68HC11 อิน - เซอร์กิต อีมูเลเตอร์

68HC11 In - circuit Emulator

โดย

นาย ศักรินทร์ กล้าหาญ

นาย สุกิจ แก้วกลิ่นจันทร์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขา วิศวกรรมการวัดคุมทางอุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์ ปีการศึกษา 2540

ภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม

สาขาวิชา วิศวกรรมการวัดคุมทางอุตสาหกรรม

คณะ วิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง 68HC11 In - circuit Emulator

ผู้จัดทำ

1. นาย ศักรินทร์ กล้าหาญ เลขประจำตัว 38012120
2. นาย สุกิจ แก้วกลิ่นจันทร์ เลขประจำตัว 38012127

..... อาจารย์ที่ปรึกษา

(อาจารย์ ทรงชัย วีระทวีมาศ)

หัวข้อปริญญานิพนธ์	68HC11 อิน - เซอร์กิต อีมูเลเตอร์
นักศึกษา	นาย ศักรินทร์ กล้าหาญ 38012120
	นาย สุกิจ แก้วกลิ่นจันทร์ 38012127
อาจารย์ที่ปรึกษา	อาจารย์ ทรงชัย วีระทวีมาศ
ระดับการศึกษา	วิศวกรรมศาสตรบัณฑิต
	สาขาวิศวกรรมการวัดคุมทางอุตสาหกรรม
ปีการศึกษา	พ.ศ. 2540

บทคัดย่อ

ในการสร้างวงจรการใช้งานใด ๆ ที่มีไมโครโปรเซสเซอร์เป็นอุปกรณ์หลักในการทำงาน อุปกรณ์หรือเครื่องมือที่ใช้ตรวจสอบการทำงานของไมโครโปรเซสเซอร์ที่มีประสิทธิภาพอย่างหนึ่งคือ อุปกรณ์ที่เรียกว่า อิน - เซอร์กิต อีมูเลเตอร์ ซึ่งอุปกรณ์ชนิดนี้มีความสามารถในการขอดูหรือแก้ไขรีจิสเตอร์ หน่วยความจำ รวมถึงพอร์ทต่าง ๆ ได้อย่างมีประสิทธิภาพ ซึ่งจะทำให้ผู้ใช้งานรู้ถึงความสามารถในการทำงานของไมโครโปรเซสเซอร์ได้เป็นอย่างดี ว่าในขณะใดขณะหนึ่งของการทำงานนั้น มีผลลัพธ์เป็นเช่นใด เป็นไปตามที่ผู้ใช้ต้องการหรือไม่ และทำให้การเรียนรู้เพื่อใช้งานไมโครโปรเซสเซอร์ทำได้สะดวกและง่ายขึ้น

Thesis : 68HC11 In - circuit Emulator

Students : Sakarin Klaharn 38012120

Sukit Kaewklinjan 38012127

Advisor : Songchai Weerathaweemas

Education Level : Bachelor of industrial instrument engineering

Education Year : 1997

Abstract

Microprocessor is the principal part in creating any circuit. Consequenly , the equipment or tool for checking the potential of Microprocessor is important. Such tool is called " In - circuit Emulator ". Which its capacity in checking and adjusting register, memory , including any port , user can know efficiency level of Microprocessor whether it work well. This eventually make it easier in learning Microprocessor for usage and ensure convenience and user can gain.

สารบัญ

หน้า

สารบัญรูปประกอบ

สารบัญตาราง

บทที่ 1
บทนำ

1.1 บทนำ

เมื่อมีการออกแบบวงจรการใช้งานในระบบควบคุมใดๆ โดยมีไมโครคอนโทรลเลอร์เป็นอุปกรณ์หลักในการทำงาน ปัญหาที่พบได้บ่อยคือ ผลลัพธ์ของการใช้งานที่เกิดขึ้นไม่เป็นไปตามต้องการของผู้ใช้งาน ซึ่งความผิดพลาดของวงจรที่สร้างขึ้นอาจแบ่งได้ 2 สาเหตุใหญ่ ๆ คือ ความผิดพลาดทางฮาร์ดแวร์ และความผิดพลาดทางซอฟต์แวร์ ในกรณีของความผิดพลาดที่เกิดจากฮาร์ดแวร์ ผู้ใช้งานอาจใช้เครื่องมือวัดทางด้านอิเล็กทรอนิกส์ เช่น ออสซิลโลสโคปหรือลอจิกโพรบตรวจหาความผิดพลาด แต่ถ้าเป็นความผิดพลาดในกรณีของซอฟต์แวร์นั้น ผู้ใช้จะไม่สามารถทราบได้เลยว่าโปรแกรมทำงานผิดพลาดช่วงใด เนื่องจากในกรณีที่ไมโครคอนโทรลเลอร์กำลังประมวลผลอยู่หรือทำงานอยู่นั้น จะไม่สามารถทราบได้ว่าไมโครคอนโทรลเลอร์ทำงานตามคำสั่งใด เกิดผลลัพธ์เป็นไปตามต้องการมากน้อยเพียงใด วิธีการหนึ่งซึ่งอาจใช้ได้ผลดีในการตรวจสอบการทำงานคือ การใช้โปรแกรมการจำลองการทำงานของไมโครคอนโทรลเลอร์เบอร์นั้นๆ แต่เมื่อพิจารณาอีกด้านหนึ่งนั้นจะพบว่าโปรแกรมการจำลองการทำงานนั้น เป็นเพียงการให้เครื่องคอมพิวเตอร์เลียนแบบการทำงานเท่านั้น ซึ่งในทางปฏิบัติแล้ว อาจไม่ได้ผลตามที่โปรแกรมนั้นแสดงผลก็ได้ ทั้งนี้เนื่องจากสาเหตุแตกต่างกันออกไป กรณีเช่น ผู้ใช้ต้องการเขียนข้อมูลลงไปยังหน่วยความจำ แต่ไม่สามารถเขียนข้อมูลลงไปได้ เนื่องจากหน่วยความจำตรงส่วนนั้นเกิดขัดข้องขึ้น หรือในกรณีที่ผู้ใช้ต้องการอ่านค่าหน่วยความจำภายนอกแต่ไม่สามารถอ่านค่าข้อมูลเข้ามาได้ เนื่องจากสายสัญญาณในการอ่านเกิดความผิดพลาดขึ้น เป็นต้น ในกรณีเช่นนี้จะเห็นได้ว่า เทคนิคการเลียนแบบการทำงานโดยใช้โปรแกรมจำลองการทำงานนั้น ไม่สามารถรับรู้ได้ว่าปัญหาที่เกิดขึ้นจริง ๆ ในทางปฏิบัติเป็นเช่นใด

ทางแก้ปัญหาในกรณีเช่นนี้จึงจำเป็นต้องให้ทางฮาร์ดแวร์ทำงานจริง ๆ และหาสิ่งหนึ่งสิ่งใดมาติดตามการทำงานของไมโครคอนโทรลเลอร์ตัวนั้น ๆ อุปกรณ์ที่ช่วยในการทำงานหรือช่วยในการพัฒนาเหล่านี้ที่ปรากฏตามท้องตลาดโดยทั่วไปได้แก่ อีพรอมอีมูเลเตอร์ (Eprom Emulator) เครื่องไมโครคอมพิวเตอร์แบบแผ่นวงจรพิมพ์เดี่ยว (Single board microcomputer) และอิน - เซอร์กิต อีมูเลเตอร์ (In - circuit Emulator) ซึ่งความสามารถข้อจำกัดในการใช้งานในแต่ละชนิดก็จะแตกต่างกันออกไป ซึ่งสามารถเปรียบเทียบความสามารถในการทำงานได้ดังตารางที่ 1.1 ซึ่งจะพบว่าอุปกรณ์อิน - เซอร์กิต อีมูเลเตอร์นั้น สามารถใช้งานได้มีประสิทธิภาพมากที่สุด และเสมือนหนึ่งว่า ผู้ใช้ได้เข้าไปในโลกปิดของตัวไมโครคอนโทรลเลอร์ได้ ทั้งนี้ก็เนื่องมาจากว่าขณะเวลาใดเวลาหนึ่ง ผู้ใช้สามารถทราบได้ว่าไมโครคอนโทรลเลอร์นั้น

กำลังปฏิบัติงานตามคำสั่งใด เกิดข้อผิดพลาดอะไร หรือค่าผลลัพธ์ที่ได้นั้นผิดพลาดไปมากน้อยเพียงใด แนวทางการแก้ไขและพัฒนาจึงเป็นไปอย่างถูกต้องและมีประสิทธิภาพยิ่งขึ้น และข้อดีของอุปกรณ์อิน - เซอร์กิต อีมูเลเตอร์ที่เห็นได้ชัดเจน คือความสามารถในการเปิดโอกาสให้ผู้ใช้งานสามารถใช้หน่วยความจำได้เต็มที่ 64 กิโลไบต์ (ในกรณีที่ไมโครโปรเซสเซอร์มีขนาด 8 บิต) ทำให้ผู้ใช้งานมีความยืดหยุ่นในการทำงานได้ชัดเจนกว่าเครื่องมือ หรืออุปกรณ์อื่น

ความสามารถ	Single board	Eprom Emulator	In-circuit Emulator
1. การติดต่อหน่วยความจำได้ทั้ง 64 กิโลไบต์	ไม่ได้	ได้	ได้
2. ติดต่อกับพอร์ทได้ทั้งหมด	ไม่ได้	ได้	ได้
3. การเขียนและอ่านหน่วยความจำโดยตรง	ได้	ไม่ได้	ได้
4. การอ่านและตั้งค่าของรีจีสเตอร์	ได้	ไม่ได้	ได้
5. การใช้งานของสัญญาณควบคุม	ไม่ได้	ไม่ได้	ได้
6. กำหนดจุด break	ได้	ไม่ได้	ได้
7. การทำ Assembler	ไม่ได้	ได้	ได้
8. การทำ Disassembler	ไม่ได้	ไม่ได้	ได้

ตารางที่ 1.1 ตารางเปรียบเทียบความสามารถของอุปกรณ์ช่วยพัฒนาไมโครโปรเซสเซอร์ [1]

1.2 วัตถุประสงค์และขอบเขตของโครงงาน

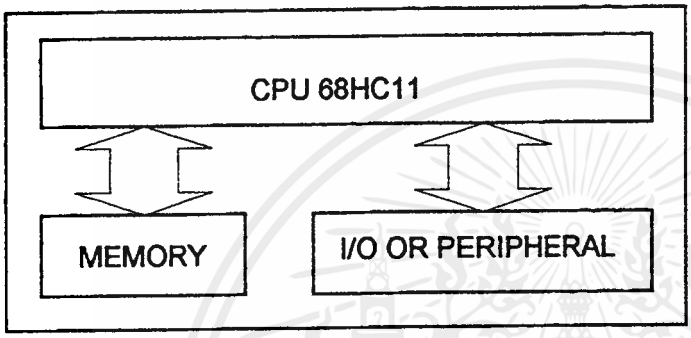
เนื่องด้วยการใช้งานไมโครคอนโทรลเลอร์ตระกูล 68HC11 มีการใช้งานกันอย่างกว้างขวางมากยิ่งขึ้นในปัจจุบัน แต่ยังขาดอุปกรณ์ที่ช่วยในการพัฒนาอยู่มาก ซึ่งอุปกรณ์ที่ช่วยในการพัฒนาระบบไมโครโปรเซสเซอร์ตั้งแต่ขนาดเล็กจนถึงขนาดใหญ่ที่มีประสิทธิภาพนี้คือ อิน - เซอร์กิต อีมูเลเตอร์ ดังนั้นโครงงานนี้จึงจัดทำอิน - เซอร์กิต อีมูเลเตอร์ สำหรับไมโครคอนโทรลเลอร์ 68HC11 ให้สามารถนำไปใช้งานได้ในระดับหนึ่ง

---

[1] วิทยานิพนธ์ " ระบบพัฒนาไมโครโปรเซสเซอร์ระดับบอร์ด " นส. สุภนันณ์ หิรัญอภิสุทธิกุล จุฬาลงกรณ์มหาวิทยาลัย

1.3 หลักการทำงานของอิน - เซอร์กิต อีมูเลเตอร์โดยทั่วไป

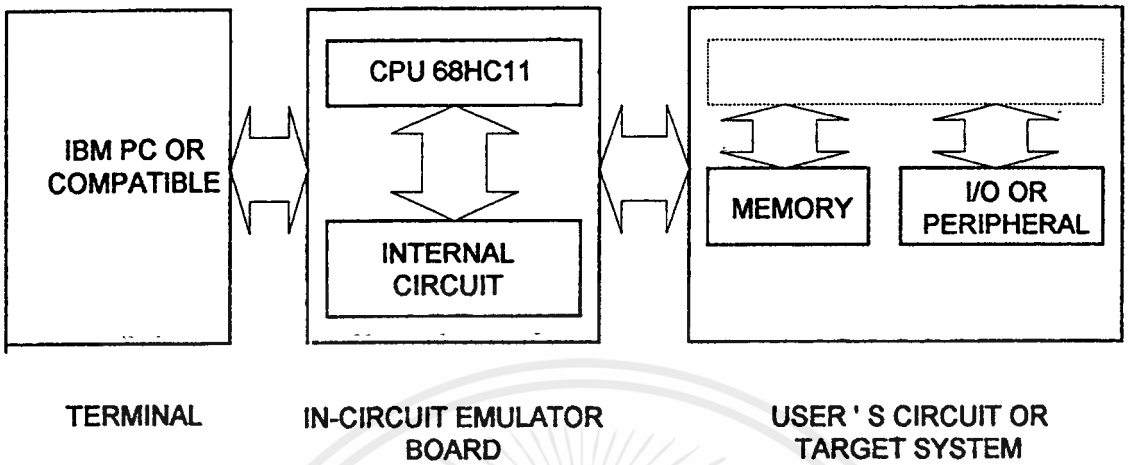
จากที่กล่าวมาข้างต้น เมื่อระบบใดระบบหนึ่งมีการสร้างหรือมีการพัฒนาการใช้งานเกิดขึ้น โดยส่วนมากจะพบปัญหาที่ตามมาคือ การทำงานไม่ได้ผลลัพธ์ตามต้องการ ทางแก้ปัญหาจากที่กล่าวมาแล้ว คือการใช้อุปกรณ์ที่เรียกว่า อิน - เซอร์กิต อีมูเลเตอร์เข้าตรวจสอบหาข้อผิดพลาด (bug) ของการทำงาน โดยยกตัวอย่างการใช้งานได้ดังนี้ว่า สมมุติผู้ใช้สร้างแผ่นวงจรขึ้นมาหนึ่งชุด โดยมีไมโครคอนโทรลเลอร์เป็นอุปกรณ์หลักในการทำงาน



รูปที่ 1.1 บอร์ดที่ใช้พัฒนา

แต่ถ้าผลลัพธ์ที่เกิดขึ้นไม่เป็นไปตามต้องการ ผู้ใช้สามารถถอดเอา CHIP CPU จากระบบที่พัฒนาออก (Target Processor) และนำเอาสายต่อจากอิน - เซอร์กิต อีมูเลเตอร์ไปเสียบแทนที่ จากนั้นผู้ใช้สามารถตรวจสอบการทำงานได้จากอุปกรณ์เทอร์มินอล (ในที่นี้คือเครื่องคอมพิวเตอร์ส่วนบุคคล) ว่าการทำงานนั้นเป็นไปตามต้องการหรือไม่อย่างไร

ซึ่งโดยทั่วไประบบของผู้ใช้ที่จะนำมาตรวจสอบ จะต้องเป็นหน่วยความจำแบบ RAM เนื่องจากโปรแกรมต้นฉบับจะถูกส่งผ่านมายังเทอร์มินอล ทำให้ผู้ใช้สามารถเก็บโปรแกรมใช้งานเข้าไปยัง RAM ได้ ยกเว้นในกรณีที่ผู้ใช้มีโปรแกรมวางตัวใน ROM หรือ EPROM อยู่แล้ว และไม่ต้องการแก้ไขโปรแกรมต้นฉบับอีก แต่ต้องการติดตามการทำงานของโปรแกรมเดิมเท่านั้น ดังนั้นจากรูปที่ 1.2 เมื่อระบบที่ใช้พัฒนาถูกเลียนแบบการทำงานทางฮาร์ดแวร์ (emulate) จึงเปรียบเสมือนชุดอิน - เซอร์กิต อีมูเลเตอร์เป็นซีพียูเบอร์นั้น ๆ และการสั่งงานจะถูกส่งผ่านโดยคอมพิวเตอร์ซึ่งจะเป็นเทอร์มินอล ซึ่งการติดตามการทำงานนั้นผู้ใช้จะต้องทราบถึงรายละเอียดของชุดที่จะพัฒนาด้วย เช่น มีการจัดวางตำแหน่งหน่วยความจำที่ตำแหน่งใด การติดตามควรหยุดดูในช่วงใด เป็นต้น



รูปที่ 1.2 การเชื่อมต่อบอร์ดที่ใช้พัฒนาเข้ากับอิน-เซอร์กิต อีมูเลเตอร์

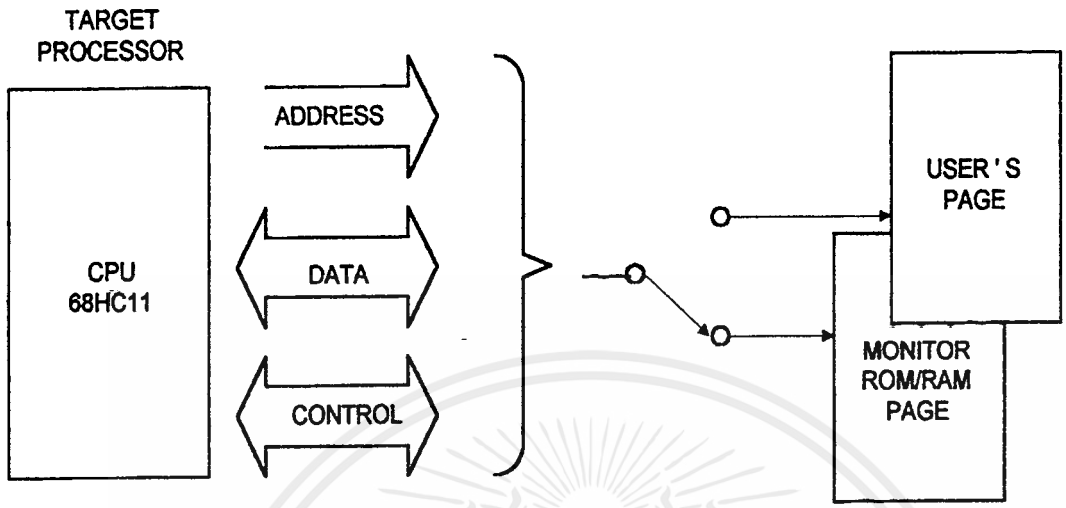
1.4 โครงสร้างของ 68HC11 อิน - เซอร์กิต อีมูเลเตอร์

โครงสร้างของอิน - เซอร์กิต อีมูเลเตอร์ โดยทั่วไปแบ่งได้ดังนี้

1. แบบซีพียูเดี่ยว (Single Process) ซึ่งซีพียูของอิน - เซอร์กิต อีมูเลเตอร์จะทำหน้าที่เป็นทั้งซีพียูของระบบที่จะพัฒนา และทำหน้าที่ของการควบคุมการทำงานของอิน - เซอร์กิต อีมูเลเตอร์ด้วย

2. แบบหลายซีพียู (Multi Process) ในแบบนี้ซีพียูตัวหนึ่งจะเป็นซีพียูของระบบที่พัฒนา (Target System) ส่วนอีกตัวหนึ่งจะทำหน้าที่ควบคุมการทำงานของอิน - เซอร์กิต อีมูเลเตอร์

ในการจัดทำปริญญานิพนธ์ฉบับนี้ยึดแนวทางการทำงานแบบซีพียูเดี่ยว เนื่องจากมีความสะดวกในการออกแบบและใช้งานมากกว่า รวมถึงการพัฒนาต่อไปจะสามารถทำได้สะดวก



รูปที่ 1.3 การสลับการทำงานของซีพียูระหว่างหน่วยความจำสองเพจ

ปัญหาที่พบโดยทั่วไปในการใช้งานอิน - เซอร์กิต อีมูเลเตอร์คือ การที่ต้องแบ่งเนื้อที่หน่วยความจำในการเก็บโปรแกรมมอนิเตอร์ โดยส่วนมากจะใช้พื้นที่ประมาณ 8 กิโลไบต์ ซึ่งหมายความว่าถ้าเป็นไมโครโปรเซสเซอร์ขนาด 8 บิต จะเหลือหน่วยความจำให้ผู้ใช้ได้เพียง 56 กิโลไบต์ ซึ่งโดยปกติในระบบควบคุมที่ใช้ไมโครโปรเซสเซอร์ หรือไมโครคอนโทรลเลอร์เป็นอุปกรณ์หลักในการทำงาน จะไม่เพียงพอต่อความต้องการของผู้ใช้ ทำให้เกิดความยุ่งยากในการใช้งาน จึงก่อให้เกิดแนวความคิดที่จะเปิดโอกาส ให้ผู้ใช้งานได้ใช้หน่วยความจำได้เต็มความสามารถ หรืออีกกรณีหนึ่งคือ ผู้ใช้สามารถออกแบบหน่วยความจำไว้ในช่วงตำแหน่งใดก็ได้ โดยใช้หลักการคือ การวางหน่วยความจำที่จะใช้เก็บโปรแกรมควบคุมระบบวางซ้อนกันอยู่กับหน่วยความจำของผู้ใช้ โดยมีการสลับการทำงานระหว่างเพจของผู้ใช้ และเพจของระบบ ซึ่งจะทำให้อิน - เซอร์กิต อีมูเลเตอร์นี้ ไม่เข้าไปยุ่งเกี่ยว หรือขอใช้พื้นที่หน่วยความจำของผู้ใช้เลย สามารถแสดงการทำงานอย่างง่ายได้ดังรูปที่ 1.3

จากรูปที่ 1.3 ซีพียูจะมีการสลับการทำงาน ระหว่างระบบของผู้ใช้และระบบของมอนิเตอร์ ซึ่งจากการทำงานดังกล่าว จะสามารถทำให้ซีพียูติดต่อหน่วยความจำหรือ I/O ได้ 64 กิโลไบต์ เต็มทั้ง 2 ชุด ซึ่งในเวลาขณะใดขณะหนึ่งที่ซีพียูจะต้องทำตัวอยู่ในโหมดใดโหมด

หนึ่ง คือโหมดอีมูเลชัน และโหมดอินเทอร์โรเกชัน [2] โหมดอีมูเลชันคือ ช่วงเวลาที่ซีพียูทำงานตามปกติของโปรแกรมผู้ใช้ ส่วนเวลาอื่นๆ คือเป็นโหมดอินเทอร์โรเกชัน ซึ่งถ้าเปรียบเทียบจากหลักการทำงานดังรูปที่ 1.3 ช่วงเวลาที่ซีพียูทำงานของเพจผู้ใช้ ถือว่าซีพียูทำงานในโหมดอีมูเลชัน และเมื่อซีพียูทำงานที่โปรแกรมมอนิเตอร์คือช่วงที่ซีพียูจะอยู่ในโหมดอินเทอร์โรเกชันนั่นเอง

1.5 รูปแบบของฟังก์ชันการทำงานที่ใช้ในโครงงานนี้

ฟังก์ชัน	รูปแบบ	ความหมาย
B	B XXXX	Set break point
D	D XXXX	Display memory
G	G XXXX	Run (free step)
R	R	View register
	R r nn	Set register
T	T	Single step (trace)

ตารางที่ 1.2 รูปแบบการทำงานแต่ละฟังก์ชัน

โดยที่ XXXX คือ start address
nn คือ data
r คือ ชื่อรีจีสเตอร์

ในการทำความเข้าใจในการทำงานของ 68HC11 ชนิดไซเคิลต่อไซเคิล (cycle by cycle) เป็นสิ่งจำเป็นในการออกแบบการทำงานของอิน - เซอร์กิต อีมูเลเตอร์ เนื่องจากจะช่วยให้เข้าใจถึงธรรมชาติในการทำงานของไมโครคอนโทรลเลอร์ตัวนี้ ในการทำงานไซเคิลต่อไซเคิลจะอธิบายตามโปรแกรมตัวอย่างต่อไปนี้

```
                ORG     $F800
F800 B7 C1 5A   STAA    $C15A
```

[2] วิทยานิพนธ์ " อิน - เซอร์กิต อีมูเลเตอร์สำหรับ Z80 และ 8085 " นาย เสกสันณ์ วัฒนะโชติ จุฬาลงกรณ์มหาวิทยาลัย

จากโปรแกรมเป็นการโอนย้ายข้อมูลจากแอกคิวมูเลเตอร์ A ไปยังหน่วยความจำแอดเดรส $C15A โดยข้อมูลใน A คือ $75

