลงสู่หน่วยความจำ ซึ่งสามารถเขียนข้อมูลได้ถึง 00 80 FE FFH ไบต์ซึ่งจะรวมถึงชั้นที่ของสแตคด้วยโดย สแตคของ 68000 จะเป็นสแตคที่มีพอยต์เตอร์แบบลบ เมื่อนำค่าลงสแตค และเพิ่มขึ้นเมื่อนำข้อมูลออกจากสแตค นอกจากจะออกแบบหน่วยความจำในส่วนของ ROM และ RAM แล้วนั้น การออกแบบตำแหน่งของหน่วยความจำของอุปกรณ์เชื่อมต่อก็มีความสำคัญ เช่นกัน โดยตำแหน่งของอุปกรณ์เชื่อมต่อทั้งคีย์บอร์ดและ LCD นั้นจะถูกกำหนดโดยการออกแบบมาก่อนแล้ว ทางฮาร์ดแวร์นั้นจะมีตำแหน่งตรงกับหน่วยความจำ ดังในรูปที่ 3.19



รูปที่ 3.19 ตำแหน่งของภาคแสดงผลแบบผลึกเหลว และคีย์บอร์ดในหน่วยความจำ

7. ตำแหน่ง A0 00 00 - A0 00 06 จะถูกถอดรหัสตำแหน่งข้อมูลไปยัง 8255 ที่ควบคุม LCD โดยเรียงตามพอร์ตดังรูปที่ 3.19

8. ตำแหน่ง A0 00 08 - A0 00 0E จะถูกถอดรหัสตำแหน่งข้อมูลไปยัง 8255 ที่ควบคุมคีย์บอร์ด โดยเรียงตามพอร์ตดังรูปที่ 3.19

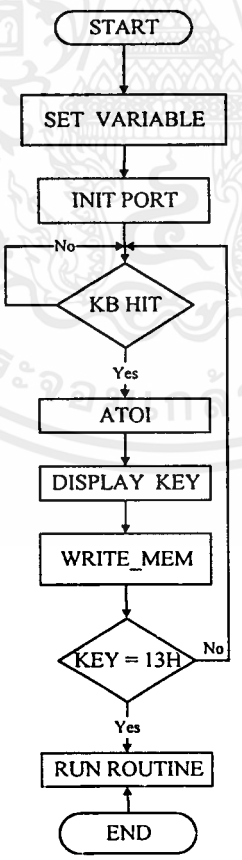
3.3.4 การกำหนดความสามารถของโปรแกรมควบคุมกับอุปกรณ์เชื่อมต่อ

จากการเชื่อมต่อ 68000 กับอุปกรณ์อินพุตและเอาต์พุตสามารถกำหนดความสามารถควบคุมได้ดังนี้

1. คีย์บอร์ดสามารถส่งรหัสตำแหน่งของคีย์บอร์ดได้ทั้ง 83 คีย์

2. LCD สามารถแสดงผลการประมวลผลที่ผู้ใช้ต้องการได้ในการเขียน LCD ตำแหน่งใดๆบน LCD

จากข้อที่ 3.3.4 ที่กล่าวมาแล้ว สามารถเขียนเป็นผังการทำงานของโปรแกรมมอนิเตอร์ได้ ดังรูปที่ 3.20



รูปที่ 3.20 แผนผังการทำงานของมอนิเตอร์โปรแกรม

จากรูปที่ 3.20 สามารถอธิบายการทำงานในแต่ละลำดับขั้นตอนได้ดังนี้ คือ

START : จะเป็นการเริ่มต้นของโปรแกรมในส่วนนี้จะเป็นส่วนของสภาวะ RESET และ HALT ของ 68000 เป็นหลัก

SET VARIABLE : เป็นการกำหนดตัวแปรที่จะใช้ในโปรแกรม

INIT PORT : เป็นการกำหนดสภาวะเริ่มต้นการทำงานของอุปกรณ์อินพุตและเอาท์พุต คือ คีย์บอร์ด และ LCD

KB HIT : เป็นการตรวจสอบการกดคีย์บอร์ดว่าคีย์ใดๆถูกกดหรือไม่หากถูกกดจะทำงานในขั้นต่อไป หากไม่ถูกกดจะรอรับการกดจนคีย์ถูกกด

ATOI : เป็นการแปลงคีย์ที่ถูกกดจากรหัสสแกน คีย์บอร์ดเป็นรหัสตัวอักษร ซึ่งตรงกับตำแหน่งของหน่วยความจำตัวอักษรใน LCD และ ทำการแปลงเป็นค่าตัวเลขตามคีย์ที่ถูกกด

DISPLAY KEY : เป็นการนำค่าจากการกดคีย์มาแสดงผลออกทางจอภาพ

WRITE_MEM : นำค่าของตัวอักษรที่ทำการกดเก็บลงในหน่วยความจำของ RAM โดยเริ่มต้นที่ 80 01 00 H

KEY = 13H : เป็นการตรวจสอบการสิ้นสุดในการรับข้อมูล หากกดคีย์ Enter จะสิ้นสุด การรับข้อมูล ถ้าไม่ใช่ ให้กลับไปรับข้อมูลใหม่ใน KBHIT

RUN_ROUTINE : เป็นการประมวลผลตามข้อมูลของผู้ใช้ซึ่งเป็นรหัสคำสั่งที่ถูกเก็บไว้ในหน่วยความจำ 80 01 00 H เป็นต้นไป

END : หากการประมวลผลของผู้ใช้สิ้นสุด จะจบการทำงานของมอนิเตอร์โปรแกรม

3.3.5 ตัวแปรที่ใช้ในโปรแกรมมอนิเตอร์

DATA_LCD = A0 00 00 H ตำแหน่งพอร์ต A ใช้ในการส่งข้อมูลไปยัง LCD

SIGN_LCD = A0 00 00 H ตำแหน่งพอร์ต B ใช้ในการส่งรหัสควบคุมไปยัง LCD

LED = A0 00 00 H ตำแหน่งพอร์ต C ใช้ในการส่งข้อมูลให้ LED บนบอร์ด

CONT_LCD = A0 00 00 H ตำแหนงของพอร์ตควบคุมในการโปรแกรมการทำงานของ 8255 ที่เชื่อมต่อกับ LCD

CHAROUT = A0 00 08 H เป็นตำแหน่งพอร์ต A ในการส่งรหัสของคีย์ที่ถูกกดจะแสดงทาง LED

CHARIN = A0 00 0A H เป็นตำแหน่งพอร์ต B ใช้รับข้อมูลจากคีย์บอร์ด

CHKCHAR = A0 00 0C H เป็นพอร์ต C เพื่อทำการตรวจสอบสถานะของคีย์บอร์ดและการควบคุมคีย์บอร์ด

CHARCON = A0 00 0E H เป็นพอร์ตควบคุมในการโปรแกรมการทำงานของ 8255 ที่เชื่อมต่อคีย์บอร์ด

A3 = User Program Counter เป็นตำแหน่งเริ่มต้นในการเขียนข้อมูลจากผู้ใช้ลงหน่วยความจำจากตำแหน่ง 80 01 00 H เป็นต้นไป

D6 = CHARACTER เป็นตัวแปรใช้เก็บค่ารหัสที่ถูกกดจากคีย์บอร์ด

D5 = COUNTER ใช้ในการเก็บค่าบรรทัดของ LCD ใช้แสดงผลบรรทัดที่ 1-5

3.3.6 การทำงานของฟังก์ชั่นต่างๆ ในโปรแกรมมอนิเตอร์

MAIN : เป็นส่วนของโปรแกรมหลักซึ่งมีการทำงานตามผังงานที่ได้ออกแบบไว้

PUSHWD0 : เป็นฟังก์ชั่นของการนำค่าของ D0 ซึ่งมีขนาด 1 เวิร์ดลงสแตค

POP STACK : เป็นการนำค่าจาก สแตคปัจจุบันมาเก็บไว้ที่ D0

ASCII TO NUMBERIC : เป็นฟังก์ชั่นที่ทำการเปลี่ยนแปลงข้อมูลเป็นรหัสคีย์บอร์ดจาก D6 และเปลี่ยนแปลงเป็นค่าของตัวเลขใน D4

RUN ROUTINE : เป็นฟังก์ชั่นที่ 68000 ไปประมวลผลที่ตำแหน่ง 80 01 00 H เป็นต้นไป

INTIAL KEYBOARD : เป็นการสร้างสภาวะเริ่มต้นของคีย์บอร์ด โดยทำการโปรแกรมโปรแกรมพอร์ต8255 โดยพอร์ต B เป็นพอร์ตอินพุตรับค่ารหัสคีย์บอร์ดพอร์ต C เป็นพอร์ต ควบคุมการทำงานของคีย์บอร์ด พอร์ต A เป็นพอร์ตการแสดงผลโดย LED

CLEAR CHARACTER : ทำการล้างคำสั่งรหัสจากคีย์บอร์ดโดยเป็น 0 ทั้งหมดเพื่อรอรับการกดคีย์ต่อไป

KEYBOARD HIT : การตรวจสอบการกดคีย์ หากกดคีย์ 1 ครั้ง จะส่งรหัสคีย์บอร์ดผ่านทางค่า D6

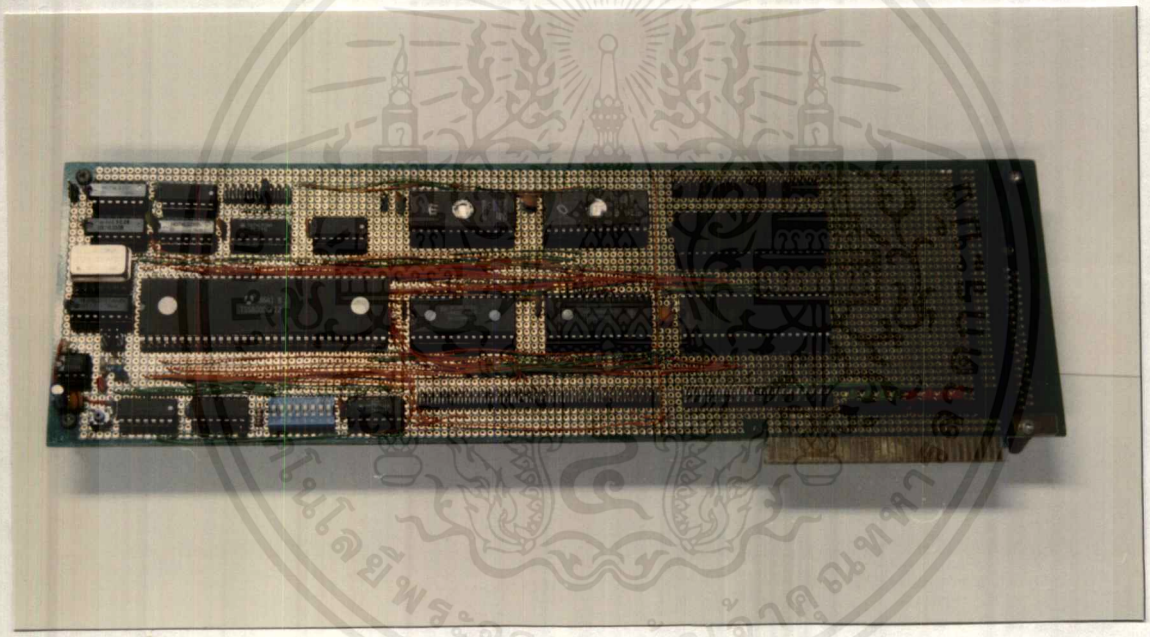
DELAY ROUNTINE : เป็นการสร้างสภาวะการคอย โดยมี 2 ระดับ คือ DELAY และ DELAY2 โดย DELAY2 มีสภาวะการคอยนานกว่า DELAY

CHARACTER IN	: เป็นการแสดงค่ารหัสคีย์บอร์ดที่ถูกกดออกทาง LED
INITLCD	: เป็นการตัดคำเริ่มต้นโปรแกรมการแสดงผลออกทาง LCD โดยมีรายละเอียด คือ LCD ขนาด 16 คอลัมน์ 4 แถว สามารถทำการแสดงผลได้โดยเคอร์เซอร์กระพริบ เมื่อมีการแสดงผลแล้วตัวอักษร 1 ตัว เคอร์เซอร์จะเลื่อนไปทางขวา
CLEAR SCREEN	: ทำหน้าที่ลบจอภาพทั้งหมด และเคอร์เซอร์อยู่ตำแหน่งบรรทัดแรกและคอลัมน์แรก
EPLUSE	: กำเนิดสัญญาณนาฬิกาให้กับ LCD
GOTOX1Y1	: กำหนด เคอร์เซอร์ไปตำแหน่งแรกของบรรทัดที่ 1
GOTOX2Y2	: กำหนด เคอร์เซอร์ไปตำแหน่งแรกของบรรทัดที่ 2
GOTOX3Y3	: กำหนด เคอร์เซอร์ไปตำแหน่งแรกของบรรทัดที่ 3
GOTOX4Y4	: กำหนด เคอร์เซอร์ไปตำแหน่งแรกของบรรทัดที่ 4
PRINTCHAR	: ทำการแสดงผลรหัสคีย์ ที่ถูกกดออกทาง LCD ตำแหน่งเคอร์เซอร์ ปัจจุบันโดยจะแสดงค่าตามคีย์ที่อยู่ใน D6
WRITELINE	: เป็นการแสดงข้อความ 1 บรรทัด ออกทาง LCD โดย D0 จะบอกถึงบรรทัดและ A4 คือ ตำแหน่งของข้อความที่ต้องการแสดงผล
TITLE SHOW	: เป็นการแสดงหน้าจอเริ่มการทำงาน

บทที่ 4

การทดลองและผลการทดลอง

ในการใช้งานของ 68000 เพื่อให้ได้ผลการดำเนินงานที่ถูกต้องนั้น จะประกอบด้วยวงจรควบคุมการทำงานของขาสัญญาณต่างๆ ของ 68000 ให้การทำงานไมโครโปรเซสเซอร์เป็นไปตามเป้าหมาย วงจรควบคุมเหล่านี้เป็นวงจรพื้นฐานในการเริ่มต้นให้เกิดการทำงานของ 68000 เกิดขึ้น ดังจะได้อธิบายผลการทดลองของวงจรพื้นฐานดังกล่าวต่อไปนี้

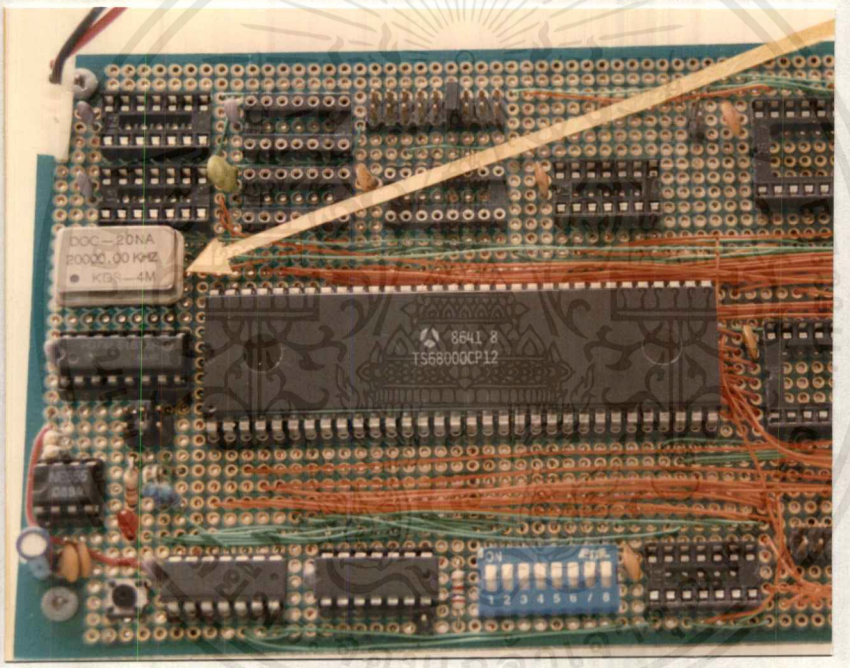


รูปที่ 4.1 บอร์ดทดลองของ 68000 ซิงเกิ้ลบอร์ด คอมพิวเตอร์

4.1 การทดลองวงจรกำเนิดสัญญาณนาฬิกา (Clock)

