การทดลองไมโครคอนโทรลเลอร์ 68HC11

LAB MICROCONTROLLER 68HC11

โดย

นางสาวทุติยา สุดสงวน

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิศวกรรมการวัดคุมทางอุตสาหกรรม
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2541

ปริญญานิพนธ์ ปีการศึกษา 2541

ภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม
สาขาวิชา วิศวกรรมการวัดคุมทางอุตสาหกรรม
คณะ วิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง LAB MICROCONTROLLER 68HC11

ผู้จัดทำ
นางสาวทุติยา สุดสงวน รหัสประจำตัว 39013384

.....อาจารย์ที่ปรึกษา
(อาจารย์ทรงชัย วีระทวีมาศ)

หัวข้อปริญญานิพนธ์ การทดลองไมโครคอนโทรลเลอร์ 68HC11

นักศึกษา นางสาวทุติยา สุดสงวน 39013384

อาจารย์ที่ปรึกษา อาจารย์ทรงชัย วีระทวีมาศ

ระดับการศึกษา วิศวกรรมศาสตรบัณฑิต
สาขาวิศวกรรมการวัดคุมทางอุตสาหกรรม

ปีการศึกษา 2541

บทคัดย่อ

ปัจจุบันไมโครคอนโทรลเลอร์ 68HC11 ได้รับความนิยมเพิ่มมากขึ้นแต่การเรียนรู้ 68HC11 นี้ยังทำได้ยาก เนื่องจากอุปกรณ์การทดลองต่าง ๆ นั้นยังมีราคาแพงมาก

ในโครงงานนี้จึงได้จัดสร้างอุปกรณ์เพื่อใช้ประกอบการทดลอง โดยประกอบด้วยแผงวงจรไมโครคอมพิวเตอร์แบบแผ่นพิมพ์เดียว (Single Board) ซึ่งมีไมโครคอนโทรลเลอร์ เบอร์ 68HC11 เป็นตัวประมวลผล โดยมีคีย์บอร์ดเป็นอินพุต และ LCD เป็นตัวแสดงผล นอกจากนั้นยังสามารถรับส่งข้อมูลกับไมโครคอมพิวเตอร์ได้โดยผ่านสายสัญญาณ RS-232 และสามารถตรวจสอบโปรแกรมโดยใช้ไมโครคอมพิวเตอร์เป็นตัวแสดงผลได้ด้วย

ส่วนเอกสารประกอบการทดลอง ประกอบด้วยการทดลองทั้งหมด 10 การทดลอง เพื่อให้ผู้ใช้ได้ศึกษาอย่างมีประสิทธิภาพมากยิ่งขึ้น

Thesis : LAB MICROCONTROLLER 68HC11

Student : Thutiya Soodsa-nguan 39013384

Advisor : Songchai Weerathaweemas

Education Level : Bachelor of industrial instrument engineering

Education Year : 2541

Abstract

Microcontroller No.68HC11 is popular device. But it difficult to learn about it because this device and experiment equipment are high price device.

This project provides equipment for experiment and document. This equipment consists of single board, microcontroller that have processing unit (68HC11), input unit (keybord) and Display with LCD, other can be comunucation pass to port RS-232. This equipment can or receive and transmit data between microcomputer. You can check program and display by microcomputer also.

Part of document consists of 10 sections to experiment for user learns about 68HC11 to high efficiency to go on.

กิตติกรรมประกาศ

ในการทำโครงงานนี้ ต้องขอขอบพระคุณทุก ๆ ท่านเป็นอย่างยิ่งที่ให้ความช่วยเหลือ ความเมตตากรุณาและความร่วมมือในทุก ๆ ด้านของการทำโครงงานนี้ รวมทั้งปริญญานิพนธ์ ฉบับนี้ ให้สำเร็จบรรลุล่วงตามวัตถุประสงค์ที่ตั้งไว้ จึงขอขอบพระคุณไว้ ณ. โอกาสนี้

นางสาวทุติยา สุดสงวน 39013384

สารบัญ

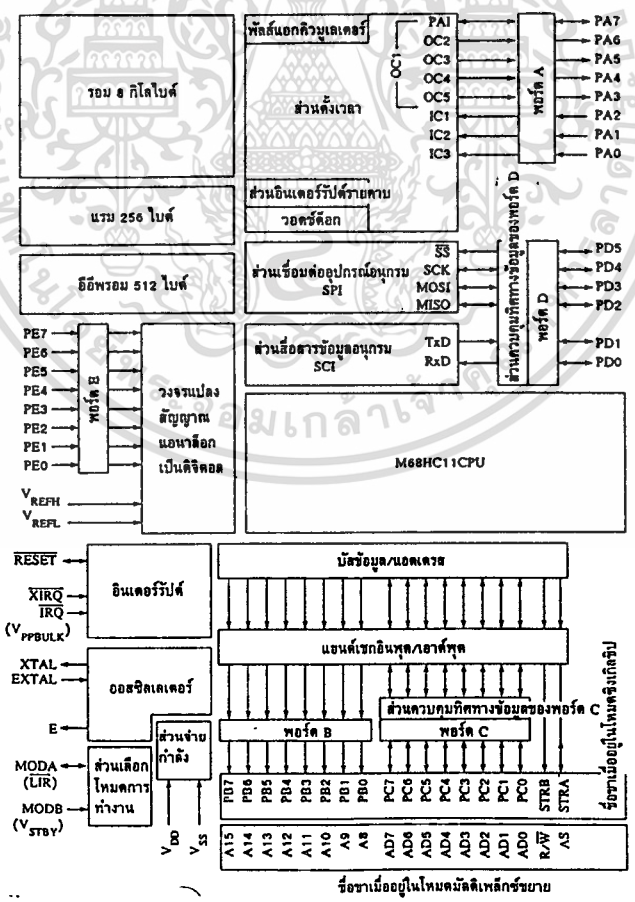
สารบัญ (ต่อ)

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของโครงงาน

ในการออกแบบไมโครคอมพิวเตอร์เพื่อใช้ในงานควบคุม หรืองานเฉพาะอย่างโดยส่วนใหญ่จะใช้ไอซีประเภทไมโครคอนโทรลเลอร์เพราะมีการทำงานที่รวดเร็วและเที่ยงตรงแม่นยำ นอกจากนั้นยังสามารถเปลี่ยนเงื่อนไขของงานโดยเพียงแต่เปลี่ยนซอฟต์แวร์เท่านั้น เหตุผลอีกประการหนึ่งที่นิยมใช้ไมโครคอนโทรลเลอร์คือมีการรวมอุปกรณ์สนับสนุน ทางด้านอินพุตเอาต์พุตและอินเตอร์เฟสต่าง ๆ ไว้อย่างพร้อมมูล

68HC11 เป็นไมโครคอนโทรลเลอร์อีกตัวหนึ่ง ของบริษัท โมโรล่า ที่ได้ถูกออกแบบมาโดยรวบรวมอุปกรณ์ที่จำเป็นในการใช้งาน ไว้ภายในตัวของมันเองได้แก่ อินพุตเอาต์พุต วงจรตั้งเวลา วงจรนับ วงจรอ่าน สัญญาณแบบอะนาล็อก ระบบการอินเตอร์รัพต์แบบเวกเตอร์อินเตอร์รัพต์ และระบบป้องกันความผิดพลาดที่จะเกิดขึ้นจากโปรแกรมดังรูปที่ 1.1



รูปที่ 1.1 บล็อกไดอะแกรมแสดงโครงสร้างของ 68HC11

คุณสมบัติของ 68HC11

- เป็นซีพียูขนาด 8 บิต
- มีหน่วยความจำภายในสำหรับเก็บโปรแกรมชนิดรอมขนาด 4, 8 หรือ 12 กิโลไบต์
- มีหน่วยความจำภายในสำหรับเก็บข้อมูลชนิดอีอีพรอม 512 ไบต์ หรือ 2 กิโลไบต์
- มีหน่วยความจำภายในสำหรับเก็บข้อมูลชนิดแรม 192, 256 หรือ 512 ไบต์
- มีวงจรตั้งเวลา/วงจรนับขนาด 16 บิต
- มีวงจรนับพัลส์ขนาด 8 บิต
- มีวงจรรับส่งข้อมูลอนุกรมได้สองทิศทาง (Universal Synchronous Receiver Transmitter : USRT) ขนาดที่ใช้ในการติดต่อสำหรับวงจรการส่งข้อมูลอนุกรมเป็นแบบมัลติโปรเซสเซอร์
- มีวงจรเปลี่ยนสัญญาณจากอะนาลอกเป็นดิจิตอล 8 ช่อง
- มีวงจรรีลไทม์อินเตอร์รัพต์
- มีระบบวอตช์ด็อกที่ตั้งเวลาได้
- มีวงจรตรวจสอบสัญญาณนาฬิกาให้กับซีพียู
- มีระบบอินเตอร์รัพต์ 2 ระดับ จาก 21 แหล่งสามารถติดต่อกับหน่วยความจำภายนอกได้ 64 กิโลไบต์

68HC11 เป็นไมโครคอนโทรลเลอร์ที่แบ่งออกเป็นหลายรุ่น โดยแบ่งตามลักษณะของหน่วยความจำที่อยู่ภายในซีพียู ซึ่งแสดงตามตารางที่ 1 รูปแบบตัวถังของชิพตระกูลนี้มีด้วยกัน 3 แบบ คือ

PLCC (Plastic Leaded Chip Carrier) เป็นตัวถังรูปสี่เหลี่ยม 52 ขา

DIP (Dual In-line Package) มีขนาด 48 ขา ในตัวถังแบบนี้จะไม่มีขาสัญญาณของพอร์ต PE4-PE7

QFP (Quad Fiat Pack) เป็นตัวถังรูปสี่เหลี่ยมขนาด 64 ขา

ตารางที่ 1 แสดงตระกูลของ 68HC11

เบอร์ชิพ	รอม	อีอีพรอม	แรม	รีจิสเตอร์ CONFIG	หมายเหตุ
MC68HC11A8	8 k	512	256	$0F	เป็นตัวมาตรฐานของชิพตระกูลนี้
MC68HC11A1	0	512	256	$0D	เหมือน MC68HC11A8 แต่ไม่รีรอม
MC68HC11A 0	0	0	256	$0C	เหมือน MC68HC11A8 แต่ไม่มีทั้งรอมและอีอีพรอม

เบอร์ชิพ	รอม	อีอีพรอม	แรม	รีจิสเตอร์ CONFIG	หมายเหตุ
XC68HC11B8	8 k	512*	256	$0F	เป็นรุ่นทดลองรุ่นแรก
XC68HC11B1	0	512*	256	$CD	เหมือน XC68HC11B8 แต่ไม่มีรอบ
XC68HC11B0	0	0	256	$0C	เหมือน XC68HC11B8 แต่ไม่มีทั้ง ROM และ RAM
MC68HC11E9	12 k	512	512	$0F	มี4 แคปเจอร์อินพุต แรมขนาดใหญ่ และรอม12 กิโลไบต์
MC68HC11E1	0	512	512	$0D	เหมือน MC68HC11E9 แต่ไม่มีรอบ
MC68HC11E0	0	0	512	$0C	เหมือน MC68HC11E9 แต่ไม่มีทั้ง รอม และ อีอีพรอม
MC68HC11E2	0	2 k**	256	$FF	ไม่สามารถเพิ่มรอบได้
MC68HC11D3	4 k	0	192	N/A	เป็นรุ่นประหยัดไม่มีเอทูดีคอนเวอร์เตอร์และหน่วยความจำขนาดเล็ก