ในไซเคิลแรกของการทำงาน 68HC11 จะเฟตช์ออปโค้ด $B7 จากอีพรอมออกมา ในช่วงนี้ขา R/W จะเป็น "1" เพื่อแสดงว่ากำลังอ่านข้อมูลจากหน่วยความจำ จากนั้น 68HC11 ก็จะทำการเฟตช์ออปโค้ดคำสั่งที่เหลือ โดยจะเลื่อนแอดเดรสที่ทำการอ่านโดยอัตโนมัติ จนกระทั่งถึงไซเคิลสุดท้าย จะต้องเขียนข้อมูลลงในหน่วยความจำแรม ขา R/W จะเป็น "0" จากนั้น 68HC11 ก็จะเขียนข้อมูลลงแรม โดยค่าดังกล่าวคือ $75 (ตามที่กำหนดตั้งแต่แรก)

ในไซเคิลที่ 2 และ 3 จะเป็นการเฟตช์ออปโค้ดของแอดเดรสไบต์สูงและต่ำ และสุดท้ายในไซเคิลสุดท้ายจะเฟตช์โอเปอแรนด์

ในตารางที่ 1.3 เป็นการสรุปการทำงานไซเคิลต่อไซเคิลของ 68 HC11 เมื่อกระทำคำสั่ง STAA ส่วนในตารางที่ 1.4 แสดงผลที่เกิดขึ้นจริงในบัสแอดเดรสและข้อมูลเมื่อกระทำคำสั่ง STAA

ไซเคิลรวม	ไซเคิลที่	บัสแอดเดรส	ลอจิกที่ขา R/W	บัสข้อมูล
4	1	ออปโค้ดแอดเดรส	1	ออปโค้ด
	2	ออปโค้ดแอดเดรส + 1	1	โอเปอแรนด์ของแอดเดรสไบต์สูง
	3	ออปโค้ดแอดเดรส + 2	1	โอเปอแรนด์ของแอดเดรสไบต์ต่ำ
	4	โอเปอแรนด์แอดเดรส	0	ข้อมูลแอกคิวมูเลเตอร์

ตารางที่ 1.3 สรุปการทำงานไซเคิลต่อไซเคิลของ 68HC11 เมื่อเอ็กซีคิวต์คำสั่ง STAA

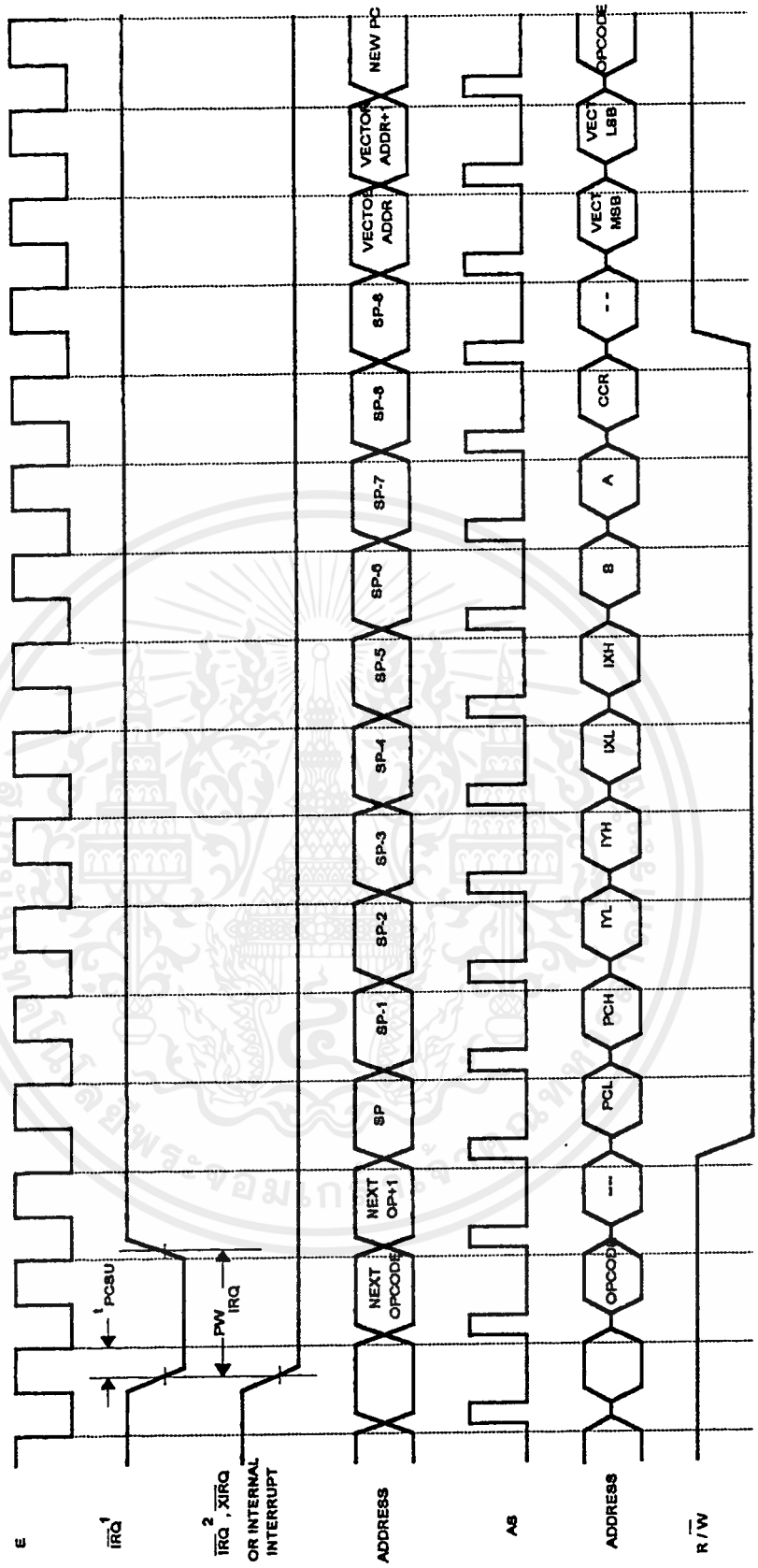
ไซเคิลรวม	ไซเคิลที่	ข้อมูลบนบัสแอดเดรส	ลอจิกที่ขา R / W	บัสข้อมูล
4	1	$F800	1	$B7
	2	$F801	1	$C1
	3	$F802	1	$5A
	4	$C15A	0	$75

ตารางที่ 1.4 ผลการทำงานไซเคิลต่อไซเคิลของ 68HC11 เมื่อเอ็กซีคิวต์คำสั่ง STAA โดยข้อมูลในแอกคิวมูเลเตอร์ A เท่ากับ $75

ในการออกแบบอิน - เซอร์กิต อีมูเลเตอร์ ของไมโครโปรเซสเซอร์โดยทั่วไป สิ่งที่สำคัญอีกอย่างคือ การเข้าใจการทำงานของการอินเตอร์รัปท์ของไมโครโปรเซสเซอร์เบอร์นั้น ๆ ซึ่งในซีพียู 68HC11 มีช่วงเวลาขบวนการอินเตอร์รัปท์แสดงดังรูปที่ 1.4

จะเห็นได้ว่าหลังจากเกิดการอินเตอร์รัปท์แล้ว ซีพียูจะเก็บค่าสำคัญต่าง ๆ ลงในสแต็ก ก่อนที่จะเฟตช์เวกเตอร์อินเตอร์รัปท์ ซึ่งค่าเวกเตอร์อินเตอร์รัปท์นี้ไมโครคอนโทรลเลอร์ 68HC11 ยอมให้ผู้ใช้งานสามารถกำหนดค่าเวกเตอร์อินเตอร์รัปท์ได้เอง จากนั้นซีพียูจะกลับจากการบริการอินเตอร์รัปท์ได้ก็ต่อเมื่อซีพียูเอ็กซีคิวต์คำสั่ง RTI (Return interrupt) ในโปรแกรมย่อยการบริการอินเตอร์รัปท์นั้น ๆ

จากรูปที่ 1.4 สัญญาณ IRQ^1 และ IRQ^2 มีความแตกต่างกัน คือแบบแรกจะตรวจจับสัญญาณขอบขาลง ส่วนแบบที่สองจะตรวจสอบสัญญาณในระดับ (level sensitive) ซึ่งในแบบที่สองนั้นมีสัญญาณที่เข้ามาจะเป็นเพียงสัญญาณที่แคบ (Impulse) แต่สามารถตรวจวัดระดับสัญญาณได้ก็สามารถทำให้ซีพียูทำงานตามรูปที่ 1.4 ได้เช่นกัน ซึ่งทั้งสองหัวข้อที่ได้กล่าวมาคือการทำงานแบบไซเคิลต่อไซเคิล และการอินเตอร์รัปท์จะใช้เป็นส่วนที่สำคัญในการออกแบบอิน - เซอร์กิต อีมูเลเตอร์สำหรับ 68HC11 ที่ใช้ในโครงงานนี้ดังจะได้กล่าวถึงในบทต่อไป

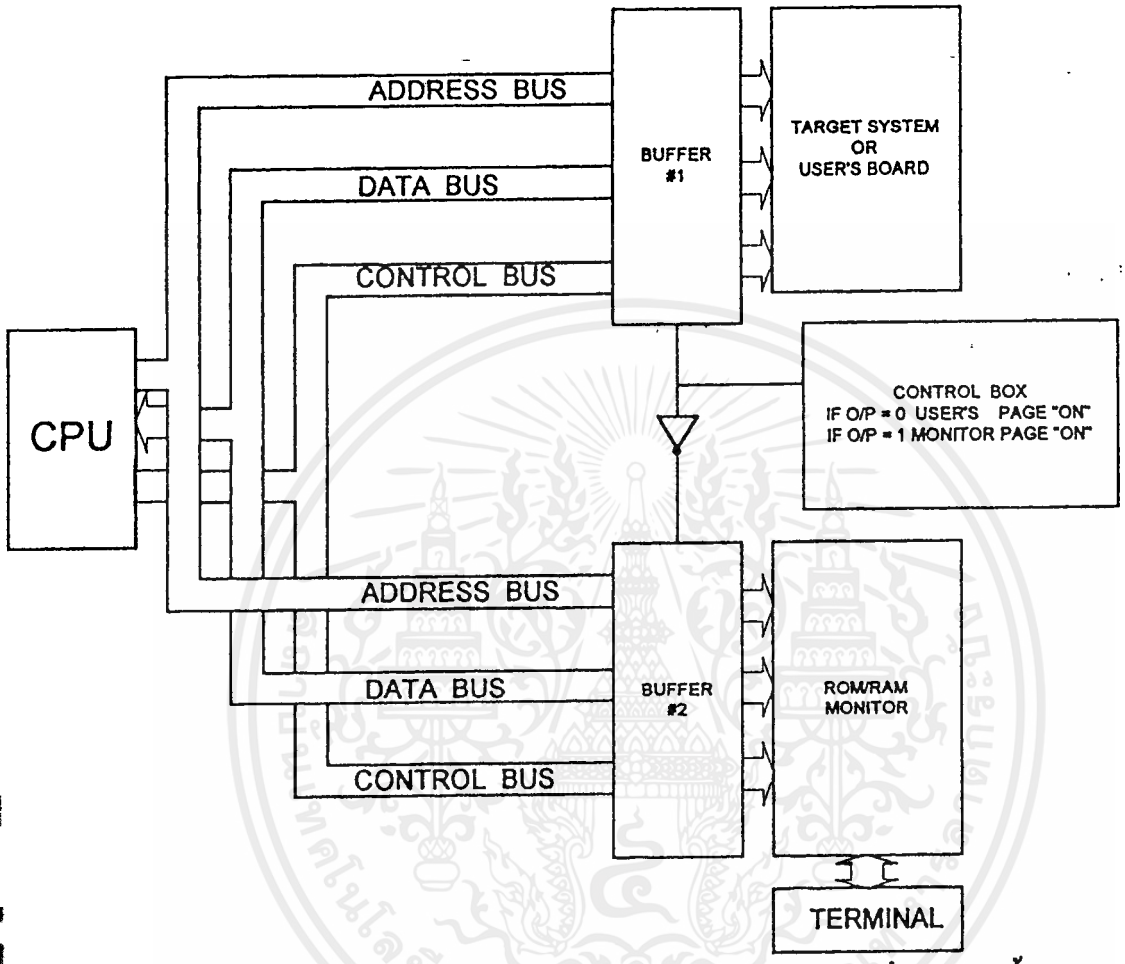


รูปที่ 1.4 แสดงการทำงานของสัญญาณอินเตอร์รัปท์ของ 68HC11

บทที่ 2

หลักการออกแบบระบบ

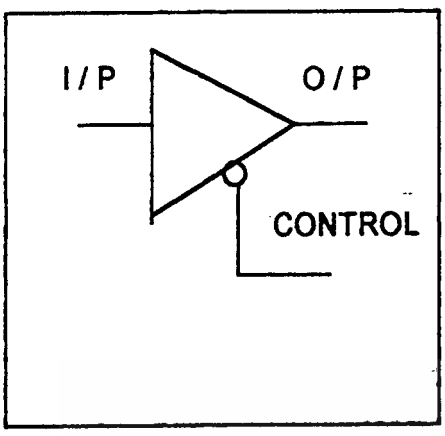
2.1 หลักการออกแบบระบบเบื้องต้น



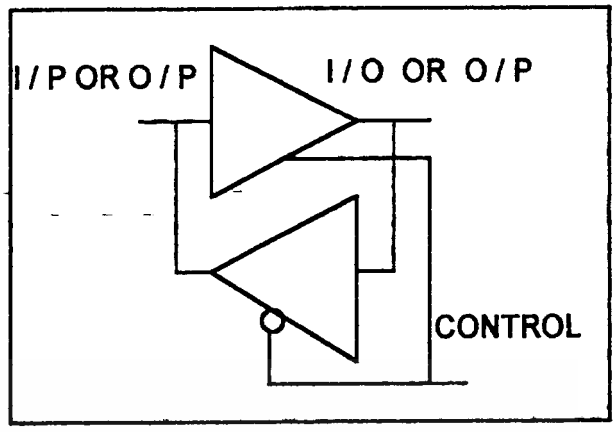
รูปที่ 2.1 รูปแบบการทำงานโดยรวมของ ICE (In - circuit Emulator) ที่ออกแบบขึ้น

จากรูปที่ 2.1 เป็นรูปแบบโดยรวมของ ICE (In - circuit Emulator) ที่ออกแบบขึ้นใช้งาน โดยสัญญาณทุกเส้นที่เข้าและออกจากซีพียูจะถูกเชื่อมต่อออกเป็น 2 เส้น เส้นที่จะไปเข้ากับระบบ 2 ระบบ โดยเป็นระบบของผู้ใช้และเป็นระบบของมอนิเตอร์ และก่อนที่จะเข้าทั้ง 2 ระบบนี้ จะให้สายสัญญาณทั้งหมดผ่านอุปกรณ์บัฟเฟอร์เพื่อที่จะควบคุมให้สัญญาณสามารถเข้าหรือออกไปสู่ระบบได้

โดยปกติสัญญาณที่จะเข้าหรือออกจากซีพียูจะมี 2 ลักษณะคือ เป็นสัญญาณที่ถูกส่งออกอย่างเดียวเท่านั้น (directional bus) เช่นบัสแอดเดรส และสัญญาณที่มีทั้งการเข้าและการออกจากตัวซีพียู (bidirection bus) เช่นบัสข้อมูล ดังนั้นการต่ออุปกรณ์บัฟเฟอร์ในกรณีแรกจะเป็นไปตามรูปที่ 2.2 และการต่ออุปกรณ์บัฟเฟอร์ในกรณีที่สองจะเป็นไปตามรูปที่ 2.3



รูปที่ 2.2 บัฟเฟอร์ทิศทางเดียว (74LS244)



รูปที่ 2.3 บัฟเฟอร์สองทิศทาง (74LS245)

สัญญาณที่ใช้ควบคุมการทำงานของบัฟเฟอร์จะเป็นสัญญาณที่ได้มาจาก control box กล่าวคือ ถ้าเอาท์พุทของ control box มีสถานะ "1" จะทำให้บัฟเฟอร์ชุดที่ 2 สามารถส่งผ่านข้อมูลไปยังเพจมอนิเตอร์ได้ และถ้าเอาท์พุทของ control box มีสถานะเป็น "0" จะทำให้บัฟเฟอร์ชุดที่ 1 ส่งผ่านข้อมูลไปยังเพจของผู้ใช้ได้ นั่นหมายความว่าบัฟเฟอร์ทั้งสองชุดนั้นจะไม่สามารถทำงานพร้อมกันได้เลย ซึ่งการที่จะให้สถานะเอาท์พุทของ control box มีสถานะเป็นเช่นใดนั้นจะขึ้นอยู่กับฟังก์ชันการทำงานที่ออกแบบขึ้น

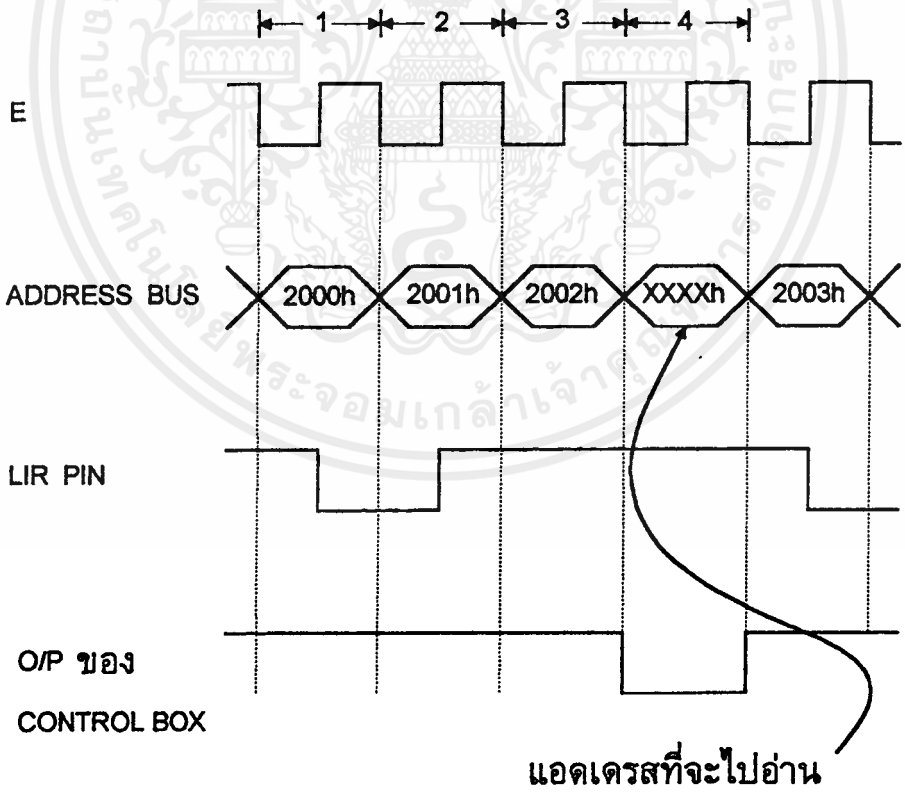
2.2 การสลับการทำงานในแต่ละเพจ

จากหัวข้อที่แล้วจะเห็นว่าซีพียูในขณะหนึ่งขณะใดนั้น จะทำงานในแต่ละระบบหรือแต่ละเพจเท่านั้น ซึ่งโดยเริ่มแรกจะต้องให้ ซีพียูมารันที่เพจมอนิเตอร์ก่อนเสมอ เพื่อรอรับค่าหรือรอรับคำสั่งการทำงานจากเทอร์มินอล (ในที่นี้คือ เครื่องคอมพิวเตอร์ส่วนบุคคล) ว่าจะให้ทำงานคำสั่งใด จากนั้นจึงจะพิจารณาได้ว่าจะให้เอาท์พุทของ control box ส่งสถานะใดออกมา ซึ่งโดยปกติทั่วไปแล้วนั้นชุดคำสั่งในการทำงานของ ICE จะแบ่งออกได้ 3 ลักษณะดังนี้

1. กลุ่มคำสั่งที่เกี่ยวกับการอ่านหรือเขียนข้อมูล ได้แก่ คำสั่ง D, R
2. กลุ่มคำสั่งที่ใช้รันโปรแกรมของผู้ใช้ ได้แก่ คำสั่ง G
3. กลุ่มคำสั่งที่ใช้ตั้งค่าจุดหยุดและคำสั่งที่ใช้รันโปรแกรมของผู้ใช้ทีละคำสั่ง (Single Step) ได้แก่คำสั่ง B และคำสั่ง T

2.2.1 กลุ่มคำสั่งที่เกี่ยวกับการอ่านหรือเขียนข้อมูล

ในการอ่านหรือเขียนข้อมูลไปยังเพจของผู้ใช้นั้น เมื่อเริ่มแรกที่ซีพียูได้รับคำสั่งก็จะกระโดดไปทำงานยังโปรแกรมย่อยของการอ่านหรือเขียนข้อมูล ซึ่งมีหลักการทำงานคือ จะเป็นการอ่าน หรือเขียนข้อมูลเพียงระยะเวลาหนึ่ง นั่นคือจะให้บัฟเฟอร์ชุดที่ 1 ทำงานเพียงชั่วขณะที่มีการอ่านหรือเขียนข้อมูล และให้ซีพียูกลับมาทำงานยังโปรแกรมมอนิเตอร์ต่อทันที ยกตัวอย่างเช่น เมื่อผู้ใช้ต้องการเขียนค่าลงไปยังหน่วยความจำตำแหน่ง 7000h ซีพียูจะสั่งให้มีการเขียนคำสั่ง STAA $7000 ไปยังแรมมอนิเตอร์ตำแหน่ง 2000h ของเพจมอนิเตอร์ ฮาร์ดแวร์ส่วนหนึ่งของ control box จะทำหน้าที่ให้สถานะทางเอาท์พุทเป็น " 0 " เมื่อมีการเขียนข้อมูลจากรีจิสเตอร์ A ไปยังหน่วยความจำที่ 7000h ชั่วขณะและ control box ก็จะส่งเอาท์พุท " 1" เมื่อมีการเขียนข้อมูลเสร็จเรียบร้อยแล้ว ซึ่งจะเห็นได้ว่าเป็นการเขียนข้อมูลไปยังเพจของผู้ใช้เพียงชั่วขณะเท่านั้น ดังนั้นจึงจำเป็นต้องพิจารณาการทำงานให้ถูกต้อง เพื่อที่จะทำให้แน่ใจได้ว่าค่าที่ได้จากการอ่าน หรือค่าที่ได้จากการเขียนลงไปมีความถูกต้องแน่นอน ขั้นตอนการออกแบบที่เกี่ยวข้องกับการอ่านหรือเขียนข้อมูลไปยังหน่วยความจำ หรือพอร์ทของผู้ใช้จะมีข้อตกลง (Protocal) ในการออกแบบดังนี้



รูปที่ 2.4 การทำงานของซีพียูเมื่อรันคำสั่ง LDAA $XXXX
หรือ STAA $XXXX

ในกรณีที่เป็นการอ่านข้อมูลจากเพจของผู้ใช้ จะเขียนคำสั่ง LDAA $XXXX ไปยังแรมมอนิเตอร์ ตำแหน่ง 2000h โดยที่ XXXX คือแอดเดรสที่ผู้ใช้ต้องการอ่าน