วงจรกำเนิดสัญญาณนาฬิกาเป็นวงจรที่ป้อนให้ 68000 เพื่อสร้างวัฏจักรการทำงานให้เกิดขึ้นกับ 68000 วงจรกำเนิดสัญญาณนาฬิกาเป็นวงจรที่ให้กำเนิดสัญญาณพัลซ์ โดยสามารถดูรูปวงจรกำเนิดสัญญาณนาฬิกาได้ดังรูปในรูปที่ 4.2 วงจรกำเนิดสัญญาณนาฬิกาบนบอร์ดทดลอง 68000

ในโครงงานนี้เลือกใช้ ไมโครโปรเซสเซอร์ทำงานที่ความถี่ 10 เมกกะเฮิตซ์ ดังนั้น คาบเวลาแต่ละคาบที่ป้อนให้ 68000 $= \frac{1}{10\ \text{MHz}} = 10$ ไมโครวินาที กำหนดดิวตี้ไซเกิลเท่ากับ 50 % แต่ไมโครโปรเซสเซอร์มีขาสัญญาณอยู่สัญญาณหนึ่งคือ ขา E ขานี้ทำหน้าที่หารความถี่ที่ป้อนให้กับไมโครโปรเซสเซอร์ลง 10 เท่า $= \frac{10\ \text{MHz}}{10} = 1$ เมกกะเฮิตซ์ คาบเวลาจึงเท่ากับ 1 ไมโครวินาที ผลของวงจรกำเนิดสัญญาณนาฬิกาและขาสัญญาณ E ที่ใช้ลอจิกอนาไลเซอร์วัดผลการทำงานดังแสดงในรูปที่ 4.2



รูปที่ 4.2 วงจรกำเนิดสัญญาณนาฬิกาบนบอร์ดทดลอง 68000

ขั้นตอนการทดลอง

1. ใช้ลอจิกอนาไลเซอร์วัดสัญญาณกำเนิดสัญญาณนาฬิกา จากวงจรผลิตกำเนิดสัญญาณนาฬิกา
2. เปิดสวิทซ์เริ่มต้นการทำงานของ 68000
3. ให้ลอจิกอนาไลเซอร์ Run การทำงาน

ผลการทดลอง

ผลการทดลองวัดสัญญาณกำเนิดสัญญาณนาฬิกาที่ป้อนให้ 68000 และวัดสัญญาณขา E ของ 68000 ดังแสดงในรูปที่ 4.3 การวัดสัญญาณกำเนิดสัญญาณนาฬิกาที่ป้อนให้ 68000 และวัดสัญญาณขา E ของ 68000

รูปที่ 4.3 ผลการวัดที่ขาสัญญาณกำเนิดสัญญาณนาฬิกาและขาสัญญาณ E ของ 68000

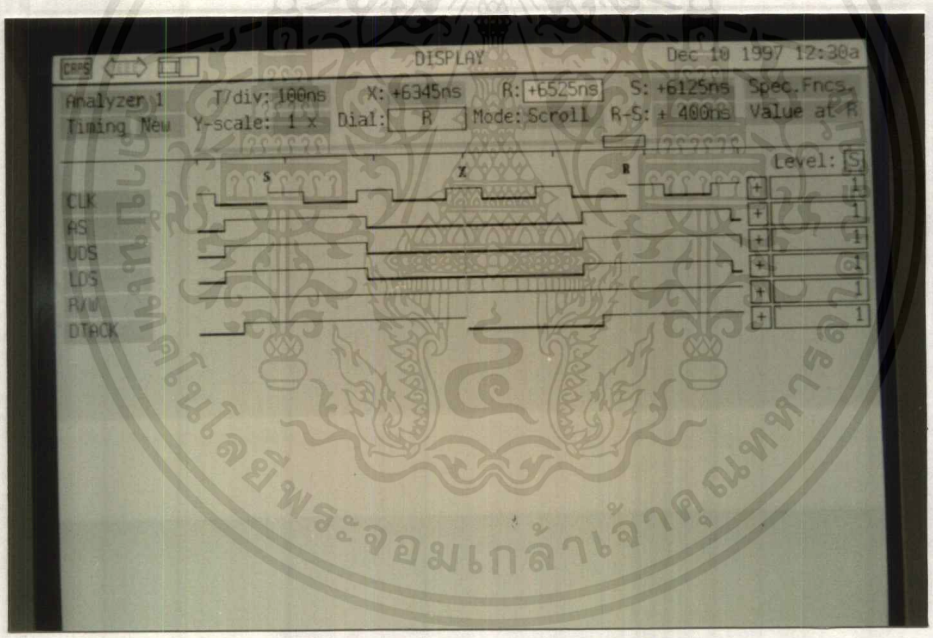
4.2 การทดลองวงจร $\overline{\text{DTACK}}$ (Data Transfer Acknowledge)

ในการทำงานของ 68000 นั้นเมื่อทำการติดต่อกับหน่วยความจำหรืออุปกรณ์ อินพุต หรือเอาต์พุต ขาหนึ่งใน 64 ขาของ 68000 จะเป็นขาที่เป็นอินพุตสัญญาณเข้า 68000 ซึ่งอินพุตที่กล่าวถึงนี้จะทำหน้าที่รับสัญญาณจากอุปกรณ์ภายนอก เป็นการตอบสนองจากอุปกรณ์ภายนอก เมื่ออุปกรณ์ภายนอกทำงานใดๆ ที่ได้รับมอบหมายจาก 68000 จบสิ้นแล้ว แต่ถ้า 68000 ติดต่อกับอุปกรณ์ภายนอกไม่ได้รับสัญญาณที่กล่าวมานี้ กลับคืนมาไซเกิลการทำงานของ 68000 จะถูกหน่วงออกไปหรือเกิดสภาวะ wait ขึ้น สัญญาณที่มีผลต่อการทำงานของ 68000 นี้ คือสัญญาณ $\overline{\text{DTACK}}$ ในการทดลองของวงจร $\overline{\text{DTACK}}$ สามารถใช้การหน่วงเวลาให้

เกิดขึ้นได้โดยควบคุมการหน่วงเวลาของสัญญาณ $\overline{\text{DTACK}}$ โดยเลือกการหน่วงเวลาได้เป็น 1 wait จนถึง 7 wait

ขั้นตอนการทดลอง

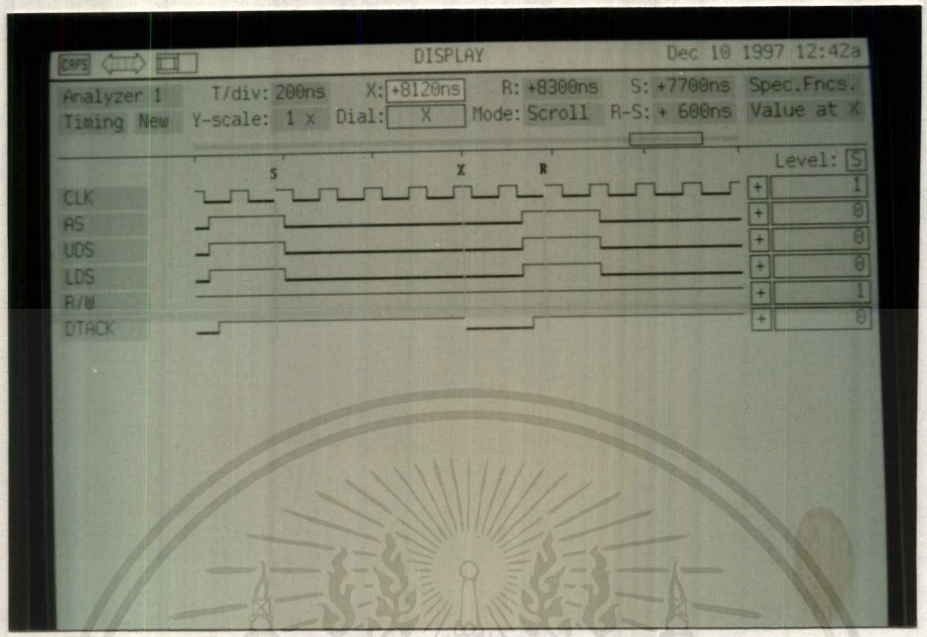
1. ควบคุมการหน่วงเวลาของสัญญาณ $\overline{\text{DTACK}}$ ได้โดยทำการ jumper บนบอร์ดของ 68000 โดยการทดลองนี้เลือก jumper หน่วงเวลาเป็น 0 สเตท, 3 สเตท, 5 สเตท และ 7 สเตท เลือก jumper บนบอร์ดทีละครั้ง
2. ใช้ลอจิกอนาไลเซอร์วัดขาสัญญาณ $\overline{\text{DTACK}}$
3. เป็นสวิทซ์เริ่มต้นการทำงานของ 68000
4. ให้ลอจิกอนาไลเซอร์ Run การทำงาน



รูปที่ 4.4 ผลตอบสนองของวงจร $\overline{\text{DTACK}}$ แบบไม่มีการหน่วงเวลา

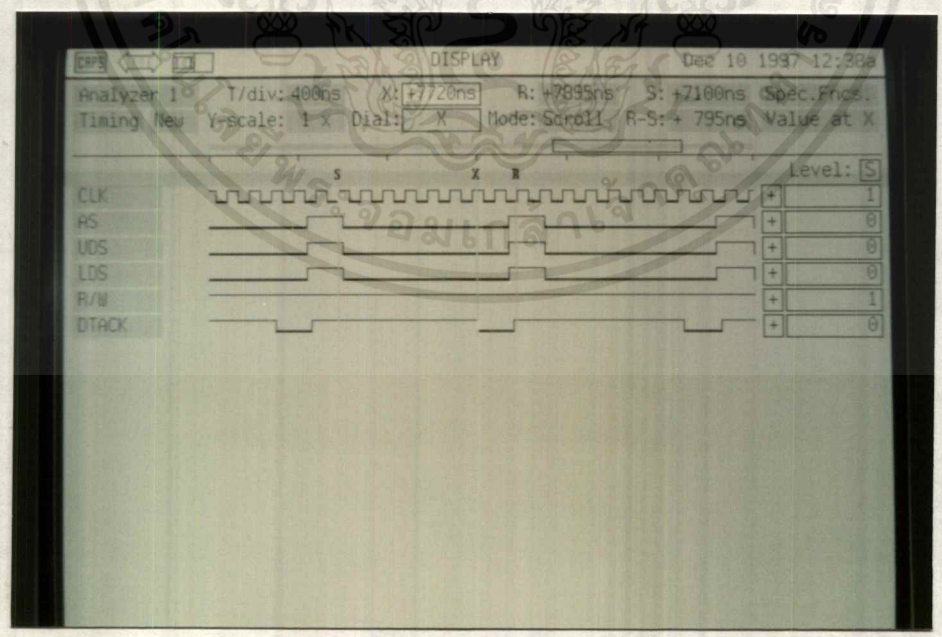
ผลการทดลอง

1. เมื่อสภาวะไม่มีการหน่วงเวลาเกิดขึ้นดังรูปที่ 4.4 แสดงสภาวะไม่มีการหน่วงเวลา
2. เมื่อสภาวะมีการหน่วงเวลา 3 สเตท เกิดขึ้นรูปที่ 4.5 สภาวะการหน่วงเวลา 3 สเตท



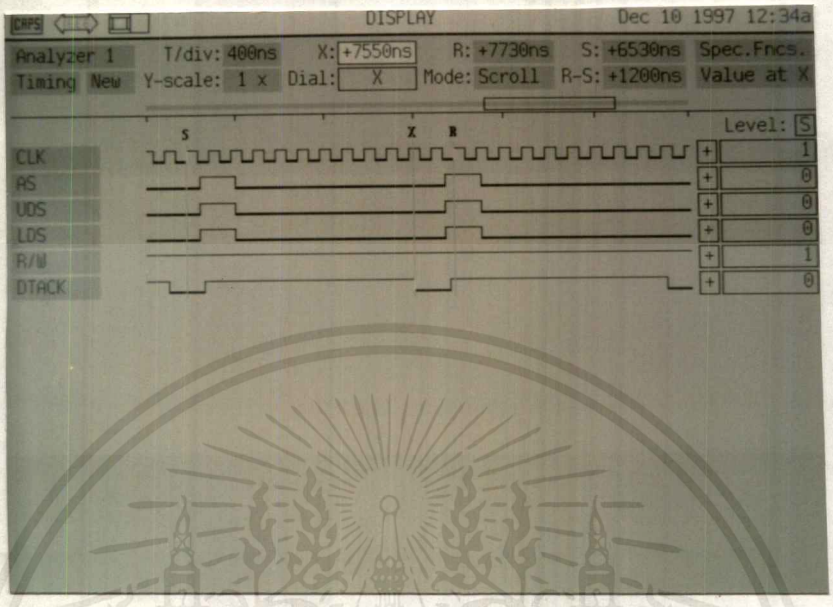
รูปที่ 4.5 ผลตอบสนองของวงจร $\overline{\text{DTACK}}$ แบบมีการหน่วงเวลา 3 สเตท

3. เมื่อสภาวะมีการหน่วงเวลา 5 สเตท ในรูปที่ 4.6 สภาวะการหน่วงเวลา 5 สเตท



รูปที่ 4.6 ผลตอบสนองของวงจร $\overline{\text{DTACK}}$ แบบมีการหน่วงเวลา 5 สเตท

4. เมื่อสภาวะมีการหน่วงเวลา 7 สเตท เกิดขึ้นรูปที่ 4.7 สภาวะการหน่วงเวลา 7 สเตท



รูปที่ 4.7 ผลตอบสนองของวงจร $\overline{\text{DTACK}}$ แบบมีการหน่วงเวลา 7 สเตท

4.3 การทดลองวงจร RESET

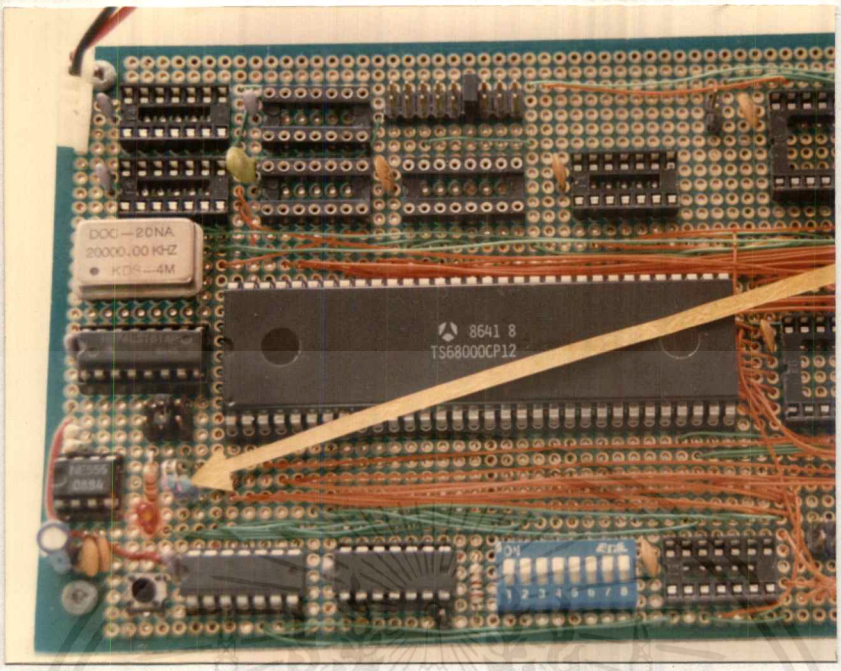
เป็นวงจรที่จะถูกดำเนินการให้เกิดการทำงานเมื่อเริ่มเปิดเครื่อง หรือมีการรีเซ็ทเกิดขึ้นจากอุปกรณ์ภายนอก ผลการทำงานในสภาวะเริ่มเปิดเครื่องนั้น แหล่งจ่าย Vcc จะต้องเป็น 5 โวลท์ และเมื่อแหล่งจ่าย Vcc มีขนาดเป็น 5 โวลท์ แล้วที่ขาสัญญาณ $\overline{\text{RESET}}$ และ $\overline{\text{HALT}}$ ต้องเป็นศูนย์อย่างน้อย 100 มิลลิวินาที 68000 จึงจะเริ่มกระบวนการทำงานเกิดขึ้น ผลการทำงานในสภาวะเริ่มเปิดเครื่องจะแสดงสภาวะในการเริ่มต้นการทำงานโดยใช้ แอลอีดี เป็นตัวแสดงผลโดยแอลอีดีที่ต่ออยู่นี้ จะติดอยู่ชั่วเวลาไม่นานนัก แต่ถ้าแอลอีดีติดอยู่เวลานานไม่ยอมดับเป็นการบอกให้รู้ว่า 68000 มีสภาวะที่ไม่ปกติในการทำงาน เช่น ไมโครโปรเซสเซอร์เกิดสภาวะ Halt หรือ No operate เกิดขึ้น รูปที่ 4.8 แสดงสภาวะการทำงานเริ่มเปิดเครื่อง สังเกตได้จากดวงไฟของแอลซีดี