* สำหรับใน 68HC11 อนุกรม B นั้น อีอีพรอมภายในตัวไมโครคอนโทรลเลอร์ต้องการไฟบวก 19 โวลต์จากภายนอกสำหรับการโปรแกรม

** สามารถปรับเป็น 4 กิโลไบต์ โดยการกำหนดที่บิตภายในรีจิสเตอร์คอนฟิก (config register)

1.2 วัตถุประสงค์และขอบเขตของโครงงาน

1. ศึกษาคุณสมบัติและรายละเอียดส่วนต่าง ๆ ของไมโครคอนโทรลเลอร์ เบอร์ 68HC11

2. สามารถนำความรู้ที่ได้ศึกษามาจัดทำเป็นชุดทดลองไมโครคอนโทรลเลอร์ เบอร์ 68HC11 เพื่อใช้ในการศึกษา

3. จัดทำเป็นคู่มือประกอบให้กับผู้ที่สนใจที่จะนำเอาไมโครคอนโทรลเลอร์ เบอร์ 68 HC11 ไปใช้งาน

บทที่ 2

ทฤษฎีและหลักการ

2. ทฤษฎีและหลักการ

2.1 โหมดการทำงาน

68HC11 มีโหมดการทำงาน 4 โหมด สามารถกำหนดที่ขาสัญญาณ MODA และ MODB โดยมีลักษณะการทำงานดังนี้

1. SINGLE CHIP OPERATING MODE โหมดนี้โปรแกรมควบคุมการทำงานจะอยู่ภายในตัวซีพียู และขาสัญญาณต่าง ๆ สามารถใช้เป็นทั้งอินพุตและเอาต์พุต

2. EXPANDED MULTIPEXED OPERATING MODE การทำงานในโหมดนี้ โปรแกรมควบคุมการทำงานจะอยู่ภายนอก โดยใช้พอร์ตอินพุตเอาต์พุตเป็นแอดเดรส และข้อมูลในการติดต่อกับหน่วยความจำภายนอก และสัญญาณควบคุมต่าง ๆ

3. SPECIAL BOOTSTRAP OPERATING MODE การทำงานในโหมดนี้จะมีลักษณะคล้ายกับ SINGLE CHIP OPERATING MODE คือมีโปรแกรมควบคุมการทำงาน อยู่ภายในซีพียู แต่ตำแหน่งในการรันและเวกเตอร์อินเตอร์รัพต์จะต่างกันและมีลักษณะ การทำงานที่พิเศษกว่าSINGLECHIP OPERATING MODE โดยสามารถป้องกันการ คัดลอกหรือเลียนแบบได้

4. SPCIAL TEST OPERATING MODE โหมดนี้จะใช้ในการทดสอบการทำงานต่าง ๆ ตามโรงงานที่ผลิตเป็นจำนวนมาก โดยมีการทำงานคล้ายกับ EXPANDED MULTIPEXED OPERATING MODE

ลักษณะของสัญญาณและการจัดขา

$\overline{RESET}$ เป็นขาที่ทำงานแบบสองทิศทางคือ เป็นอินพุตสำหรับการทำให้ซีพียูเริ่มต้นการทำงาน และเป็นเอาต์พุตแสดงถึงการทำงานภายในผิดพลาดซึ่งเกิดขึ้นได้ 3 กรณีคือ

- ความผิดพลาดจากสัญญาณนาฬิกา
- ความผิดพลาดจากโปรแกรมในการใช้ระบบวอตช์ด็อก และ
- การเอ็กซีคิวต์ออปโค้ดผิดพลาด

XTAL, EXTAL เป็นขาอินพุตสำหรับต่อคริสตอลเพื่อผลิตสัญญาณนาฬิกาให้กับซีพียู หรือจะใช้สัญญาณนาฬิกาจากภายนอกป้อนเข้าที่ขา EXTAL โดยตรงก็ได้ โดยทีมีตัวต้านทาน 10-100 กิโลโอห์มต่อลงกราวด์เพื่อป้องกันสัญญาณรบกวน E เป็นเอาต์พุตโดยจะมีความถี่สัญญาณนาฬิกาน้อยกว่าความถี่ของสัญญาณนาฬิกาที่ขา EXTAL และ XTAL 4 เท่า และยังเป็นขาที่แสดงการทำงานของซีพียูด้วย คือ ถ้ามีสัญญาณเป็นลอจิก "0" แสดงว่าซีพียูกำลังประมวล

ผลภายใน และหากเป็นลอจิก "1" หมายถึงขณะนี้ซีพียูกับข้อมูล เมื่ออยู่ใน STOP โหมดจะไม่มีสัญญาณ E ออกมา

IRQ เป็นขาอินพุตทำงานที่ลอจิก "0" หรือขอบขาลงเป็นการขออินเตอร์รัพต์ซีพียู โดยที่ขานี้ต้องต่อตัวต้านทานภายนอกค่า 4.7 กิโลโอห์ม กับไฟเลี้ยง +5 โวลต์ เพื่อทำให้สภาวะปกติเป็นลอจิก "1"

$\overline{XIRQ}$ เป็นขาอินพุตทำงานที่ลอจิก "0" เป็นการขออินเตอร์รัพต์แบบนอนมาสเคเบิลอินเตอร์รัพต์

MODA, MODB เป็นขาที่ใช้เลือกโหมดการทำงานของ CPU ซึ่งแสดงการเลือกโหมดดังตารางที่ 2

ตารางที่ 2 แสดงลักษณะการเลือกโหมดการทำงาน

MODB	MODA	Mode Selected
1	0	Single Chip
1	1	Expanded
0	1	Special Test

หลังจากที่ซีพียูเลือกโหมดการทำงานแล้ว ขา MODA จะเป็นขาเอาต์พุตลอจิก "0" เพื่อแสดงถึงการเฟตช์ของซีพียูที่จะใช้ในการทำให้ซีพียูทำงานทีละคำสั่ง และขา MODB จะเป็นขาอินพุตสำหรับป้อนไฟเพื่อไม่ให้ข้อมูลสูญหายหลังจากไม่มีไฟเลี้ยงซีพียู

VRL, VRH เป็นขาอินพุตสำหรับป้อนแรงดันอ้างอิงในวงจรแปลงอะนาลอกเป็นดิจิตอล

STRB / R / $\overline{W}$ เป็นขาเอาต์พุตแสดงผลการทำงานในโหมด SINGLE CHIP ขานี้จะเป็นเอาต์พุตแสดงการแฮนด์เชคข้อมูล และเมื่อทำงานในโหมด EXPANDED MULTIPLECED ขานี้จะเป็นเอาต์พุตแสดงการอ่านเขียนของซีพียู โดยถ้าเป็นลอจิก "0" หมายถึงการเขียนข้อมูล

STRA / AS เป็นทั้งขาอินพุตและเอาต์พุต ถ้าทำงานในโหมด SINGLE CHIP ขานี้จะเป็นอินพุตในการแฮนด์เชคข้อมูล และถ้าทำงานในโหมด EXPANDED MULTIPLECED จะเป็นเอาต์พุตในการแยกสัญญาณระหว่างแอดเดรส 8 บิตล่างกับสัญญาณข้อมูล

PORT A เป็นพอร์ตอินพุตเอาต์พุต โดยบิตที่ 0, 1, 2 จะถูกกำหนดเป็นอินพุตและบิตที่ 4, 5, 6 เป็นเอาต์พุต ส่วนบิตที่ 7 สามารถกำหนดเป็นอินพุตหรือเอาต์พุตก็ได้

PORT B เป็นพอร์ตเอาต์พุตเมื่อทำงานในโหมด SINGLE CHIP และจะเป็นแอดเดรส 8 บิตบน เมื่อทำงานในโหมด EXPANDED MULTIPLECED

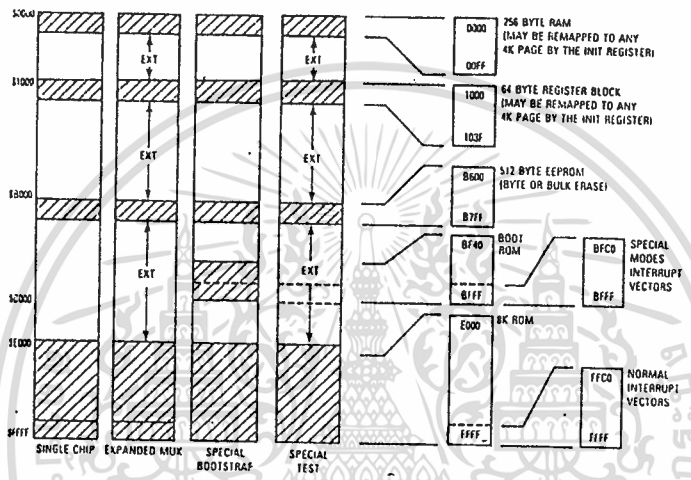
PORT C จะเป็นพอร์ตอินพุตเอาต์พุต เมื่อทำงานในโหมด SINGLE CHIP และเมื่อทำงานในโหมด EXPANDED MULTIPLECED พอร์ตนี้จะเป็นทั้งสายสัญญาณแอดเดรสและข้อมูล โดยมีสัญญาณ AS เป็นตัวแยกระหว่างแอดเดรสและข้อมูล ซึ่งต้องใช้งานร่วมกับสัญญาณ R/W

PORT D เป็นพอร์ตอินพุตเอาต์พุตที่สามารถกำหนดได้ว่าจะให้บิตใดเป็นอินพุตหรือเอาต์พุต และยังใช้เป็นพอร์ตในการส่งและรับข้อมูลแบบอนุกรมได้อีกด้วย

PORT E เป็นพอร์ตอินพุตสำหรับใช้งานทั่วไป และยังใช้เป็นพอร์ตอินพุตสำหรับวงจรเปลี่ยนสัญญาณอะนาลอกเป็นดิจิตอลด้วย

การจัดหน่วยความจำ

68HC11 เป็นซีพียูที่มีการจัดหน่วยความจำอย่างมีประสิทธิภาพ โดยสามารถติดต่อกับหน่วยความจำได้ถึง 64 กิโลไบต์ รวมทั้งหน่วยคามจำที่อยู่ภายในตัวซีพียู การจัดหน่วยความจำแบ่งออกเป็น 4 แบบ ตามโหมดการทำงาน ดังรูปที่ 2.1



รูปที่ 2.1 แสดงการจัดหน่วยความจำในโหมดต่าง ๆ

ตารางที่ 3 สรุปการใช้ขาของ 68HC11 ในโหมดการทำงานต่าง ๆ

68HC11 ในโหมดการทำงานต่าง ๆ

Port – Bit	**Single–Chip and Bootstrap Mode**	**Expanded Multiplexed And Special Test Mode**
A-0	PA0 IC3	PA0 IC3
A-1	PA1 IC2	PA1 IC2
A-2	PA2 IC1	PA2 IC1
A-3	PA3 OC5 and – or OC1	PA3 OC 5and – or OC1
A-4	PA4 OC4 and – or OC1	PA4 OC4and – or OC1
A-5	PA5 OC3 and – or OC1	PA5 OC3and – or OC1
A-6	PA6 OC2	PA6 OC2and – or OC1
A-7	PA7 PAI and – or OC1	PA7 PAI and – or OC1