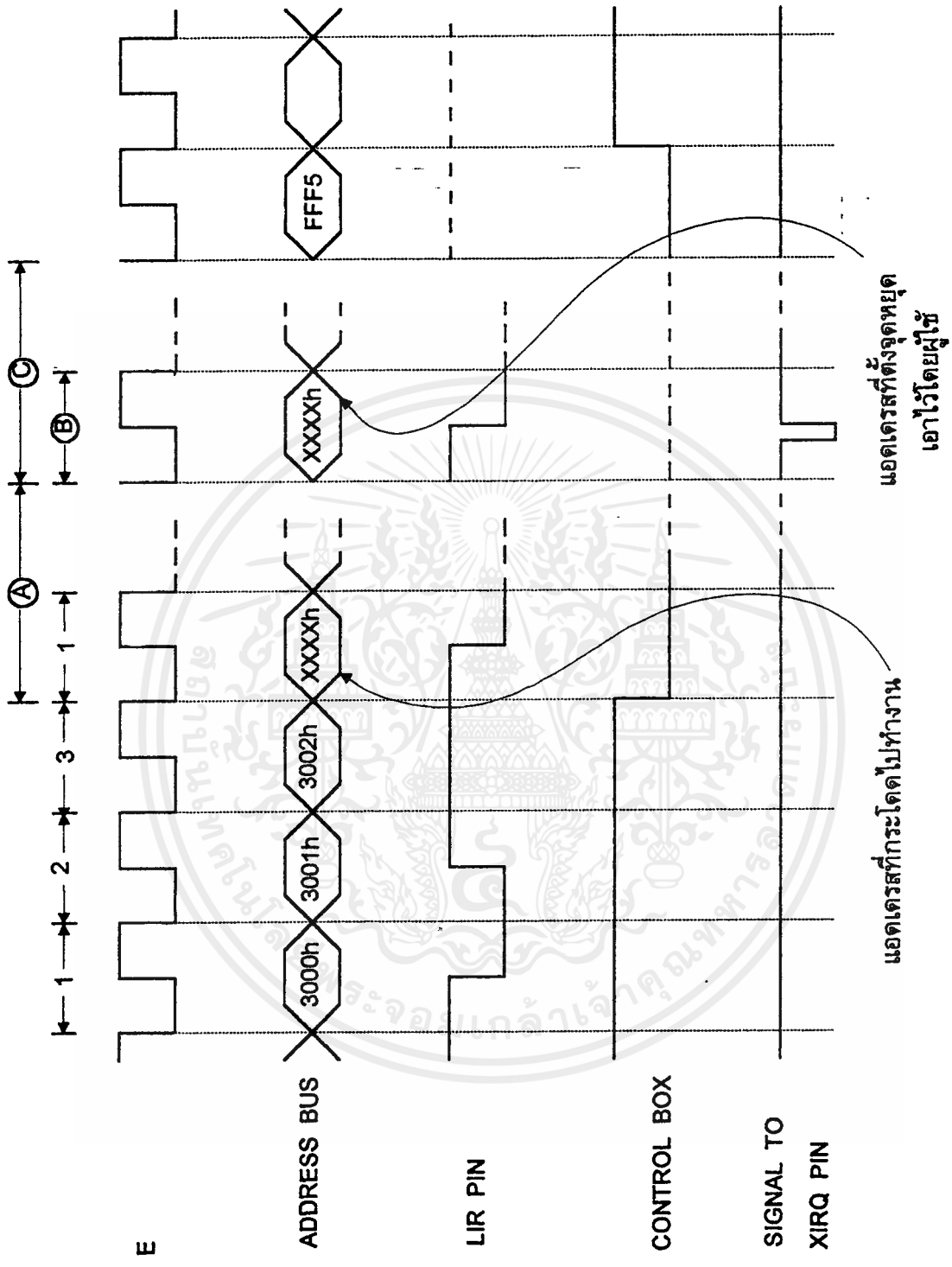
ในกรณีที่เป็นการเขียนข้อมูลไปยังเพจผู้ใช้ จะเขียนคำสั่ง STAA $XXXX ไปยังแรมมอนิเตอร์ ตำแหน่ง 2660h โดยที่ XXXX คือแอดเดรสที่ผู้ใช้ต้องการเขียน

ซึ่งโดยปกติแล้วสัญญาณของบัสแอดเดรสนั้น จะมีการคาบเกี่ยวกันกับสัญญาณของบัสข้อมูลในหนึ่งไซเคิล ดังนั้นช่วงเวลาที่ไซเคิลที่ 4 ในการประมวลผลนั้น จะต้องให้เอาท์พุทของ control box นั้นมีสถานะเป็น " 0 " ซึ่งหมายความว่าแอดเดรสที่ไปอ่านหรือเขียนข้อมูลจะอยู่ในเพจของผู้ใช้ และเมื่อการอ่านหรือเขียนเสร็จแล้วก็จะกลับมารันที่แอดเดรส 2003h ซึ่งจะอยู่ในเพจของมอนิเตอร์ต่อไป

2.2.2 กลุ่มคำสั่งที่รันโปรแกรมของผู้ใช้

ในการรันโปรแกรมของผู้ใช้ จะสั่งให้ซีพียูเขียนคำสั่ง jmp $XXXX ไปยังแอดเดรส 3000h ของแรมมอนิเตอร์ (โดยที่ XXXX คือแอดเดรสที่ผู้ใช้ต้องการให้เริ่มรันโปรแกรม) ซีพียูจะกระโดดไปทำงานยังแรมมอนิเตอร์ตำแหน่ง 2000h นี้ และช่วงเวลาที่เสร็จจากการรันโปรแกรมคำสั่งนี้ control box จะส่งเอาท์พุทสถานะ " 0 " ออกมา เพื่อที่แอดเดรสต่อไปที่จะทำการรันโปรแกรมนั้นเป็นแอดเดรสที่อยู่ในเพจของผู้ใช้

การรันโปรแกรมของผู้ใช้จะรันจนกว่าจะถึงจุดหยุด (Break point) ที่ผู้ใช้ตั้งไว้ก่อนที่จะสั่งรัน เมื่อซีพียูรันคำสั่งสุดท้ายที่พบจุดหยุดเสร็จสิ้น ฮาร์ดแวร์ของ control box ก็จะส่งสถานะ " 1 " ออกมาเพื่อให้ซีพียูรันโปรแกรมต่อไปที่เพจของมอนิเตอร์ ปัญหาของขั้นตอนนี้คือ เมื่อซีพียูหยุดแอดเดรสใดแอดเดรสหนึ่งจากเพจของผู้ใช้แล้ว เมื่อสลับมารันที่เพจมอนิเตอร์จะไม่สามารถทราบได้เลยว่า ที่แอดเดรสต่อไปนั้นเป็นที่เก็บโปรแกรมของมอนิเตอร์ใดหรือแอดเดรสนั้นของโปรแกรมมอนิเตอร์ใช้ทำงานใดอยู่ ดังนั้นเพื่อแก้ปัญหาเหล่านี้ เมื่อฮาร์ดแวร์ของ control box พบจุดหยุดที่ต้องการแล้ว จะต้องมีเอาท์พุทส่วนหนึ่งเป็นสัญญาณอินเตอร์รัปท์ไปสู่ซีพียู ซึ่งเมื่อซีพียูได้รับสัญญาณการร้องขอบริการอินเตอร์รัปท์แล้ว ก็จะปฏิบัติงานที่แอดเดรสนั้นให้เสร็จสิ้นก็จะไปทำงานให้บริการอินเตอร์รัปท์



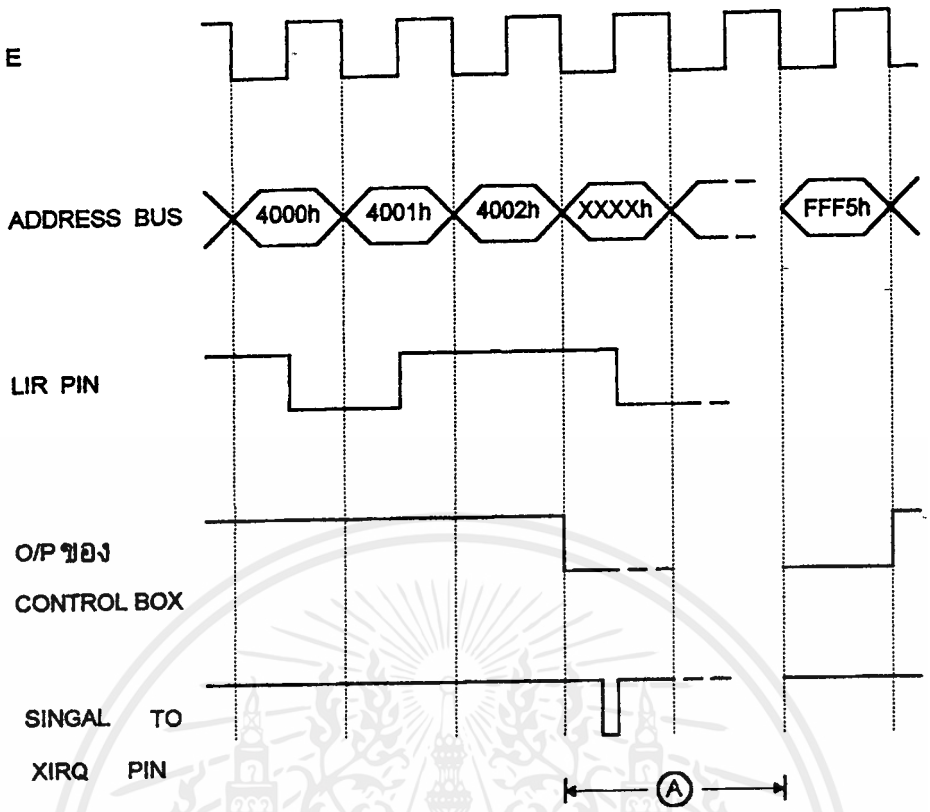
รูปที่ 2.5 การทำงานของซีพียูที่สอดคล้องกับ control box ในฟังก์ชันการรันโปรแกรมของผู้ใช้

เมื่อจังหวะที่ซีพียูกำลังให้บริการอินเตอร์รัปท์ ซึ่งในที่นี้ใช้การอินเตอร์รัปท์แบบนอนมาสเบิล (XIRQ Pin) ก็จะไปเฟตช์เวกเตอร์แอดเดรสที่อยู่ในแอดเดรส FFF4h , FFF5h จังหวะที่ซีพียูเฟตช์แอดเดรสนี้นั้นฮาร์ดแวร์ส่วนหนึ่งของ control box ก็จะทำให้สถานะทางเอาท์พุทมีสถานะเป็น " 1 " ซึ่งฮาร์ดแวร์ที่จัดทำขึ้นนี้ที่แอดเดรส FFF4h จะมีค่า 30 และที่แอดเดรส FFF5h จะมีค่า F1 ดังนั้นเมื่อซีพียูตอบสนองการอินเตอร์รัปท์ จะทำให้แอดเดรสที่จะเฟตช์ต่อไปจะเป็นแอดเดรส 30F1h ซึ่งเป็นแรมมอนิเตอร์ของระบบ และที่ 30F1h ของแรมมอนิเตอร์ก็จะเขียนคำสั่งให้ซีพียูกลับสู่โปรแกรมมอนิเตอร์หลักได้ พิจารณาไซเคิลการทำงานจะได้ดังรูปที่ 2.5 ซึ่งจะเริ่มจากการให้ซีพียูเฟตช์คำสั่ง jmp $XXXX ที่แอดเดรส 3000h ในแรมมอนิเตอร์

จากรูปที่ 2.5 ช่วงเวลาที่ A คือช่วงเวลาที่ซีพียูจะกระโดดไปทำงาน ซึ่งในขณะเวลาเดียวกันนั้นเอาท์พุทของ control box จะเปลี่ยนจากสถานะ " 1 " เป็น " 0 " และยังคงค่า " 0 " อยู่ตลอดจนกว่าแอดเดรสบัสจะมีค่า FFF5h ซึ่งจะหมายถึงว่าซีพียูเสร็จสิ้นช่วงที่ได้รับการ break พอดี นั่นคือช่วงเวลาที่ C เป็นช่วงเวลาที่ซีพียูจะกระทำคำสั่งสุดท้ายที่พบแอดเดรสการ break พอดี และช่วงที่ฮาร์ดแวร์ของ control box ได้รับแอดเดรสการเบรคตรงกับสัญญาณ LIR พอดีนั้น จะมีสัญญาณไปอินเตอร์รัปท์ที่ยังตัวซีพียู (ช่วงเวลาที่ B) ซึ่งก็จะทำให้ซีพียูกระทำคำสั่งที่มีอยู่จนเสร็จสิ้น แล้วจึงไปตอบสนองการอินเตอร์รัปท์ ซึ่งจะสามารถทำให้แน่ใจได้ว่าซีพียูได้กระทำตามคำสั่งที่อยู่ในระหว่างการได้รับสัญญาณอินเตอร์รัปท์จนเสร็จสิ้นพอดี สาเหตุที่จะไม่ให้ฮาร์ดแวร์ของ control box นั้นนับค่าสัญญาณนาฬิกา E ก็เนื่องมาจากเราไม่ทราบได้ว่าคำสั่งที่อยู่ในแอดเดรสนั้นมีกี่ไซเคิลนั่นเอง

2.2.3 การรันโปรแกรมผู้ใช้ทีละคำสั่ง

หลักการให้ซีพียูรันโปรแกรมของผู้ใช้ทีละคำสั่งหรือ single step นั้น จะมีหลักการคล้ายกับคำสั่งรันโปรแกรมของผู้ใช้ แต่จะให้มีสถานะการอินเตอร์รัปท์ทุกครั้งหลังจากที่รันโปรแกรม ทั้งนี้เนื่องจากเหตุผลเดียวกัน คือเราไม่สามารถทราบล่วงหน้าได้ว่า คำสั่งต่อไปนั้นจะมีช่วงเวลาการทำงานกี่ไซเคิล หลักการทำงานทั้งหมดจะมี Timing diagram การทำงานดังรูปที่ 2.6 โดยพิจารณาเริ่มตั้งแต่การให้ซีพียูรับคำสั่ง jmp ที่แอดเดรส $XXXX ของแรมมอนิเตอร์

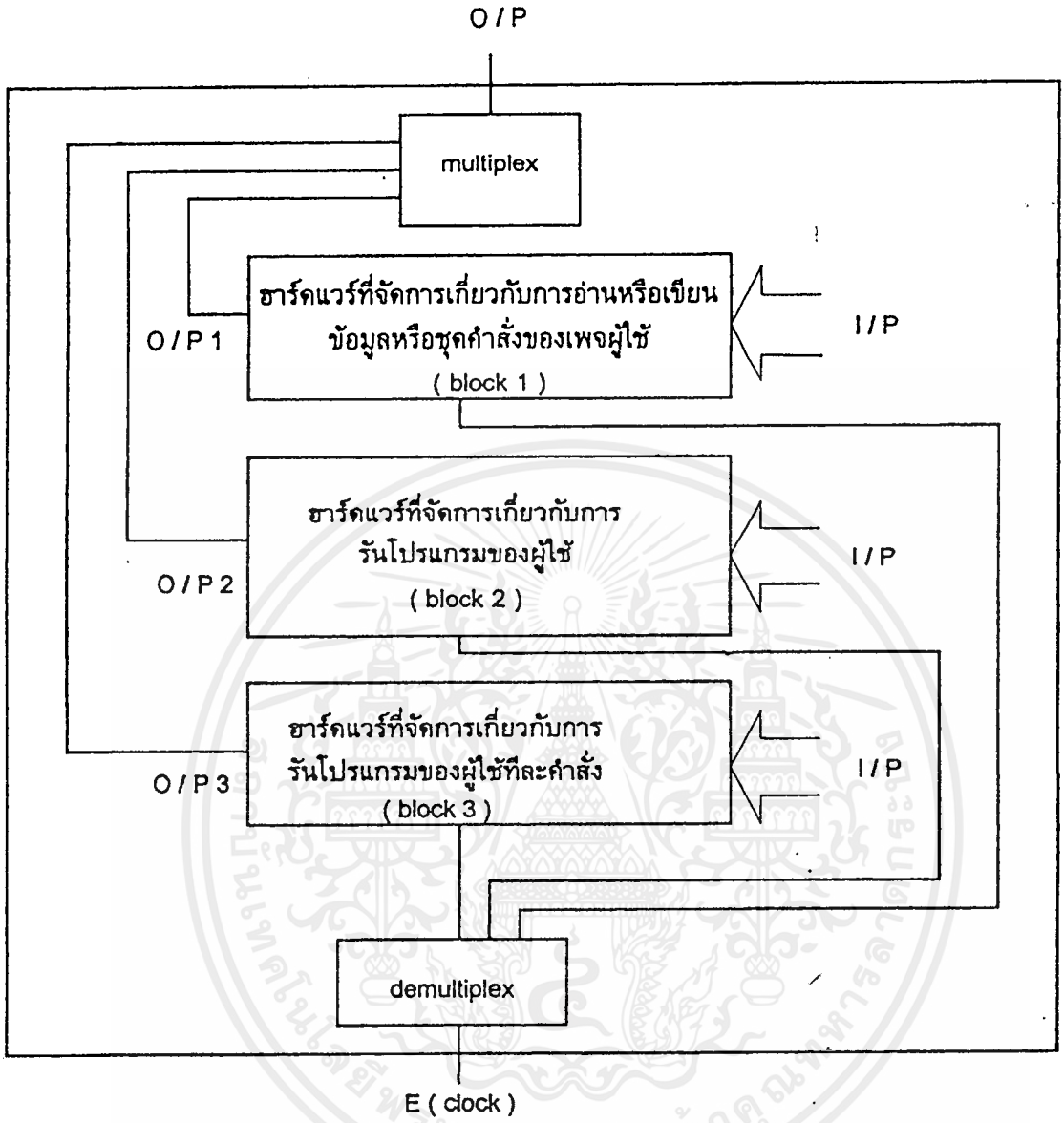


รูปที่ 2.6 การทำงานของซีพียูที่สอดคล้องกับ control box
ในฟังก์ชันการรันโปรแกรมของผู้ใช้ทีละคำสั่ง

จากรูปที่ 2.6 ช่วงที่ A จะเป็นช่วงที่ซีพียูปฏิบัติงานตามคำสั่งนั้นจนเสร็จสิ้น ทั้งหลังจากได้รับสัญญาณการอินเตอร์รัปท์ และเมื่อสิ้นสุดสัญญาณแอดเดรสบัสที่มีค่า FFF5h ก็จะทำให้เอาท์พุทของ conrtol box มีสถานะเป็น "1" อีกครั้ง นั่นคือทำให้ซีพียูกลับมาทำงานยังเพจของมอนิเตอร์

2.3 การออกแบบการทำงานในแต่ละกลุ่มคำสั่ง

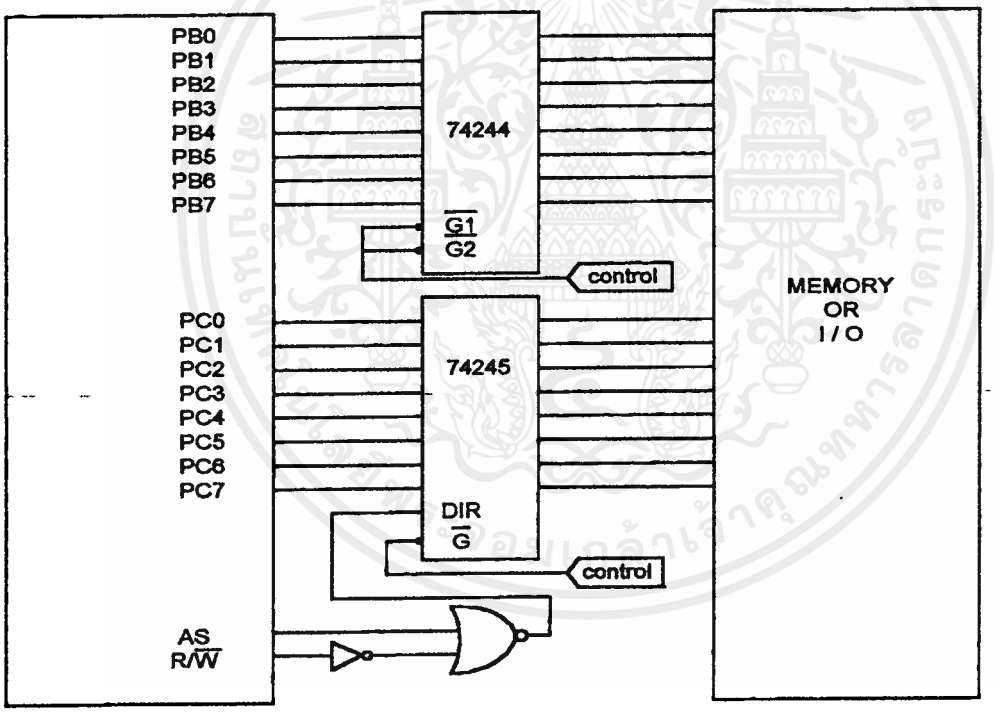
จากหัวข้อ 2.2 จะเห็นว่า เป็นการแบ่งกลุ่มการทำงานออกเป็น 3 กลุ่มใหญ่ๆ ดังนั้นในการออกแบบสิ่งที่อยู่ใน conrtol box จะต้องแบ่งเป็น 3 บล็อกย่อยๆ เป็นจำนวน 3 บล็อกที่มีการทำงานไม่พร้อมกัน ดังรูปที่ 2.7



รูปที่ 2.7 แสดงรายละเอียดของ block diagram ภายใน control box

จากรูปที่ 2.7 สัญญาณนาฬิกาอ้างอิงจะเป็นอินพุทของ block ทั้ง 3 และจะต้องมีการเลือกออกสัญญาณ E เพียงเส้นเดียวเท่านั้น เพื่อเป็นการทำงานของ block เพียงชุดเดียว และขาเอาท์พุททั้ง 3 block นั้น จะต้องมีการเลือกสัญญาณออกไปควบคุมเพียงเส้นเดียวเช่นเดียวกัน ซึ่งโดยปกติจะให้ block ที่ 1 เป็น block ทำงานปกติ และสัญญาณเข้าออกของสัญญาณนาฬิกา E จะถูกควบคุมโดยชุดมอนิเตอร์เพจ

จากรูปที่ 2.1 และ 2.2 การควบคุมสัญญาณเข้าออกของบัฟเฟอร์จะพิจารณาจากลักษณะของบัสในแต่ละประเภท โดยซีพียู 68HC11 นี้เมื่อทำงานในโหมดมัลติเพล็กขยาย พอร์ท B จะทำหน้าที่เป็นบัสแอดเดรสไบต์สูง (A8 - A15) และพอร์ท C จะทำหน้าที่เป็นบัสแอดเดรสไบต์ต่ำ (A0 - A7) และบัสข้อมูล ดังนั้นพอร์ท B จะไม่มีปัญหาเรื่องของสัญญาณการควบคุม นั่นคือจะใช้การอีนาเบิลการนำสัญญาณไว้โดยตลอด (จนกว่าจะมีการสลับเพจการทำงาน) แต่ในส่วนของพอร์ท C จะมีลักษณะของการควบคุมทิศทางการเข้าออกของสัญญาณได้ 2 กรณี คือ กรณีที่มีสัญญาณออก ซึ่งจะเป็นได้ทั้งมีการส่งสัญญาณแอดเดรส และการเขียนข้อมูลกับหน่วยความจำ กรณีที่ 2 คือสัญญาณเข้าจะเป็นได้เพียงกรณีเดียว คือเป็นการอ่านข้อมูลจากหน่วยความจำหรืออินพุท เอาท์พุทเข้าสู่ตัวซีพียู ดังนั้นการควบคุมสัญญาณเข้าออกให้เป็นไปตามเงื่อนไขดังกล่าวมาแล้วข้างต้น และต่อเป็นรูปวงจรได้ดังรูปที่ 2.8

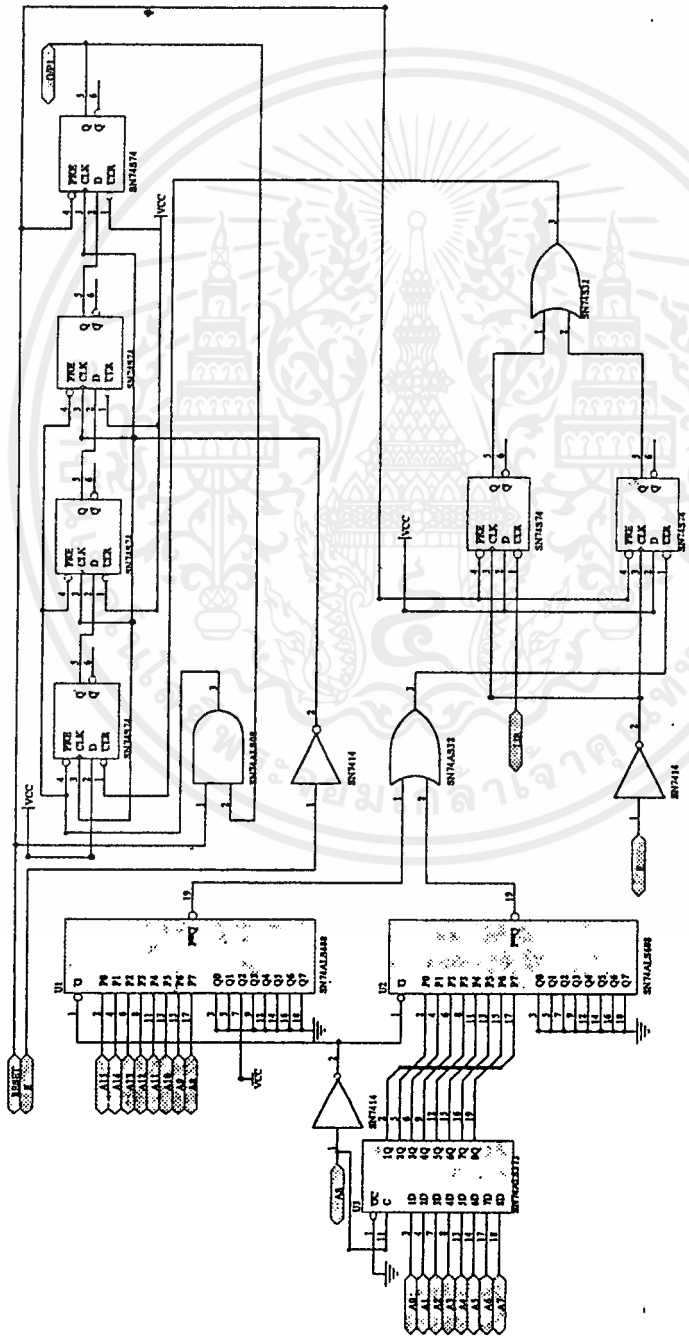


รูปที่ 2.8 วงจรการควบคุมสัญญาณบัสของซีพียู 68HC11 โดยผ่านบัฟเฟอร์

ในกรณีเดียวกันกับบัฟเฟอร์ชุดที่สอง การต่อการใช้งานจะมีลักษณะเช่นเดียวกันกับรูปที่ 2.8 ดังนั้นสัญญาณการอีนาเบิลของบัฟเฟอร์ในแต่ละชุด จะถูกควบคุมโดยสัญญาณ control เพียงอย่างเดียว ซึ่งสัญญาณเส้นนี้จะมาจากเอาท์พุทของ control box ซึ่งจะกล่าวถึงในหัวข้อต่อไป