ขั้นตอนการทดลอง

เปิดสวิตซ์การทำงานของ 68000

ผลการทดลอง

ในการใช้ลอจิกอนาไลเซอร์วัดผลอออกมาดังรูปที่ 4.8



รูปที่ 4.8 วงจรรีเซ็ตบนบอร์ดทดลอง 68000

4.4 การทดลองวงจรถอดรหัส (Decode)

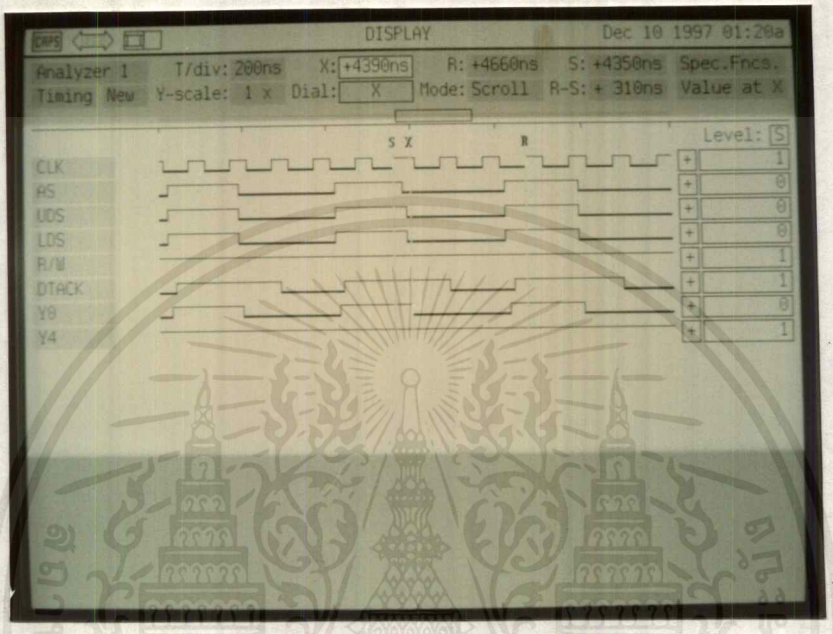
ในโครงงานนี้เลือกการถอดรหัสหน่วยความจำโดยใช้ไอซี 74LS138 ซึ่งไอซี 74LS138 สามารถแบ่งหน่วยความจำออกได้เป็น 8 ส่วนจากเอาต์พุตของไอซี 74LS138 ความสามารถในการอ้างอิงตำแหน่งของ68000 ทำได้ถึง 16 เมกกะไบต์ ช่วงทำงานแต่ละเอาต์พุต Y_0–Y_7 ของ 74LS138 จึงถูกแบบออกเป็นช่วงละ 2 เมกกะไบต์ วงจรดีโค็ดในโครงงานนี้เลือกใช้เป็นตัวแอคทีฟหน่วยความจำ ROM ที่เอาต์พุตY_0และแอคทีฟหน่วยความจำ RAM ที่เอาต์พุต Y_4 โดย Y_0 แอคทีฟตำแหน่งในหน่วยความจำที่ร่ตำแหน่ง 00 00 00 00 H และ Y_4 แอคทีฟที่ตำแหน่ง 80 00 00 00 H ในหน่วยความจำ ผลของการแอคทีฟของ Y_0 และ Y_4 ที่ใช้ลอจิกอนาไลเซอร์วัดผลการทำงานดังในรูปที่ 4.8 และรูปที่ 4.9 โดย Y_0 และ Y_4 ผลัดกันแอคทีฟ การแอคทีฟของแต่ละสัญญาณจะเป็น $\overline{AS}$ เมื่อเป็นศูนย์พร้อมกับ $\overline{UDS}$ และ $\overline{LDS}$ เป็นศูนย์

ขั้นตอนการทดลอง

1. ใช้ลอจิกอนาไลเซอร์วัดขาสัญญาณ Y_0 และ Y_4
2. เปิดสวิทซ์เริ่มต้นการทำงานของ 68000
3. ให้ลอจิกอนาไลเซอร์ RUN การทำงาน

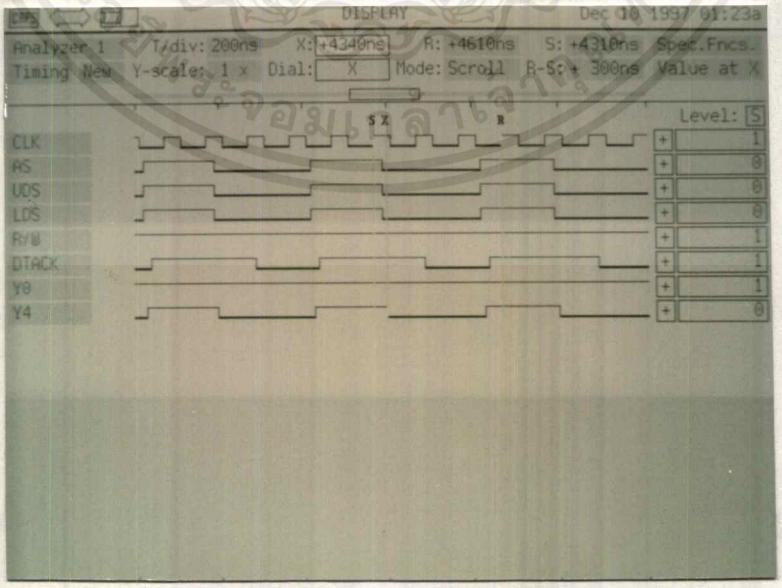
ผลการทดลอง

1. การแอคทีฟ Y_0 ของไอซี 74LS138 เพื่อไป Enable หน่วยความจำ ROM ดังแสดงในรูปที่ 4.9



รูปที่ 4.9 การแอคทีฟY_0 ของไอซี 74LS138

2. การแอคทีฟ Y_4 ของไอซี 74LS138 เพื่อไป Enable หน่วยความจำ RAM ในรูปที่ 4.10



รูปที่ 4.10 การแอคทีฟ Y_4 ของไอซี 74LS138

4.5 การทดลองวงจร U_$\overline{WE}$, U_RD, L_$\overline{WE}$ และ L_RD

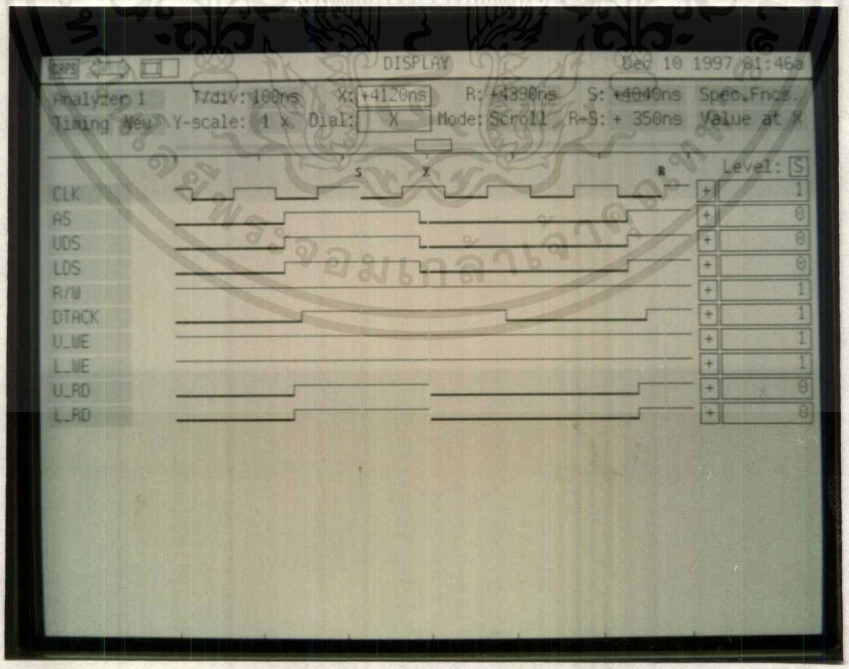
เราใช้ขา U_$\overline{WE}$, U_RD, L_$\overline{WE}$ และ L_RD ในการบอกการติดต่อกับอุปกรณ์ภายนอก แต่เนื่องจากบัสข้อมูลของ 68000 เป็นขนาด 16 บิต จึงต้องมีการบอกการติดต่อกับอุปกรณ์ภายนอกเป็นแบบ 8 บิตบนและ 8 บิตล่าง แต่ในการทดลองวงจร U_$\overline{WE}$, U_RD, L_$\overline{WE}$ และ L_RD นี้ เป็นการทดลองในสภาวะ free running หรือสภาวะการเริ่มต้นการทำงานของ 68000 จะอยู่ในสภาวะ free running เสมอ ที่สภาวะ free running นี้ 68000 จะทำการอ่านข้อมูลตลอดเวลา U_$\overline{WE}$ และ L_$\overline{WE}$ จึงเป็นศูนย์ แต่ U_RD และ L_RD จะเป็นหนึ่งเสมอ

ขั้นตอนการทดลอง

1. ใช้ลอจิกอนาไลเซอร์วัดขาสัญญาณ U_$\overline{WE}$, U_RD, L_$\overline{WE}$ และ L_RD
2. เปิดสวิตซ์เพื่อเริ่มต้นการทำงานให้ 68000
3. ให้ลอจิกอนาไลเซอร์ Run การทำงาน

ผลการทดลอง

เมื่อใช้ลอจิกอนาไลเซอร์วัดแล้วได้ผลดังรูปที่ 4.11



รูปที่ 4.11 ผลตอบสนองของสัญญาณ U_$\overline{WE}$, U_RD, L_$\overline{WE}$ และ L_RD

บทที่ 5
บทสรุป ปัญหา แนวทางแก้ไข และพัฒนา

5.1 บทสรุป

การจัดสร้าง 68000 ซิงเกิ้ลบอร์ดไมโครคอมพิวเตอร์ในปริญญานิพนธ์ฉบับนี้ ถูกจัดทำขึ้น เพื่อศึกษาหลักการของสถาปัตยกรรมของไมโครโปรเซสเซอร์ 68000 ที่จะนำมาใช้สร้าง 68000 ซิงเกิ้ลบอร์ดไมโครคอมพิวเตอร์ ซึ่งสามารถจัดสร้างโดยนำเอาอุปกรณ์และเครื่องมือที่มีอยู่ภายในประเทศมาใช้ โดยสามารถติดต่อกับอุปกรณ์อินพุตและเอาต์พุตได้รวมไปถึงการจัดสร้างยังสามารถปรับปรุงและพัฒนาในการนำ 68000 ไปใช้งานในด้านต่างๆ ได้ โดยการสร้างอุปกรณ์เชื่อมต่อกับคอนเนคเตอร์ที่ได้ทำการออกแบบไว้แล้ว เนื่องจาก 68000 เป็นไมโครโปรเซสเซอร์ขนาด 16 บิต จะทำให้การประมวลผลมีความเร็วกว่าขนาด 8 บิต ซึ่งสามารถประยุกต์ใช้งานกับอุปกรณ์ขนาด 16 บิตได้ เนื่องจากการศึกษาและการใช้งาน 68000 นั้น เรายังต้องนำซิงเกิ้ลบอร์ดขนาด 16 บิตเข้ามาจากต่างประเทศ จึงนับได้ว่าโครงงานนี้จะมีประโยชน์ต่อผู้ที่สนใจจะนำ 68000 ไปใช้งานและทำการพัฒนาให้ดีขึ้น ทั้งทางด้านฮาร์ดแวร์และซอฟแวร์ต่อบุคคลที่สนใจ

5.2 ปัญหาและแนวทางแก้ไข

5.2.1 ปัญหา

1. การเขียนโปรแกรมด้วยภาษาแอสเซมบลี้ของ 68000 นั้น เป็นรูปแบบเฉพาะตัว ซึ่งจะแตกต่างจากไมโครโปรเซสเซอร์ในตระกูลอื่น ๆ ทำให้การพัฒนาโปรแกรมขนาดใหญ่มีความล่าช้า

2.เนื่องจาก 68000 ซิงเกิ้ลบอร์ดไมโครคอมพิวเตอร์ ยังขาดโปรแกรมที่ทำการควบคุมการทำงานหรือโปรแกรมมอนิเตอร์ ทำให้การทดสอบหรือแก้ไขการทำงานของซิงเกิ้ลบอร์ดเป็นไปอย่างล่าช้า

3. การเขียนโปรแกรมและนำโปรแกรมลงสู่หน่วยความจำของ 68000 นั้น มีขนาด 16 บิต แต่เครื่องมือที่ใช้อยู่นั้น สามารถนำโปรแกรมลงสู่หน่วยความจำได้ทีละ 8 บิต ทำให้ล่าช้าและไม่สะดวกในการพัฒนาโปรแกรม

5.2.2 แนวทางแก้ไข

1. จัดหาคอมไพล์เลอร์หรือตัวแปลภาษาระดับสูง เพื่อช่วยในการเขียนโปรแกรมจะช่วยให้พัฒนาโปรแกรมที่มีขนาดใหญ่ได้สะดวกกว่าและภาษาระดับสูงง่ายต่อการเข้าใจและเขียนโปรแกรม

2. จัดสร้างโปรแกรมมอนิเตอร์ เพื่อที่จะนำมาควบคุมการทำงานและทำการตรวจสอบการทำงานของ 68000 ด้วย

3. จัดสร้างเครื่องมือที่นำมาโปรแกรมลงสู่หน่วยความจำได้ครั้งละ 16 บิต โดยใช้ไมโครคอมพิวเตอร์เป็นตัวช่วยในการนำข้อมูลลงสู่หน่วยความจำซิงเกิ้ลบอร์ด

5.3 แนวทางในการพัฒนา

ปริญญานิพนธ์ฉบับนี้ คณะผู้จัดทำได้พยายามทำให้ 68000 ไมโครคอมพิวเตอร์ แบบแผ่นพิมพ์เดี่ยว โดยใช้ ไอซีเบอร์ 68000 ซึ่งเป็นอุปกรณ์ที่มีความสามารถสูงและมีคุณภาพมากที่สุดเท่าที่จะสามารถทำได้ แต่ด้วยระยะเวลาและงบประมาณที่จำกัด ประกอบกับตำราที่ใช้ศึกษาเป็นตำราจากต่างประเทศและประสบการณ์ของคณะผู้จัดทำ ยังไม่มากพอสำหรับการทำงานนี้จึงทำให้เครื่องที่จัดทำขึ้นมา ยังมีข้อจำกัด และสมควรที่จะได้รับการพัฒนาให้ดีขึ้นได้อีกดังต่อไป