B-0	PB0	A8
B-1	PB1	A9
B-2	PB2	A10
B-3	PB3	A11
B-4	PB4	A12
B-5	PB5	A13
B-6	PB6	A14
B-7	PB7	A15
C-0	PC0	A0 D0
C-1	PC1	A1 D1
C-2	PC2	A2 D2
C-3	PC3	A3 D3
C-4	PC4	A4 D4
C-5	PC5	A5 D5
C-6	PC6	A6 D6
C-7	PC7	A7 D7
D-0		PD0 RxD
D-1		PD1 TxD
D-2		PD2 MISO
D-3		PD3 MOSI
D-4		PD4 SCK
D-5		PD5 $\overline{SS}$
		AS
		R $\overline{W}$
E-0	PE0 AND	PE0 AND
E-1	PE1 AN1	PE1 AN1
E-2	PE3 AN2	PE3 AN2
E-3	PE3 AN3	PE3 AN3
E-4	PE4 AN4##	PE4 AN4##
E-5	PE5 AN5##	PE5 AN5##
E-6	PE6 AN6##	PE6 AN6##
E-7	PE7 AN7##	PE7 AN7##

พื้นที่หน่วยความจำแอดเดรส 0000H - 00FFH เป็นพื้นที่ของหน่วยความจำแรม ที่อยู่ภายในชิพของทุกโหมดการทำงาน แอดเดรส 0100H - 0FFFH เป็นพื้นที่หน่วยความจำภายนอกในโหมดการทำงาน EXPANDED MUX กับ SPECIAL TEST ที่แอดเดรส1000H-103F เป็นพื้นที่ของรีจิสเตอร์ภายในชิพของทุกโหมดการทำงานแอดเดรส 1040H - B5FFH เป็นพื้นที่หน่วยความจำภายนอกใน EXPANDED MUX กับ SPEACIAL TEST พัดมาที่แอดเดรส B600 - B7FFH เป็นพื้นที่หน่วยความจำอีอีพรอมภายในชิพแอดเดรส BF40H - BFFFH เป็นพื้นที่บูตรอมในโหมด SPECIAL BOOT STARP

ส่วนแอดเดรส B800H - FFFH เป็นพื้นที่สำหรับหน่วยความจำภายนอกและที่แอดเดรส FFC0H - FFFFH เป็นพื้นที่สำหรับเก็บเวกเตอร์อินเตอร์รัพต์ของอินเตอร์รัพต์ทุกประเภท

หน่วยความจำแรมภายในซีพียู

หน่วยความจำแรมภายในซีพียู แบ่งออกเป็น 2 ประเภทคือ แรมที่ใช้ในการอ่านเขียนข้อมูล และรีจิสเตอร์โดยตั้งแต่แอดเดรส 0000H - 00FFH (256 ไบต์) เป็นหน่วยความจำที่ใช้ในการเขียนอ่านข้อมูลทั่วไป ซึ่งหน่วยความจำบริเวณนี้จะสามารถเก็บข้อมูลได้แม้ไฟเลี้ยงของวงจรจะหายไป ข้อมูลแรมส่วนนี้ในสภาวะปกติจะได้ไฟเลี้ยงจาก V_{DD} แต่เมื่อ V_{DD} หายไปจะรับไฟเลี้ยงจากอีกทางหนึ่ง โดยก่อนที่ V_{DD} จะหายไปจะต้องมีสัญญาณรีเซตเพื่อทำให้ซีพียู หยุดการทำงานเพื่อป้องกันไม่ให้มีการเขียนข้อมูลในหน่วยความจำ และเมื่อมี V_{DD} เข้ามาก็จะไปจ่ายให้กับแรมในส่วนนี้แทนไฟเลี้ยงอีกชุดหนึ่งได้ทันที โดยที่ข้อมูลที่อยู่ในแรมจะไม่เปลี่ยนแปลง กระแสที่ใช้ในการเลี้ยงแรมนี้ใช้กระแสไฟน้อยมาก จึงสามารถใช้แบตเตอรี่เล็ก ๆ เป็นไฟเลี้ยงให้กับแรมนี้ได้ส่วนแรมอีกส่วนหนึ่งที่เป็นรีจิสเตอร์จะถูกใช้ในการควบคุมการทำงานต่าง ๆ ของซีพียู โดยมีหน้าที่ที่แตกต่างกันออกไปดังนี้

DDRD	รีจิสเตอร์ควบคุมทิศทางการติดต่อของพอร์ต
PORTE	รีจิสเตอร์พอร์ต E ใช้สำหรับอ่านข้อมูลของพอร์ต E
CFORC	รีจิสเตอร์บอกผลการเปรียบเทียบของไทเมอร์ต่าง ๆ
OC1M	รีจิสเตอร์สำหรับกำหนดบิตของข้อมูลในพอร์ต Aที่มีการเปรียบเทียบกับ OC1
OC1D	รีจิสเตอร์สำหรับเก็บข้อมูลของการเปรียบเทียบระหว่างพอร์ต A กับ OC1
OC1M	รีจิสเตอร์สำหรับเซตผลการเปรียบเทียบระหว่างพอร์ต A กับรีจิสเตอร์ TOC1
TCNT	รีจิสเตอร์ 16 บิต ใช้เป็นตัวฟรีเคาน์เตอร์ของระบบไทเมอร์ โดยจะทำการนับตั้งแต่ 0000 - FFFF แล้วกลับมาเริ่มต้นใหม่
TIC 1	รีจิสเตอร์ 16 บิต เป็นอินพุตไทเมอร์ 1
TIC2	รีจิสเตอร์ 16 บิต เป็นอินพุตไทเมอร์ 2
TIC 3	รีจิสเตอร์ 16 บิต เป็นอินพุตไทเมอร์ 3
TOC1	รีจิสเตอร์ 16 บิต ใช้ในการเปรียบเทียบกับไทเมอร์เคาน์เตอร์
TOC2	รีจิสเตอร์ 16 บิต ใช้ในการเปรียบเทียบกับไทเมอร์เคาน์เตอร์
TOC3	รีจิสเตอร์ 16 บิต ใช้ในการเปรียบเทียบกับไทเมอร์เคาน์เตอร์

TOC4 รีจิสเตอร์ 16 บิต ใช้ในการเปรียบเทียบกับไทเมอร์เคาน์เตอร์

TOC5 รีจิสเตอร์ 16 บิต ใช้ในการเปรียบเทียบกับไทเมอร์เคาน์เตอร์

TCTL1 เป็นรีจิสเตอร์ที่ใช้ในการเลือกลักษณะเอาต์พุตจากการใช้ฟังก์ชันการเปรียบเทียบเอาต์พุต

TCTL2 เป็นรีจิสเตอร์ที่ใช้ในการเลือกลักษณะของอินพุตเพื่อฟังก์ชั่นอินพุตแคบเจอร์ (input capture) CAPTURE ฟังก์ชั่น

TMSK1 เป็นรีจิสเตอร์ที่ใช้สำหรับอีนาเบิลหรือดิสเอเบิลอินเตอร์รัพต์ของตัวเปรียบเทียบเอาต์พุตกับตัวเปรียบเทียบอินพุต

TFLG1 เป็นรีจิสเตอร์ที่แสดงถึงการเท่ากันของไทเมอร์เคาน์เตอร์กับตัวเปรียบเทียบเอาต์พุตกับตัวเปรียบเทียบอินพุต

TMSK2 เป็นรีจิสเตอร์ที่ใช้ในการอีนาเบิลหรือดิสเอเบิลอินเตอร์รัพต์ของรีลไทม์ เป็นตัวกำหนดตัวหารของเวลารีลไทม์

TFLG2 เป็นรีจิสเตอร์ที่ใช้แสดงการทำงานต่างๆ เป็นรีลไทม์อินเตอร์รัพต์โอเวอร์ไฟลว์

PACTL เป็นรีจิสเตอร์ที่ใช้ในการควบคุมพัลส์แอกคิวมูเลเตอร์

PACNT เป็นรีจิสเตอร์เคาน์เตอร์ของพัลส์แอกคิวมูเลเตอร์

APCR เป็นรีจิสเตอร์ที่ใช้ในการควบคุมการทำงานแบบ SPI (Serial Perpheral Interface)

SPSR เป็นรีจิสเตอร์ที่ใช้ในการบอกลักษณะการติดต่อแบบ SPI

SPDR เป็นรีจิสเตอร์ที่ใช้ในการรับข้อมูลและส่งข้อมูลของ SPI

BACD เป็นรีจิสเตอร์ที่ใช้ในการกำหนดเร็วในการส่งข้อมูลแลล SCI (Serial Communication Interface)

SCCR1 เป็นรีจิสเตอร์ที่ใช้ในการกำหนดรูปแบบในการส่งข้อมูลแบบ SCI CC-R2 เป็นรีจิสเตอร์ที่ใช้ในการควบคุมการอินเตอร์รัพต์

R2 เป็นรีจิสเตอร์ที่ใช้ในการควบคุมการอินเตอร์รัพต์

SCCR เป็นรีจิสเตอร์ที่ใช้ในบอกสถานะการรับส่งข้อมูลแบบ SCI

SCDR เป็นรีจิสเตอร์ที่ใช้ในการเก็บข้อมูลจากการรับหรือส่งข้อมูลแบบ SCI

ADCTL เป็นรีจิสเตอร์ที่ใช้เลือกการอ่านข้อมูลแบบอะนาลอกและแสดงสถานะการอ่านข้อมูล

ADR1 เป็นรีจิสเตอร์ที่ใช้เก็บข้อมูลที่ได้จากวงจร A/D

ADR2 เป็นรีจิสเตอร์ที่ใช้เก็บข้อมูลที่ได้จากวงจร A/D

ADR3 เป็นรีจิสเตอร์ที่ใช้เก็บข้อมูลที่ได้จากวงจร A/D

ADR4 เป็นรีจิสเตอร์ที่ใช้เก็บข้อมูลที่ได้จากวงจร A/D

OPTION เป็นรีจิสเตอร์ที่ใช้ในการกำหนดเวลาของการใช้ระบบ COP (Computer Operating Properly) การเปิดสัญญาณนาฬิกาสำหรับการ

แซมปลิ้งในวงจร A/D

COPRST การเปิดสัญญาณนาฬิกาเลือกรีเซตค่าเวลาในระบบ COP

PPROG เป็นรีจิสเตอร์ที่ใช้กำหนดรูปแบบของการติดต่อกับอีพรอม

HPIRO เป็นรีจิสเตอร์ที่ใช้ในการจัดลำดับการอินเตอร์รัพต์

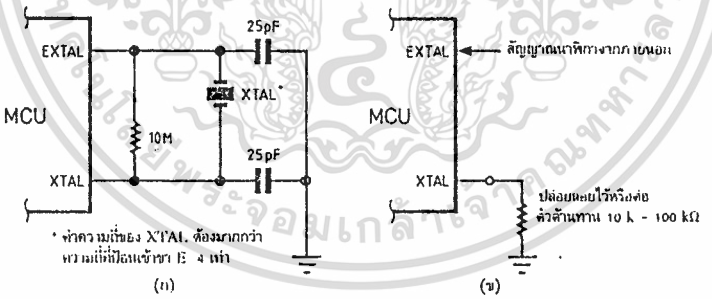
INIT เป็นรีจิสเตอร์ที่ใช้ในการกำหนดตำแหน่งของแรมและรีจิสเตอร์