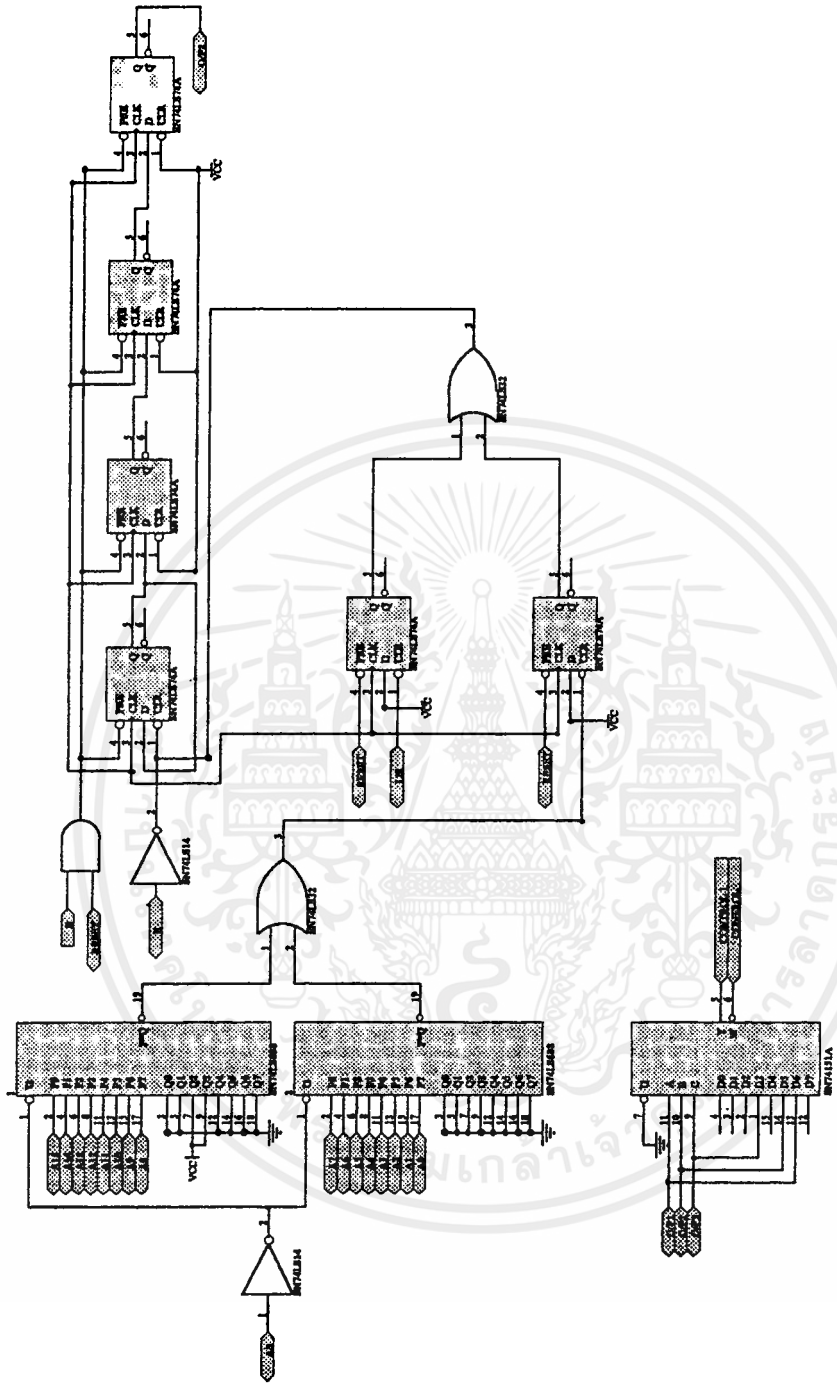
2.4 การออกแบบวงจรการใช้งานจริงในแต่ละส่วน

จากรูปที่ 2.4 การควบคุมการอ่านหรือเขียนข้อมูลในแต่ละครั้งจะใช้สัญญาณของแอดเดรสบัสที่มีค่า 2000h พร้อมกับสัญญาณการแอกทีฟ (active) ของขา LIR ในแต่ละ 1 clock ในการอีนาเบิลการนับ clock ของสัญญาณนาฬิกา E เพื่อให้เอาท์พุทที่ได้แอกทีฟ "0" ที่สัญญาณลูกที่ 4 และการนับจะใช้การเลื่อน "0" ของ clock ในแต่ละลูกโดยใช้ชิฟท์รีจิสเตอร์ (shift register) ที่จะเป็นการนำ D flip flop มาต่อคาดเคสกัน 4 ตัว และการให้สัญญาณอีนาเบิลในการนับนั้นจะเป็นการเคลียร์ flip flop ตัวแรกเพียง 1 ครั้ง ดังแสดงในรูปที่ 2.9

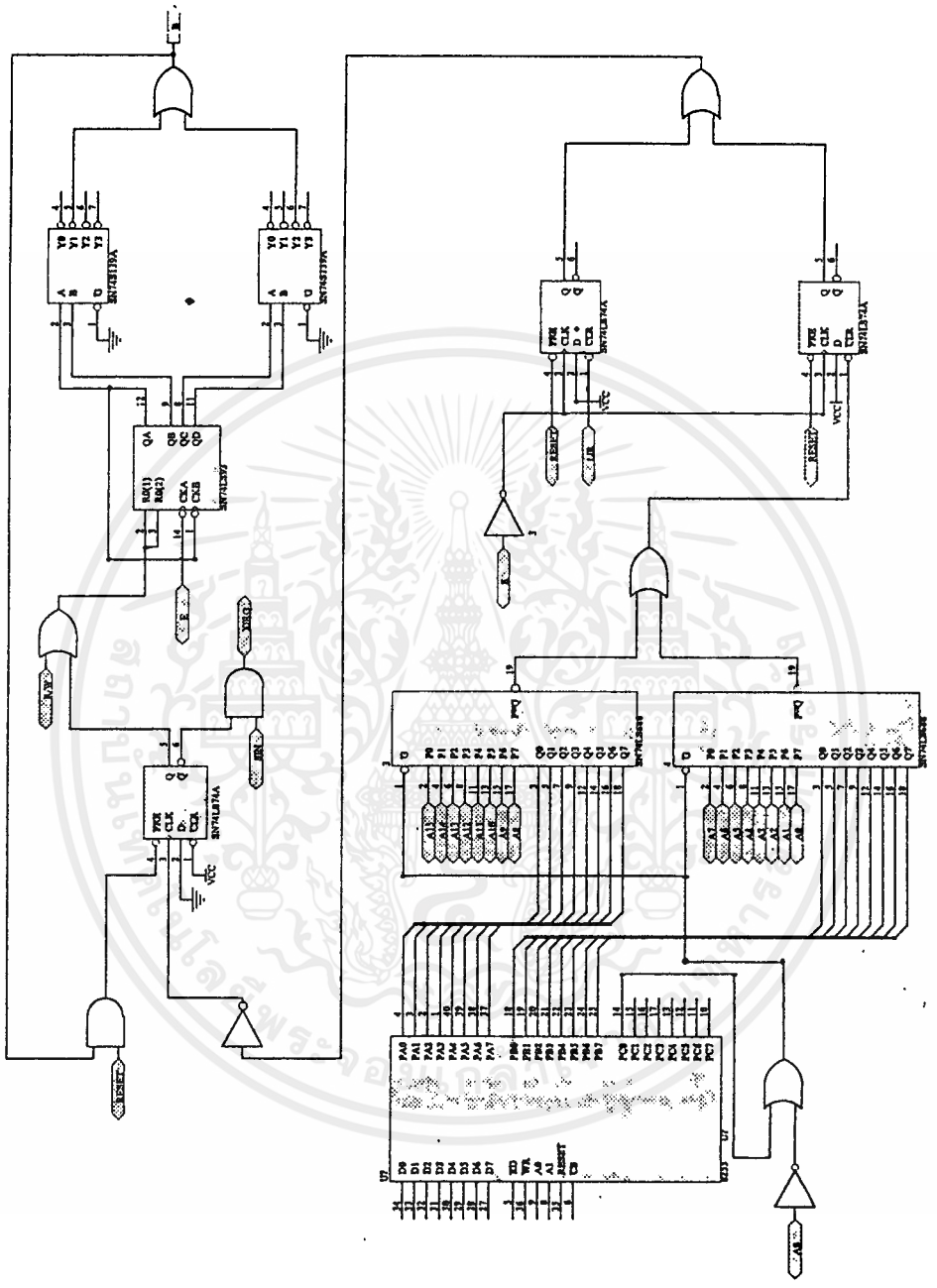


รูปที่ 2.9 วงจรการทำงานควบคุมการอ่านหรือเขียนข้อมูล

ในส่วนของการจัดการการรันโปรแกรมของผู้ใช้ จะมีเทคนิคการสลับการทำงานเริ่มต้นคล้ายกับชุดของการควบคุมการอ่านหรือเขียนข้อมูล คือจะใช้ชิฟท์รีจิสเตอร์เลื่อน " 0 " ออกมาทางเอาท์พุทในไซเคิลที่ 4 ของสัญญาณนาฬิกา เพียงแต่ไม่มีการสลับการทำงานกลับในไซเคิลต่อไป เพื่อที่จะต้องการให้ซีพียูรันโปรแกรมของผู้ใช้โดยตลอด และจะกลับคืนสู่มอนิเตอร์หลักได้เพียง 2 ทางคือ โดยการรีเซต หรือพบสัญญาณ break ที่ตั้งไว้โดยผู้ใช้เท่านั้น ซึ่งสัญญาณ break ดังกล่าวจะมีหลักการทำงานคือ เมื่อผู้ใช้ตั้งจุดหยุดผ่านทางเทอร์มินอล โปรแกรมมอนิเตอร์จะเขียนแอดเดรสที่ต้องการจะหยุดให้กับพอร์ท 8255 โดยจะให้พอร์ท A เป็นส่วนของแอดเดรสไบต์สูง และพอร์ท B เป็นส่วนของแอดเดรสไบต์ต่ำ พอร์ททั้งคู่จะถูกเชื่อมต่อกับส่วนของวงจรเปรียบเทียบสัญญาณ (74688) และให้การอีนาเบิล การตั้งจุดหยุดทางพอร์ท C (บิต C0) เอาท์พุทของวงจรเปรียบเทียบนี้จะเชื่อมต่อไปสู่ขาอินเตอร์รัปท์ของ 68HC11 โดยจะใช้ขาอินเตอร์รัปท์แบบนอนมาสเบิล เนื่องจากการอินเตอร์รัปท์แบบนี้ถูกจัดให้มีระดับความสำคัญสูงสุด (highgest piority) เมื่อซีพียูได้รับสัญญาณอินเตอร์รัปท์แล้วจะมี Timing diagram การทำงานตอบสนองการอินเตอร์รัปท์ดังรูปที่ 1.4 ซึ่งจาก Timing diagram จะเห็นว่าเมื่อซีพียูได้รับสัญญาณร้องขอการอินเตอร์รัปท์และเอ็กซีคิวต์คำสั่งสุดท้ายจนเสร็จสิ้นแล้ว ซีพียูจะใช้ช่วงเวลาหนึ่งในการเก็บค่าต่าง ๆ ที่สำคัญลงในพื้นที่สแต็ก ดังนั้นการออกแบบวงจรที่ใช้งานในขั้นตอนนี้คือ จะต้องคอยเช็คสัญญาณว่าหลังจากมีสัญญาณเข้ามาสู่ซีพียูเมื่อขอบริการอินเตอร์รัปท์นี้แล้ว สัญญาณขา R / W มีสถานะลอจิก " 0 " ติดต่อกันเกิน 9 ไซเคิลหรือยัง ถ้ายังนั่นหมายถึงว่าซีพียูจะยังไม่เก็บค่าต่าง ๆ ที่สำคัญลงในสแต็ก แต่ถ้าให้สัญญาณ R / W มีสถานะลอจิก " 0 " ติดต่อกัน 9 ไซเคิลแล้วแสดงว่าช่วงเวลานั้น ซีพียูกำลังเก็บค่าต่าง ๆ ลงบนสแต็ก ดังนั้นในขั้นตอนต่อไปคือ เมื่อสัญญาณ R / W มีสถานะลอจิก " 0 " ติดต่อกันครบ 9 ไซเคิล จะใช้ขอบขาลงของสัญญาณนาฬิกาในลูกต่อไป เพื่อทำการสลับเพจจากระบบผู้ใช้เข้าสู่ระบบมอนิเตอร์หลักต่อไป เพื่อให้ซีพียูเฟตช์เวกเตอร์แอดเดรสที่ต้องการใช้ วงจรการทำงานที่ใช้ควบคุมการรันโปรแกรมของผู้ใช้แสดงไว้ดังรูปที่ 2.10 และวงจรควบคุมการหยุด (break point circuit) แสดงไว้ดังรูปที่ 2.11



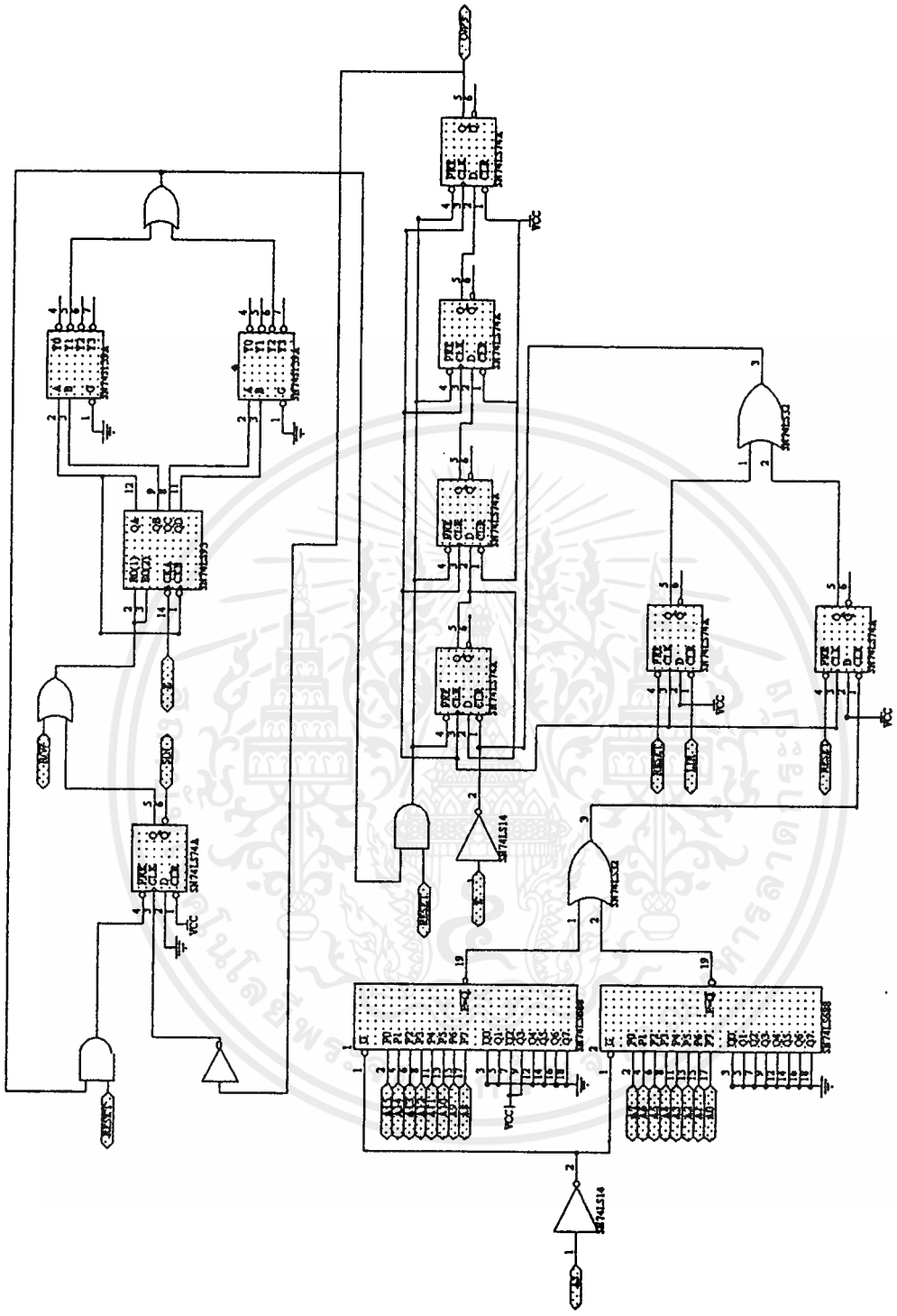
รูปที่ 2.10 วงจรการทำงานควบคุมการรันโปรแกรมของผู้ใช้



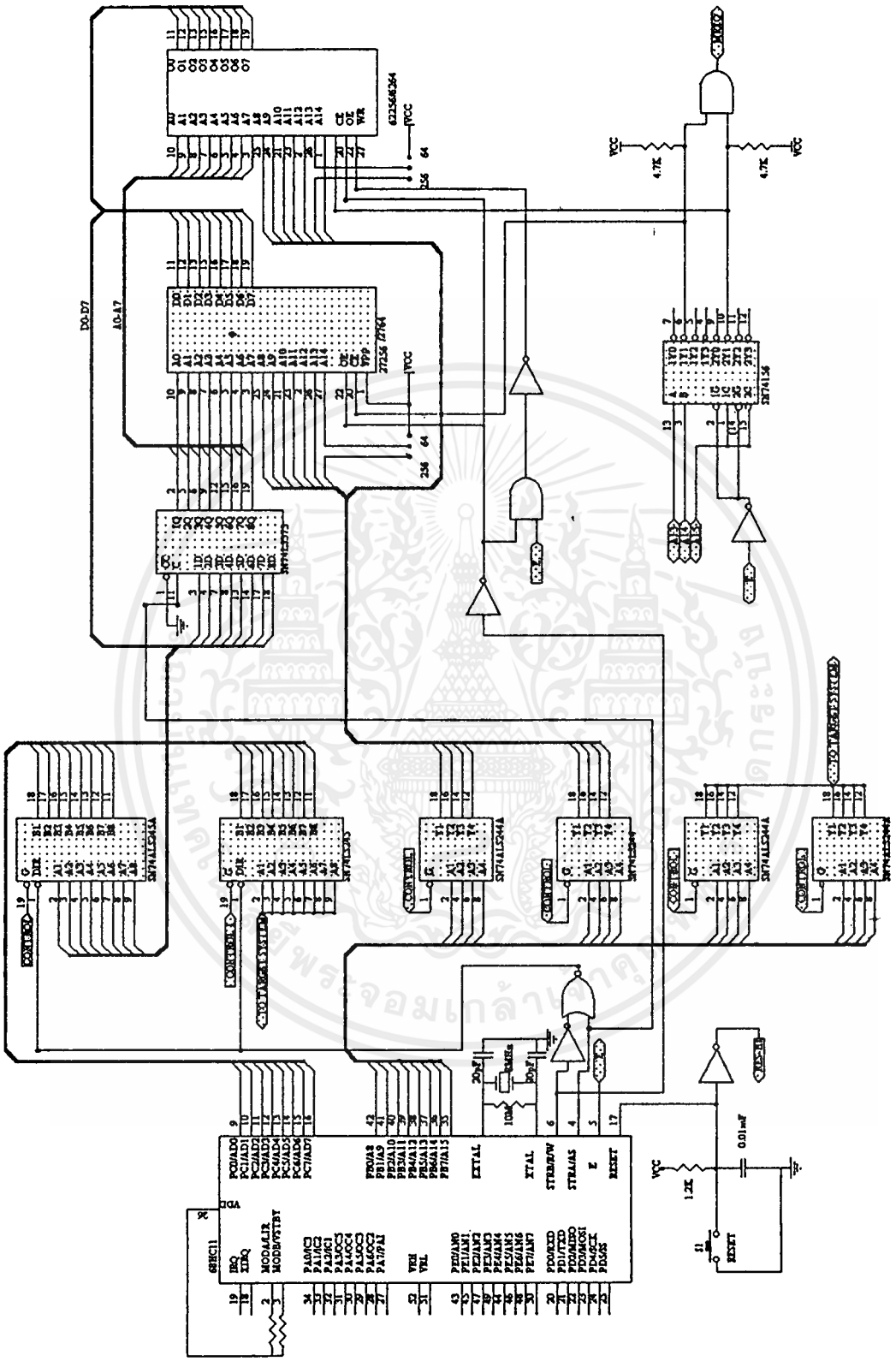
รูปที่ 2.11 วงจรการทำงานควบคุมการหยุดของซีพียู

โดยปกติแล้วการอินเตอร์รัปท์ของไมโครโปรเซสเซอร์ทุกเบอร์ จะมีลักษณะที่คล้ายกันอย่างหนึ่งคือ เมื่อเกิดการอินเตอร์รัปท์แล้วซีพียูจะกระโดดจากการทำงานเดิมไปสู่การบริการอินเตอร์รัปท์เซอร์วิสรูทีน และจะกลับมาทำงานต่อจากเดิมที่ทำค้างไว้ได้ก็ต่อเมื่อได้เอ็กซีคิวต์คำสั่ง return interrupt เท่านั้น ซึ่งจากการออกแบบวงจรการทำงานข้างต้นหากซีพียูกลับสู่ระบบมอนิเตอร์หลักได้แล้ว จะไม่สามารถตั้งจุด break อีกได้ เนื่องจากถ้าโปรแกรมสั่งให้ซีพียูเอ็กซีคิวต์คำสั่ง return interrupt จะไม่มีทางทราบได้เลยว่า ซีพียูจะกระโดดไปยังส่วนใดของโปรแกรมมอนิเตอร์ การแก้ปัญหาในขั้นตอนนี้คือ หลังจากที่ระบบกลับสู่มอนิเตอร์หลักแล้ว จะต้องแก้ค่าในโปรแกรมเคาท์เตอร์ใหม่ เพื่อให้ซีพียูกลับการทำงานไปยังที่ต้องการในโปรแกรมมอนิเตอร์ได้ โดยการ pop ค่าโปรแกรมเคาท์เตอร์ที่อยู่ในสแต็กออกมาแก้ไข และ push ค่าลงไปใหม่ จากนั้นจึงให้ซีพียูเอ็กซีคิวต์คำสั่ง return interrupt ซึ่งก็จะทำให้บิตมาสก์ต่าง ๆ ที่เกี่ยวข้องกับการอินเตอร์รัปท์ถูกเคลียร์เพื่อเตรียมพร้อมรับการอินเตอร์รัปท์ใหม่ได้อีก

ในส่วนของวงจรการจัดการควบคุมการรันโปรแกรมของผู้ใช้ทีละคำสั่ง จะคล้ายกับการรันโปรแกรมผู้ใช้ดังที่ได้กล่าวมาแล้วข้างต้น แต่จะมีข้อแตกต่างกันคือ จะไม่ใช้วงจร break เข้าช่วย แต่จะใช้หลักการนำสัญญาณควบคุมไปอินเตอร์รัปท์ซีพียูเลยทันที ดังนั้นเมื่อซีพียูรันโปรแกรมของผู้ใช้ก็จะเกิดการอินเตอร์รัปท์ขึ้นเลยในขณะเดียวกัน ซึ่งซีพียูจะรันคำสั่งนั้นเพียงคำสั่งเดียวก็จะไปตอบสนองการบริการอินเตอร์รัปท์ และจะใช้หลักการเข้าสู่ชุดมอนิเตอร์เป็นหลักการเดียวกับชุด break point วงจรที่ใช้แสดงดังรูปที่ 2.12 และวงจรการทำงานควบคุมระบบแสดงได้ดังรูปที่ 2.13