1. การจัดการเกี่ยวกับการร้องขออินเตอร์รัพต์จากอุปกรณ์ภายนอก
2. การติดต่อกับไมโครคอมพิวเตอร์โดยผ่านทางพอร์ตอนุกรม RS232
3. การจัดสร้างโปรแกรมมอนิเตอร์ เพื่อตรวจสอบแก้ไขข้อมูลในหน่วยความจำ
4. การติดต่อกับหน่วยความจำชนิด DRAM ซึ่งจะช่วยทำให้ผู้ใช้สามารถติดต่อกับหน่วยความจำได้มากขึ้น
5. การติดต่อกับอุปกรณ์อินพุตและเอาต์พุตชนิดอื่นๆ เช่น จอมอนิเตอร์ หน่วยความจำสำรองอัน ได้แก่ เครื่องอ่านแถบแม่เหล็กชนิดอ่อน (Floppy disk) และเครื่องอ่านแถบแม่เหล็กชนิดแข็ง (Hard disk) ฯลฯ

ภาคผนวก ก
โปรแกรมการทำงาน

```
        CPU   "68000.TBL"
        HOF   "BIN16"


        ORG   00000H
STACK:  DWM   80H,0FFF0H        ;RESET :  initial SSP
        DWM   0,START           ;RESET :  initial PC


;************************************************
;               VARIABLE
;************************************************

DATA_LCD    EQU   0A00000H  ; PA=OUT FOR DATA
SIGN_LCD    EQU   0A00002H  ; PB=OUT FOR SIGN
LED         EQU   0A00004H  ; PC=ON  BOARD LED
CONT_LCD    EQU   0A00006H  ; PORT CONTROL

CHAROUT     EQU   0A00008H  ; OUT   CHAR
CHARIN      EQU   0A0000AH  ; PB=IN CHAR
CHKCHAR     EQU   0A0000CH  ; PCU=IN,PCL=OUT
                            ; PC5=START BIT,PC1=RESET KEY
CHARCON     EQU   0A0000EH  ; PCU XX1X XXXX=NO CHAR ,
                            ; XX0X XXXX=ENABLE CHAR
                            ; PCL XXXX XX0X=RESET KEY, XXXX XX1X
                            ; WAIT

;A3  PROGRAM COUTER
;D6  CHARECTER
```

```
;D5  COUNTER
;*************************************************
;                       MAIN
;*************************************************
                ORG     0400H
START:

                MOVEA.L  #0803FF0H,A7

                MOVEA.L  #USER_PROGRAM,A3
;FILL_MEM:       ADDA.L   #2,A3
;               MOVE.W   #0000H,(A3)
;               CMPA.L   #00803FFFH,A3
;               BNE    FILL_MEM
                MOVE.L   #0000001H,D5
                BSR    INITLCD
                BSR    TITLE
                BSR    INIT_KEY
                BSR    CLR_KEY
                BSR    KBHIT
                BSR    DELAY2
                BSR    CLRSCR

LOOP_KEY:
N1:             BSR    CLR_KEY  ; BYTE ONE
                BSR    KBHIT
                BSR    IN_KEY
                BSR    ATOI       ; NUMBERIC IN D4 IS MSB
```

```
        MOVEA.L  #MSB,A0
        MOVE.W   D4,(A0)
        CMP.W    #0FF00H,D4
        BEQ      N1
        BSR      PRINTCHAR
        BSR      CLR_KEY

N2:     BSR      KBHIT
        BSR      IN_KEY
        BSR      ATOI          ; NUMBERIC IN D4 IS LSB
        MOVEA.L  #LSB,A0
        MOVE.W   D4,(A0)
        CMP.W    #0FF00H,D4
        BEQ      N2
        BSR      PRINTCHAR
        BSR      CLR_KEY

        MOVEA.L  #LSB,A0
        MOVE.W   (A0),D0
        MOVEA.L  #MSB,A0
        MOVE.W   (A0),D1
        LSL.W    #4,D1
        OR.W     D1,D0    ; ONE BYTE DATA IN D0

        MOVEA.L  #BUFFER_BYTE_ONE,A0
        MOVE.W   D0,(A0)
        MOVEA.L   #LED,A0
        MOVE.W    D0,(A0)
```

```
N3:     BSR     CLR_KEY  ; BYTE TWO
        BSR     KBHIT
        BSR     IN_KEY
        BSR     ATOI        ; NUMBERIC IN D4 IS MSB
        MOVEA.L  #MSB,A0
        MOVE.W  D4,(A0)
        CMP.W     #0FF00H,D4
        BEQ     N3
        BSR     PRINTCHAR
        BSR     CLR_KEY

N4:     BSR     KBHIT
        BSR     IN_KEY
        BSR     ATOI        ; NUMBERIC IN D4 IS LSB
        MOVEA.L #LSB,A0
        MOVE.W  D4,(A0)
        CMP.W  #0FF00H,D4
        BEQ     N4
        BSR     PRINTCHAR
        BSR     CLR_KEY
        MOVEA.L  #LSB,A0
        MOVE.W    (A0),D0
        MOVEA.L  #MSB,A0
        MOVE.W    (A0),D1
        LSL.W     #4,D1
        OR.W      D1,D0    ; ONE BYTE DATA IN D0

        MOVEA.L  #BUFFER_BYTE_TWO,A0
```

```
MOVE.W    D0,(A0)
MOVEA.L   #LED,A0
MOVE.W    D0,(A0)

MOVEA.L   #BUFFER_BYTE_ONE,A0
MOVE.W    (A0),D0
MOVEA.L   #BUFFER_BYTE_TWO,A0
MOVE.W    (A0),D1
LSR.W     #8,D1
OR.W      D1,D0

;LSL.W    #8,D0

AND.L     #0000FFFFH,D0
MOVE.W    D0,(A3)

MOVE.W    (A3),D0

LSL.W     #8,D0

MOVEA.L   #LED,A0
MOVE.W    D0,(A0)


ADDA.L    #2,A3

ADD.W     #1,D5
```

```
                CMP.B     #2,D5
                BEQ       ROW2
                CMP.B     #3,D5
                BEQ       ROW3
                CMP.B     #4,D5
                BEQ       ROW4
                CMP.B     #5,D5
                BEQ       ROW1

ROW1            MOVE.W    #0001H,D5
                BSR       CLRSCR
                BRA       LOOP_KEY

ROW2            BSR       GOTOX2Y2
                BRA       LOOP_KEY

ROW3            BSR       GOTOX3Y3
                BRA       LOOP_KEY

ROW4            BSR       GOTOX4Y4
                BRA       LOOP_KEY

                BRA       LOOP_KEY

;************************************************
;                         PUSHW   STACK
;IN      D0              VALUE TO PUSH
```

```
;A7        SSP


;*************************************************
PUSHWD0:        MOVE.W   D0,(A7)
                SUBA.W   #2,A7
                RTS
;*************************************************
;               POPW  STACK
;IN        A7
;OUT       D0
;*************************************************
POPWD0:         MOVE.W  (A7)+,D0
                RTS
;*************************************************
;               ASCII TO NUMBERIC
;IN        D6=SCANKEY   UPPER    BYTE
;OUT       D4=NUMBERIC  UPPER    BYTE
;*************************************************
ATOI:           MOVE.W   D6,D4
                AND.W #07F00H,D4
                CMP.W #0200H,D4  ; 1
                BEQ     NUM
                CMP.W #0300H,D4  ; 2
                BEQ     NUM
                CMP.W #0400H,D4  ; 3
                BEQ     NUM
                CMP.W #0500H,D4  ; 4
```

```
BEQ     NUM

CMP.W #0600H,D4  ; 5
BEQ     NUM

CMP.W #0700H,D4  ; 6
BEQ     NUM

CMP.W #0800H,D4  ; 7
BEQ     NUM

CMP.W #0900H,D4  ; 8
BEQ     NUM

CMP.W #0A00H,D4 ; 9
BEQ     NUM

CMP.W #0B00H,D4 ; 0
BEQ     NUM0

CMP.W #1E00H,D4  ; A
BEQ     NUM_A

CMP.W #200H,D4    ; B
BEQ     NUM_B

CMP.W #2E00H,D4  ; C
BEQ     NUM_C
```

```
                CMP.W #2000H,D4  ; D
                BEQ   NUM_D

                CMP.W #1200H,D4  ; E
                BEQ   NUM_E

                CMP.W #2100H,D4  ; F
                BEQ   NUM_F

                CMP.W  #01C00H,D4          ; ENTER
                BEQ    RUN_ROUTINE

                MOVE.W   #0FF00H,D4        ; ELSE KEY
                BRA    RETURN

NUM:            SUB.W      #0100H,D4
                BRA        RETURN
NUM0:           MOVE.W     #0000H,D4
                BRA        RETURN
NUM_A:          MOVE.W     #0A00H,D4
                BRA    RETURN
NUM_B:          MOVE.W  #0B00H,D4
                BRA    RETURN
NUM_C:          MOVE.W     #0C00H,D4
                BRA    RETURN
NUM_D:          MOVE.W  #0D00H,D4
                BRA    RETURN
```

```
NUM_E:          MOVE.W   #0E00H,D4
                BRA      RETURN
NUM_F:          MOVE.W   #0F00H,D4
                BRA      RETURN
RETURN:         MOVEA.L #LED,A0
                MOVE.W   D4,(A0)
                RTS
;************************************************
;               RUN    ROUTINE
;************************************************

RUN_ROUTINE: BRA    USER_PROGRAM

;************************************************
;         INITIAL  KEYBOARD
;************************************************
INIT_KEY:       MOVEA.L  #CHARCON,A0
                MOVE.W   #08A00H,(A0) ;CONTROL PORT
                RTS
;************************************************
;               CLEAR   CHARECTER
;************************************************
CLR_KEY:        MOVEA.L  #CHKCHAR,A1
                MOVE.W   #0000H,(A1) ;RESET KEY
                BSR      DELAY
                MOVE.W   #0FF00H,(A1) ;WAIT FOR KEYBOARD HIT
                RTS
```

```
;*************************************************
;                KEYBOARD HIT
;*************************************************
KBHIT:          MOVEA.L  #CHKCHAR,A1
KBHIT_LOOP:     MOVE.W    (A1),D0
                AND.W  #2000H,D0
                CMP.W  #2000H,D0
                BEQ     KBHIT_LOOP
                RTS
;*************************************************
;                CHARECTER IN
;*************************************************
IN_KEY:         BSR     DELAY
                MOVEA.L   #CHARIN,A2
                MOVE.W   (A2),D6
                MOVEA.L   #CHAROUT,A0
                MOVE.W   D6,(A0)
                RTS

;*************************************************
;          DELAY          ROUTINE
;*************************************************

DELAY:          MOVE.L     #0FFH,D0
LOOP_DELAY:     SUBQ.L  #1,D0
                BNE     LOOP_DELAY
                RTS
```

```
DELAY2:         MOVE.L    #007FFFH,D0
LOOP_DELAY2:    SUBQ.L  #1,D0
                BNE     LOOP_DELAY2
                RTS


;*************************************************


INITLCD:        MOVEA.L #CONT_LCD,A2
                MOVE.W  #8000H,(A2)    ; CONTROL PROT LCD

                MOVEA.L #SIGN_LCD,A1

                MOVE.W    #0000H,(A1)       ; OUT B FOR SIGN
                MOVEA.L #DATA_LCD,A0
                MOVE.W    #03800H,(A0)      ; OUT A FOR DATA
                BSR     EPLUSE              ; DL=1 8 BIT,N=1 1/16 DUTY,
                                            ; F=0  5 X 7
                BSR     DELAY


                MOVE.W    #0F00H,(A0)       ; OUT A FOR DATA
                BSR     EPLUSE              ; MODE SET D=1,C=1 CURSOR
                                            ; ON,B=1 BLINK

                MOVE.W    #0600H,(A0)       ; OUT A FOR DATA
                BSR     EPLUSE              ; I/D =1 INC,S=0 RIGHT
```

```
                MOVE.W    #0100H,(A0)       ; OUT A FOR DATA
                BSR       EPLUSE            ; CLEAR ALL DISPLAY
                BSR       DELAY
                RTS

;*********************************************************
;                       CLEAR SCREEN
;*********************************************************
CLRSCR:
                MOVEA.L #SIGN_LCD,A1
                MOVE.W    #0000H,(A1)       ; OUT B FOR SIGN
                MOVEA.L #DATA_LCD,A0

                MOVE.W    #0100H,(A0)       ; OUT A FOR DATA
                BSR       EPLUSE            ; CLEAR ALL DISPLAY
                BSR       DELAY
                RTS

;************************************************
EPLUSE:         MOVEA.L #SIGN_LCD,A1
                MOVE.W    (A1),D1
                OR.W    #0400H,D1
                MOVE.W    D1,(A1)           ;E PLUSE 1
                BSR       DELAY
                AND.W #0FBFFH,D1
                MOVE.W    D1,(A1)           ;E PLUSE 0
                BSR       DELAY
                RTS
```

```
;************************************************

GOTOX1Y1:   MOVEA.L #DATA_LCD,A0
            MOVEA.L #SIGN_LCD,A1

            MOVE.W#0000H,D0          ; ADDRESS FOR START LINE 1
            OR.W   #8000H,D0         ; SET BIT 7
            MOVE.W   D0,(A0)         ; OUT A FOR DATA
            AND.W #0000H,D0          ;
            MOVE.W   D0,(A1)         ; OUT B SIGN
            BSR    EPLUSE
            RTS

;************************************************

GOTOX2Y2:   MOVEA.L #DATA_LCD,A0
            MOVEA.L #SIGN_LCD,A1

            MOVE.W   #4000H,D0       ; ADDRESS FOR START LINE 1
            OR.W   #8000H,D0         ; SET BIT 7
            MOVE.W   D0,(A0)         ; OUT A FOR DATA
            AND.W #0000H,D0          ;
            MOVE.W   D0,(A1)         ; OUT B SIGN
            BSR    EPLUSE
            RTS

;************************************************
```

```
GOTOX3Y3:    MOVEA.L #DATA_LCD,A0
             MOVEA.L #SIGN_LCD,A1

             MOVE.W #1000H,D0          ; ADDRESS FOR START LINE 1
             OR.W #8000H,D0            ; SET BIT 7
             MOVE.W   D0,(A0)          ; OUT A FOR DATA
             AND.W #0000H,D0           ;
             MOVE.W   D0,(A1)          ; OUT B SIGN
             BSR     EPLUSE
             RTS

;*************************************************
GOTOX4Y4:    MOVEA.L #DATA_LCD,A0
             MOVEA.L #SIGN_LCD,A1

             MOVE.W#5000H,D0           ; ADDRESS FOR START LINE 1
             OR.W#8000H,D0             ; SET BIT 7
             MOVE.W   D0,(A0)          ; OUT A FOR DATA
             AND.W #0000H,D0           ;
             MOVE.W   D0,(A1)          ; OUT B SIGN
             BSR     EPLUSE
             RTS

;*************************************************

PRINTCHAR:   MOVEA.L #DATA_LCD,A0
             MOVEA.L #SIGN_LCD,A1
             MOVE.W   #0100H,(A1)   ;SIGN FOR WRITE
```

```
BSR    DELAY
MOVEA.L #CHAR_MAP,A6
MOVE.L     #00007F00H,D0
AND.L  D0,D6
MOVEA.L #0A00004H,A5
;MOVE.W  D6,(A5)
BSR      DELAY
LSL.W  #1,D6
;MOVE.W  D6,(A5)
BSR      DELAY
AND.L  #0000FF00H,D6
LSR.L  #8,D6

;MOVE.L#00000002,D6

ADDA.W    D6,A6
MOVE.W    (A6),D6
MOVE.W  D6,(A0)              ;DATA BYTE
BSR      EPLUSE
BSR      DELAY2
RTS
```