TEST1 เป็นรีจิสเตอร์ที่ใช้ในการทดสอบของโรงงานผู้ผลิต

CONFIG เป็นรีจิสเตอร์ที่ใช้ในการบ่งบอกอุปกรณ์ภายในตัวซีพียู

วงจรกำเนิดสัญญาณนาฬิกา

68HC11 สามารถต่อสัญญาณนาฬิกาได้สองรูปแบบคือ ใช้วงจรกำเนิด สัญญาณนาฬิกาภายในโดยการต่อคริสตอลภายนอกกับตัวต้านทาน และตัวเก็บประจุสองตัว ดังรูปที่ 2.1 ก และการต่อสัญญาณนาฬิกาจากภายนอกเข้าที่ขา EXTAL และต่อตัวต้านทานเข้าที่ขา XTAL ลงกราวด์ ดังรูปที่ 2.2 ข. นอกจากนั้นสัญญาณนาฬิกาจากการต่อ EXTAL จากไมโครคอนโทรลเลอร์ตัวที่หนึ่งยังสามารถใช้เป็นสัญญาณนาฬิกาให้กับไมโครคอนโทรลเลอร์ตัวต่อไปได้อีก ดังรูปที่ 2.4 โดยสัญญาณนาฬิกาที่ให้กับซีพียู้ต้องมีความถี่สูงกว่าสัญญาณที่ขา E ประมาณ 4 เท่า



รูปที่ 2.2 การต่อสัญญาณนาฬิกาให้กับ 68HC 11
(ก) สัญญาณนาฬิกาภายใน
(ข) สัญญาณนาฬิกาจากภายนอก

รีจิสเตอร์

68HC11 มีรีจิสเตอร์ที่ใช้ในการเขียนโปรแกรมต่าง ๆ ทั้ง 8 บิต และ 16 บิต อยู่ด้วยกัน 7 ตัว ได้แก่

แอกคิวมูเลเตอร์ A และ B เป็นรีจิสเตอร์ขนาด 8 บิต ที่ใช้ในการคำนวณทางด้านคณิตศาสตร์ และตรรกศาสตร์ นอกจากนั้นยังรวมกันเป็นรีจิสเตอร์ D ขนาด 16 บิต โดยแอกคิวมูเลเตอร์ A เป็น 8 บิตสูงและแอกคิวมูเลเตอร์ B เป็น 8 บิตต่ำ

อินเด็กซ์รีจิสเตอร์ IX เป็นรีจิสเตอร์ขนาด 16 บิต ใช้เป็นตัวชี้ตำแหน่งของข้อมูลและเป็นตัวตั้งในการบวกแบบ 16 บิตกับ แอกคิวมูเลเตอร์ B

อินเด็กซ์รีจิสเตอร์ IY เป็นรีจิสเตอร์ขนาด 16 บิต มีลักษณะคล้ายกับรีจิสเตอร์ IX แต่คำสั่งที่ใช้จะมีออปโค้ดขนาดสองไบต์

สแต็คพอยเตอร์ SP เป็นรีจิสเตอร์ขนาด16 บิต ชี้ตำแหน่งของสแต็คในการเก็บข้อมูลต่าง ๆ จากการเรียกโปรแกรมย่อย หรือการอินเตอร์รัพต์ การเก็บข้อมูลลงสแต็คมีลักษณะแบบเข้าก่อนออกทีหลัง (FIFO)

โปรแกรมเคาน์เตอร์ PC เป็นรีจิสเตอร์ขนาอ 16 บิต ใช้เป็นตัวชี้ตำแหน่งของคำสั่งต่อไปจะประมวลผล

คอนดิชั่นโค้ดรีจิสเตอร์ CCR เป็นรีจิสเตอร์ขนาด 8 บิต เป็นตัวบ่งบอกลักษณะการทำงานต่าง ๆ ของซีพียู และกำหนดการทำงานของซีพียู เช่น การกำหนดให้อีนาเบิลหรือดิสเอเบิลอินเตอร์รัพต์และการทำงานใน STOP โหมด

วงจรเอทูดี

เป็นวงจรแปลงสัญญาณอะนาลอกให้เป็นดิจิตอลขนาด 8 บิตโดยสามารถรับอินพุตได้ถึง 8 ช่อง จะทำการรับครั้งละช่องโดยการโปรแกรมเลือกว่าจะทำการแปลงสัญญาณที่ช่องใดผลลัพธ์ที่ได้จากแปลงจะเป็นอัตราส่วนระหว่างสัญญาณอินพุตกับแหล่งจ่ายที่เป็นไฟอ้างอิงของวงจร A/Dโดยแรงดันที่ใช้จะอยู่ในช่วง 2.5-5 โวลต์ เวลาใช้ในการแปลงสัญญาณนั้นจะใช้จำนวนสัญญาณนาฬิกา 32 ลูกต่อหนึ่งครั้ง หรือ ที่เวลา 16 ไมโครวินาที ที่ความถี่ 8 เมกะเฮิรตซ์ ผลลัพธ์ที่อ่านเข้ามาแล้วจะถูกเก็บอยู่ในรีจิสเตอร์ ADR1 -ADR4 ดังในตารางที่ 5

ตารางที่ 5 การกำหนดช่องการทำงานและการเก็บข้อมูลของวงจร A/D ใน 68HC11

CD	CC	CB	CA	Channel Signal	Result in ADRx if MULT = 1
0	0	0	0	AN0	ADR1
0	0	0	1	AN1	ADR2
0	0	1	0	AN2	ADR3
0	0	1	1	AN3	ADR4
0	1	0	0	AN4	ADR1
0	1	0	1	AN5	ADR2
0	1	1	0	AN6	ADR3
0	1	1	1	AN7	ADR4

1	0	0	0	Reserved	ADR1
1	0	0	1	Reserved	ADR2
1	0	1	0	Reserved	ADR3
1	0	1	1	Reserved	ADR4
1	1	0	0	V_{RH} Pin	ADR1
1	1	0	1	V_{RL} Pin	ADR2
1	1	1	0	$(V_{RH})/2$	ADR3
1	1	1	1	Reserved	ADR4

ไทเมอร์

68HC11 มีไทเมอร์ขนาด 16 บิต ที่ทำงานในลักษณะฟรีรันนิ่งคือ จะทำการนับตั้งแต่ 0000H-FFFFH และกลับมาเริ่มต้นนับใหม่ โดยจะใช้สัญญาณนาฬิกาจากภายในตัวซีพียู ไทเมอร์นี้จะนำไปใช้เป็นตัวเปรียบเทียบกับรีจิสเตอร์ไทเมอร์อินพุตและไทเมอร์เอาต์พุต

รีจิสเตอร์ไทเมอร์อินพุต (input capture) เป็นรีจีสเตอร์ ที่อ่านได้อย่างเดียวทั้งหมด 3 ตัว ถ้ามีสัญญาณลอจิก "0" ที่ขาอินพุตของแต่ละตัวหลังจากที่ได้ทำการโหลดค่าจากฟรีรันไทเมอร์แล้ว สามารถที่จะทำการอินเตอร์รัพต์ซีพียูได้ และยังสามารถนำค่าที่โหลดมาใช้ในการวิเคราะห์คาบเวลาได้ว่าเป็นความถี่เท่าไร

รีจิสเตอร์ไทเมอร์เอาต์พุต (output compare) เป็นรีจิสเตอร์ที่เขียนและอ่านได้มีทั้งหมด 5 ตัวอยู่ตำแหน่งตั้งแต่แอดเดรส 1016H-101FH เราสามารถเขียนค่าลงไปยังรีจิสเตอร์ไทเมอร์เอาต์พุต เพื่อที่จะนำไปเปรียบเทียบกับฟรีรันไทเมอร์ เมื่อค่าทั้งสองเท่ากันจะทำการอินเตอร์รัพต์ซีพียู หรือจะทำการหารความถี่จากสัญญาณนาฬิกาไปยังเอาตพุตได้ทั้ง 5 ตัว ซึ่งสามารถควบคุมโดยการกำหนดค่าในรีจิสเตอร์ TMSK1 และ TCTL1

รีลไทม์

เป็นวงจรส่วนหนึ่งที่ทำหน้าที่เป็นฐานเวลาให้กับโปรแกรม โดยสามารถโปรแกรมเลือกเวลาไปยังรีจิสเตอร์ PACTL ที่บิต RTRO และ RTRO บิต RTIF ในรีจิสเตอร์ TFLG2 จะเป็นตัวกำหนดว่าจะให้ทำการอินเตอร์รัพต์หรือไม่ ถ้าต้องการก็ควบคุมให้บิต RTIF มีเท่ากับ "1"

พัลส์แอกคิวมูเลเตอร์

68HC11 มีรีจิสเตอร์สำหรับนับพัลส์จากภายนอกขนาด 8 บิต โดยความถี่ที่สามารถนับได้สูงสุดจะเท่ากับความถี่ของสัญญาณนาฬิกาภายในตัวซีพียูหารด้วย 64 ลักษณะของการนับจะเป็นการนับขึ้น และถ้านับจนเกินค่าของรีจิสเตอร์แล้วจะทำการอินเตอร์รัพต์ซีพียูได้ โดยการเซ็ตที่บิต PAOVI ของรีจิสเตอร์ TMSK2

รีเซต

การรีเซตของ 68HC11 จะแบ่งออกเป็น 3 ชนิด คือ

(1) การรีเซตจากภายนอก ซึ่งเกิดจากการเปิดเครื่องครั้งแรกหรือสัญญาณ

รีเซตจากภายนอก

(2) การรีเซตจากวงจรวอตช์ดอก

(3) การรีเซตเนื่องจากการผิดพลาดของสัญญาณนาฬิกาการรีเซตจากการเปิดเครื่องจะใช้จำนวนพัลส์สัญญาณนาฬิกา 4064 ลูกหลังจากที่สัญญาณนาฬิกาลูกแรกทำงาน ซึ่งถ้าใช้คริสตอล ภายนอกที่ 8 MHz จะใช้เวลา 2 มิลลิวินาที

ถ้าซีพียูทำงานในโหมด Bootstrap หรือ Spectral test หลังจากที่ซีพียูรีเซตจะไปเฟตช์ข้อมูลที่เวกเตอร์แอดเดรส FFFCH-FFFFH หรือ BFFEH-BFFFH ในกรณีการรีเซตจากวงจรวอตช์ด็อกจะทำให้สัญญาณที่จะรีเซตเป็นลอจิก "0" ด้วยและจะไปเฟตช์ข้อมูลที่เวกเตอร์แอดเดรส FFFCH-FFFDH