รูปที่ 2.12 วงจรการทำงานควบคุมการรันโปรแกรมของผู้ใช้ทีละคำสั่ง

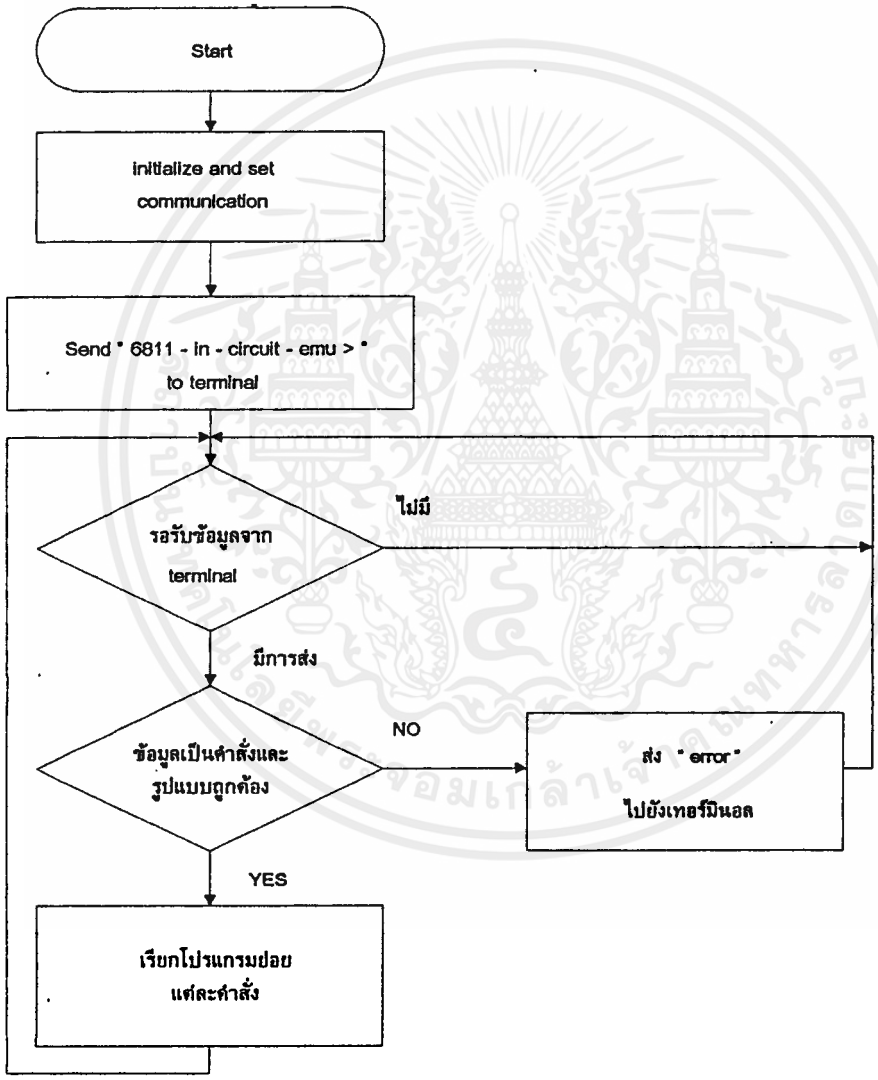


รูปที่ 2.13 วงจรการทำงานควบคุมระบบ

บทที่ 3

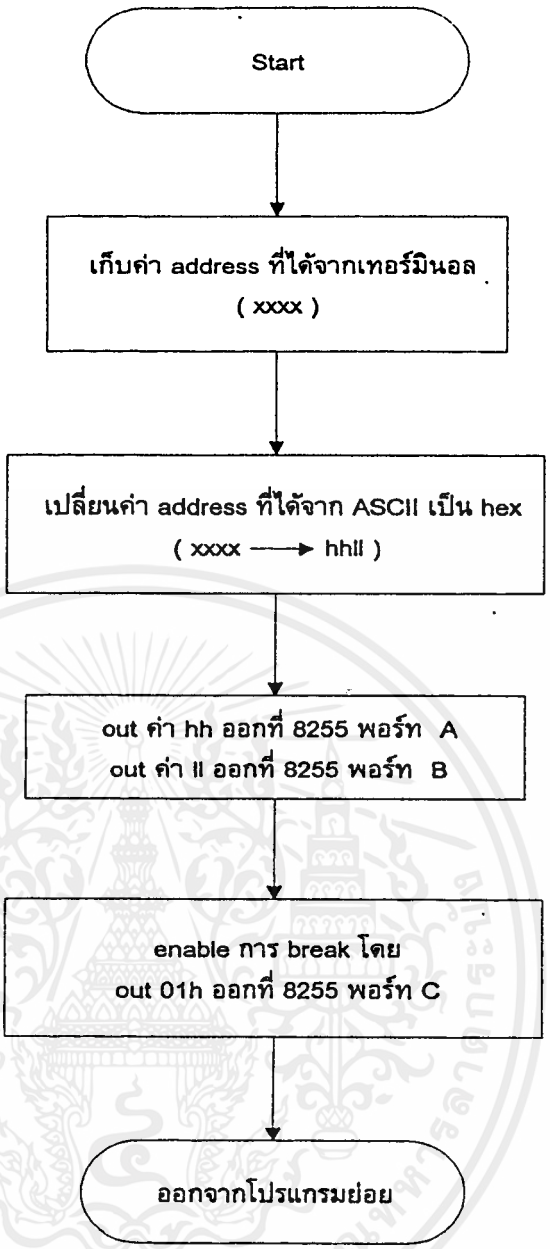
รายละเอียดการทำงานในแต่ละฟังก์ชัน

การทำงานของโปรแกรมมอนิเตอร์ จะมีส่วนโปรแกรมหลักดังรูปที่ 3.1 ซึ่งหลังจากที่กำหนดค่าเริ่มต้นให้กับระบบแล้วก็จะส่ง promt คำว่า " 6811 - in - circuit - emu > " ออกสู่เทอร์มินอลเพื่อแสดงความพร้อมในการรับคำสั่ง และแสดงว่าระบบทั้งสองคือ เทอร์มินอล และอิน - เซอร์กิต อีมูเลเตอร์ได้ถูกเชื่อมต่อกันเรียบร้อยแล้ว



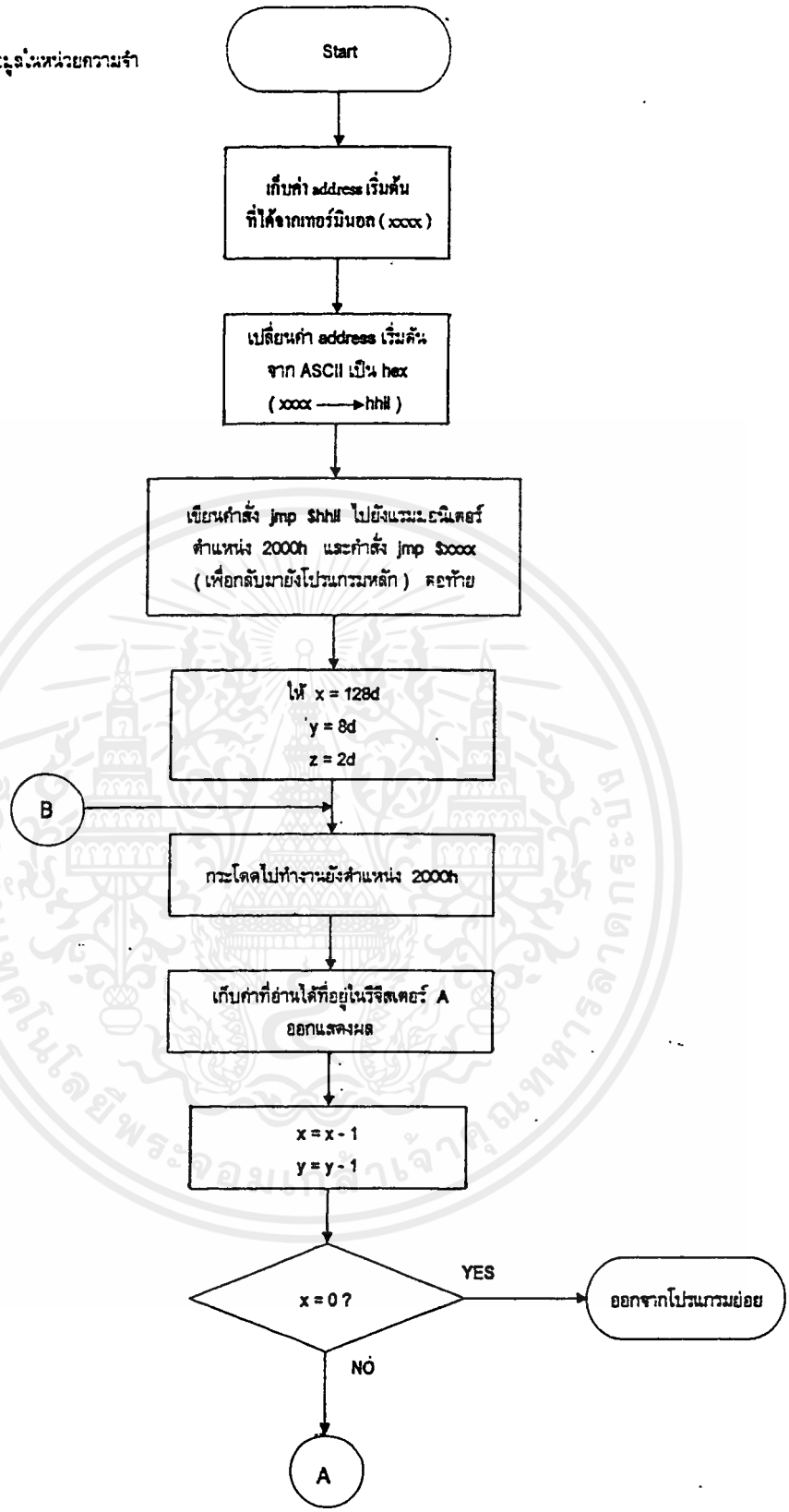
รูปที่ 3.1 การทำงานของโปรแกรมมอนิเตอร์

ฟังก์ชัน Break
หน้าที่ ตั้งจุดหยุด
รูปแบบ B xxxx

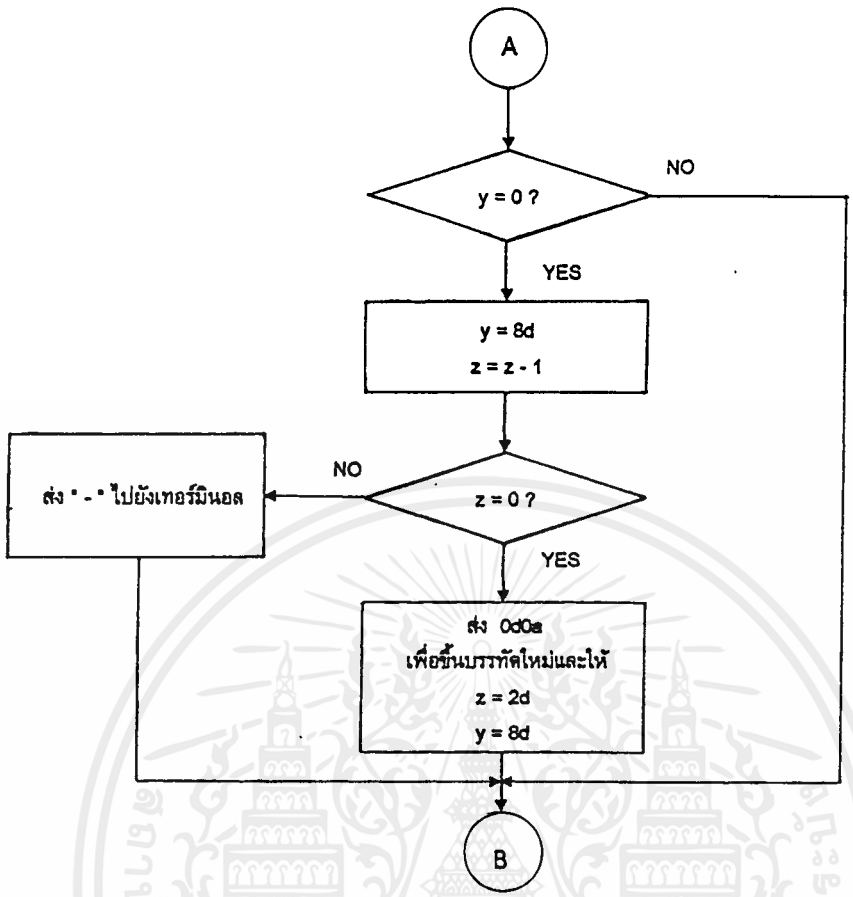


รูปที่ 3.2 การทำงานของฟังก์ชัน Break

ฟังก์ชัน dump
หน้าที่ ขอดูข้อมูลในหน่วยความจำ
รูปแบบ d xxxx



รูปที่ 3.3 การทำงานของฟังก์ชัน dump



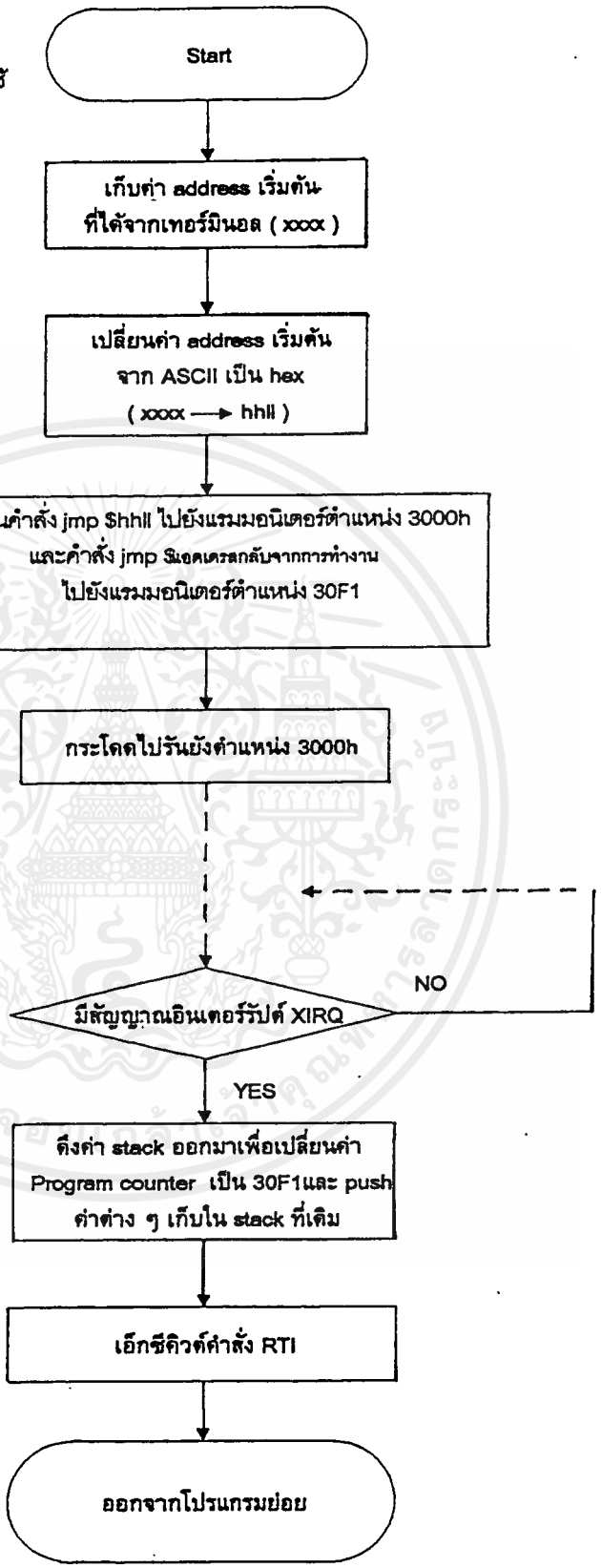
หมายเหตุ ตัวแปร x = ใช้นับจำนวนข้อมูลทั้งหมดที่ต้องแสดงผล (128 byte)

ตัวแปร y = ใช้นับจำนวนการแสดงผลในแต่ละครั้งแถว

ตัวแปร z = ใช้นับจำนวนการแสดงผลในแต่ละแถว

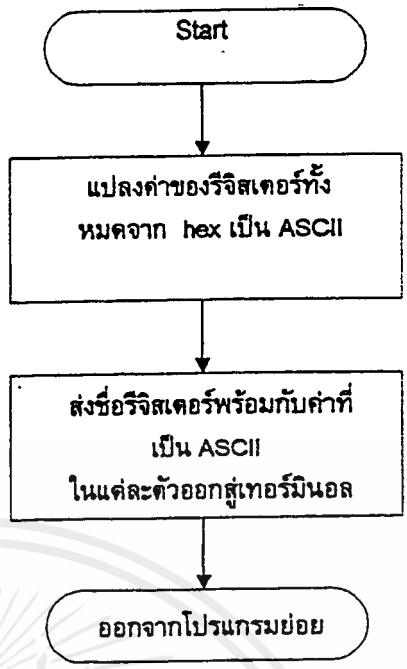
รูปที่ 3.3 การทำงานของฟังก์ชัน dump (ต่อ)