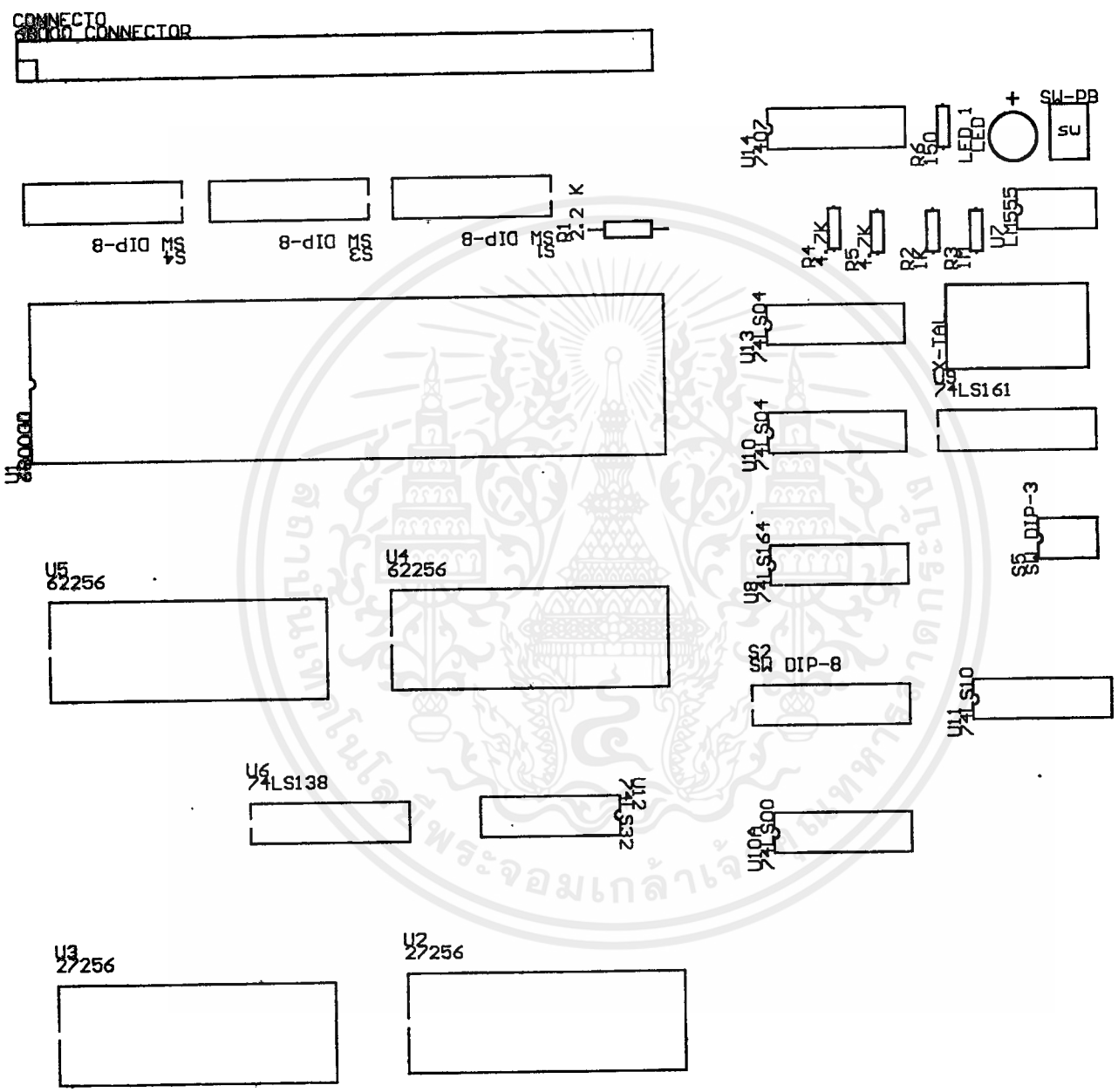
```
;*************************************************
;          WRITE  LINE
; D0 = 0000H LINE1
;      = 4000H LINE2
;      = 1000H LINE3
```

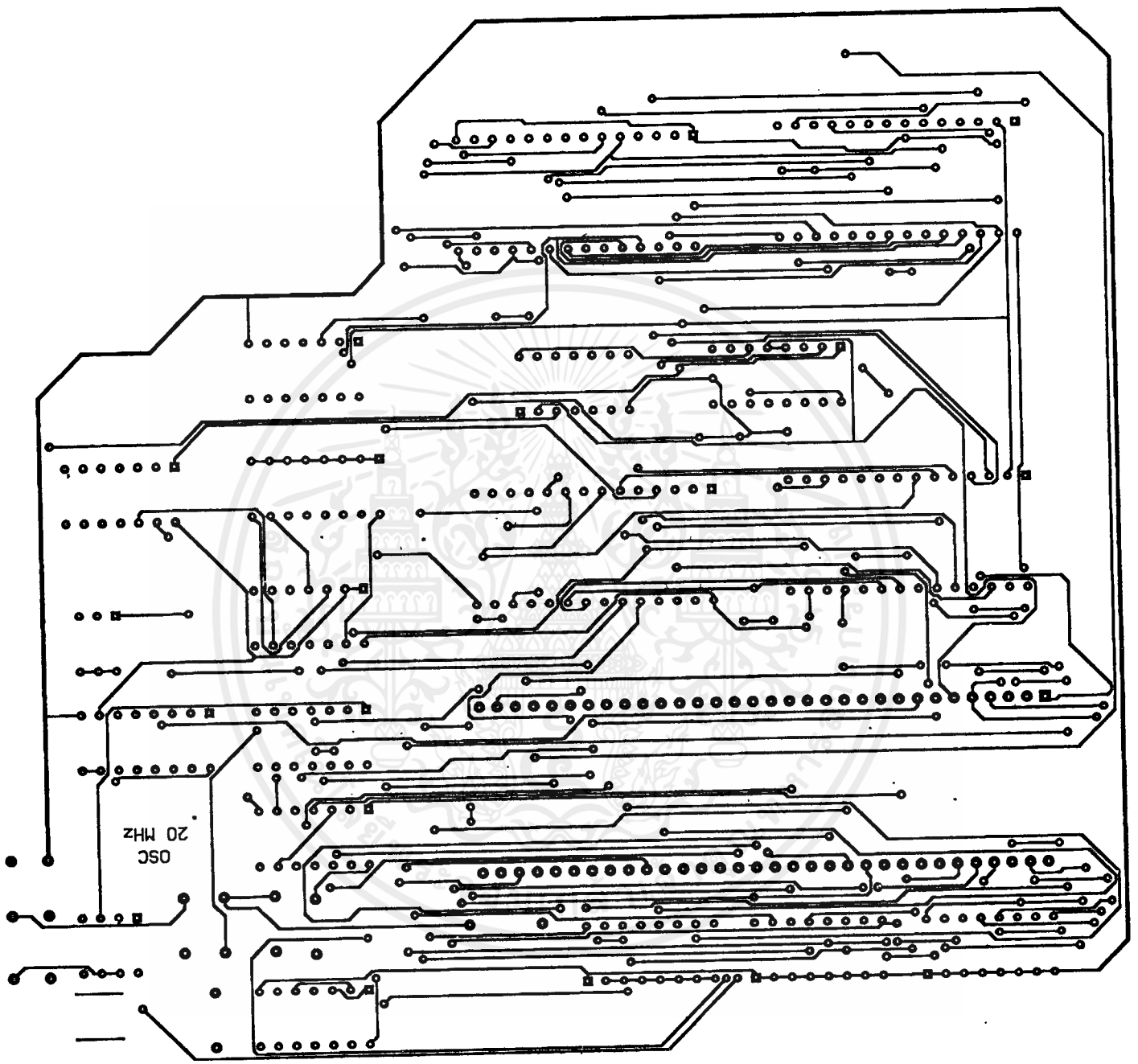
```
;       = 5000H LINE4
;  A4 = ADDRESS BEGIN LINE
;*************************************************

WRITELINE:      MOVEA.L #DATA_LCD,A0
                MOVEA.L #SIGN_LCD,A1
                OR.W    #8000H,D0  ; SET BIT 7
                MOVE.W    D0,(A0)           ; OUT A FOR DATA
                AND.W #0000H,D0             ;
                MOVE.W    D0,(A1)           ; OUT B SIGN
                BSR     EPLUSE
                MOVE.L  #0000000FH,D4       ; 16 CHARECTOR
NEXT_CHAR:      MOVE.W    (A4)+,D0
                MOVE.W#0100H,(A1)           ;SIGN FOR WRITE
                MOVE.W  D0,(A0)             ;DATA BYTE
                BSR     EPLUSE
                SUBQ.B#1,D4
                BNE     NEXT_CHAR

                RTS
;*************************************************************
;           TITLE  SHOW
;*************************************************************
TITLE:
                MOVE.W  #0000H,D0
                MOVEA.L #TEXT_LINE1,A4
                BSR    WRITELINE
```

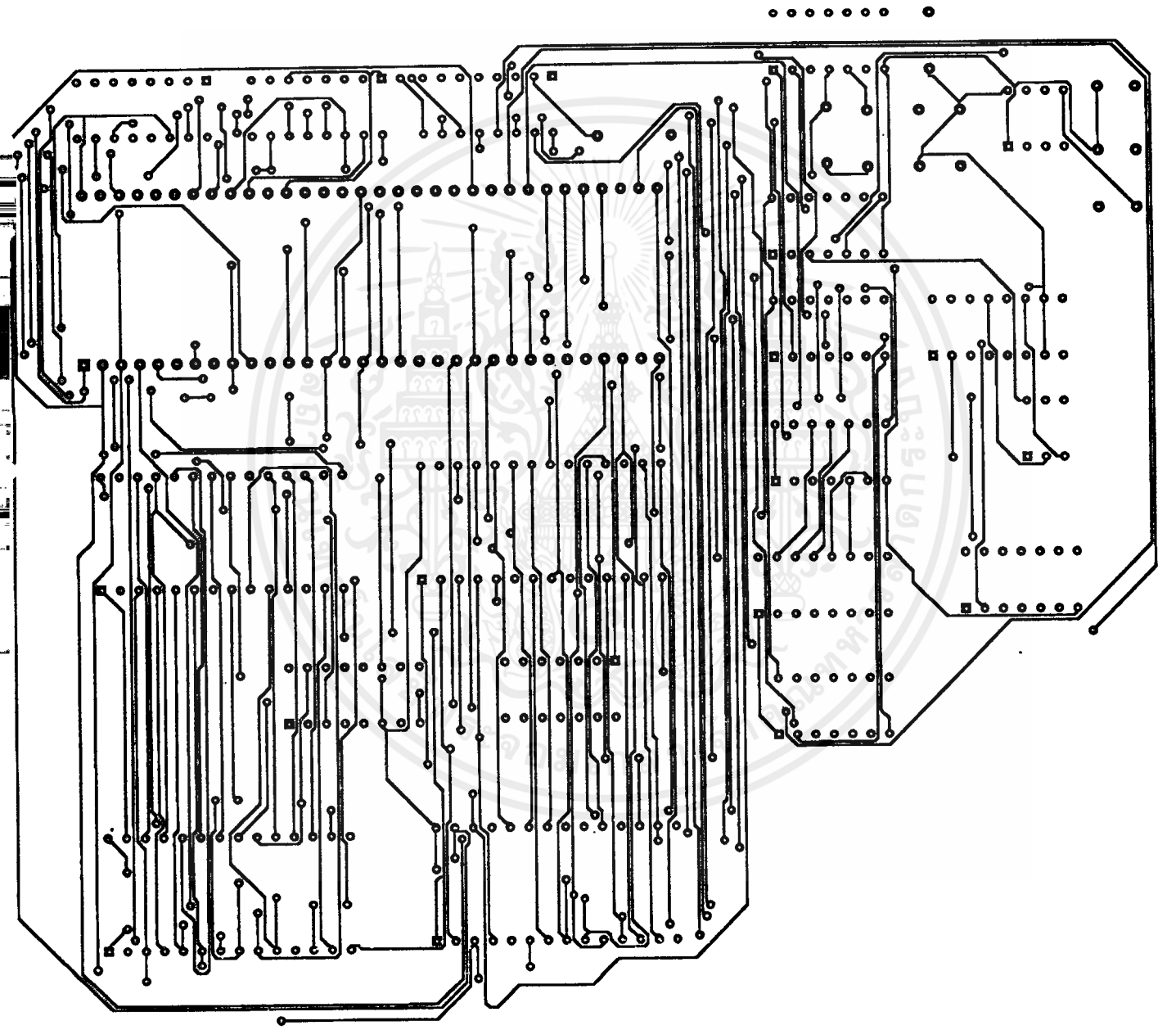
```
                MOVE.W  #4000H,D0
                MOVEA.L #TEXT_LINE2,A4
                BSR     WRITELINE


                MOVE.W  #1000H,D0
                MOVEA.L #TEXT_LINE3,A4
                BSR     WRITELINE


                MOVE.W  #5000H,D0
                MOVEA.L #TEXT_LINE4,A4
                BSR     WRITELINE

                RTS

;*************************************************************


TEXT_LINE1:     DFB     " * * * * * *6*8*0*0*0* * * * * *"
TEXT_LINE2:     DFB     " * *S*I*N*G*L*E* *B*O*R*A*D* * *"
TEXT_LINE3:     DFB     " *M*I*C*R*O*C*O*M*P*U*T*E*R*.* *"
TEXT_LINE4:     DFB     " *K*M*I*T*L*.* * *I*D*.*E*D*.* *"

CHAR_MAP:       DFB     " \1234567890-="
                DFB     "  QWERTYUIOP[]"
                DFB     13H,00H
```

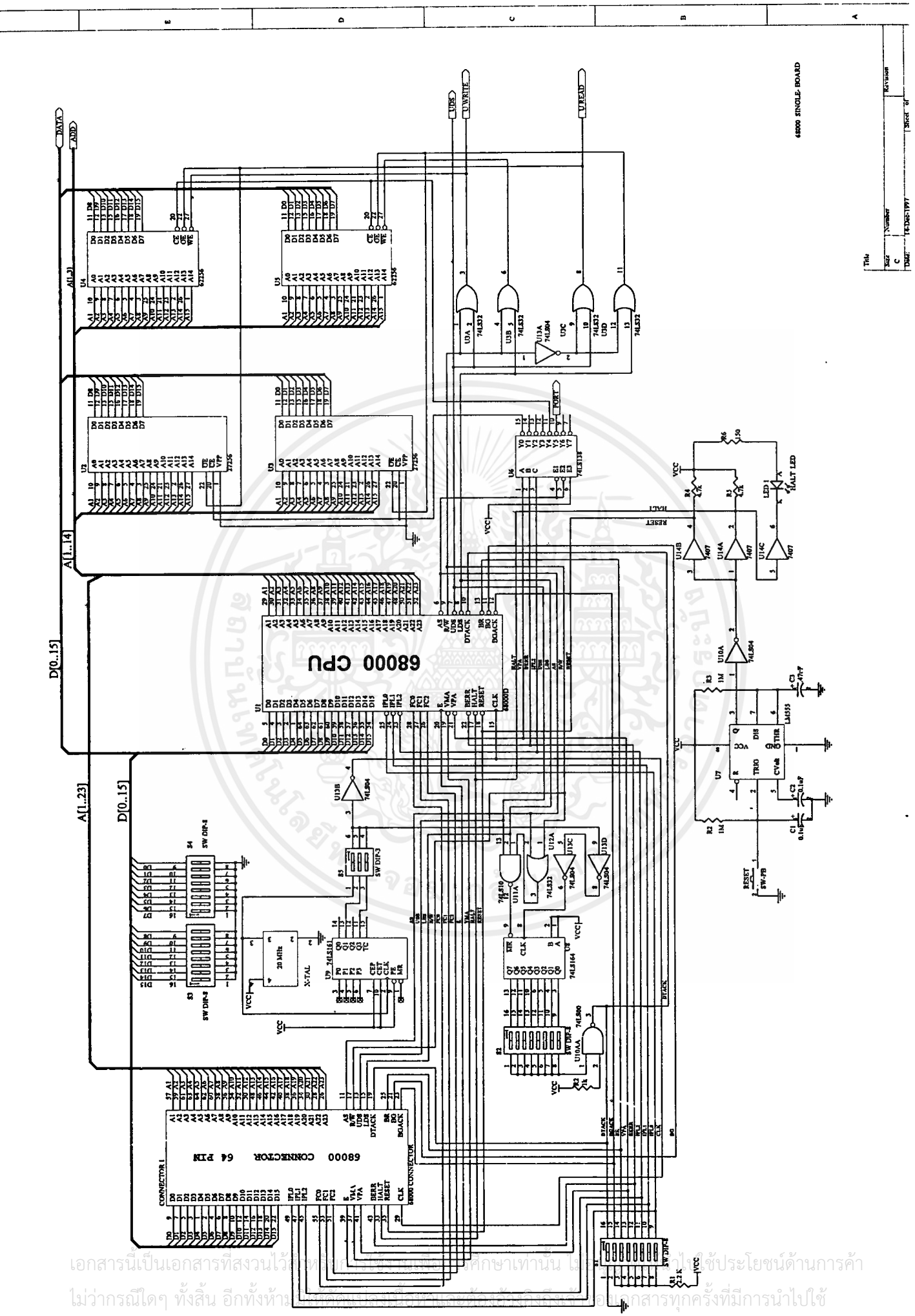
```
                DFB    " ASDFGHJKL;''"
                DFB    " \ZXCVBNM,./ "
                DFB    "*^ c          "
                DFB    "ns789-456+1230."

        ORG   0800000H
MSB:            DFB    00H,00H
LSB:            DFB    00H,00H

BUFFER_BYTE_ONE: DFB    00H,00H
BUFFER_BYTE_TWO: DFB    00H,00H

                ORG   0800100H
USER_PROGRAM      DFB       "        "
                  DFB       "        "
                  DFB       "        "
              DFB    "   "
                  DFB       "        "
                  END
```