อินเตอร์รัพต์

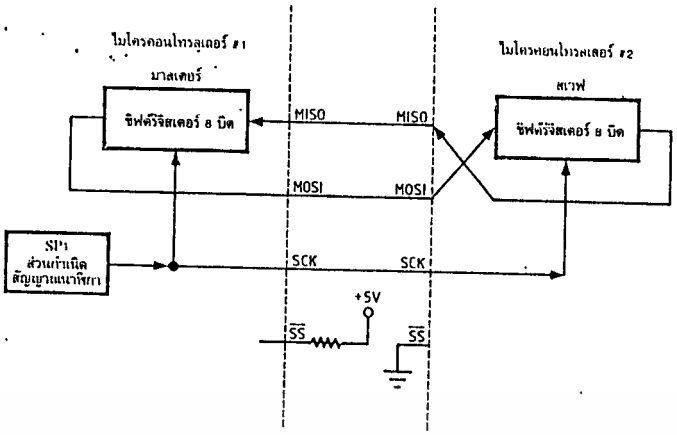
ลักษณะการอินเตอร์รัพต์ แบ่งออกเป็น 2 แบบคือ ฮาร์ดแวร์ และซอฟต์แวร์อินเตอร์รัพต์ ฮาร์ดแวร์อินเตอร์รัพต์ ยังแบ่งออกได้เป็นการอินเตอร์รัพต์ จากภายในและจากภายนอก สามารถจัดลำดับความสำคัญของการขออินเตอร์รัพต์ ได้ด้วยการควบคุมรีจิสเตอร์ HPRIO การอินเตอร์รัพต์จะทำงานเป็นแบบเวกเตอร์อินเตอร์รัพต์

พอร์ตอนุกรม

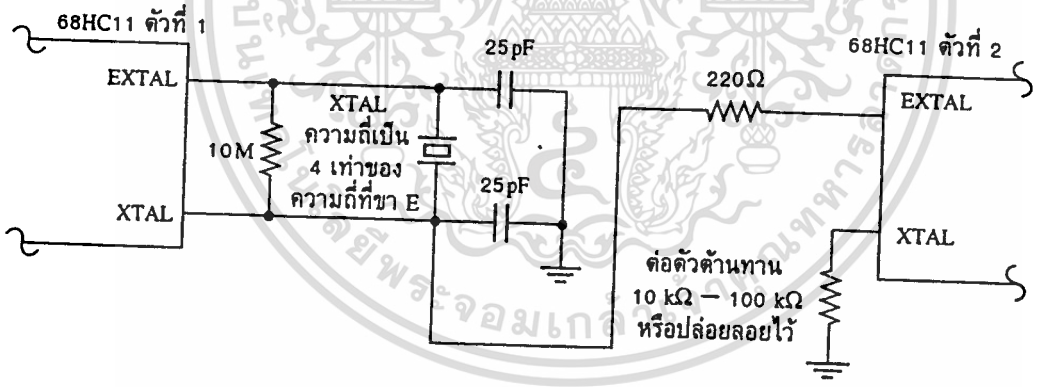
พอร์ตอนุกรมของ 68HC11 มีลักษณะเหมือนกับ พอร์ตอนุกรมใน ไมโครคอนโทรเลอร์โดยทั่วไป การรับส่งข้อมูลเป็นแบบ ฟูลดูเพล็กซ์คือ สามารถรับส่งข้อมูลได้พร้อมกันในเวลาเดียวกัน ก่อนจะรับข้อมูลมายังรีจิสเตอร์ จะต้องอ่านข้อมูลเดิมออกไปก่อน มิฉะนั้นข้อมูลที่ส่งมาใหม่จะทับข้อมูลเดิม แอดเดรสของรีจิสเตอร์อยู่ที่ตำแหน่ง 102FH การรับส่งข้อมูลสามารถที่จะกำหนดการรับส่งข้อมูลได้ว่าจะเป็นแบบ10 บิต หรือ 11 บิต แต่ลักษณะที่พิเศษของพอร์ตอนุกรมสำหรับ 68HC11 นี้คือ สามารถที่จะทำการตรวจสอบสัญญาณรบกวนได้ โดยความกว้างของสัญญาณรบกวนจะต้องไม่เกิน 1/16 ของอัตราความเร็วในการส่ง

มัลติโปรเซสเซอร์

นอกจากการติดต่อข้อมูลแบบพอร์ตอินพุต เอาต์พุต และอนุกรมแล้ว 68HC11 ยังมีการติดต่อแบบมัลติโปรเซสเซอร์ที่สามารถนำซีพียูหลาย ๆ ตัวได้ โดยมีซีพียูตัวหนึ่งเป็นมาสเตอร์ และมีซีพียูตัวอื่น ๆ เป็นสเลฟจะใช้สัญญาณที่ขา $\overline{SS}, SCK, MISO, MOSI$ เป็นตัวติดต่อดังแสดงในรูปที่ 2.3



รูปที่ 2.3 การเชื่อมต่อไมโครคอนโทรลเลอร์ 68HC11 ให้ทำงานแบบมัลติโปรเซสเซอร์



รูปที่ 2.4 การต่อสัญญาณนาฬิกาจากไมโครคอนโทรลเลอร์ตัวหนึ่งไปยังอีกตัวหนึ่ง

2.2 การอ้างแอดเดรสและชุดคำสั่งของ 68HC11

การศึกษาทำความรู้จักการควบคุมการทำงาน ตั้งแต่การกำหนดแอดเดรส และตารางชุดคำสั่ง เพื่อจุดประสงค์ให้คุณรู้จัก เข้าใจ และนำไปใช้งานได้ เริ่มต้นด้วยการอ้างแอดเดรสของ 68HC11 ซึ่งมีหลายแบบหลายลักษณะกันก่อน ตามด้วยตารางชุดคำสั่ง

การอ้างแอดเดรสของ 68HC11

68HC11 มีกระบวนการอ้างแอดเดรส (addressing) ทั้งสิ้นถึง 6 โหมดด้วยกัน ประกอบด้วย การอ้างแอดเดรสแบบทันทีทันใด (immediate), แบบโดยตรง (direct), แบบขยาย (extended), แบบอินเด็กซ์ (index), แบบอินเฮียเรนต์ (inherent) และแบบสัมพัทธ์ (relative) ในบางคำสั่งเมื่อถูกใช้ในการอ้างแอดเดรสบางแบบจำเป็นต้องเพิ่มข้อมูลเข้าไปอีก 1 ไบต์ ก่อนออปโค้ดเพื่อปรับให้ 68HC11 สามารถทำงานในลักษณะมัลติเพจออปโค้ดแมป (multi-page opcode map) ได้เหมาะสมขึ้น ข้อมูลไบต์นั้นจะเรียกว่า พรีไบต์ (prebyte)

การอ้างแอดเดรสของ 68HC11 (Immediate addressing)

ในการอ้างแอดเดรสแบบนี้ เป็นการติดต่อเพื่อจัดการข้อมูล ซึ่งเป็นค่าใดๆ โดยตรงในทันทีทันใด จำนวนไบต์ของคำสั่งในการอ้างแอดเดรสแบบนี้จะมีขนาดตั้งแต่ 2-4 ไบต์ ขึ้นอยู่กับขนาดของรีจิสเตอร์ที่ต้องเกี่ยวข้องด้วย เช่น ถ้าเป็นแอดคิวมูเลเตอร์ A ก็จะมีจำนวนไบต์ของคำสั่ง 2 ไบต์ (1 ไบต์แรกเป็นออปโค้ด อีก 1 ไบต์หลังเป็นขนาดของโอเปอร์แรนด์ ซึ่งเท่ากับขนาดของรีจิสเตอร์แอดคิวมูเลเตอร์ A)

รูปแบบของคำสั่งที่มีการอ้างแอดเดรสแบบนี้หลังจากคำสั่งแล้วต้องตามด้วยเครื่องหมาย # เสมอเพื่อเป็นการบ่งบอกให้ซีพียูทราบว่าคำสั่งที่จะกระทำต่อไปนี้ใช้การอ้างแอดเดรสแบบทันทีทันใด

การอ้างแอดเดรสแบบโดยตรง (Driect addressing)

เป็นการติดต่อเพื่อประมวลผลข้อมูลขนาด 8 บิต ที่อยู่ในหน่วยความจำแรมภายในชิพ ซึ่งมีอยู่ 256 ไบต์ โดยมีแอดเดรสตั้งแต่ $0000-$00FF ดังนั้นค่าโอเปอร์แรนด์ที่ตามหลังออปโค้ดคำสั่งก็คือ ค่าแอดเดรสของแรมนั่นเองในการอ้างแอดเดรสแบบนี้ บางทีเรียกว่า การอ้างแอดเดรสเพจศูนย์ (zero page addressing)

การอ้างแอดเดรสแบบขยาย (Extended addressing)

ในการอ้างแอดเดรสแบบนี้ ข้อมูลในไบต์ที่ 2 และ 3 ที่ตามหลังออปโค้ด จะเก็บค่าแอดเดรสจริงของหน่วยความจำที่ต้องการนำข้อมูลในหน่วยความจำออกมาประมวลผล หรือจัดเก็บข้อมูลลงไปใหม่เมื่อใช้การอ้างแอดเดรสแบบนี้คำสั่งจะมีขนาด 3-4 ไบต์ โดยเป็นออปโค้ดไบต์ที่ 1 หรือ 2 (กรณีถ้ามีขนาด4 ไบต์) ส่วน 2 ไบต์หลังจะเป็นค่าแอดเดรสที่อ้างถึงในหน่วยความจำ

การอ้างแอดเดรสแบบอินเด็กซ์ (Indexed addressing)

การอ้างแอดเดรสแบบนี้จะมีการนำรีจิสเตอร์ชี้ข้อมูล (index register) ทั้ง 2 ตัว คือ IX และ IY นำมาใช้ในการคำนวณค่าแอดเดรสที่ต้องการเรียกใช้ข้อมูล ดังนั้นค่าแอดเดรสที่ต้องการใช้งานจะเปลี่ยนแปลงหรือไม่ ขึ้นอยู่กับองค์ประกอบหลัก 2 ประการ คือ

1. ค่าแอดเดรสที่ถูกกำหนดอยู่ในปัจจุบัน
2. ค่าออฟเซต 8 บิต ที่บรรจุอยู่ในคำสั่ง (หรือค่าโอเปอแรนด์)

ในการอ้างแอดเดรสแบบนี้ สามารถที่จะใช้หน่วยความจำที่ตำแหน่งใดก็ได้จำนวน 64 กิโลไบต์ เป็นจุดอ้างอิง ส่วนในด้านขนาดของคำสั่งในการอ้างแอดเดรสแบบนี้ถ้าใช้รีจิสเตอร์ IX จะมีขนาด 2 ไบต์ โดยไบต์แรกเป็นออปโค้ด ส่วนไบต์ที่สองเป็นค่าออฟเซต หากใช้รีจิสเตอร์ IY จะมีขนาด 3 ไบต์ ไบต์แรกก็คือ พรีไบต์ ตามด้วยออปโค้ด และค่าออฟเซตขนาด 8 บิต

การอ้างแอดเดรสแบบอินเฮียเรนต์ (Inherent addressing)

การอ้างแอดเดรสแบบนี้จะไม่ยุ่งเกี่ยวกับข้อมูลในหน่วยความจำแต่อย่างใด แต่จะเข้าไปจัดการในรีจิสเตอร์แทน ดังนั้นขนาดของคำสั่งในการอ้างแอดเดรสแบบนี้จะมีเพียง 1-2 ไบต์ โดยเป็นออปโค้ดทั้งสิ้นไม่มีโอเปอแรนด์