ฟังก์ชัน Go
หน้าที่ รันโปรแกรมของผู้ใช้
รูปแบบ G xxxx

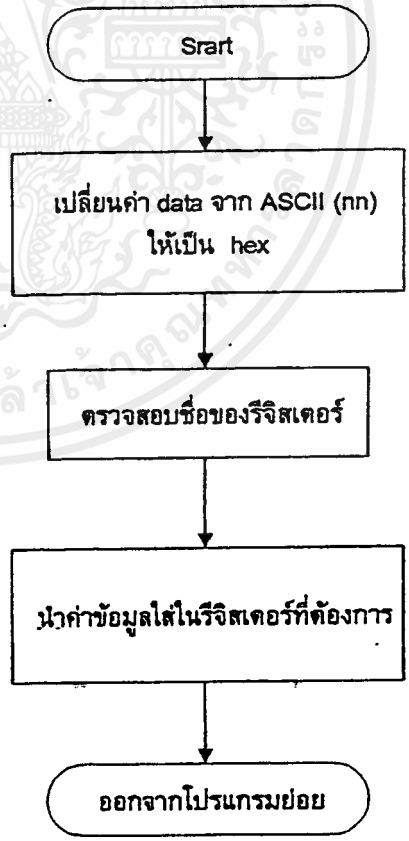


รูปที่ 3.4 การทำงานของฟังก์ชัน Go

ฟังก์ชัน View Register
หน้าที่ ขอดูค่าในรีจิสเตอร์
รูปแบบ R

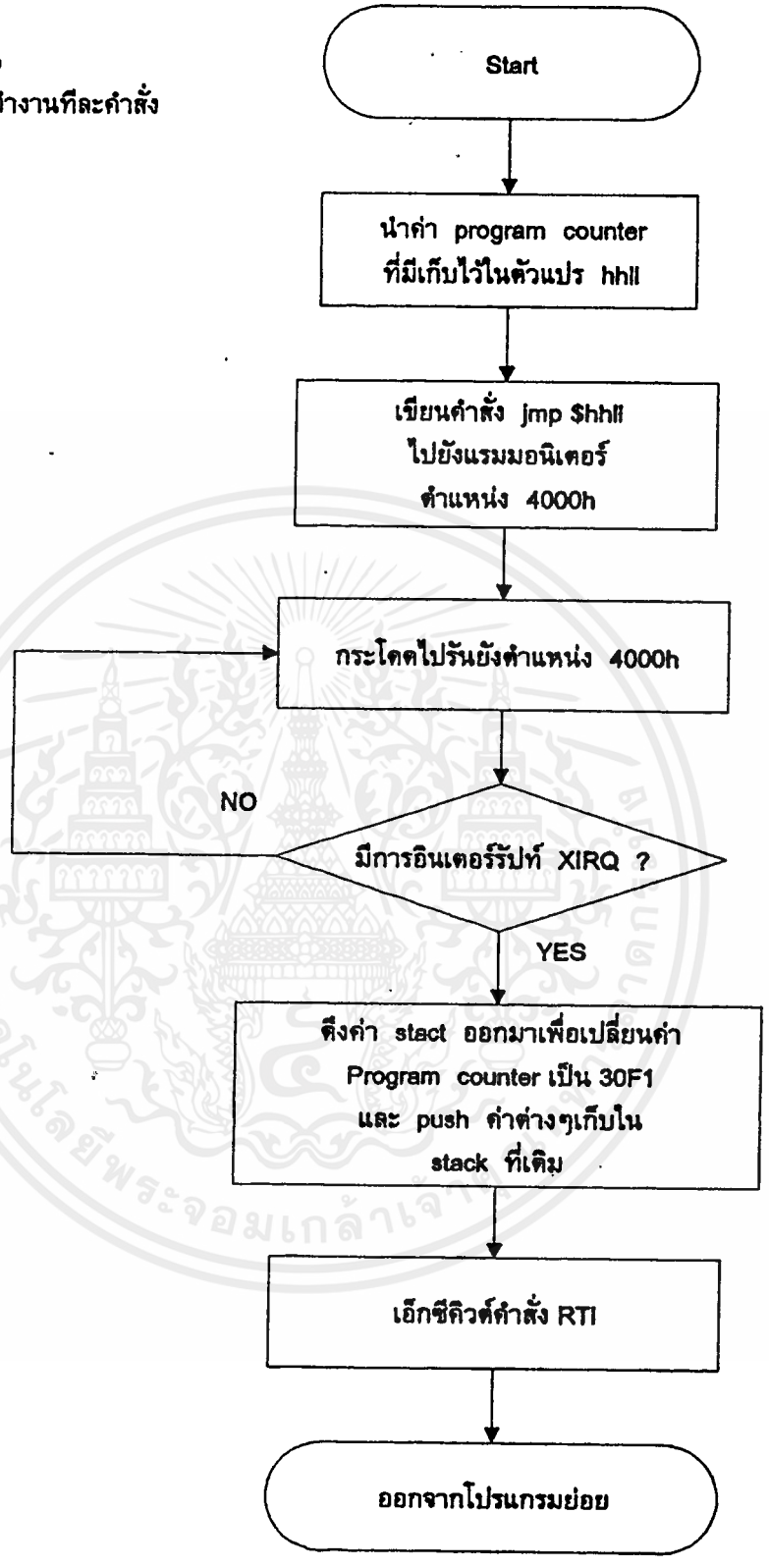


ฟังก์ชัน Set Register
หน้าที่ เก็บค่าในรีจิสเตอร์ที่ต้องการ
รูปแบบ R r nn



รูปที่ 3.5 การทำงานของฟังก์ชัน View Register และ Set Register

ฟังก์ชัน trace
หน้าที่ การทำงานทีละคำสั่ง
รูปแบบ t

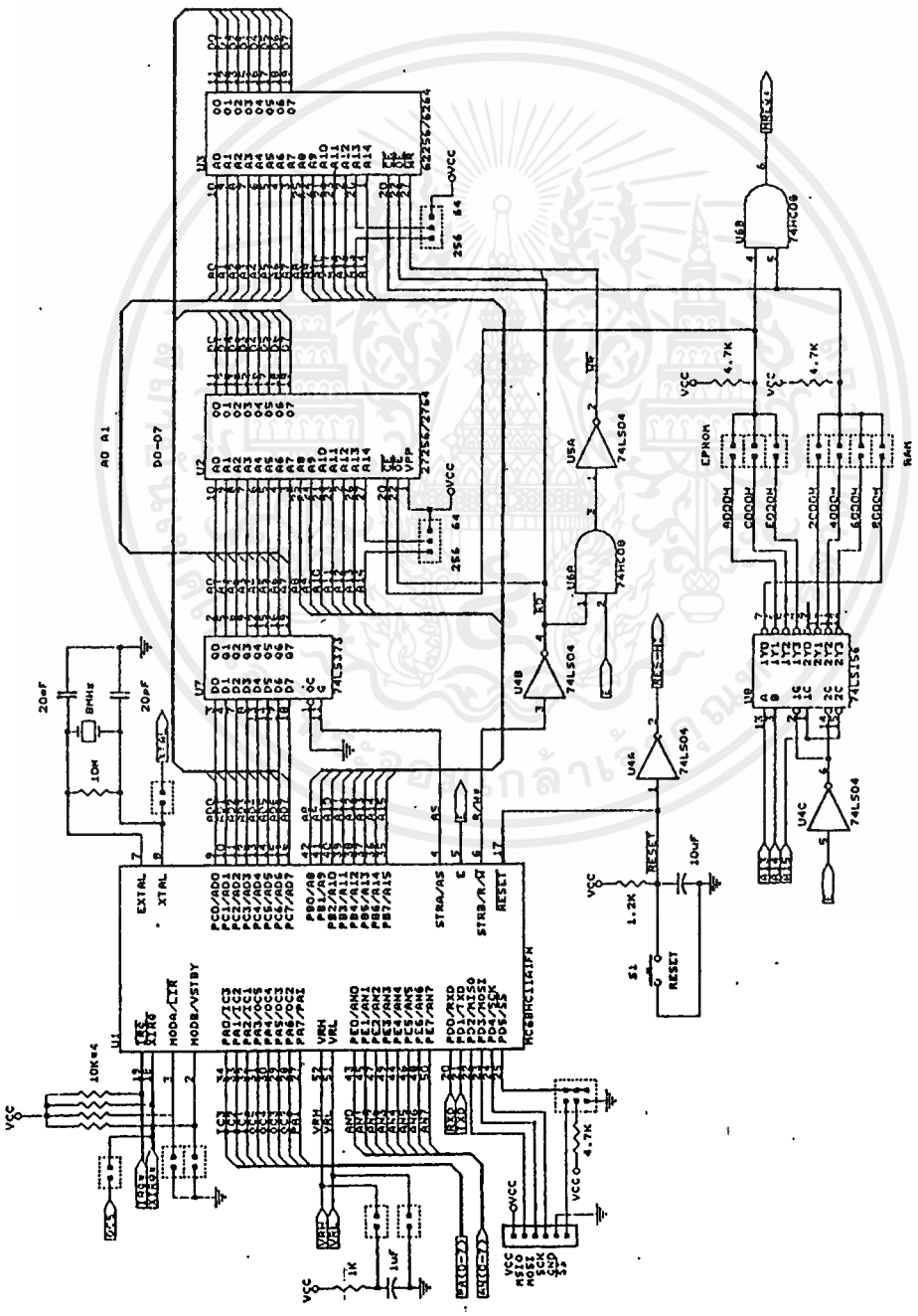


รูปที่ 3.6 การทำงานของฟังก์ชัน trace

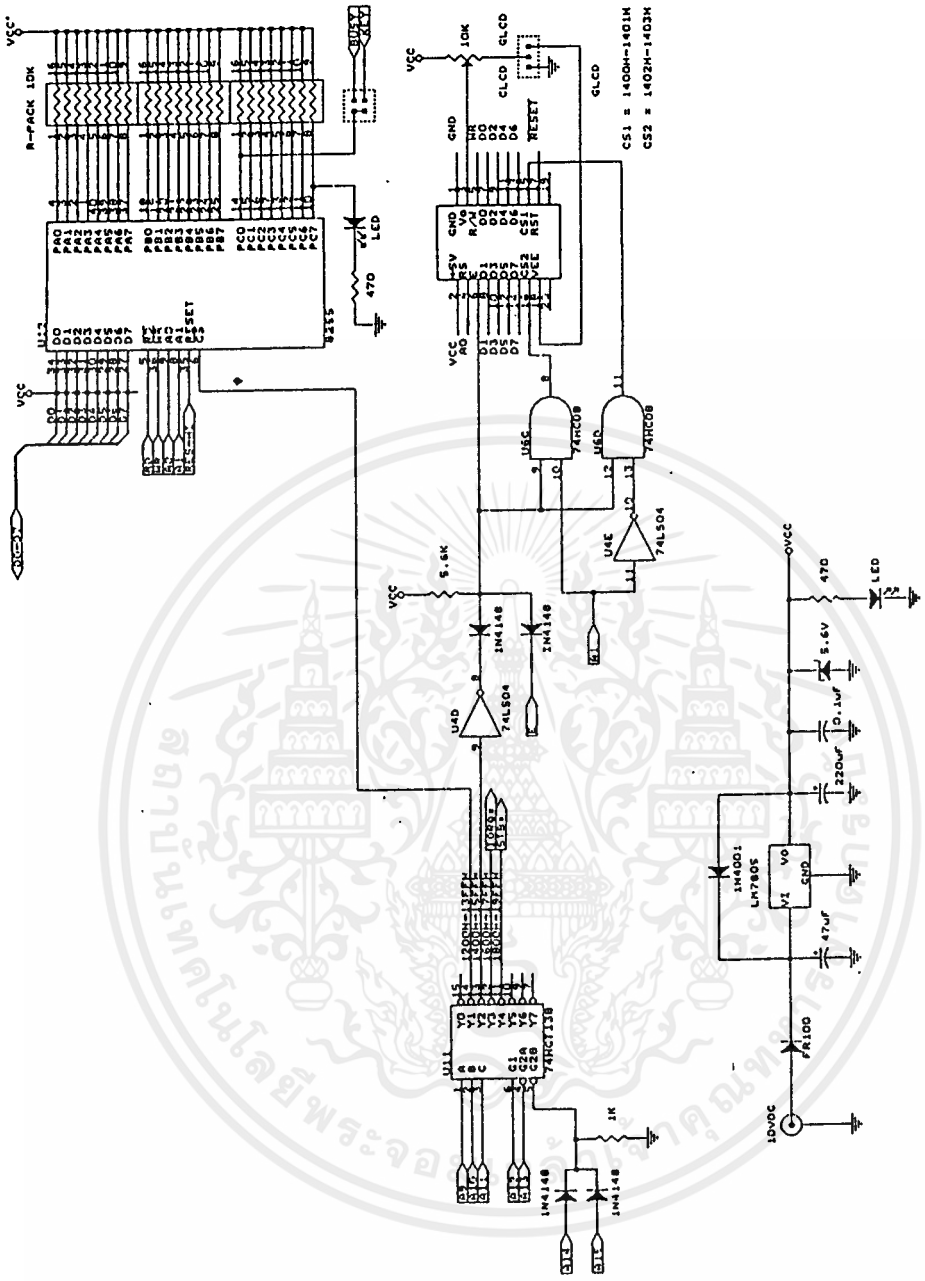
บทที่ 4

สรุปและวิจารณ์ผล

จากการออกแบบอิน - เซอร์กิต อีมูเลเตอร์จะเห็นได้ว่า การทำงานจะแบ่งออกเป็นสองส่วนคือ ฮาร์ดแวร์และซอฟต์แวร์ ซึ่งในส่วนของซอฟต์แวร์จะเป็นการทำงานที่คล้ายกับโปรแกรมมอนิเตอร์โดยทั่วไป แต่จะใช้หลักการควบคุมการสลับการทำงานของซีพียูเพียงตัวเดียว ที่สามารถแยกโปรแกรมมอนิเตอร์ให้อิสระจากชุดฮาร์ดแวร์ใดๆได้ โดยออกแบบวงจรจับจังหวะการทำงานต่างๆของซีพียูให้ถูกต้อง และจากการออกแบบที่ได้กล่าวมา สามารถทดลองได้ดังนี้คือ ชุดทดลองจะใช้วงจรดังรูปที่ 4.1



รูปที่ 4.1 วงจรที่ใช้ทดลอง



รูปที่ 4.1 วงจรที่ใช้ทดลอง (ต่อ)

โดยจะให้ชุดพัฒนานี้แสดงตัวอักษรผ่านทาง LCD 4 หลัก โดยมีโปรแกรมการทำงานดังนี้

```
                ;---------------------------------
0000            cpu   "68hc11.tbl"
0000            hof   "int8"

                ;---------------------------------
1400=           lcd_inst:     equ $1400
1401=           lcd_data:     equu $1401
                ;---------------------------------
A000                          org $0a000

A000 8E00FF                   lds #$00ff      ;for stack pointer
                ;---------------------------------
                ;     system delay
                ;---------------------------------
A003 4F                       clra
A004 5F         sysdly:       clrb
A005 5A         sys:          decb
A006 26FD                     bne  sys
A008 4A                       deca
A009 26F9                     bne  sysdly
                ;---------------------------------
                ;     initial lcd
                ;---------------------------------
A00B 8638       ini_lcd:      ldaa #$38       ;function set
A00D B71400                   staa lcd_inst
A010 C605                     ldab #05d
A012 BDA08C                   jsr  delay
A015 860C                     ldaa #$0c       ;display on/off
A017 B71400                   staa lcd_inst
A01A BDA098                   jsr  busy_delay
A01D 8606                     ldaa #$06       ;entry mode set
A01F B71400                   staa lcd_inst
A022 BDA098                   jsr  busy_delay
A025 BDA07D                   jsr  clr_disp
A028 7EA02B                   jmp  main
A02B CEA09D     main:         ldx  #table     ;load data
A02E 8680                     ldaa #$80       ;ADDRESS OF FIRST LINE LCD
A030 BDA056                   jsr  write_lcd

A033 CEA0AD                   ldx  #table1
```

```
A036 86C0                   ldaa  #$0C0
A038 BDA056                 jsr  write_lcd

A03B CEA0BD                 ldx  #table2
A03E 8690                   ldaa  #$90
A040 BDA056                 jsr write_lcd

A043 CEA0CD                 ldx  #table3
A046 86D0                   ldaa  #$0D0     ;load address of ddram
A048 BDA056                 jsr  write_lcd

A04B BDA07D                 jsr  clr_disp
A04E C678                   ldab #120d
A050 BDA08C                 jsr  delay
A053 7EA02B                 jmp  main

A056 BDA05D   write_lcd:    jsr  goto_addr
A059 BDA067                 jsr  write_data
A05C 39                     rts

A05D B71400   goto_addr:    staa  lcd_inst
A060 BDA098                 jsr  busy_delay
A063 BDA098                 jsr  busy_delay
A066 39                     rts

A067 8610     write_data:   ldaa #16d
A069 B72000                 staa  $2000
A06C A600     again:        ldaa  0h,x
A06E B71401                 staa  lcd_data
A071 C646                   ldab  #70d
A073 BDA08C                 jsr  delay
A076 08                     inx
A077 7A2000                 dec  $2000
A07A 26F0                   bne  again
A07C 39                     rts

A07D 8601     clr_disp:     ldaa #$01       ;clear display
A07F B71400                 staa  lcd_inst
A082 BDA098                 jsr busy_delay
A085 BDA098                 jsr busy_delay
A088 BDA098                 jsr busy_delay
A08B 39                     rts
```

```
                ;-------------------------------
                ;       delay 1 mS
                ;-------------------------------
A08C 3C         delay:  pshx
A08D CE01B2     del1:   ldx #434d
A090 09         del:    dex
A091 26FD               bne  del
A093 5A                 decb
A094 26F7               bne del1
A096 38                 pulx
A097 39                 rts

                ;-------------------------------
                ;       wait for busy
                ;-------------------------------
A098 5F         busy_delay:  clrb
A099 5A         _del:        decb
A09A 26FD                    bne  _del
A09C 39                      rts
A09D 496E737472 table:       dfb  "Instrumentation"
A0AD 20204546E67 table:      dfb  "Engineering"
A0BD 2020204465  table:      dfb  "Deparument"
A0CD 202020204B  table:      dfb  "KMITL"

0FF4                         org $0ff4
0FF4 20F1                    dfb $20,$0f1    ;vector XIRQ non markable interrupt
FFFE                         org $0fffe
FFFE A000                    dfb  $0a0,$00   ;vector reset
```

การทดลองจะถอดเอา CHIP CPU ในระบบพัฒนาออก และเสียบสายจากอิน - เซอร์กิต อีมูเลเตอร์เข้าไปแทนที่ CPU ตัวเดิม จากนั้นต่อพอร์ทอนุกรม RS - 232 (ในการทดลองนี้ใช้พอร์ท COM 2) เพื่อเชื่อมต่อกับระบบคอมพิวเตอร์ และในโครงงานนี้จะใช้โปรแกรมเทอร์มินอล Telix โดยตั้งอัตราบอดไว้ที่ 9600 เมื่อจ่ายไฟให้กับอิน - เซอร์กิต อีมูเลเตอร์และกด Reset ถ้าทั้งสองระบบเชื่อมต่อเข้าหากันโดยสมบูรณ์จะมี promt ขึ้น เพื่อแสดงการรอรับคำสั่งจากผู้ใช้

4.1 การทดสอบโปรแกรมมอนิเตอร์

4.1.1 การทดลองคำสั่ง dump

จากโปรแกรมที่ใช้ทดสอบ เมื่อทดลองใช้คำสั่ง dump จะปรากฏผลดังนี้

```
6811-in-circuit-emu > d a000
8e 00 ff 4f 5f 5a 26 fd  -  4a 26 f9 86 38 b7 14 00
c6 05 bd a0 8c 86 0c b7  -  14 00 bd a0 98 86 06 b7
14 00 bd a0 98 bd a0 7d  -  7e a0 2b ce a0 9d 86 80
bd a0 56 ce a0 ad 86 c0  -  bd a0 56 ce a0 bd 86 90
bd a0 56 ce a0 cd 86 d0  -  bd a0 56 bd a0 7d c6 78
bd a0 8c 7e a0 2b bd a0  -  5d bd a0 67 39 b7 14 00
bd a0 98 bd a0 98 39 86  -  10 b7 20 00 a6 00 b7 14
01 c6 46 bd a0 8c 08 7a  -  20 00 26 f0 39 86 01 b7

6811-in-circuit-emu >
```

เมื่อเปรียบเทียบกับข้อมูลที่อยู่ใน Eprom ของผู้ใช้ จะเห็นได้ว่ามีค่าตรงกับที่ dump มาดูบนหน้าจอคอมพิวเตอร์

4.1.2 การทดลองคำสั่ง Go

เมื่อใช้คำสั่ง G xxxx จะเป็นการสั่งให้รันโปรแกรมของผู้ใช้ตามแอดเดรสที่กำหนด ซึ่งจะหยุดการรันโปรแกรมได้อย่างเดียวคือ การกด Reset และจะแสดงการทำงานดังนี้

```
6811-in-circuit-emu > g a000
6811-in-circuit-emu >
```

4.1.3 การทดลองคำสั่ง Break

เมื่อใช้คำสั่ง Break ก่อนการรันโปรแกรม จะเป็นการอินาเบิลให้มีการหยุดการทำงานเกิดขึ้นเมื่อมีการรันโปรแกรม ซึ่งโดยปกติเมื่อเริ่มต้นให้อิน - เซอร์กิต อีมูเลเตอร์ทำงานจะเป็นการดิสเบิลการ Break โดยปริยาย (default) การแสดงการทำงานของคำสั่ง Break ร่วมกับคำสั่ง Go จะได้ดังนี้

```
6811-in-circuit-emu > b a033
set break ok
6811-in-circuit-emu > g a000
6811-in-circuit-emu >
```

4.1.4 การทดลองคำสั่ง Register

คำสั่งเกี่ยวกับ Register แบ่งออกเป็นสองอย่างคือ เกี่ยวกับการแก้ไขค่าภายในรีจิสเตอร์ และคำสั่งเกี่ยวกับการขอดูค่ารีจิสเตอร์ภายใน การแสดงการทำงานจะได้ดังนี้

```
6811-in-circuit-emu > r
a = 00  b = 00  x = a0ad  y = c6c1
6811-in-circuit-emu > r a 56
ok
6811-in-circuit-emu > r
a = 56  b = 00  x = a0ad  y = c6c1
6811-in-circuit-emu >
```

จะเห็นได้ว่าค่าในรีจิสเตอร์ A จะถูกเซตให้มีค่า 56

4.1.5 การทดลองคำสั่ง trace

การใช้คำสั่ง trace หรือ Single Step จะให้การทำงานโดยผู้ใช้กด " t " แล้วตามด้วย enter เมื่อเกิด promt ใหม่พร้อมกับคำว่า " ok " ขึ้นมา แสดงว่ามีการรันแบบ Single Step เกิดขึ้นแล้วดังนี้

```
6811-in-circuit-emu > t
ok
6811-in-circuit-emu >
```

4.1.6 การทดลองการป้อนคำสั่งที่ผิดรูปแบบ

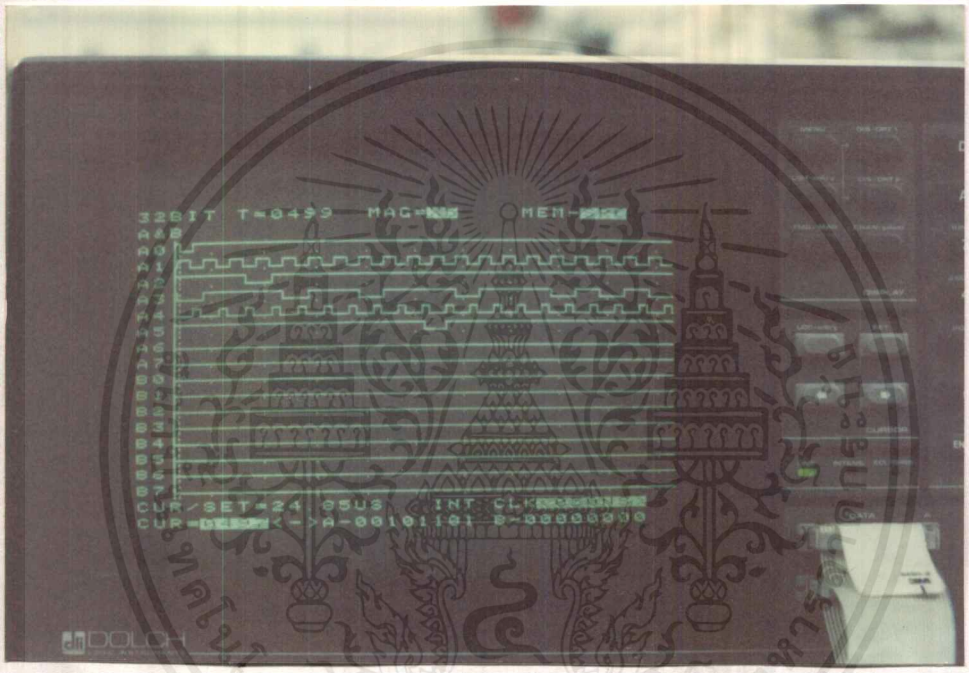
เมื่อผู้ใช้ป้อนคำสั่งผิด อิน - เซอร์กิต อิมูเลเตอร์จะส่ง " error " ออกมาที่จอคอมพิวเตอร์ดังนี้

```
6811-in-circuit-emu > ty emryh
error
6811-in-circuit-emu >
```

4.2 การทดสอบชุดฮาร์ดแวร์ของอิน - เซอร์กิต อีมูเลเตอร์

4.2.1 การทดสอบชุดฮาร์ดแวร์ที่เกี่ยวกับการอ่านเขียนข้อมูล

จากทฤษฎีการออกแบบในหัวข้อ 2.1 จะเห็นได้ว่าเมื่อมีการอ่านเขียนข้อมูลไปยังหน่วยความจำ จะพิจารณาจากไซเคิลการทำงานของคำสั่ง LDAA $XXXX หรือคำสั่ง STAA $XXXX ซึ่งจากการออกแบบวงจรการทำงานดังรูปที่ 2.9 จะใช้ลอจิกอนาไลเซอร์จับสัญญาณการทำงานได้ดังรูปที่ 4.2



รูปที่ 4.2 Timing diagram การทำงานของชุดควบคุมการอ่านเขียนข้อมูล
ที่ได้จากลอจิกอนาไลเซอร์

จากรูปสัญญาณแต่ละเส้นมีความหมายดังนี้

A0 คือ สัญญาณการอีนาเบิลการนับที่ได้จากวงจรเปรียบเทียบแอดเดรส

A1 คือ สัญญาณนาฬิกา E

A2 คือ สัญญาณ output 1

A3 คือ สัญญาณขา LIR

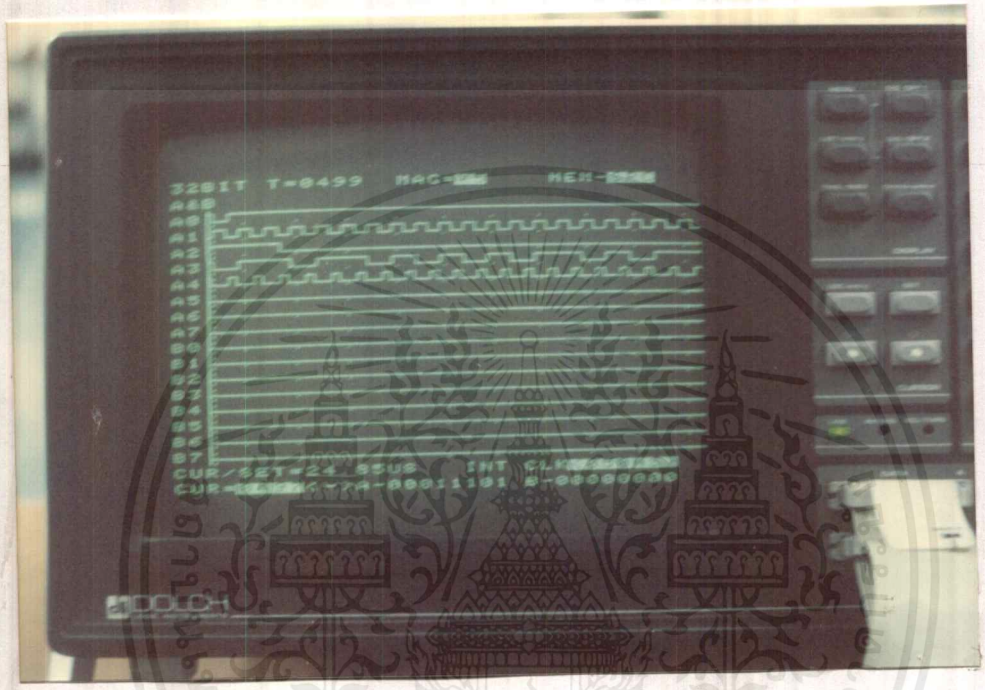
A4 คือ สัญญาณ address stobe

A5 คือ สัญญาณ read / write

ซึ่งจากรูปจะเห็นได้ว่ามี Timing diagram ตรงกับที่ออกแบบไว้ นั่นคือมีการสลับการทำงานไปยังเพจของผู้ใช้เพียง 1 ไซเคิลการทำงานของสัญญาณนาฬิกา E

4.2.2 การทดสอบชุดฮาร์ดแวร์ที่เกี่ยวกับการรันโปรแกรมของผู้ใช้

จากวงจรควบคุมการรันโปรแกรมของผู้ใช้ เมื่อทดสอบโดยใช้ลอจิกอนาไลเซอร์จับสัญญาณจะได้ดังรูปที่ 4.3



รูปที่ 4.3 Timing diagram การทำงานของชุดควบคุมการรันโปรแกรมของผู้ใช้ที่ได้จากลอจิกอนาไลเซอร์

จากรูปสัญญาณในแต่ละเส้นมีความหมายดังนี้

A0 คือ สัญญาณการอีนาเบิลการนับที่ได้จากวงจรเปรียบเทียบแอดเดรส

A1 คือ สัญญาณนาฬิกา E

A2 คือ สัญญาณ output 2

A3 คือ สัญญาณขา LIR

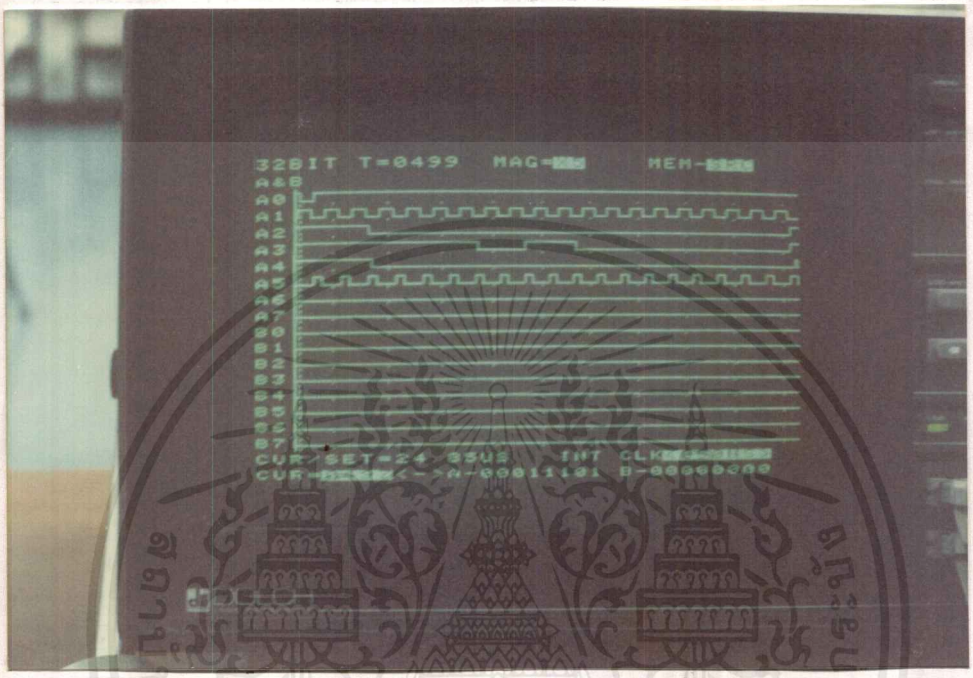
A4 คือ สัญญาณ address stobe

A5 คือ สัญญาณ read / write

จากรูปที่ 4.3 จะเห็นได้ว่าสัญญาณ output 2 นั้นมีสถานะ " 0 " ไปตลอด ตั้งแต่มีการนับสัญญาณนาฬิกา E ครบ 3 ลูก นั่นคือจะทำให้บัฟเฟอร์ของส่วนที่ติดต่อกับฮาร์ดแวร์ชุดพัฒนาถูกเชื่อมต่อกับซีพียูตั้งแต่ไซเคิลนี้เป็นต้นไป

4.2.3 การทดสอบชุดฮาร์ดแวร์ที่เกี่ยวกับการ break ของซีพียู

จากวงจรควบคุมการ break ซีพียู เมื่อใช้ลอจิกอนาไลเซอร์จับสัญญาณเมื่อมีสัญญาณ break เกิดขึ้นจะได้ดังรูปที่ 4.4



รูปที่ 4.4 Timing diagram การทำงานของชุดควบคุมการ break ของซีพียู ที่ได้จากลอจิกอนาไลเซอร์