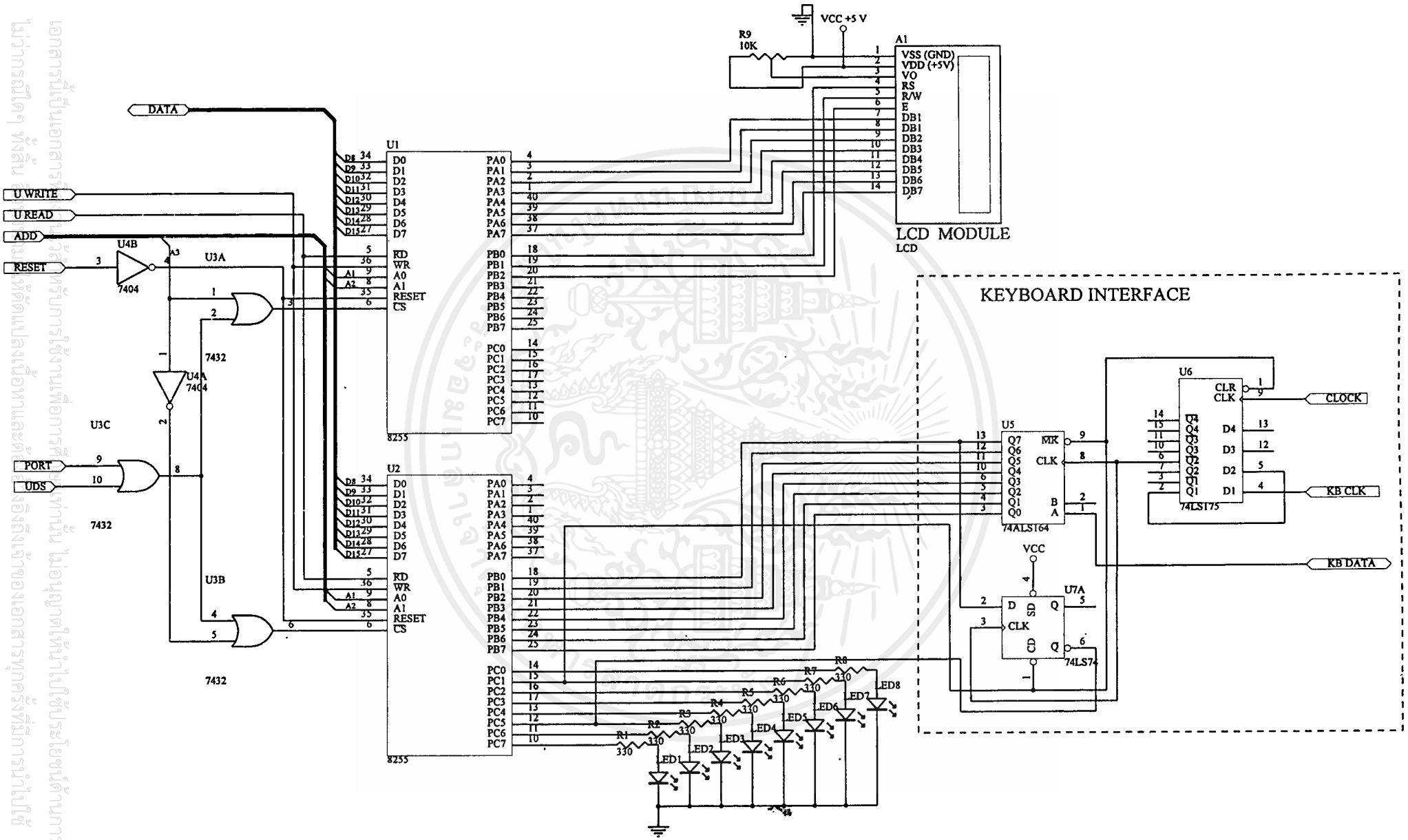
ภาคผนวก ข
ลายวงจรและแผ่นวงจรพิมพ์










VCC +5 V
R9
10K
A1
VSS (GND)
VDD (+5V)
VO
RS
R/W
E
DB1
DB1
DB2
DB3
DB4
DB5
DB6
DB7
LCD MODULE
LCD
KEYBOARD INTERFACE
DATA
U WRITE
U READ
ADD
RESET
U4B
7404
U3A
7432
U4A
7404
U3C
PORT
UDS
7432
U3B
7432
U1
U2
8255
8255
RD
WR
A0
A1
RESET
CS
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
U5
74ALS164
U6
74LS175
CLOCK
KB CLK
KB DATA
VCC
U7A
74LS74
R1
R2
R3
R4
R5
R6
R7
R8
330
LED1
LED2
LED3
LED4
LED5
LED6
LED7
LED8
126
Title
Size
B
Number
Revision
Date:
14-Dec-1997
Sheet of

ภาคผนวก ค
คุณสมบัติของอุปกรณ์

MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

MC68000

Technical Summary
16-/32-Bit Microprocessor

This document contains both a summary of the MC68000 as well as a detailed set of parametrics. The purpose is twofold - to provide an introduction to the MC68000 and support for the sophisticated user. For detailed information on the MC68000, refer to the *MC68000 16- 32-Bit Microprossor Advance Information Data Sheet.*

The MC68000 is the first implementation of the M68000 16 32 microprocessor architecture. The MC68000 has a 16-bit data bus and 24-bit address bus while the full architecture provides for 32-bit address and data buses. It is completely code-compatible with the MC68008 8-bit data bus implementation of the M68000 and is upward code compatible to the MC68010 MC68012 virtual extensions and the MC68020 32-bit implementation of the architecture. Any user-mode programs written using the MC68000 instruction set will run unchanged on the MC68008, MC68010, MC68020. This is possible because the user programming model is identical for all five processors and the instruction sets are proper sub-sets of the complete architecture. Resources available to the MC68000 user consists of the following:

- 17 32-Bit Data and Address Registers
- 16 Megabyte Direct Addressing Range
- 56 Powerful Instruction Types
- Operations on Five Main Data Types
- Memory Mapped I O
- 14 Addressing Modes

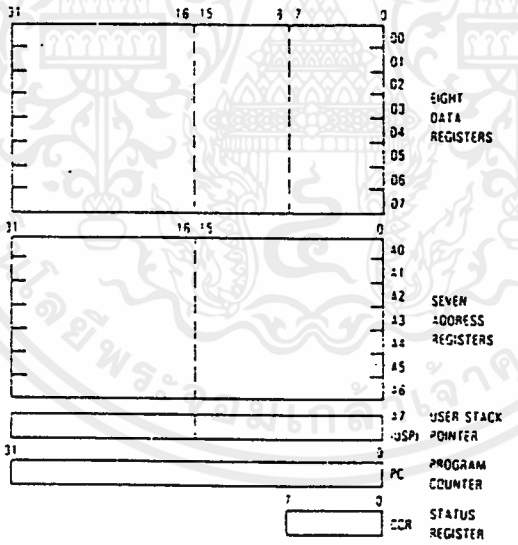


Figure 1. User Programming Model

INTRODUCTION

As shown in the user programming model (Figure 1), the MC68000 offers 16 32-bit registers and a 32-bit program counter. The first eight registers (D0-D7) are used as data registers for byte (8-bit), word (16-bit), and long word (32-bit) operations. The second set of seven registers (A0-A6) and the user stack pointer (USP) may be used as software stack pointers and base address registers. In addition, the registers may be used for word and long word operations. All of the 16 registers may be used as index registers.

In supervisor mode, the upper byte of the status register and the supervisor stack pointer (SSP) are also available to the programmer. These registers are shown in Figure 2.

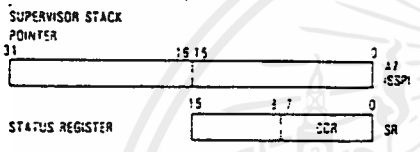


Figure 2. Supervisor Programming Model Supplement

The status register (Figure 3) contains the interrupt mask (eight levels available) as well as the condition codes: extend (X), negative (N), zero (Z), overflow (V), and carry (C). Additional status bits indicate that the processor is in a trace (T) mode and in a supervisor (S) or user state.

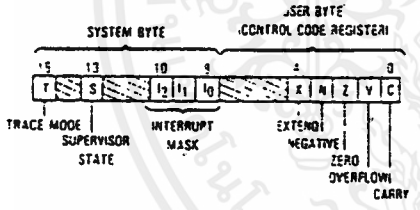


Figure 3. Status Register

DATA TYPES AND ADDRESSING MODES

Five basic data types are supported. These data types are:

- Bits
- BCD Digits (4-Bits)
- Bytes (8 Bits)
- Words (16 Bits)
- Long Words (32 Bits)

In addition, operations on other data types such as memory addresses, status word data, etc. are provided in the instruction set.

The 14 addressing modes, shown in Table 1, include six basic types:

- Register Direct
- Register Indirect
- Absolute
- Program Counter Relative
- Immediate
- Implied

Included in the register indirect addressing modes is a capability to do postincrementing, predecrementing, offsetting, and indexing. The program counter relative mode can also be modified via indexing and offsetting.

Table 1. Addressing Modes

Addressing Modes	Syntax
Register Direct Addressing	
Data Register Direct	Dn
Address Register Direct	An
Absolute Data Addressing	
Absolute Short	xxx.W
Absolute Long	xxx.L
Program Counter Relative Addressing	
Relative with Offset	d_{16}(PC)
Relative with Index Offset	d_8(PC,Xn)
Register Indirect Addressing	
Register Indirect	(An)
Postincrement Register Indirect	(An)+
Predecrement Register Indirect	-(An)
Register Indirect with Offset	d_{16}(An)
Indexed Register Indirect with Offset	d_8(An,Xn)
Immediate Data Addressing	
Immediate	#xxx
Quick Immediate	#1-#8
Implied Addressing	
Implied Register	SR USP SP PC

NOTES:
Dn = Data Register
An = Address Register
Xn = Address of Data Register used as Index Register
SR = Status Register
PC = Program Counter
SP = Stack Pointer
USP = User Stack Pointer
( ) = Effective Address
d_8 = 8-Bit Offset (Displacement)
D_{16} = 16-Bit Offset (Displacement)
#xxx = Immediate Data

INSTRUCTION SET OVERVIEW

The MC68000 instruction set is shown in Table 2. Some additional instructions are variations, or sub-sets, of these and they appear in Table 3. Special emphasis has been given to the instruction set's support of structured high-level languages to facilitate ease of programming. Each instruction, with few exceptions, operates on bytes, words, and long words and most instructions can use any of the 14 addressing modes. Combining instruction types, data types, and addressing modes, over 1000 useful instructions are provided. These instructions include signed and unsigned, multiply and divide, "quick" arithmetic operations, BCD arithmetic, and expanded operations (through traps).

Table 2. Instruction Set Summary

Mnemonic	Description
ABCD	Add Decimal With Extend
ADD	Add
AND	Logical AND
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
Bcc	Branch Conditionally
BCHG	Bit Test and Change
BCLR	Bit Test and Clear
BRA	Branch Always
BSET	Bit Test and Set
BSR	Branch to Subroutine
BTST	Bit Test
CHK	Check Register Against Bounds
CLR	Clear Operand
CMP	Compare
DBcc	Test Condition, Decrement and Branch
DIVS	Signed Divide
DIVU	Unsigned Divide
EOR	Exclusive OR
EXG	Exchange Registers
EXT	Sign Extend
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link Stack
LSL	Logical Shift Left
LSR	Logical Shift Right
MOVE	Move
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NOP	No Operation
NOT	One's Complement
OR	Logical OR
PEA	Push Effective Address
RESET	Reset External Devices.
ROL	Rotate Left without Extend
ROR	Rotate Right without Extend
ROXL	Rotate Left with Extend
ROXR	Rotate Right with Extend
RTE	Return from Exception
RTR	Return and Restore
RTS	Return from Subroutine
SBCD	Subtract Decimal with Extend
Scc	Set Conditional
STOP	Stop
SUB	Subtract
SWAP	Swap Data Register Halves
TAS	Test and Set Operand
TRAP	Trap
TRAPV	Trap on Overflow
TST	Test
UNLK	Unlink

Table 3. Variations of Instruction Types

Instruction Type	Variation	Description
ADD	ADD	Add
	ADDA	Add Address
	ADDQ	Add Quick
	ADDI	Add Immediate
	ADDX	Add with Extend
AND	AND	Logical AND
	ANDI	AND Immediate
	ANDI to CCR	AND Immediate to Condition Codes
	ANDI to SR	AND Immediate to Status Register
CMP	CMP	Compare
	CMPA	Compare Address
	CMPM	Compare Memory
	CMPI	Compare Immediate
EOR	EOR	Exclusive OR
	EORI	Exclusive OR Immediate
	EORI to CCR	Exclusive OR Immediate to Condition Codes
	EORI to SR	Exclusive OR Immediate to Status Register
MOVE	MOVE	Move
	MOVEA	Move Address
	MOVEM	Move Multiple Registers
	MOVEP	Move Peripheral Data
	MOVEQ	Move Quick
	MOVE from SR	Move from Status Register
	MOVE to SR	Move to Status Register
	MOVE to CCR	Move to Condition Codes
	MOVE USP	Move User Stack Pointer
NEG	NEG	Negate
	NEGX	Negate with Extend
OR	OR	Logical OR
	ORI	OR Immediate
	ORI to CCR	OR Immediate to Condition Codes
	ORI to SR	OR Immediate to Status Register
SUB	SUB	Subtract
	SUBA	Subtract Address
	SUBI	Subtract Immediate
	SUBQ	Subtract Quick
	SUBX	Subtract with Extend

SIGNAL DESCRIPTION

The input and output signals are illustrated functionally in Figure 4 and are described in the following paragraphs.