การอ้างแอดเดรสแบบสัมพัทธ์ (Relative addressing)

การอ้างแอดเดรสแบบนี้จะใช้ในชุดคำสั่งกระโดด (jump and branch instructions) ถ้าหากเงื่อนไขในการกระโดดเป็นจริง ค่าออฟเซตขนาด 8 บิต ที่อยู่ตามหลังออปโค้ด ก็จะถูกบวกเข้าไปในรีจิสเตอร์โปรแกรมเคาน์เตอร์ (PC) เพื่อกำหนดแอดเดรสต่อไปที่จะข้ามไปทำงานของซีพียู ปกติแล้วขนาดของคำสั่งในการอ้างแอดเดรสแบบนี้จะเท่ากับ 2 ไบต์ โดยไบต์แรกเป็นออปโค้ด ส่วนไบต์ที่สองเป็นค่าออฟเซตเพื่อบอกจำนวนที่จะเข้าไปเพิ่มใน PC

พรีไบต์ (prebyte)

เป็นข้อมูลขนาด 8 บิต (1 ไบต์) ที่ใส่เข้าไปก่อนหน้าออปโค้ด เพื่อประโยชน์ในการบอกให้ซีพียูทราบถึงลักษณะของการจัดเพจของออปโค้ด โดยถ้าหาก 68HC11 ทำงานในเพจ 1 ข้อมูลพรีไบต์ก็ไม่ต้องมี แต่ถ้าทำงานในเพจ 2 จะต้องเพิ่มค่าพรีไบต์เท่ากับ $18 เข้าไปก่อนออปโค้ดเสมอ แล้วตามด้วยออปโค้ด ในกรณีเพจ 3 จะใช้ค่า $1A และใช้ค่า $CD สำหรับเพจ 4 ค่าพรีไบต์จะเข้าไปเกี่ยวข้อง เมื่อมีคำสั่งใด ๆ ก็ตามให้รีจิสเตอร์ IY ทำงาน

ชุดคำสั่ง 68HC11

ซีพียูภายในไมโครคอนโทรลเลอร์ 68HC11 มีลักษณะการทำงานคล้าย ๆ กับ ซีพียู MC6801 แต่ได้มีการเพิ่มเติมความสามารถของ 68HC11 นี้ด้วยการเพิ่มริจิสเตอร์และคำสั่งให้มากขึ้น เช่น การเพิ่มรีจิสเตอร์อินเด็กซ์ขนาด 16 บิตอีก 1 ตัวคือ IY เพิ่มคำสั่งเกี่ยวกับการหารเลข 16 บิตอีก 2 แบบ, เพิ่มคำสั่งควบคุมและเพิ่มคำสั่งจัดการข้อมูลระดับบิต (bit manipulation instruction)

ในที่นี้จะแบ่งชุดคำสั่ง 68HC11 ออกเป็น 7 กลุ่มย่อย ๆ ดังนี้

1. กลุ่มคำสั่งจัดการเกี่ยวกับข้อมูล (Data handling instructions)
2. กลุ่มคำสั่งทางคณิตศาสตร์ (Arithmatics instructions)
3. กลุ่มคำสั่งทางลอจิก (Logic instructions)
4. กลุ่มคำสั่งการกระโดด (Jump and b ranch instructions)
5. กลุ่มคำสั่งเปรียบเทียบและตรวจสอบข้อมูล (Data test instructions)
6. กลุ่มคำสั่งจัดการกับรีจิสเตอร์รหัสเงื่อนไข (CCR instructions)
7. กลุ่มคำสั่งควบคุม (control instructions)

ซึ่งจะแจงรายละเอียดของกลุ่มคำสั่งทั้ง 7 กลุ่ม

1. กลุ่มคำสั่งจัดการเกี่ยวกับข้อมูล (data handling instructions) แบ่งออกได้ 3 กลุ่มคือ

กลุ่มเคลื่อนย้ายข้อมูล กลุ่มคำสั่งชุดนี้แยกออกได้อีก 3 กลุ่ม คือ

- กลุ่มเคลื่อนย้ายข้อมูลระดับบิต มีคำสั่งดังนี้

BSET (Set bit) เป็นกลุ่มคำสั่งที่ใช้ในการทำให้บิตใด ๆ ของข้อมูลที่อ้างถึงเป็น "1" แล้วนำข้อมูลที่ทำการเซตบิตแล้วกลับเข้าไปเก็บในตำแหน่งเดิม รูปแบบการทำงานคือ M + mm -> M (M คือค่าของข้อมูลที่อ้างถึงในหน่วยความจำ) mm คือ บิตใด ๆ ที่ต้องการเซตส่วนเครื่องหมาย + หมายถึงการออร์ (OR)

BCLR (Clear bit) มีลักษณะตรงข้ามกับ BSET คือ ทำให้บิตใด ๆของข้อมูลที่อ้างถึงเป็น "0" มีรูปแบบการทำงานดังนี้ M. $\overline{mm}$ -> M (. คือการแอนด์ (AND))

- กลุ่มคำสั่งเคลื่อนย้ายข้อมูลระดับไบต์ มี 15 คำสั่ง

LDAA (Load Accumulator A) เป็นคำสั่งที่ใช้โหลดข้อมูลที่อ้างถึงมาเก็บไว้ในแอกคิวมูเลเตอร์ A

LDAB (Load Accumulator B) เป็นคำสั่งที่ใช้โหลดข้อมูลที่อ้างถึงมาเก็บไว้ในแอกคิวมูเลเตอร์ B

STAA (Store Accumulator A) เป็นคำสั่งที่ใช้ในการอ่านค่าจากแอกคิวมูเลเตอร์ A มาเก็บไว้ในหน่วยความจำ จะมีลักษณะการทำงานตรงข้ามกับ LDAA

STAB (Store Accumulator B) เหมือนกับ STAA แต่จะอ่านค่าจากแอกคิวมูเลเตอร์ B

TAB (Transfer B to A) เป็นคำสั่งในการถ่ายเทข้อมูลจากแอกคิวมูเลเตอร์ A ไป B

TBA (Transfer B to B) เป็นคำสั่งในการถ่ายเทข้อมูลจากแอกคิวมูเลเตอร์ B ไป A

TAP (Transfer A to CCR) เป็นคำสั่งในการถ่ายเทข้อมูลจากแอกคิวมูเลเตอร์ A ไปยังรีจิสเตอร์รหัสเงื่อนไข (CCR)

CCLRA และ CLRB (Clear Accumulator A & Clear Accumulator B) เป็นคำสั่งที่ใช้ในการเคลียร์ข้อมูลในแอกคิวมูเลเตอร์ A และ B

CLR (Clear memoryu byte) เป็นคำสั่งในการเคลียร์ข้อมูลในหน่วยความจำ 1 ไบต์ นั่นคือนำข้อมูล "0" เขียนลงในหน่วยความจำ

PSHA และ PSHB (Push A onto Stack & Push B onto Stack) เป็นคำสั่งที่ใช้เก็บค่าที่ข้อมูลในแอกคิวมูเลเตอร์ A หรือ B ไปไว้ในสแต็ก เมื่อทำคำสั่งนี้แล้วค่าของรีจิสเตอร์ชี้สแต็ก (SP) จะลดลง

PULA และ PULB (Pull A from stack & Pull B from stack) เป็นคำสั่งที่ใช้เก็บค่าที่ข้อมูลในแอกคิวมูเลเตอร์ A หรือ B ไปไว้ในสแต็ก กลับมาเก็บไว้ในแอกคิวมูเลเตอร์ A และ B อย่างเดิม ค่าของ SP จะเพิ่มขึ้นเมื่อทำคำสั่งนี้แล้ว

- กลุ่มคำสั่งเคลื่อนย้ายข้อมูลระดับเวิร์ด มีกลุ่มคำสั่ง 18 คำสั่งคือ

LDD (Load Double Accumulator D) เป็นการโหลดข้อมูล 16 บิต มาเก็บไว้ในแอกคิวมูเลเตอร์ D โดยไบต์ต่ำจะเก็บไว้ในแอกคิวมูเลเตอร์ A ส่วนไบต์สูงจะเก็บไว้ในแอกคิวมูเลเตอร์ B (แอกคิวมูเลเตอร์ D เกิดจากการรวมกันของ A และ B)

STD (Store Accumulator D) เป็นคำสั่งในการเก็บข้อมูลจาก D ไปไว้ในหน่วยความจำ โดยมีการแยกเก็บคือ ข้อมูลใน A จะถูกเก็บไว้ในหน่วยความจำแอดเดรสต่ำ ส่วนข้อมูลใน B จะเก็บไว้ในหน่วยความจำแอดเดรสถัดไป

XGDX และ XGDY (Exchange D with X & Exchange D with Y) เป็นคำสั่งในการเปลี่ยนข้อมูลระหว่างรีจิสเตอร์ข้อมูล IX และ IY กับแอกคิวมูเลเตอร์ D โดยXGDX เป็นการเปลี่ยนระหว่าง IX กับ D และ XGDY เป็นการเปลี่ยนระหว่าง IY กับ D

LDX และ LDY (Load Index Register X & Load Index Register Y) เป็นคำสั่งในการใช้โหลดข้อมูลขนาด 16 บิต จากหน่วยความจำ 2 แอดเดรส (แอดเดรสละ 8 บิต) มาเก็บไว้ใน IX และ IY

STX STY (Store Index Register X & Store Index Register Y) เป็นคำสั่งในการเก็บข้อมูลจาก IX และ IY ลงในหน่วยความจำ โดยไบต์ต่ำเก็บในแอดเดรสต่ำ ส่วนไบต์สูงเก็บในแอดเดรสถัดไป

LDS (Load Stack Pointer) เป็นคำสั่งใช้ในการกำหนดค่าแอดเดรสของสแต็กให้แก่รีจิสเตอร์ตัวชี้สแต็ก (SP)

STS (Store Stack Pointer) เป็นคำสั่งใช้ในการนำค่าของ SP เก็บลงในหน่วยความจำ

PSHX และ PSHY (Push X onto Stack & Push Y onto Stack) เป็นการเก็บค่า IX และ IY กลับมาคืนจากสแต็ก โดยข้อมูลไบต์สูงของ IX และ IY จะออกมาก่อน SP จะเพิ่มขึ้น 2 ค่า (SP=SP+2)

TSX และ TYS (Transfer Stack pointer to X & Transfer Stack pointer to Y) เป็นการนำค่าของตัวชี้สแต็กที่เพิ่มขึ้น 1ค่า (SP+1) มาเก็บไว้ใน IX หรือ IY

TXS และ TYS (Transfer X to Stack pointer & Transfer Y to Stack pointer) เป็นการนำค่าของ IX หรือ IY ที่ลดลง 1 แอดเดรส (IX-1 หรือ IY-1) ไปเก็บใน SP