จากรูปสัญญาณในแต่ละเส้นมีความหมายดังนี้

A0 คือ สัญญาณอีนาเบิลการนับเพื่อเริ่มให้มีการ break

A1 คือ สัญญาณนาฬิกา E

A2 คือ สัญญาณ output 3

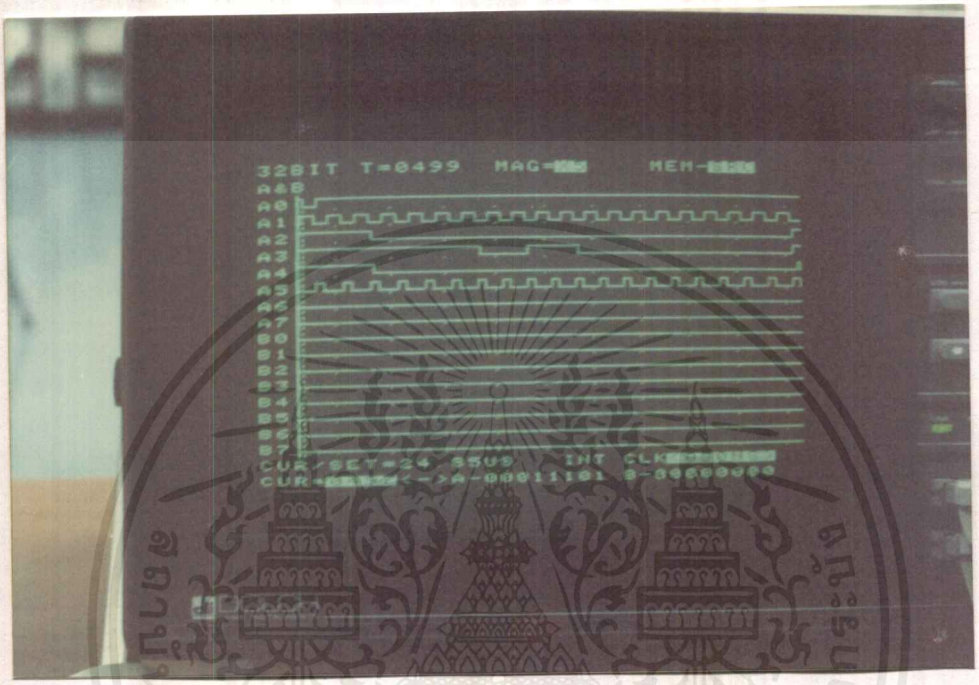
A3 คือ สัญญาณขา read / write

A4 คือ สัญญาณขา XIRQ

A5 คือ สัญญาณ address stobe

4.2.4 การทดสอบชุดฮาร์ดแวร์ที่เกี่ยวกับการรันโปรแกรมของผู้ใช้ทีละคำสั่ง

จากวงจรควบคุมการรันโปรแกรมของผู้ใช้ทีละคำสั่ง เมื่อใช้ลอจิกอนาไลเซอร์จับสัญญาณการทำงานจะได้ดังรูปที่ 4.5



รูปที่ 4.5 Timing diagram การทำงานของชุดควบคุมการรันโปรแกรมของผู้ใช้ทีละคำสั่งที่ได้จากลอจิกอนาไลเซอร์

จากรูปสัญญาณในแต่ละเส้นมีความหมายดังนี้

A0 คือ สัญญาณอีนาเบิลการนับที่ได้จากวงจรเปรียบเทียบแอดเดรส

A1 คือ สัญญาณนาฬิกา E

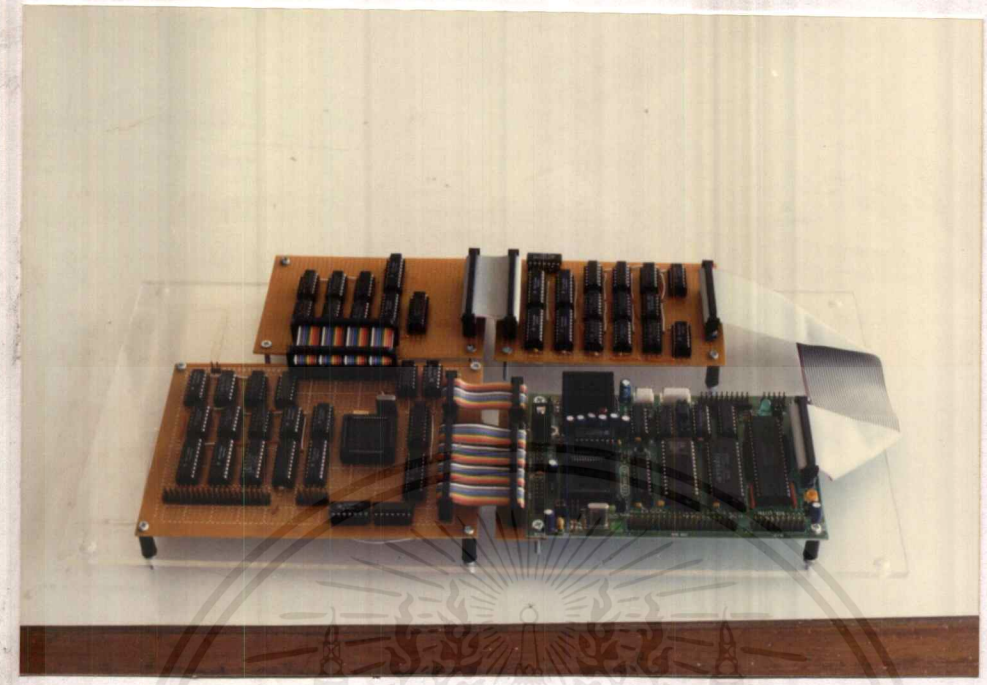
A2 คือ สัญญาณ output 3

A3 คือ สัญญาณ read / write

A4 คือ สัญญาณ XIRQ

A5 คือ สัญญาณ address stobe

จากรูปจะเห็นได้ว่า เมื่อมีการสลับการทำงานไปยังเพจของผู้ใช้ สัญญาณ XIRQ จะแอคทีฟทันที ซึ่งหมายความว่าซีพียูจะรันคำสั่งนั้นเพียงคำสั่งเดียวก็จะตอบสนองการอินเตอร์รัปท์ทันที



รูปที่ 4.6 อิน - เซอร์กิต อีมูเลเตอร์ที่ได้จัดทำขึ้น

4.3 วิจารณ์ผลการทดลองและแนวทางการพัฒนา

จากการทดสอบโปรแกรมและชุดควบคุมการทำงานของซีพียู จะเห็นได้ว่าการทำงานทั้งสองส่วนนั้นสอดคล้องกัน และสามารถทำงานได้ถูกต้องตรงตามที่ออกแบบไว้ทุกประการ

ปัญหาที่พบในโครงงานนี้คือ

1. การนำสัญญาณของอุปกรณ์บัฟเฟอร์ที่มีความไวไม่เท่ากัน ทำให้เกิดปัญหาในส่วนของบัสทำงานไม่สอดคล้องต่อเนื่องกัน เช่นกรณีของการสลับการทำงานเพียง 1 ไซเคิลเพื่ออ่านหรือเขียนข้อมูล สัญญาณของ address bus และ address stobe จะเข้าไปยังชุด target system ของผู้ไม่ใช้พร้อมกัน จะทำให้การอ่านหรือเขียนข้อมูลเกิดความผิดพลาดได้ เนื่องจากการอ่านค่าของ address bus ผิดพลาดไป ในกรณีนี้แก้ไขโดยการเพิ่มการนำสัญญาณของ address stobe ให้ช้าลง เพื่อรอให้สัญญาณ address bus มีค่าที่แน่นอนก่อนที่จะสโตรปเข้าหาชุดฮาร์ดแวร์ของผู้ใช้ ซึ่งจะทำให้ค่าที่ได้มีความถูกต้องสูงยิ่งขึ้น

2. กรณีของสัญญาณลอจิกรบกวนต่างๆ เช่น Gitch logic ที่เกิดจากธรรมชาติของสารกึ่งตัวนำในแต่ละไอซี และโดยเฉพาะการใช้ลักษณะการทำงานในแบบของ Ripple counter ซึ่งจะทำให้เกิดลักษณะของ Gitch logic ได้ง่ายขึ้นกว่าเดิม

3. ปัญหาของการเหลื่อมล้ำไม่เท่ากันของสัญญาณนาฬิกาในแต่ละจุด (clock skew) ที่ทำให้เกิดความผิดพลาดขึ้นในบางครั้ง แต่เนื่องจากซีพียู 68HC11 ทำงานด้วยความเร็วค่อนข้างต่ำ จึงไม่เกิดปัญหาในส่วนนี้มากนัก

แนวทางการพัฒนาต่อไป จะทำได้โดยการเขียนโปรแกรมให้สามารถทำงานได้หลากหลาย และอำนวยความสะดวกให้ดียิ่งขึ้นกว่าเดิม และในส่วนของฮาร์ดแวร์อาจสามารถยุบลอจิกต่างๆให้อยู่ในไอซีสำเร็จรูปที่สามารถโปรแกรมลอจิกได้ เช่น PAL หรือ GAL ซึ่งจะทำให้มีการกินกำลังงานไฟฟ้าที่ต่ำลง และลดความผิดพลาดที่เกิดจากปัญหาข้างต้นได้มาก

ไมโครคอนโทรลเลอร์ยังมีส่วนสนับสนุนส่วนอื่นๆ เช่น Input capture , Output compare , SPI หรือแม้กระทั่งในส่วนของ Analog to digital converter ซึ่งในโครงงานนี้ยังมิได้มีการนำเทคนิคใดๆ มาทำการอิมูเลทส่วนนี้ได้ รวมถึงในขณะที่ซีพียูอยู่ในโหมดอินเทอร์โรเกชันนั้น อาจเกิดปัญหาในส่วนของสัญญาณอินเตอร์รัปท์จากฮาร์ดแวร์ของผู้ใช้ ซึ่งจุดนี้อาจจะมีการเช็คการอินเตอร์รัปท์อยู่เสมอ เพื่อที่จะทำให้ตรวจสอบได้เหมือนซีพียูจริงมากยิ่งขึ้น

กิตติกรรมประกาศ

ขอขอบพระคุณ บิดา มารดา ของผู้จัดทำ ผู้ซึ่งให้กำเนิดและเล็งเห็นความสำคัญของการศึกษาของลูกๆเสมอมา จนผู้จัดทำมีวันนี้

อาจารย์ทรงชัย วีระทวีมาศ ผู้ซึ่งให้ความอนุเคราะห์ในส่วนของเครื่องมือ สถานที่จัดทำ รวมถึงการแนะนำติชม จนโครงงานสำเร็จลุล่วงไปด้วยดี

ศักรินทร์ กล้าหาญ
สุกิจ แก้วกลิ่นจันทร์

บรรณานุกรม

1. ชัยวัฒน์ ลิ้มพรจิตรวิไล , " เรียนรู้และใช้งานไมโครคอนโทรลเลอร์ 68HC11 " , บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน) .
2. สุภนันณ์ หิรัญยพิสุทธิกุล ,"ระบบพัฒนาไมโครโปรเซสเซอร์ระดับบอร์ด " , วิทยานิพนธ์วิศวกรรมศาสตร์มหาบัณฑิต . กรุงเทพฯ : คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย 2534.
3. เสกสันณ์ วัฒนะโชติ , " อินเซอร์กิตอีมูเลเตอร์สำหรับ Z80 และ 8085 " , วิทยานิพนธ์วิศวกรรมศาสตร์มหาบัณฑิต . กรุงเทพฯ : คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย 2535.
4. Frederick F. Driscoll , Robert F. coughlim , Robert S. Villanucci , " **Data Acquisition and Process Control with the M68HC11 Microcontroller** " , Macmillan Publishing Company , 1994.
5. G. J Lipovski , " **Single - and Multiple - Chip Microcomputer Interfacing** " , Prentice - Hall International , Inc . , 1998.
6. Gene H. Miller , " **Microcomputer Engineering** " , Prentice - Hall International , Inc. , 1993.
7. John B. Peatmam , " **Design with Microcontrollers** " , McGraw - Hill Book Company , 1998 .
8. Peter Spasov , " **Microcontroller Technology The 68HC11** " , Prentice - Hall International , Inc. , 1993.
9. Technical Data , " **HC11 MC68HC11A8** " , MOTOROLA INC.,1991 .
10. Theodore F. Bogart , Jr. , " **Introduction To Digital Circuits** " , McGraw - Hill International Editions , 1992 .

ภาคผนวก

ผ - 1 ส่วนประกอบภายในของ 68HC11

ภายในชิปไมโครคอนโทรลเลอร์ 68HC11 ประกอบด้วยส่วนหลักๆ 6 ส่วนคือ

1. ซีพียู
2. หน่วยความจำ
3. แอกคิวมูเลเตอร์-วอตช์ด็อก-ตัวตั้งเวลาหลัก
4. พอร์ทอินพุท เอาท์พุท
5. วงจรแปลงสัญญาณแอนาล็อกเป็นดิจิตอล
6. สไตรบ / แฮนด์เชก

ซีพียู ในส่วนนี้มีส่วนประกอบอีก 3 ส่วนคือ ส่วนควบคุมโหมดการทำงานของชิป (MODE) , ส่วนกำเนิดสัญญาณนาฬิกา (CLK) และส่วนลอจิกอินเตอร์รัปท์ (INT) โดยทั้งสามส่วนนี้จะได้รับการควบคุมการทำงานจากแก่นซีพียู

หน่วยความจำ ภายในชิป 68HC11 จะมีหน่วยความจำครบทั้ง 3 แบบคือ รอม แรม และอีอีพรอม ซึ่งการต่อและเรียกใช้หน่วยความจำนี้จะขึ้นอยู่กับส่วนซีพียูเป็นหลัก

แอกคิวมูเลเตอร์-วอตช์ด็อก-ตัวตั้งเวลาหลัก ในส่วนนี้จะมีการติดต่อกับพอร์ท A โดยใช้พอร์ท A เป็นทางผ่านของข้อมูล พัลส์แอกคิวมูเลเตอร์จะใช้พอร์ท PA7 ในขณะที่ส่วนตัวตั้งเวลาหลักจะใช้ PA3 - PA6 ในชิปยังมีวงจรวอตช์ด็อกเพื่อช่วยให้ชิปสามารถทำงานได้อย่างต่อเนื่อง แม้ว่าจะมีรีเซตอยู่บ่อยๆ ก็ตาม

พอร์ทอินพุทเอาท์พุท ใน 68HC11 จะมีพอร์ทอินพุทเอาท์พุทด้วยกัน 5 พอร์ท ดังนี้

พอร์ท A เป็นทั้งพอร์ทอินพุทและเอาท์พุท โดยมีการทำงานแยกกันคือ PA7 เป็นพอร์ทที่สามารถส่งผ่านข้อมูลได้สองทิศทาง ในขณะที่ PA3 - PA6 เป็นพอร์ทเอาท์พุทของตัวตั้งเวลาหลัก และ PA0 - PA2 เป็นพอร์ทอินพุท

พอร์ท B เป็นพอร์ทเอาท์พุท ในขณะที่พอร์ท C เป็นพอร์ทอินพุทและเอาท์พุท สามารถส่งข้อมูลได้ 2 ทิศทาง โดยที่พอร์ท C นี้จะทำหน้าที่เป็นบัสแอสเดรสไบต์ต่ำ และบัสข้อมูลด้วย โดยได้รับการควบคุมการทำงานแบบมัลติเพล็กซ์จึงไม่เกิดความสับสน

พอร์ท D เป็นพอร์ทที่ใช้ในการถ่ายทอดสัญญาณ จากส่วนเชื่อมต่ออุปกรณ์อนุกรมและส่วนสื่อสารข้อมูลอนุกรม จึงมีลักษณะเป็นพอร์ทอินพุทเอาท์พุทที่สามารถส่งผ่านข้อมูลได้สองทิศทาง ในขณะที่พอร์ท E เป็นพอร์ทอินพุทสำหรับวงจรแปลงสัญญาณแอนาล็อกเป็นดิจิตอล (Analog Digital Converter)

วงจรแปลงสัญญาณแอนาล็อกเป็นดิจิตอล (ADC) เป็นวงจรที่ช่วยเสริมให้ชิปไมโครคอนโทรลเลอร์มีประสิทธิภาพมากขึ้น ปกติแล้ว 68HC11 จะมีวงจร ADC 8 ช่อง การเรียกใช้งานวงจรส่วนนี้จะได้รับการควบคุมจากซีพียู

สโตรบ / แฮนด์เชก ในส่วนนี้จะทำงานร่วมกับส่วนขยายบัสด้วย ทั้งนี้เพื่อให้ซีพียูสามารถทำงานได้กับแอดเดรสถึง 16 บิต ที่ส่วนนี้จะมีสัญญาณที่สำคัญ ๆอยู่ 2 สัญญาณ คือ STRB / R / W และ STRA / AS โดยทั้งสองสัญญาณคือ สัญญาณสโตรบเพื่อให้ชิปทำการอ่านเขียนข้อมูลได้

และที่ส่วนนี้ยังมีวงจรแฮนด์เชก เพื่อตรวจสอบความพร้อมในการรับส่งข้อมูลของชิปกับส่วนอุปกรณ์ภายนอก

ผ - 2 รายละเอียดสัญญาณและการจัดขาของ 68HC11

ขาไฟเลี้ยงและกราวด์ (Vdd , Vss)

เป็นขาที่ใช้สำหรับจ่ายแรงดันเพื่อให้ 68HC11 ทำงาน โดยขา Vdd ต้องป้อนแรงดัน +5 โวลต์ ในขณะที่ขา Vss เป็นกราวด์ และเนื่องจาก 68HC11 มีโครงสร้างมาจากอุปกรณ์จำพวกซีมอส ดังนั้นจะต้องระมัดระวังอย่างมากในการจ่ายแรงดันไฟเลี้ยง แหล่งจ่ายไฟต้องมีคุณภาพค่อนข้างดี แรงดันคงที่และต้องต่อตัวเก็บประจุแบบเซรามิกค่า 0.1 ไมโครฟารัดคร่อมระหว่างขา Vdd และ Vss และต้องติดตั้งตัวเก็บประจุนี้ให้ใกล้ขาใดขาหนึ่งมากที่สุด

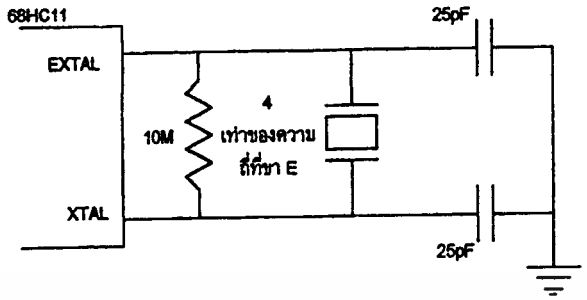
ขา RESET

เป็นขาอินพุทรับสัญญาณเพื่อให้ 68HC11 เริ่มต้นทำงานใหม่อันเกิดจากสภาวะที่ซีพียูเพิ่งได้รับไฟเลี้ยงให้เริ่มทำงาน หรือเกิดจากการทำงานภายในวงจรเกิดความผิดพลาด แล้วถูกตรวจจับได้โดยวงจรวอตช์ด็อก จากนั้นวงจรวอตช์ด็อกก็จะป้อนสัญญาณมาที่ขานี้เพื่อกระตุ้นให้ซีพียูเริ่มทำงานใหม่หมด โดยสัญญาณที่ป้อนเข้าขานี้ต้องมีสภาวะลอจิก " 0 "

สัญญาณ RESET ของ 68HC11 มีความพิเศษแตกต่างไปจากสัญญาณรีเซตของไมโครคอนโทรลเลอร์ตัวอื่น

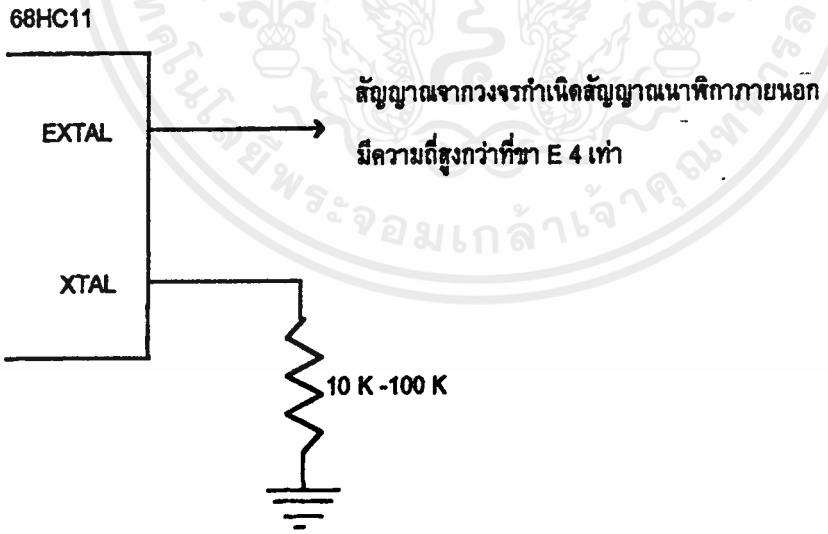
ขา XTAL , EXTAL

ทั้ง 2 ขานี้ถูกจัดไว้ให้ต่อกับคริสตอล หรือวงจรกำเนิดสัญญาณนาฬิกา เพื่อควบคุมวงจรกำเนิดสัญญาณนาฬิกาในชิป ความถี่ที่ปรากฏอยู่ที่ขานี้มีค่าสูงกว่าที่ขา E อยู่ประมาณ 4 เท่า การจัดวงจรเพื่อต่อคริสตอลเข้ากับขาทั้งสองนี้แสดงดังรูปที่ ผ - 1



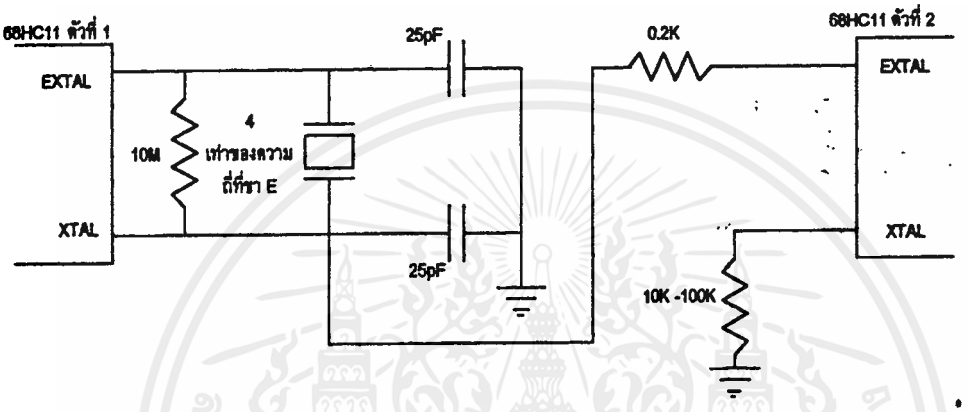
รูปที่ ผ - 1 การจัดวงจรเพื่อต่อคริสตอลให้แก่ 68HC11

ถ้าหากใช้วงจรกำเนิดสัญญาณนาฬิกาภายนอก ขา XTAL ต้องปล่อยลอยไว้หรือต่อตัวต้านทานค่าตั้งแต่ 10 - 100 กิโลโอห์มลงกราวด์ แล้วป้อนสัญญาณนาฬิกาเข้าที่ขา EXTAL โดยความถี่ที่ป้อนนี้ต้องสูงกว่าที่ขา E ประมาณ 4 เท่า ดังรูปที่ ผ - 2



รูปที่ ผ - 2 การต่อวงจรกำเนิดสัญญาณนาฬิกาภายนอกให้แก่ 68HC11

ในการต่อวงจรเพื่อกำเนิดสัญญาณนาฬิกา ควรต่อบัฟเฟอร์ที่มีอิมพีแดนซ์สูง เช่น ใช้ไอซีเบอร์ 74HC04 ทั้งนี้ก็เผื่อไว้สำหรับในกรณีที่ต้องการต่อวงจรกำเนิดสัญญาณนาฬิกานี้จ่ายให้แก่ไมโครคอนโทรลเลอร์ตัวอื่นด้วย อย่างไรก็ดี ถ้าหากต้องการต่อวงจรกำเนิดสัญญาณนาฬิกาเข้ากับไมโครคอนโทรลเลอร์เพียง 2 ตัว สามารถทำได้โดยต่อสัญญาณจากขา XTAL ผ่านตัวต้านทาน 200 โอห์ม เข้าที่ขา EXTAL ของไมโครคอนโทรลเลอร์อีกตัวหนึ่งก็ได้ ดังรูปที่ ผ - 3