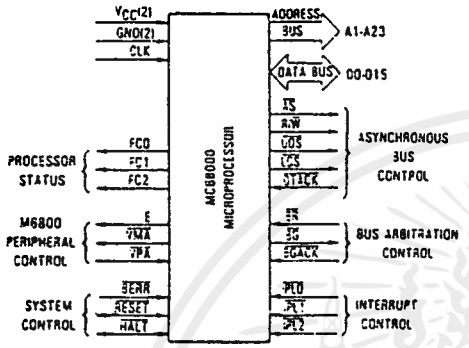


Figure 4. Input and Output Signals

ADDRESS BUS (A1 THROUGH A23)

This 32-bit, unidirectional, three-state bus is capable of addressing 16 megabytes of data. It provides the address for bus operation during all cycles except interrupt cycles. During interrupt cycles, address lines A1, A2, and A3 provide information about what level interrupt is being serviced while address lines A4 through A23 are set to a logic high.

DATA BUS (D0 THROUGH D15)

This 16-bit, bidirectional, three-state bus is the general purpose data path. It can transfer and accept data in either word or byte lenght. During an interrupt acknowledge cycle, the external device supplies the vector number on data lines D0-D7.

ASYNCHRONOUS BUS CONTROL

Asynchronous data transfers are handled using the following control signals: address strobe, read/write, upper and lower data strobes, and data transfer acknowledge. These signals are explained in the following paragraphs.

Address Strobe ($\overline{AS}$)

This signal indicates that there is a valid address on the address bus.

Read/Write ($R/\overline{W}$)

This signal defines the data bus transfer as a read or write cycle. The $R/\overline{W}$ signal also works in conjunction with the data strobes as explained in the following paragraph.

Upper and Lower Data Strobe ($\overline{UDS}$, $\overline{LDS}$)

These signals control the flow of data on the data bus, as shown in Table 4. When the $R/\overline{W}$ line is high, the processor will read from the data bus as indicated. When the $R/\overline{W}$ line is low, the processor will write to the data bus as shown.

Data Transfer Acknowledge ($\overline{DTACK}$)

This input indicates that the data transfer is completed. When the processor recognizes $\overline{DTACK}$ during a read cycle, data is latched and the bus cycle terminated. When $\overline{DTACK}$ is recognized during a write cycle, the bus cycle is terminated.

BUS ARBITRATION CONTROL

The three signals, bus request, bus grant, and bus grant acknowledge, form a bus arbitration circuit to determine which device will be the bus master device.

Bus Request ($\overline{BR}$)

This input is wire-ORed with all other devices that could be bus masters. This input indicates to the processor that some other device desires to become the bus master.

Bus Grant ($\overline{BG}$)

This output indicates to all other potential bus master devices that the processor will release bus control at the end of the current bus cycle.

Bus Grant Acknowledge ($\overline{BGACK}$)

This input indicates that some other device has become the bus master. This signal should not be asserted until the following four conditions are met:

1. a bus grant has been received,
2. address strobe is inactive which indicates that the microprocessor is not using the bus,
3. data transfer acknowledge is inactive which indicates that neither memory nor peripherals are using the bus, and
4. bus grant acknowledge is inactive which indicates that no other device is still claiming bus mastership.

Table 4. Data Strobe Control of Data Bus

$\overline{UDS}$	$\overline{LDS}$	$R/\overline{W}$	D8-D15	D0-D7
High	High	—	No Valid Data	No Valid Data
Low	Low	High	Valid Data Bits 8-15	Valid Data Bits 0-7
High	Low	High	No Valid Data	Valid Data Bits 0-7
Low	High	High	Valid Data Bits 8-15	No Valid Data
Low	Low	Low	Valid Data Bits 8-15	Valid Data Bits 0-7
High	Low	Low	Valid Data Bits 0-7*	Valid Data Bits 0-7
Low	High	Low	Valid Data Bits 8-15	Valid Data Bits 8-15*

*These conditions are a result of current implementation and may not appear on future devices.

INTERRUPT CONTROL ($\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$)

These input pins indicate the encoded priority level of the device requesting an interrupt. Level seven is the highest priority while level zero indicates that no interrupts are requested. Level seven cannot be masked. The least significant bit is given in $\overline{IPL0}$ and the most significant bit is contained in $\overline{IPL2}$. These lines must remain stable until the processor signals interrupt acknowledge (FC0-FC2 are all high) to insure that the interrupt is recognized.

SYSTEM CONTROL

The system control inputs are used to either reset or halt the processor and to indicate to the processor that bus errors have occurred. The three system control inputs are explained in the following paragraphs.

Bus Error ($\overline{BERR}$)

This input informs the processor that there is a problem with the cycle currently being executed. Problems may be a result of:

1. nonresponding devices,
2. interrupt vector number acquisition failure,
3. illegal access request as determined by a memory management unit, or
4. other application dependent errors.

The bus error signal interacts with the halt signal to determine if the current bus cycle should be re-executed or if exception processing should be performed.

Reset ($\overline{RESET}$)

This bidirectional signal line acts to reset (start a system initialization sequence) the processor in response to an external reset signal. An internally generated reset (result of a $\overline{RESET}$ instruction) causes all external devices to be reset and the internal state of the processor is not affected. A total system reset (processor and external devices) is the result of external $\overline{HALT}$ and $\overline{RESET}$ signals applied at the same time.

Halt ($\overline{HALT}$)

When this bidirectional line is driven by an external device, it will cause the processor to stop at the completion of the current bus cycle. When the processor has been halted using this input, all control signals are inactive and all three-state lines are put in their high-impedance state.

When the processor has stopped executing instructions, such as in a double bus fault condition, the $\overline{HALT}$ line is driven by the processor to indicate to external devices that the processor has stopped.

M6800 PERIPHERAL CONTROL

These control signals are used to allow the interfacing of synchronous M6800 peripheral devices with the asynchronous MC68000. These signals are explained in the following paragraphs.

Enable (E)

This signal is the standard enable signal common to all M6800 type peripheral devices. The period for this output is ten MC68000 clock periods (six clocks low, four clocks high). Enable is generated by an internal ring counter which may come up in any state (i.e., at power on, it is impossible to guarantee phase relationship of E to CLK). E is a free-running clock and runs regardless of the state of the bus on the MPU.

Valid Peripheral Address ($\overline{VPA}$)

This input indicates that the device or region addressed is an M68000 Family device or region addressed is an M68000 Family device and that data transfer should be synchronized with the enable (E) signal. This input also indicates that the processor should use automatic vectoring for an interrupt during an IACK cycle.

Valid Memory Address ($\overline{VMA}$)

This output is used to indicate to M68000 peripheral devices that there is a valid address on the address bus and the processor is synchronized to enable. This signal only responds to a valid peripheral address ($\overline{VPA}$) input which indicates that the peripheral is an M68000 Family device.

PROCESSOR STATUS (FC0, FC1, FC2)

These function code outputs indicate the state (user or supervisor) and the cycle type currently being executed, as shown in Table 5. The information indicated by the function code outputs is valid whenever address strobe ($\overline{AS}$) is active.

Table 5. Function Code Outputs

Function Code Output			Cycle Time
FC2	FC1	FC0	
Low	Low	Low	(Undefined, Reserved)
Low	Low	High	User Data
Low	High	Low	User Program
Low	High	High	Undefined, Reserved)
High	Low	Low	(Undefined, Reserved)
High	Low	High	Supervisor Data
High	High	Low	Supervisor Program
High	High	High	Interrupt Acknowledge

CLOCK (CLK)

The clock input is a TTL-compatible signal that is internally buffered for development of the internal clocks needed by the processor. The clock input should not be gated off at any time and the clock signal must conform to minimum and maximum pulse width times.

DATA TRANSFER OPERATIONS

Transfer of data between devices involves the following leads:

1. address bus A1 through A23,
2. data bus D0 through D15, and
3. control signals.

The address and data buses are separate parallel buses used to transfer data using an asynchronous bus structure. In all cycles, the bus master assumes responsibility for deskewing all signals it issues at both the start and end of a cycle. In addition, the bus master is responsible for deskewing the acknowledge and data signals from the slave device.

The following paragraphs explain the read, write, and read-modify-write cycles. The indivisible read-modify-write cycle is the method used by the MC68000 for interlocked multiprocessor communications.

READ CYCLE

During a read cycle, the processor receives data from the memory or a peripheral device. The processor reads bytes of data in all cases. If the instruction specifies a word (or double word) operation, the processor reads both upper and lower bytes simultaneously by asserting both upper and lower data strobes. When the instruction specifies byte operation, the processor uses an internal A0 bit to determine which byte to read and then issues the data strobe required for that byte. For byte operations, when the A0 bit equals zero, the upper data strobe is issued. When the A0 bit equals one, the lower data strobe is issued. When the data is received, the processor correctly positions it internally.

WRITE CYCLE

During a write cycle, the processor sends data to either the memory or a peripheral device. The processor writes bytes of data in all cases. If the instruction specifies a word operation, the processor writes both bytes. When the instruction specifies a byte operation, the processor uses an internal A0 bit to determine which byte to write and then issues the data strobe required for that byte. For byte operations, when the A0 bit equals zero, the upper data strobe is issued. When the A0 bit equals one, the lower data strobe is issued.

READ-MODIFY-WRITE CYCLE

The read-modify-write cycle performs a read, modifies the data in the arithmetic-logic unit, and writes the data back to the same address. In the MC68000, this cycle is indivisible in that the address strobe is asserted throughout the entire cycle. The test and set (TAS) instruction uses this cycle to provide meaningful communication between processors in a multiple processor environment. This instruction is the only instruction that uses the read-modify-write cycles and since the test and set instruction only operates on bytes, all read-modify-write cycles are byte operations.

PROCESSING STATES

The MC68000 is always in one of three processing states: normal, exception, or halted.

NORMAL PROCESSING

The normal processing state is that associated with instruction execution; the memory references are to fetch instructions and operands, and to store results. A special case of normal state is the stopped state which the processor enters when a stop instruction is executed. In this state, no further references are made.

EXCEPTION PROCESSING

The exception processing state is associated with interrupts, trap instructions, tracing, and other exception conditions. The exception may be internally generated by an instruction or by an unusual condition arising during the execution of an instruction. Externally, exception processing can be forced by an interrupt, by a bus error, or by a reset. Exception processing is designed to provide an efficient context switch so that the processor may handle unusual conditions.

HALTED PROCESSING

The halted processing state is an indication of catastrophic hardware failure. For example, if during the exception processing of a bus error another bus error occurs, the processor assumes that the system is unusable and halts. Only an external reset can restart a halted processor. Note that a processor in the stopped state is not in the halted state, nor vice versa.

INTERFACE WITH M6800 PERIPHERALS

Motorola's extensive line of M6800 peripherals are directly compatible with the MC68000. Some of these devices that are particularly useful are:

MC6821	Peripheral Interface Adapter
MC6840	Programmable Timer Module
MC6843	Floppy Disk Controller
MC6845	CRT Controller
MC6850	Asynchronous Communications Interface Adapter
MC5854	Advanced Data Link Controller

To interface the synchronous M6800 peripherals with the asynchronous MC68000, the processor modifies its bus cycle to meet the M6800 cycle requirements whenever an M6800 device address is detected. This is possible since both the processors use memory mapped I/O.

ELECTRICAL SPECIFICATIONS

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	−0.3 to −7.0	V
Input Voltage	V_{in}	−0.3 to −7.0	V
Operating Temperature Range MC68000 MC68000C	T_A	T_L to T_H 0 to 70 −40 to 85	°C
Storage Temperature	T_{stg}	−55 to 150	°C

The device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, normal precautions should be taken to avoid application of voltages higher than maximum-rated voltages to these high-impedance circuits. Tying unused inputs to the appropriate logic voltage level (e.g., either GND or V_{CC}) enhances reliability of operation.

THERMAL CHARACTERISTICS

Characteristic	Symbol	Value	Symbol	Value	Rating
Thermal Resistance (Still Air) Ceramic, Type L,LC Ceramic, Type R,RC Plastic, Type P Plastic, Type FN	θ_{JA}	 30 33 30 45	θ_{JC}	 15* 15 15* 25*	°C/W

*Estimated

DC ELECTRICAL CHARACTERISTICS (V_{CC} = 5.0 Vdc ± 5%; GND = 0 Vdc; $T_A = T_L$ to T_H)

Characteristic		Symbol	Min	Max	Unit
Input High Voltage		V_{IH}	2.0	V_{CC}	V
Input Low Voltage		V_{IL}	GND −0.3	0.8	V
Input Leakage Current @ 5.25 V	$\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$, CLK, $\overline{IPL0}$-$\overline{IPL2}$, $\overline{VPA}$ $\overline{HALT}$, $\overline{RESET}$	I_{IN}	— —	2.5 20	μA
Three-State (Off State) Input Current @ 2.4 V/0.4 V	$\overline{AS}$, A1-A23, D0-D15, FC0-FC2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$	I_{TSI}	—	20	μA
Output High Voltage (I_{OH} = −400 μA) (I_{OH} = −400 μA)	E* E, $\overline{AS}$, A1-A23, $\overline{BG}$, D0-D15, FC0-FC2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$	V_{OH}	V_{CC} − 0.75 2.4	— 2.4	V
Output Low Voltage (I_{OL} = 1.6 mA) (I_{OL} = 3.2 mA) (I_{OL} = 5.0 mA) (I_{OL} = 5.3 mA)	 $\overline{HALT}$ A1-A23, $\overline{BG}$, FC0-FC2 $\overline{RESET}$ E, $\overline{AS}$, D0-D15, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$	V_{OL}	 — — — —	 0.5 0.5 0.5 0.5	V
Power Dissipation (see POWER CONSIDERATIONS)		P_D***	—	—	W
Capacitance (V_{in} = 0 V, T_A = 25°C, Frequency = 1 MHz)**		C_{in}	—	20.0	pF
Load Capacitance	$\overline{HALT}$ All Others	C_L	— —	70 130	pF

*With external pullup resistor of 1.1 kΩ.
**Capacitance is periodically sampled rather than 100% tested.
***During normal operation instantaneous V_{CC} current requirements may be as high as 1.5 A.

POWER CONSIDERATIONS

The average die-junction temperature, T_J, in °C can be obtained from:

$$T_J = T_A + (P_D \cdot \theta_{JA}) \quad (1)$$

where:

T_A = Ambient Temperature, °C
θ_{JA} = Package Thermal Resistance, Junction-to-Ambient, °C/W
$P_D = P_{INT} + P_{I/O}$
$P_{INT} = I_{CC} \times V_{CC}$, Watts — Chip Internal Power
$P_{I/O}$ = Power Dissipation on Input and Output Pins — User Determined

For most applications $P_{I/O} < P_{INT}$ and can be neglected.

An appropriate relationship between P_D and T_J (if $P_{I/O}$ is neglected) is:

$$P_D = K \div (T_J + 273\ °C) \quad (2)$$

Solving equations (1) and (2) for K gives:

$$K = P_D \cdot (T_A + 273°C) + \theta_{JA} \cdot P_D^2 \quad (3)$$

where K is a constant pertaining to the particular part. K can be determined from equation (3) by measuring P_D (at thermal equilibrium) for a known T_A. Using this value of K, the values of P_D and T_J can be obtained by solving equations (1) and (2) iteratively for any value of T_A.

The curve shown in Figure 11-1 gives the graphic solution to the above equations for the specified power dissipation of 1.5 watts over the ambient temperature range of −55 °C to 125 °C using a maximum θ_{JA} of 45 °C/W. Ambient temperature is that of the still air surrounding the device. Lower values of θ_{JA} cause the curve to shift downward slightly; for instance, for θ_{JA} of 40 °/W, the curve is just below 1.4 watts at 25 °C.

The total thermal resistance of a package (θ_{JA}) can be separated into two components, θ_{JC} and θ_{CA}, representing the barrier to heat flow from the semiconductor junction to the package (case) surface (θ_{JC}) and from the case to the outside ambient air (θ_{CA}). These terms are related by the equation:

$$\theta_{JA} = \theta_{JC} + \theta_{CA} \quad (4)$$

θ_{JC} is device related and cannot be influenced by the user. However, θ_{CA} is user dependent and can be minimized by such thermal management techniques as heat sinks, ambient air cooling, and thermal convection. Thus, good thermal management on the part of the user can significantly reduce θ_{CA} so that θ_{JA} approximately equals θ_{JC}. Substitution of θ_{JC} for θ_{JA} in equation 1 results in a lower semiconductor junction temperature.