กลุ่มคำสั่งเปลี่ยนแปลงแก้ไขข้อมูล แบ่งออกเป็น 3 ส่วน

- กลุ่มคำสั่งเพิ่มค่า (Increment) มี 6 คำสั่ง

INCA และ INCB (Increment Accumulator A & Increment Accumulator B) เป็นคำสั่งใช้เพิ่มค่าแอกคิวมูเลเตอร์ A หรือ B ครั้งละ 1 ค่า

INC (Increment memory byte) เป็นคำสั่งใช้เพิ่มค่าของข้อมูลที่อ้างถึง 1 ค่า

INX และ INY (Increment Index register X & Increment Index register Y) เป็นคำสั่งใช้เพิ่มค่า IX หรือ IY ครั้งละ 1 ค่า

INS (Increment stack pointer) เป็นคำสั่งใช้เพิ่มค่าของรีจิสเตอร์ตัวชี้สแต็ก (SP) ขึ้นครั้งละ 1 ค่า

- กลุ่มคำสั่งลดค่า (Decrement) มี 6 คำสั่ง

DECA และ DECB (Decrement Accumulator A & Decrement Accumulator B) เป็นคำสั่งใช้ลดค่าของแอกคิวมูเลเตอร์ A หรือ B

DEC (Decrement memory byte) เป็นคำสั่งใช้ลดค่าของข้อมูลที่อ้างถึง

DECX และ DECY (Decrement IX & (Decrement IY) เป็นคำสั่งใช้ลดค่าของ IX หรือ IY ลงครั้งละ 1 ค่า

DES (Decrement stack pointer) เป็นคำสั่งใช้ลดค่าของรีจิสเตอร์ตัวชี้สแต็ก (SP) ลงครั้งละ 1 ค่า

- กลุ่มคำสั่งแปลงค่า (Complement) มี 3 คำสั่ง คือ

COM A และ COM B (1's Complement A & 1's Complement B) เป็นคำสั่งที่ใช้ในการแปลงค่าข้อมูลในแอกคิวมูเลเตอร์ A หรือ B เป็น 1's Complement โดยนำค่า $FF ลบด้วยค่าในแอกคิวมูเลเตอร์ แล้วนำมาเก็บไว้ในแอกคิวมูเลเตอร์

COM (1's Complement memory byte) เช่นเดียวกับ COM A และ COM B หากแต่จะเปลี่ยนจากข้อมูลในแอกคิวมูเลเตอร์มาใช้กับข้อมูลที่อยู่ในหน่วยความจำ แล้วเก็บเข้าไปในแอดเดรสเดิม รายละเอียดของการแปลงข้อมูลทำเช่นเดียวกับ COM A และ COM B

กลุ่มการเลื่อนและหมุนข้อมูล มี 3กลุ่มย่อยคือ

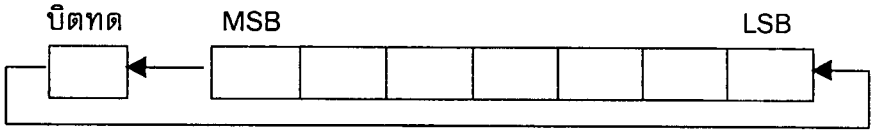
- กลุ่มคำสั่งหมุนข้อมูล มี 6 คำสั่ง คือ

ROLA (Rotate Left A) เป็นการหมุนข้อมูลในแอกคิวมูเลเตอร์ A

ROLB (Rotate Left B) เป็นการหมุนข้อมูลในแอกคิวมูเลเตอร์ B

ROL (Rotate Left) เป็นการหมุนข้อมูลในหน่วยความจำ

ทั้ง 3 คำสั่ง เป็นคำสั่งให้หมุนข้อมูลไปทางซ้าย โดยมีรูปแบบดังนี้



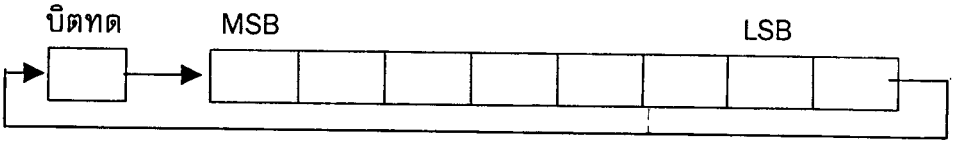
คำสั่งทั้ง 3 จะแตกต่างที่ คำสั่ง ROLA เป็นการหมุนแอกคิวมูเลเตอร์ A และคำสั่ง ROLB เป็นการหมุนข้อมูลใน B หรือ ROL เป็นคำสั่งที่ใช้ในการหมุนในหน่วยความจำ

ROLA (Rotate Right A)

ROLB (Rotate Right B)

ROL (Rotate Right)

ทั้ง 3 คำสั่ง เป็นคำสั่งให้หมุนข้อมูลไปทางขวา โดยมีรูปแบบดังนี้



จะเห็นว่ามีทิศทางตรงกันข้ามกับ ROL โดยบิตสุดท้าย (D0) จะหมุนมาแทนที่บิตทด (C) แล้วข้อมูลในบิตทดจะเลื่อนเข้าไปแทนที่ D

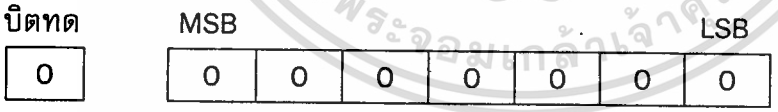
กลุ่มการเลื่อนข้อมูลคณิตศาสตร์

ASLA (Arithmetic Shift Left A) เป็นคำสั่งเลื่อนข้อมูลใน A ไปทางซ้ายโดยบิต LSB จะมีข้อมูล "0" เลื่อนเข้ามาแทนที่ที่ บิต MSB จะเลื่อนไปที่บิตทด โดยมีรูปแบบการเลื่อนข้อมูลดังรูป

สมมติให้ข้อมูล A เป็น $40 เมื่อเลื่อนข้อมูลตามคำสั่ง ASLA จะได้ผลดังนี้

$40

ก่อนทำคำสั่ง ASLA



$80

หลังทำคำสั่ง ASLA

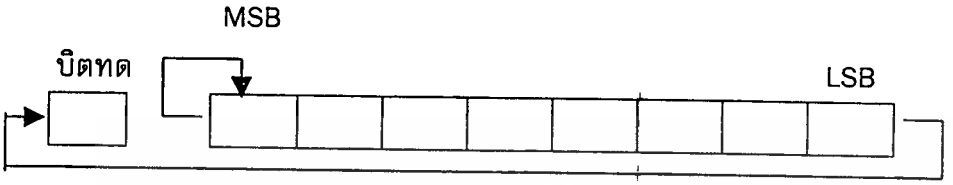
นั่นคือจากข้อมูล $40 เมื่อกระทำคำสั่ง ASLA ค่าจะกลายเป็น $80 ซึ่งก็คือเกิดการคูณ 2 ให้แก่ข้อมูลเดิม

ASLB (Arithmatic Shift Left B) มีหลักการเช่นเดียวกับ ASLA แต่เป็นการเลื่อนข้อมูลในแอกคิวมูเลเตอร์ B

ASL (Arithmatic Shift Left) เป็นคำสั่งเลื่อนข้อมูลในหน่วยความจำไปทางซ้าย มีรูปแบบทำคำสั่งเช่นเดียวกับ ASLA และ ASLB

ASLD (Arithmatic Shift Left D) เป็นคำสั่งเลื่อนข้อมูลของแอกคิวมูเลเตอร์ D ขนาด 16 บิต มีรูปแบบการเลื่อนข้อมูลเช่นเดียวกับ ASLA และ ASLB หากแต่จำนวนข้อมูลเพิ่มจาก 8 บิตเป็น 16 บิต

ASRA (Arithmatic Shift Right A) เป็นคำสั่งเลื่อนข้อมูลใน A ไปทางขวาของแอกคิวมูเลเตอร์ A โดยมีรูปแบบการเลื่อนข้อมูลดังนี้



จะเห็นว่าที่บิต MSB จะเลื่อนข้อมูลวนตำแหน่งเดิม ดังนั้นข้อมูลในบิต MSB จะไม่เปลี่ยนแปลง จากนั้นข้อมูลเดิมของ MSB ก็จะเลื่อนไปทางขวาไล่ไปเรื่อย ๆ ทีละบิตที่บิต LSB ข้อมูลจะถูกเลื่อนไปที่บิตทด

สมมติให้ A เป็น $40 เมื่อทำคำสั่ง ASRA จะได้ผลดังนี้

นั่นคือ เมื่อกระทำคำสั่ง ASRA แล้ว ค่าของข้อมูลจะถูกหารด้วย 2 จาก $40 กลายเป็น $20 ดังนั้นจึงสามารถสรุปได้ว่า ถ้ากระทำคำสั่ง ASLA จะเป็นการคูณ 2 แต่ถ้าเป็นคำสั่ง ASRA ข้อมูลจะถูกหาร 2

ASRB (Arithmetic Shift Right B) มีรูปแบบเช่นเดียวกับ ASRB แต่เป็นการเลื่อนข้อมูลที่ B

ASR (Arithmetic Shift Right) มีรูปแบบเช่นเดียวกับ ASRA และ ASRB แต่เป็นการเลื่อนข้อมูลใด ๆในหน่วยความจำ

กลุ่มคำสั่งเลื่อนข้อมูลทางลอจิก

มีทั้งเลื่อนไปทาง ซ้ายหรือขวาโดยจะให้ค่าศูนย์มาทดแทนที่บิต LSB ในกรณีเลื่อนข้อมูลไปทางซ้าย และเลื่อนค่าศูนย์เข้าที่บิต MSB ในกรณีเลื่อนข้อมูลไปทางขวา กลุ่มคำสั่งมี 8 คำสั่ง คือ

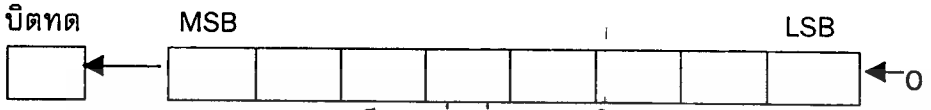
คำสั่งเลื่อนข้อมูลไปทางซ้าย

LSLA (Logic Shift Left A)

LSLB (Logic Shift Left B)

LSL (Logic Shift Left)

ทั้ง 3 คำสั่งมีรูปแบบการเลื่อนข้อมูลเหมือนกัน แต่แตกต่างที่ข้อมูลที่นำมากระทำ มีรูแบบการเลื่อนข้อมูลดังนี้



LSLD (Logic Shift Left D) เป็นคำสั่งเลื่อนข้อมูลในแอกคิวมูเลเตอร์ D ไปทางซ้าย

ทุกคำสั่งการเลื่อนข้อมูลไปทางซ้าย จะมีผลต่อแฟล็ก N,Z,V และ C ทั้งสิ้น

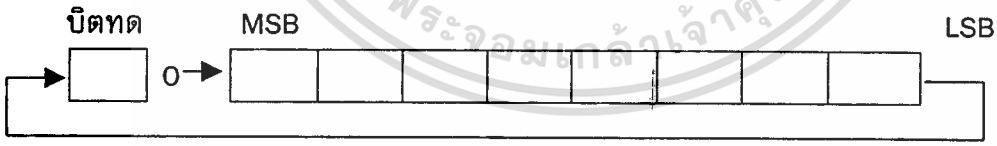
คำสั่งเลื่อนข้อมูลไปทางขวา

LSRA (Logic Shift Right A)