รูปที่ ผ - 3 การต่อวงจรกำเนิดสัญญาณนาฬิกาภายนอกให้แก่ไมโครคอนโทรลเลอร์ 2 ตัว

อีกประการหนึ่งที่ต้องคำนึงถึงในการออกแบบแผ่นวงจรพิมพ์ในส่วนที่เกี่ยวข้องกับขาทั้งสองนี้ ค่าความจุของตัวเก็บประจุที่แสดงในวงจรรูปที่ ผ - 1 ถึง ผ - 3 เป็นค่าที่รวมกับค่าความจุแฝงที่เกิดขึ้นบนแผ่นวงจรพิมพ์ด้วย ดังนั้นการเลือกใช้ค่าตัวเก็บประจุต้องใช้ค่าที่ลดลงจากวงจรที่แสดงเล็กน้อย

ขา E

เป็นขาเอาท์พุทของวงจรกำเนิดสัญญาณนาฬิกาภายในชิป สามารถใช้เป็นฐานเวลาอ้างอิงได้โดยความถี่ที่ออกจากขานี้จะมีค่า 1 / 4 เท่าของความถี่อินพุทที่ขา XTAL และ EXTAL เมื่อใดที่ขา E นี้มีสภาวะเป็น " 0 " หมายความว่า ขณะนี้ซีพียูกำลังทำการประมวลผลภายในอยู่ ถ้าหากเป็น " 1 " หมายถึง ขณะนี้ข้อมูลได้รับการเรียกใช้จากซีพียู ถ้าหากไมโครคอนโทรลเลอร์อยู่ในโหมด STOP สัญญาณที่ขา E จะไม่มีออกมาหรือเกิดสภาวะ High Impedance นั่นเอง

ขา IRQ (Interrupt Request)

เป็นขาอินพุทสำหรับเรียกการอินเตอร์รัปท์แบบอะซิงโครนัสของชิป 68HC11 โดยการรับสัญญาณเพื่อการอินเตอร์รัปท์นี้สามารถจะเลือกได้ว่า จะรับสัญญาณขอบขาลง (Negative

Edge) หรือจะรับการทริกเป็นแบบระดับสัญญาณ โดยปกติจะกำหนดให้รับการอินเตอร์รัปท์ด้วยการทริกแบบระดับสัญญาณ นั่นคือ ทำงานที่ระดับลอจิก " 0 " เมื่อต่อใช้งานปกติต้องต่อตัวต้านทานค่า 4.7 กิโลโอห์มพูลอัปเข้ากับไฟเลี้ยง

ขา XIRQ (Non - Maskable Interrurt)

เป็นขาอินพุทสำหรับตอบรับการอินเตอร์รัปท์แบบนอน - มาสเคเบิล หลังจากที่มีการรีเซตซีพียูสามารถรับการอินเตอร์รัปท์ด้วยการทริกแบบระดับสัญญาณ โดยทำงานที่ลอจิก " 0 " และต้องต่อตัวต้านทานพูลอัปกับไฟเลี้ยงเข้าที่ขา XIRQ นี้ด้วย ในขณะที่ไม่มีการส่งสัญญาณอินเตอร์รัปท์

ขา MODA / LIR และ MODB / Vstby (Mode A / load instruction register และ Mode B / Standby voltage)

หลังจากที่มีการรีเซตไมโครคอนโทรลเลอร์ ขานี้ (ทั้ง MODA และ MODB) จะถูกใช้ให้เป็นขาสำหรับเลือกโหมดการทำงานของ 68HC11 ซึ่งเลือกได้เพียง 1 โหมด จาก 4 โหมด โดยแสดงการเลือกโหมดตามตารางที่ ผ - 1

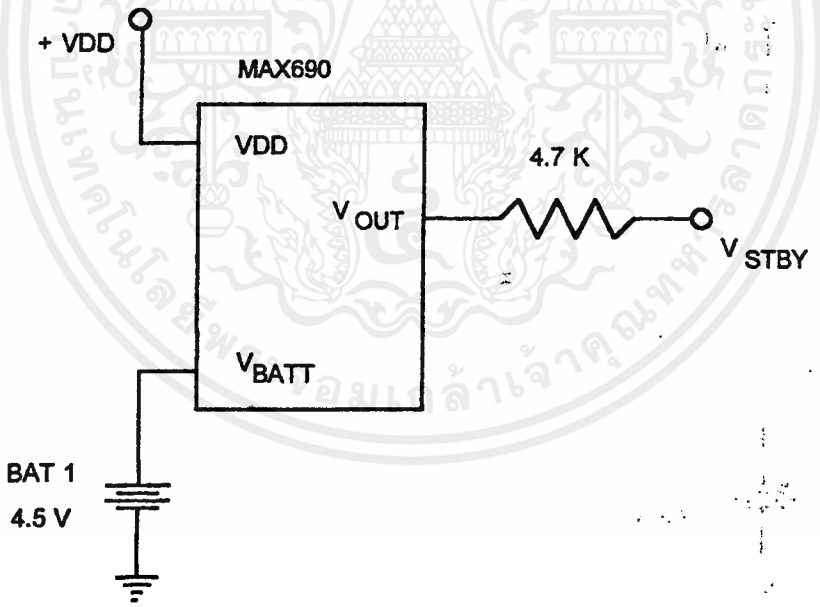
MODB	MODA	โหมดการทำงาน
1	0	ซิงเกิลชิป (Single chip)
1	1	มัลติเพล็กขยาย (Extended multiplexed)
0	0	บูตสแตร็ปพิเศษ (Special bootstrap)
0	1	ทดสอบพิเศษ (Special test)

ตารางที่ ผ - 1 การเลือกโหมดการทำงานของ 68HC11

หลังจากเลือกโหมดการทำงานแล้ว ขา MODA จะเปลี่ยนลักษณะการทำงานเป็นขา LIR แทน ซึ่งจะมีลักษณะเป็นขาเอาท์พุท ใช้ในการแสดงสภาวะว่า ขณะนี้ได้เริ่มจัดการกับคำสั่งแล้ว โดยจะแอกทีฟเป็นลอจิก" 0 " ในทันทีที่ขา E เริ่มส่งสัญญาณนาฬิกาพัลส์แรกของแต่ละ

คำสั่งออกมา ซึ่งก็คือ เมื่อซีพียูเริ่มเฟตช์ข้อมูลคำสั่ง ขา LIR นี้จะแอกทีฟทันที สัญญาณที่ขา LIR จะมีไว้ช่วยในการแก้ไขโปรแกรม (Program debugging)

เช่นเดียวกับขา MODA เมื่อขา MODB ถูกใช้ในการเลือกโหมดการทำงานไปแล้ว ที่ขา MODB นี้จะเปลี่ยนหน้าที่เป็นขาอินพุท Vstby แทน โดยอินพุท Vstby นี้เป็นอินพุทสำหรับจ่ายแรงดันเพื่อสแตนด์บายให้แก่แรมภายในชิป ถ้าหากแรงดันที่ขานี้มากกว่าแรงดันไฟเลี้ยงชิปประมาณ 0.7 โวลต์แล้ว หน่วยความจำแรมภายในชิปทั้ง 256 ไบต์ และส่วนลอจิกรีเซตจะรับแรงดันจากนี้แทนแรงดันที่จ่ายเข้าที่ขา Vdd นั่นคือ ใช้เป็นขาที่จ่ายแรงดันเพื่อเลี้ยงหน่วยความจำแรม เพื่อป้องกันข้อมูลภายในชิปสูญหาย เมื่อไม่มีการจ่ายแรงดันไฟเลี้ยงปกติ สำหรับส่วนลอจิกรีเซตจะต้องกำหนดให้มีสภาวะ " 0 " ก่อนที่จะปลดแรงดันออกจากขา Vdd และต้องรักษาสภาวะลอจิก " 0 " นี้ไปจนกว่าที่ขา Vdd จะได้รับไฟเลี้ยงในระดับปกติ ในรูปที่ ผ - 4 เป็นวงจรที่ใช้ในการเก็บรักษาข้อมูลในแรมของ 68HC11 โดยการใช้ชิป MAX690 และแบตเตอรี่ 4.5 โวลต์ เมื่อ Vdd หายไป MAX690 จะส่งแรงดันจากแบตเตอรี่ผ่านตัวต้านทาน 4.7 กิโลโอห์ม เข้าที่ขา MODB / Vstby ของ 68HC11 ก็จะสามารถเก็บรักษาข้อมูลในแรมได้



รูปที่ ผ - 4 การต่อ MAX690 เพื่อสร้างสัญญาณให้แก่ขา MODB / Vstby เพื่อใช้เก็บข้อมูลในแรมไม่ให้สูญหาย เมื่อปลดไฟเลี้ยง

ขา V_{RL} และ V_{RH} (A / D Converter Reference Voltages)

เป็นขาที่ใช้สำหรับต่อแรงดันอ้างอิง สำหรับวงจรแปลงสัญญาณแอนาล็อกเป็นดิจิตอลภายในชิป

ขา STRB / R / W (Strobe B และ Read / Write)

เป็นขาที่สามารถทำงานได้หลายหน้าที่ ขึ้นอยู่กับโหมดการทำงานของไมโครคอนโทรลเลอร์ ถ้าหาก 68HC11 อยู่ในโหมดซิงเกิลชิปขานี้จะเป็นขาเอาท์พุท STRB หรือเรียกว่าขาสโตรบสำหรับการแฮนด์เชกกับอุปกรณ์ภายนอกที่ต่อเข้าที่พอร์ทอินพุทเอาท์พุท

แต่ถ้าอยู่ในโหมดมัลติเพล็กซ์ขยาย ขานี้จะแสดงตนเป็นขา R / W ใช้ในการควบคุมทิศทางของการถ่ายเทข้อมูลภายนอกชิป ถ้าขานี้เป็นลอจิก " 0 " จะหมายความว่า ขณะนี้ซีพียูกำลังทำการเขียนข้อมูลลงไปในบัสข้อมูลภายนอก แต่ถ้าเป็นลอจิก " 1 " จะเป็นการแสดงว่า ขณะนี้กำลังอ่านข้อมูลเข้ามาเพื่อทำการประมวลผล ถ้าหากมีการเขียนข้อมูลอย่างต่อเนื่อง ขา R / W ก็จะเป็นลอจิก " 0 " ตลอดเวลา จนกว่าจะทำการเขียนข้อมูลเสร็จ นอกจากนี้เมื่อใช้งานขา R / W ร่วมกับขา E จะสามารถใช้เป็นสัญญาณอีนาเบิลสำหรับหน่วยความจำสแตติกแรมภายนอก

ขา STRA / AS (Strobe A และ Address Strobe)

เป็นขาอินพุทโดยรับสัญญาณเป็นแบบขอบขาของสัญญาณ ลักษณะการใช้งานขานี้จะขึ้นอยู่กับการกำหนดโหมดการทำงานของ 68HC11

สำหรับโหมดซิงเกิลชิปขานี้จะเป็นขา STRA คือ เป็นขาสโตรบสำหรับการแฮนด์เชกกับอุปกรณ์ภายนอกที่ต่อเข้ากับพอร์ทขนานอินพุทเอาท์พุท

ในโหมดมัลติเพล็กขยาย ขานี้จะเปลี่ยนลักษณะการทำงานเป็นขาเอาท์พุท AS โดยการใช้ในวงจรดีมัลติเพล็กซ์สัญญาณข้อมูลกับสัญญาณแอดเดรสที่พอร์ท C

ขาสัญญาณของพอร์ท

พอร์ท A , D และ E จะเป็นพอร์ทที่ไม่ขึ้นกับโหมดการทำงานของ 68HC11 นั่นคือไม่ว่า 68HC11 จะทำงานในโหมดใดก็ตาม พอร์ททั้ง 3 ยังคงมีลักษณะสัญญาณและการทำงานเช่นเดิม ในขณะที่พอร์ท B จะถูกกำหนดให้เป็นพอร์ทเอาท์พุท หาก 68HC11 ทำงานในโหมดซิงเกิลชิป แต่ถ้าเป็นโหมดมัลติเพล็กซ์ขยายจะกลายเป็นขาแอดเดรสไบต์สูง ส่วนพอร์ท C จะเป็นพอร์ทอินพุทเอาท์พุท ถ้า 68HC11 ทำงานในโหมดซิงเกิลชิป ถ้าหากทำงานใน

โหมดมัลติเพล็กซ์ขยาย พอร์ท C นี้จะใช้เป็นบัสมัลติเพล็กซ์ระหว่างแอดเดรส 8 บิตต่ำกับบัสข้อมูล

พอร์ท A (PA0 - PA7)

พอร์ท A นี้แต่ละสัญญาณสามารถทำงานได้ตั้งแต่ 2 - 3 ฟังก์ชัน โดยสามารถเป็นได้ทั้งอินพุทแคปเจอร์ (Input capture) 3 อินพุท คือ IC1 , IC2 และ IC3 เป็นเอาท์พุทของฟังก์ชันเปรียบเทียบ (Output compare) 4 ช่อง คือ OC2 , OC3 , OC4 และ OC5 หรือจะเป็นของฟังก์ชันเปรียบเทียบ (OC1)

อีกลักษณะหนึ่งคือ เป็นอินพุทของพัลส์แอกคิวมูเลเตอร์ ถ้าหากขาสัญญาณของพอร์ท A ไม่ได้ถูกใช้งานในฟังก์ชันการตั้งเวลา สามารถใช้งานเป็นพอร์ทอินพุทเอาท์พุท เพื่อเชื่อมต่อในงานทั่ว ๆ ไปได้

พอร์ท B (PB0 - PB7)

เมื่อ 68HC11 ทำงานในโหมดซิงเกิลชิป ขาสัญญาณทั้งหมดจะเป็นขาเอาท์พุทของพอร์ทเอาท์พุท และใช้งานร่วมกับสัญญาณสโตรบเอาท์พุท จะมีพัลส์เอาท์พุทมาปรากฏที่ขา STRB ในทุกๆช่วงเวลาที่มีการเขียนข้อมูลออกมาที่พอร์ท B ถ้าหากทำงานในโหมดมัลติเพล็กซ์ขยาย ขาสัญญาณทั้งหมดของพอร์ท B จะทำหน้าที่เป็นขาแอดเดรส 8 บิตสูง (A8 - A15)

พอร์ท C (PC0 - PC7)

ถ้า 68HC11 ทำงานในโหมดซิงเกิลชิป ขาของพอร์ท C ทั้งหมดจะเป็นขาพอร์ทอินพุทเอาท์พุท ถ้าเป็นพอร์ทอินพุท พอร์ท C สามารถที่จะแลตช์สัญญาณอินพุทที่ป้อนเข้ามานี้ได้ โดยการป้อนสัญญาณเข้าที่ขา STRA สัญญาณพอร์ท C มักใช้ในการแฮนด์เชกของการเชื่อมต่อพอร์ทขนาน โดยในขณะเดียวกันขา STRB จะถูกใช้เป็นสายควบคุมการแฮนด์เชก

แต่ถ้าทำงานในโหมดมัลติเพล็กซ์ขยาย ขาสัญญาณพอร์ท C จะสามารถเป็นได้ทั้งขาของแอดเดรสและข้อมูลโดยการมัลติเพล็กซ์ ปกติขานี้จะเป็นสัญญาณแอดเดรส A0 - A7 ในแต่ละไซเคิลการทำงานของไมโครคอนโทรลเลอร์ แต่ถ้าหากขา E แอกทีฟเป็น " 1 " สัญญาณที่ขานี้จะกลับเป็นขาสัญญาณข้อมูล D0 - D7 โดยทิศทางการเข้าออกของข้อมูลที่พอร์ท C จะแสดงออกมาที่ขาสัญญาณ R / W

พอร์ท D (PD0 - PD5)

ขาสัญญาณพอร์ท D ทั้ง 6 ขา ปกติจะใช้เป็นขาสัญญาณอินพุทเอาท์พุท และสามารถรองรับในการใช้งานเชื่อมต่อสัญญาณข้อมูลอนุกรม (SCI) และเชื่อมต่ออุปกรณ์อนุกรม (SPI) ด้วย ถ้าหากมีการอีนาเบิลในส่วนนี้ให้ทำงาน

ขา PD0 เป็นขาอินพุทรับข้อมูล (RxD) เมื่อใช้ในการเชื่อมต่อสัญญาณข้อมูลอนุกรม (SCI)

ขา PD1 เป็นขาเอาท์พุทส่งข้อมูล (TxD) เมื่อใช้ในการเชื่อมต่อสัญญาณข้อมูลอนุกรม (SCI)

ขา PD2 - PD5 ใช้สำหรับเชื่อมต่ออุปกรณ์อนุกรม (SPI) โดย PD2 เป็นขาสัญญาณมาสเตอร์อินสเลฟเอาท์ (Master In Slave Out : MISO) PD3 เป็นขาสัญญาณมาสเตอร์เอาท์สเลฟอิน (Master Out Slave In : MOSI)

PD4 เป็นขาสัญญาณนาฬิกาอนุกรม (SCK) และขา PD5 เป็นขาอินพุทเลือกสเลฟ (SS)

พอร์ท E (PE0 - PE7)

พอร์ท E เป็นพอร์ทอินพุท และเป็นขาอินพุทสำหรับวงจรแปลงสัญญาณแอนาล็อกเป็นดิจิตอล สัญญาณอินพุทที่จะป้อนเข้าที่พอร์ท E นี้ ต้องมีความแน่นอนของสัญญาณสูงพอสมควร เพราะความแม่นยำของการแปลงสัญญาณแอนาล็อกเป็นดิจิตอล จะขึ้นอยู่กับความเที่ยงตรงของสัญญาณอินพุทเป็นหลัก

ผ - 3 รีจีสเตอร์ของ 68HC11

ซีพียูของไมโครคอนโทรลเลอร์ 68HC11 จะมีรีจีสเตอร์ใช้งานอยู่ 7 ตัว

1. แอกคิวมูเลเตอร์ A และ B
2. แอกคิวมูเลเตอร์ D
3. รีจีสเตอร์อินเด็กซ์ IX
4. รีจีสเตอร์อินเด็กซ์ IY
5. รีจีสเตอร์ตัวชี้สแต็ก (Stack Pointer : SP)
6. รีจีสเตอร์โปรแกรมเคาน์เตอร์ (Program Counter : PC)
7. รีจีสเตอร์รหัสเงื่อนไข (Condition Code Register : CCR)

แอกคิวมูเลเตอร์ A และ B

เป็นรีจีสเตอร์ขนาด 8 บิต ที่ใช้ในการเก็บค่าโอเปอแรนด์และผลของการคำนวณทางคณิตศาสตร์หรือผลจากการจัดการข้อมูลโดยตัวซีพียู ในการประมวลผลทางคณิตศาสตร์หรือลอจิก จะต้องนำข้อมูลเหล่านั้นมาเก็บในรีจีสเตอร์ทั้งสองนี้ จึงจะสามารถประมวลผลได้

แอกคิวมูเลเตอร์ D

เป็นรีจีสเตอร์ขนาด 16 บิต ใช้ในการประมวลผลและเก็บค่าจากการคำนวณทางคณิตศาสตร์และลอจิก แอกคิวมูเลเตอร์ D ก็เกิดมาจากการรวมกันของแอกคิวมูเลเตอร์ A และ B จึงทำให้มีขนาด 16 บิต

รีจีสเตอร์อินเด็กซ์ IX

เป็นรีจีสเตอร์ขนาด 16 บิต ใช้ในการชี้ตำแหน่งแอดเดรส เพื่อเข้าไปจัดการประมวลผลกับข้อมูลในแอดเดรสนั้นๆ นอกจากนั้น IX สามารถใช้เป็นตัวนับหรือรีจีสเตอร์เก็บข้อมูลชั่วคราวได้ด้วย

รีจีสเตอร์อินเด็กซ์ IY

เป็นรีจีสเตอร์ขนาด 16 บิต มีหน้าที่เหมือนกับ IX แต่จะแตกต่างกันตรงที่ในทุกคำสั่งที่ต้องเกี่ยวข้องกับ IY จะต้องมีข้อมูลไบต์พิเศษของรหัสแมชชีน และมีไซเคิลพิเศษของช่วงเวลาในการเอกซีคิวต์คำสั่งด้วย จึงทำให้คำสั่งที่มี IY ไปเกี่ยวข้องต้องมีขนาดเพิ่มขึ้นอย่างน้อย 2 ไบต์ขึ้นไปหรือที่เรียกว่า พรีไบต์ (Prebyte)

รีจีสเตอร์ตัวชี้สแต็ก : SP

เป็นรีจีสเตอร์ขนาด 16 บิต ใช้เก็บแอดเดรสบนสแต็ก (Stack) โดยสแต็กใน 68 HC11 นี้จะมีลักษณะการเก็บข้อมูลเข้าและนำออกมาเป็นแบบ LIFO (Last - In - First - Out) หรือข้อมูลที่เข้าไปเก็บในสแต็กหลังสุด เมื่อจะเรียกออกมาจะเรียกออกมาก่อน สแต็กใช้เก็บข้อมูลของรีจีสเตอร์ เมื่อต้องมีการนำรีจีสเตอร์ไปทำงานในโปรแกรมย่อยอื่น หรือต้องใช้ไปในการตอบสนองการอินเตอร์รัปท์ เพื่อเป็นการป้องกันข้อมูลเดิมสูญหายจึงต้องเก็บข้อมูลนั้นไว้ในหน่วยความจำสำรองแห่งหนึ่ง ซึ่งก็คือสแต็กนั่นเอง ทุกครั้งที่มีการเก็บข้อมูลลงสแต็กค่าของ SP จะลดลง ในทางตรงกันข้ามถ้าเรียกข้อมูลออกจากสแต็กค่าของ SP ก็จะเพิ่มขึ้น

รีจีสเตอร์โปรแกรมเคาน์เตอร์ : PC

เป็นรีจีสเตอร์ขนาด 16 บิต ใช้เก็บค่าของแอดเดรสของคำสั่งถัดไปที่ซีพียูจะไปทำการเอกซีคิวต์

รีจีสเตอร์รหัสเงื่อนไข : CCR (Condition Code Register)

เป็นรีจีสเตอร์ขนาด 8 บิต ในแต่ละบิตจะแสดงความหมายของผลจากการกระทำคำสั่งที่เพิ่งจะเอกซีคิวต์ไปของซีพียู แต่ละบิตของ CCR เป็นอิสระต่อกัน จึงสามารถตรวจสอบสถานะได้โดยโปรแกรม และยังสามารถนำผลการตรวจสอบนั้นไปดำเนินการต่อได้ด้วย

รายละเอียดของแต่ละบิตใน CCR มีดังนี้

บิต Carry / Borrow (C) : บิตนี้จะเซตเมื่อซีพียูทำการประมวลผลทางคณิตศาสตร์แล้วเกิดการทดค่า (Carry) หรือยืมค่า (Borrow) บิตนี้สามารถที่จะใช้งานร่วมกับคำสั่งการหมุน (Rotate) และเลื่อน (Shift) ข้อมูลได้ หรือที่เรียกว่า บิตทด

บิต Over Flow (V) : บิตนี้จะเซตเป็น " 1 " เมื่อซีพียูกระทำคำสั่งคณิตศาสตร์แล้วเกิดค่าที่เกินออกมานอกเหนือจากเงื่อนไขดังกล่าว บิตนี้จะเป็น " 0 "

บิต Zero (Z) : บิตนี้จะเซตเมื่อผลของการกระทำทางคณิตศาสตร์ หรือลอจิกหรือการประมวลผลข้อมูลแล้วทำให้เกิดเป็นค่าศูนย์ขึ้นมา

บิต Negative (N) : ถ้าหากผลของการกระทำทางคณิตศาสตร์หรือลอจิก หรือการประมวลผลข้อมูลแล้ว ทำให้เกิดค่าเป็นลบ บิตนี้จะเซตผลลัพธ์ที่เป็นลบสามารถสังเกตุได้จากบิตที่มีนัยสำคัญสูงสุด (MSB) มีค่าเป็น " 1 "

บิต I - Interrupt Mask (I) : สามารถเซตบิตนี้ให้เป็น " 1 " ได้ 2 วิธีคือโดยวิธีการทางฮาร์ดแวร์ และโดยการใช้คำสั่งที่ใช้ในการดิสเอเบิลการอินเตอร์รัปท์แบบมาสเคเบิลทั้งภายในและภายนอก

บิต Half Carry (H) : จะเซตเป็น " 1 " เมื่อซีพียูกระทำคำสั่งทางคณิตศาสตร์ เช่น ADD , ABA หรือ ADC แล้วเกิดการทดข้ามจากบิตที่ 3 มายังบิตที่ 4

บิต X - Interrupt Mask (X) : บิตนี้จะถูกเซตด้วยวิธีการทางฮาร์ดแวร์เท่านั้น โดยการป้อนสัญญาณเข้าที่ขา RESET และ XIRQ และจะรีเซตบิตนี้ด้วยการใช้คำสั่ง TAP หรือ RTI เท่านั้น

บิต Stop Disable (S) : บิตนี้จะเซตเมื่อต้องการดิสเอเบิลคำสั่ง STOP และถ้ารีเซตบิตนี้ ก็จะเป็นการอีนาเบิลคำสั่ง STOP บิตนี้จะถูกควบคุมโดยโปรแกรม และเมื่อบิตนี้ถูกเซตจะทำให้คำสั่ง STOP มีผลเช่นเดียวกับคำสั่ง NOP