Table 6 summarizes maximum power dissipation and average junction temperature for the curve drawn in Figure 5, using the minimum and maximum values of ambient temperature for different packages and substituting θ_{JC} for θ_{JA} (assuming good thermal management). Table 7 provides the maximum power dissipation and average junction temperature assuming that no thermal management is applied (i.e., still air).

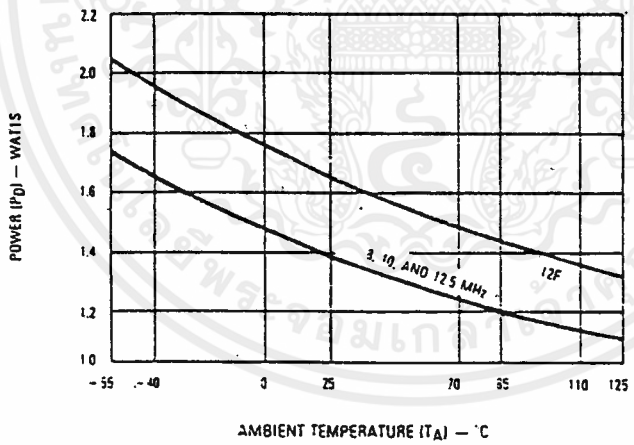


Figure 5. MC68000 Power Dissipation (P_D) vs Ambient Temperature (T_A)

Table 6. Power Dissipation and Junction Temperature vs Temperature ($\theta_{JC} = \theta_{JA}$)

Package	T_A Range	θ_{JC} (°C/W)	P_D (W) @ T_A Min.	T_J (°C) @ T_A Min.	P_D (W) @ T_A Max.	T_J (°C) @ T_A Max.
L/LC	0°C to 70°C	15	1.5	23	1.2	88
	−40°C to 85°C	15	1.7	−14	1.2	103
	0°C to 85°C	15	1.5	23	1.2	103
P	0°C to 70°C	15	1.5	23	1.2	88
R/RC	0°C to 70°C	15	1.5	23	1.2	88
	−40°C to 85°C	15	1.7	−14	1.2	103
	0°C to 85°C	15	1.5	23	1.2	103
FN	0°C to 70°C	25	1.5	38	1.2	101

NOTE: Table does not include values for the MC68000 12F.

Table 7. Power Dissipation and Junction Temperature vs Temperature ($\theta_{JC} = \theta_{JC}$)

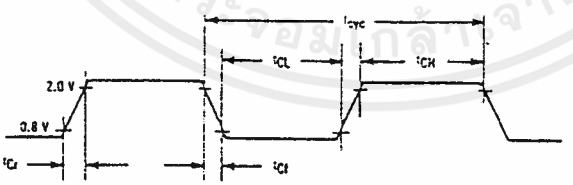
Package	T_A Range	θ_{JA} (°C/W)	P_D (W) @ T_A Min.	T_J (°C) @ T_A Min.	P_D (W) @ T_A Max.	T_J (°C) @ T_A Max.
L/LC	0°C to 70°C	30	1.5	23	1.2	88
	−40°C to 85°C	30	1.7	−14	1.2	103
	0°C to 85°C	30	1.5	23	1.2	103
P	0°C to 70°C	30	1.5	23	1.2	88
R/RC	0°C to 70°C	33	1.5	23	1.2	88
	−40°C to 85°C	33	1.7	−14	1.2	103
	0°C to 85°C	33	1.5	23	1.2	103
FN	0°C to 70°C	40	1.5	38	1.2	101

NOTE: Table does not include values for the MC68000 12F.

AC ELECTRICAL SPECIFICATIONS — CLOCK TIMING (see Figure 6)

Num.	Characteristic	Symbol	8 MHz*		10 MHz*		12.5 MHz*		16.67 MHz "12F"		Unit
			Min	Max	Min	Max	Min	Max	Min	Max	
	Frequency of Operation	f	4.0	8.0	4.0	10.0	4.0	12.5	8.0	16.7	MHz
1	Cycle Time	t_{cyc}	125	250	100	250	80	250	60	125	ns
2,3	Clock Pulse Width (Measured from 1.5 V to 1.5 V for 12F)	t_{CL}	55	125	45	125	35	125	27	62.5	ns
		t_{CH}	55	125	45	125	35	125	27	62.5	
4,5	Clock Rise and Fall Times	t_{Cr}	—	10	—	10	—	5	—	5	ns
		t_{Cf}	—	10	—	10	—	5	—	5	

*These specifications represent an improvement over previously published specifications for the 8-, 10-, and 12.5-MHz MC68000 and are valid only for product bearing date codes of 8827 and later.



NOTE: Timing measurements are referenced to and from a low voltage of 0.8 volt and high a voltage of 2.0 volts, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8 volt and 2.0 volts.

Figure 6. Clock Input Timing

DMC164

• Display Format(16chacter ×4line) • Display Fonts(5×8dots) • Driving Method(1/16D)

ABSOLUTE MAXIMUM RATINGS

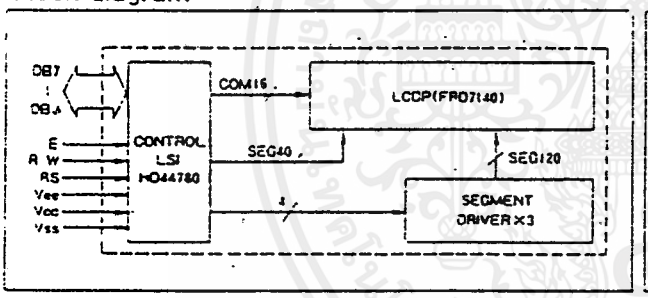
Item	Symbol	Test Condition	Standard Value min.	typ.	max.	Unit
Power Supply Voltage for Logic	Vcc−Vss	—	0	—	6.5	V
Power Supply Voltage for LCD Drive	Vcc−Vee	—	0	—	6.0	V
Input Voltage	Vi	—	Vss		Vcc	V
Operating Temperature	Ta	—	0		+50	°C
Storage Temperature	Tstg	—	−20		+70	°C

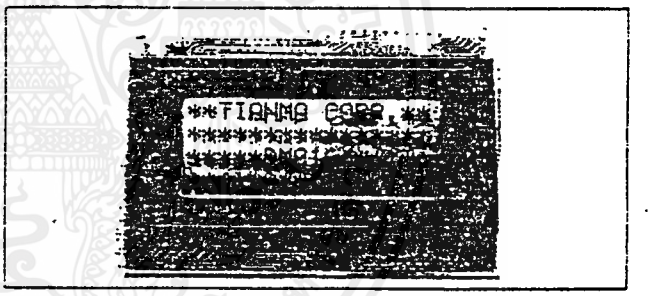
ELECTRICAL CHARACTERISTICS

Item	Symbol	Test Condition	Standard Value min.	typ.	max.	Unit
Input "High" Voltage	VIH	—	2.2	—	Vcc	V
Input "Low" Voltage	VIL	—	−0.3	—	0.6	V
Output "High" Voltage	VOH	−IOH=0.205mA	2.4			V
Output "Low" Voltage	VOL	IOH=1.2mA			0.4	V
Power Supply Current	ICC	Vcc=5.0V		2.4	4.0	mA

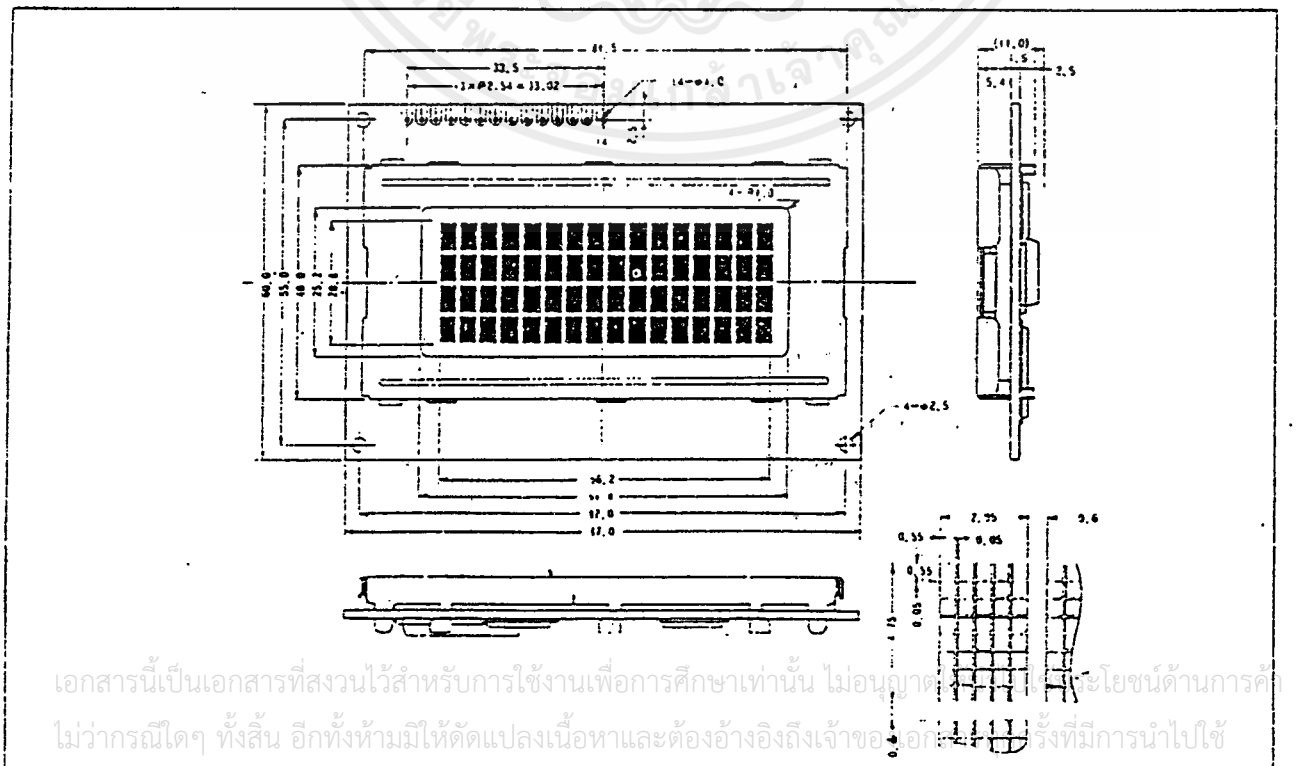
*Vcc=5.0V±5%, Ta=25°C

Block diagram





External dimensions / Display pattern



บรรณานุกรม

Alan D.Wilcox . **68000 Microcomputer Svstems** : Designing and Trouble shooting . Englewood Cliffs , New Jersey : Pentice Hall , 1987 .

James L.Antonakos . **The 68000 Microprocessor** : Hardware and software Principles and Application . Third Edition Englewood Cliffs , New Jersey : Pentice Hall , 1996 .

Walter a.Triebel and Avtar Singh . **The 68000 and 68020 Microprocessor** : Architecture , Software and Interfacting Techniques . Englewood Cliffs : Prentice Hall , 1986 .

วุฒิศักดิ์ สุริยะมณี และ ดนัย ตันสิทธิแพทย์. ซิงเกิ้ลบอร์ด คอมพิวเตอร์, ปริญญานิพนธ์วิศวกรรมศาสตร์บัณฑิต ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าธนบุรี, 2537.

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นาย คนิระ วงศ์วีแก้ว
วันเดือนปีเกิด	6 มกราคม พ.ศ.2519
สถานที่เกิด	จังหวัด สุราษฎร์ธานี
ภูมิลำเนาเดิม	100/2 หมู่ 5 ถนน สุราษฎร์-ปากน้ำ ตำบล บางกุ้ง อำเภอ เมือง จังหวัด สุราษฎร์ธานี
ที่อยู่ปัจจุบัน	362 ซอย โชติวัฒน์ ถนน ประชาชื่น เขต บางซื่อ กรุงเทพฯ 10800
โทรศัพท์	5860061
ประวัติการศึกษา	
ประถมศึกษา	โรงเรียนปกรณ์วิทยา
มัธยมศึกษา	โรงเรียนสุราษฎร์ธานี จังหวัด สุราษฎร์ธานี
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.4ปี)	สถาบันเทคโนโลยีราชมงคล วิทยาเขตนนทบุรี
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	-
คติพจน์	ตนเป็นที่พึงแห่งตน

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นาย ฉัตรชัย ไชยาภินันท์
วันเดือนปีเกิด	29 กันยายน พ.ศ.2518
สถานที่เกิด	จังหวัด ชลบุรี
ภูมิลำเนาเดิม	46 หมู่ 5 ตำบล พลิ้ว อำเภอ แหลมสิงห์ จังหวัด จันทบุรี 22190
ที่อยู่ปัจจุบัน	80/42 หมู่บ้าน เสนานิเวศน์ 1 ถนน พหลโยธิน แขวง จรเข้บัว เขต ลาดพร้าว กรุงเทพฯ
โทรศัพท์	5781891

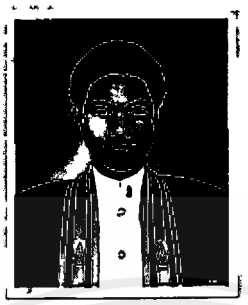
ประวัติการศึกษา

ประถมศึกษา	โรงเรียนอนุบาลจันทบุรี
มัธยมศึกษา	โรงเรียนแหลมสิงห์วิทยาคม
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคจันทบุรี
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	วิทยาลัยเทคนิคจันทบุรี
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	-
คติพจน์	ทำวันนี้ให้ดีที่สุด

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นาย ภักดี พรอุดมเดช
วันเดือนปีเกิด	10 มิถุนายน 2519
สถานที่เกิด	จังหวัด ระยอง
ภูมิลำเนาเดิม	104/14 ถนน เทศวานิช อำเภอ เมือง จังหวัด ระยอง 21000
ที่อยู่ปัจจุบัน	179/36 หมู่ มณสินี ถนน อ่อนนุช เขตลาดกระบัง กรุงเทพฯ 10520
โทรศัพท์	01-6108506, 038-611489, 7390838
ประวัติการศึกษา	
ประถมศึกษา	โรงเรียนกวงฮั้ว (ระยอง)
มัธยมศึกษา	โรงเรียนระยองวิทยาคม
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคระยอง
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	วิทยาลัยเทคนิคระยอง
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
ผลงานที่ได้รับรางวัล	รางวัลรองชนะเลิศอันดับสอง การประกวดสิ่งประดิษฐ์คนรุ่นใหม่ระดับประเทศ ปี 2537 กรมอาชีวศึกษา “เครื่องจ่ายยาด้วยคอมพิวเตอร์”
ทุนการศึกษา	-
คติพจน์	ปัญหาทุกอย่างแก้ไขได้

ประวัติผู้แต่ง



ชื่อผู้ทำปริญญานิพนธ์	นายวิฑิต ทักษิณาวาริน
วันเดือนปีเกิด	17 มกราคม พ.ศ.2518
สถานที่เกิด	กรุงเทพมหานคร
ภูมิลำเนาเดิม	กรุงเทพมหานคร
ที่อยู่ปัจจุบัน	265/8 ซอย สายน้ำทิพย์ ถนน สุขุมวิท
	แขวงคลองตัน เขตคลองเตย กรุงเทพฯ 10110
โทรศัพท์	2582309
ประวัติการศึกษา	
ประถมศึกษา	โรงเรียนสายน้ำทิพย์
มัธยมศึกษา	โรงเรียนปทุมคงคา
ประกาศนียบัตรวิชาชีพ (ปวช.)	โรงเรียนเกษมโปลีเทคนิค
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	โรงเรียนเทคโนโลยีสยาม
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์
	ภาควิชาครุศาสตร์วิศวกรรม
	คณะครุศาสตร์อุตสาหกรรม
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	-
คติพจน์	จงทำวันนี้ให้ดีกว่าเมื่อวานนี้