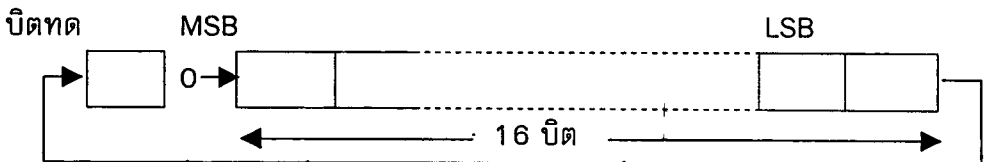
LSRB (Logic Shift Right B)

LSR (Logic Shift Right)

ทั้ง 3คำสั่งมีรูปแบบการเลื่อนข้อมูลเหมือนกัน แตกต่างที่ที่มาของข้อมูลที่นำมากระทำ คือ นำเอาข้อมูลแอกคิวมูเลเตอร์ A มากระทำ (LSRA) นำ B มากระทำ (LSRB) และนำข้อมูลหน่วยความจำมากระทำ (LSR) รูปแบบการเลื่อนข้อมูลคือ



LSRD (Logic Shift Right D)เป็นคำสั่งเลื่อนข้อมูลในแอกคิวมูเลเตอร์ D ขนาด 16 บิต ไปทางซ้าย



ทุกคำสั่งในการเลื่อนข้อมูลลอจิกไปทางขวานี้ จะมีผลต่อแฟล็ก Z,V, และ C ในขณะที่แฟล็ก N จะมีค่าเป็น “0”

2. กลุ่มคำสั่งทางคณิตศาสตร์ (Arithmetic instruction)แบ่งได้เป็น 3 กลุ่ม

กลุ่มคำสั่งการบวก มี 9 คำสั่ง

ADDA และ ADDB (Add memory to A & Add memory to B) เป็นคำสั่งใช้ในการบวกค่าของแอกคิวมูเลเตอร์ A หรือ B เข้ากับค่าข้อมูลในหน่วยความจำแล้วนำผลลัพธ์มาเก็บไว้ในแอกคิวมูเลเตอร์ A หรือ B การบวกข้อมูลทั้ง 2 คำสั่งนี้เป็นการบวกข้อมูลขนาด 8 บิต

ABA (Add Accumulators) คำสั่งนี้เป็นการบวกค่าของแอกคิวมูเลเตอร์ทั้ง 2 ตัวคือ A และ B เข้าด้วยกัน ผลลัพธ์จะเก็บไว้ที่แอกคิวมูเลเตอร์ A

ADCA และ ADCB (Add with Carry To A & Add with Carry To B) เป็นคำสั่งใช้บวกค่าของแอกคิวมูเลเตอร์กับค่าข้อมูลขนาด 8 บิต โดยมีการคิดทดด้วยผลลัพธ์ถูกนำมาเก็บไว้ที่แอกคิวมูเลเตอร์ A หรือ B

ADDD (Add 16 bit to D) เป็นคำสั่งใช้ในการบวกข้อมูลขนาด 16 บิต ระหว่างแอกคิวมูเลเตอร์ D กับหน่วยความจำ ผลลัพธ์เก็บไว้ที่ D

ADX และ ADY (Add B to X & Add B to Y) เป็นคำสั่งที่ใช้รวมค่าของแอกคิวมูเลเตอร์ B กับรีจิสเตอร์ IX และ IY โดย 8 บิตแรกของ IX และ IY จะถูกบวกด้วย 00 ส่วน 8 บิตหลังจะถูกรวมเข้ากับค่าของ B ผลลัพธ์เก็บไว้ที่ IX หรือ IY

DAA (Decimal Adjust A) เป็นคำสั่งใช้ในการปรับค่าในแอกคิวมูเลเตอร์ A ซึ่งไบนารี 8 บิตปกติเป็นเลขฐานสิบหรือรหัส BCD (Binary Code Decimal)

กลุ่มคำสั่งลบ มี 9 คำสั่ง

SUB A และ SSUB B (Subtractor memory from A & Subtractor memory from B) เป็นคำสั่งใช้ในการลบค่าในแอกคิวมูเลเตอร์ A หรือ B ด้วยค่าข้อมูลในหน่วยความจำ ผลลบถูกเก็บไว้ใน A หรือ B การลบในคำสั่งนี้เป็นการลบข้อมูลขนาด 8 บิต

SBA (Subtractor B from A) ใช้ในการลบค่าใน A ด้วย B ผลลัพธ์เก็บไว้ใน A

SBCA และ SBCB (Subtractor with Carry from A & Subtractor with Carry from B) เป็นการลบค่าใน A หรือ B ด้วยค่าข้อมูลในหน่วยความจำ โดยมีการคิดตัวทด (ตัวยืม)ด้วย ผลลัพธ์ถูกเก็บไว้ใน A หรือ B

SUBD (Subtractor memory from D) เป็นคำสั่งลบข้อมูล 16 บิต ระหว่างค่าในแอกคิวมูเลเตอร์ D กับในหน่วยความจำ ผลลัพธ์เก็บไว้ที่ D

NEGA และ NEGB (2's complement A & 2's complement B) เป็นคำสั่งที่ใช้การจัดการข้อมูลในแอกคิวมูเลเตอร์ A และ B ให้เป็นค่า 2's complement โดยนำค่า 00 ตั้งลบด้วยค่าใน A หรือ B แล้วผลลัพธ์ที่ได้จะเก็บไว้ใน A หรือ B

NEG (2's complement memory byte) เป็นการทำให้ค่าข้อมูลในหน่วยความจำเป็นตัวเลข 2's complement ด้วยการนำข้อมูลลบออกจาก 00 แล้วผลลัพธ์จะถูกนำมาเก็บไว้ในหน่วยความจำ

กลุ่มคำสั่งการคูณและการหาร มี 3 คำสั่ง

MUL (Multiply 8 by 8) เป็นคำสั่งคูณข้อมูลขนาด 8 บิต ระหว่างข้อมูลในแอกคิวมูเลเตอร์ A กับ B ผลคูณจะถูกเก็บไว้ในแอกคิวมูเลเตอร์ D

IDIV (Integer Divide 16 by 16) เป็นคำสั่งการหารเลขจำนวนเต็มขนาด 16 บิต โดยตัวตั้งจะถูกเก็บไว้ใน D ส่วนตัวหารจะถูกเก็บไว้ใน IX เศษหารจะถูกเก็บไว้ใน IX เศษของการหารเก็บไว้ใน D

FDIV (Fractional Divide 16 by 16) เป็นคำสั่งการหารเลขเศษส่วนขนาด 16 บิต โดยตัวตั้งเก็บไว้ใน D ส่วนตัวหารเก็บไว้ใน IX ผลหารจะถูกเก็บไว้ใน IX

คำสั่ง IDIV และ FDIV จะใช้งานร่วมกันเมื่อการหารเกิดเศษและต้องการหารค่าของเศษ

3. กลุ่มคำสั่งทางลอจิก (Logic Instruction) มีด้วยกัน 3 กระบวนการ คือ แอนด์ ออร์และเอ็กซ์คลูซีฟออร์ มีคำสั่งรวมทั้งสิ้น 6 คำสั่งดังนี้

ANDA และ ANDB (AND A with memory & AND B with memory)เป็นคำสั่งการแอนด์ข้อมูลขนาด 8 บิต ระหว่างแอกคิวมูเลเตอร์ A หรือ B กับค่าข้อมูลใด ๆ ผลของการแอนด์เก็บไว้ใน A หรือ B

ORAA และ ORAB (OR Accumulator A & OR Accumulator B) เป็นคำสั่งการออร์ข้อมูลขนาด 8 บิต ระหว่างแอกคิวมูเลเตอร์ A หรือ B กับข้อมูลในหน่วยความจำ ผลของการออร์ถูกนำมาเก็บไว้ใน A หรือ B

EORA EORB (Exclusive OR A with memory & Exclusive OR B Exclusive OR) เป็นคำสั่งในการเอ็กซ์คลูซีฟออร์ข้อมูลขนาด 8 บิต ระหว่างแอกคิวมูเลเตอร์ A หรือ B กับข้อมูลในหน่วยความจำ ผลนำมาเก็บไว้ใน A หรือ B

4. กลุ่มคำสั่งการเปรียบเทียบและทดสอบข้อมูล (Data test instructions) แบ่งได้ 2 ลักษณะ 9 ตามขนาดของข้อมูล 8 บิต และ 16 บิต ดังนี้

กลุ่มคำสั่งการเปรียบเทียบและตรวจสอบข้อมูลขนาด 8 บิต มีด้วยกัน 8 คำสั่ง และแต่ละคำสั่งเมื่อกระทำไปแล้วจะมีผลต่อต่อการเปลี่ยนแปลงของแฟล็ก N,Z,V และ C

CMPA และ CMPB (Compare A to memory & Compare B to memory) เป็นคำสั่งการเปรียบเทียบข้อมูลระหว่างแอกคิวมูเลเตอร์ A หรือ B กับข้อมูลในหน่วยความจำ

CBA (Compare A to B) เป็นคำสั่งการเปรียบเทียบข้อมูลระหว่างแอกคิวมูเลเตอร์ A กับ B

TSTA และ TSAB (Test for zero or minus of A & Test for zero or minus of B) เป็นการตรวจสอบว่าข้อมูลระหว่างแอกคิวมูเลเตอร์ A และ B มีค่าเป็นศูนย์หรือติดลบ

TST (Test for zero or minus of memory) เป็นการตรวจสอบว่าข้อมูลในหน่วยความจำว่ามีค่าเป็นศูนย์หรือติดลบ

BITA BITB (Bit test A with memory & Bit test B with memory) เป็นการเปรียบเทียบข้อมูลระดับบิตของแอกคิวมูเลเตอร์ A หรือ B กับข้อมูลในหน่วยความจำ นั่นคือการเปรียบเทียบระหว่างบิตต่อบิตของ A หรือ B กับข้อมูลใด ๆ

กลุ่มคำสั่งการเปรียบเทียบและตรวจสอบข้อมูลขนาด 16 บิต มี 3 คำสั่งด้วยกัน และ

แต่ละคำสั่งเมื่อกระทำไปแล้วจะมีผลต่อต่อการเปลี่ยนแปลงของแฟล็ก N,Z,V และ C เพื่อใช้ประโยชน์ในการกำหนดเงื่อนไขการกระโดดต่อไป

CPD (Compare D to memory 16 bit) เป็นคำสั่งการเปรียบเทียบข้อมูลในแอกคิวมูเลเตอร์ D กับข้อมูลขนาด 16 บิตในหน่วยความจำ

CPX (Compare X to memory 16 bit) เป็นคำสั่งการเปรียบเทียบข้อมูลใน IX กับข้อมูลขนาด 16 บิตในหน่วยความจำ

CPY (Compare Y to memory 16 bit) เป็นคำสั่งการเปรียบเทียบข้อมูลใน IY กับข้อมูลขนาด 16 บิตในหน่วยความจำ

5. กลุ่มคำสั่งการกระโดด (Jump and Branch Instruction) แบ่งออกเป็น 2 ประเภทคือ

กลุ่มกระโดดแบบไม่มีเงื่อนไข มีด้วยกัน 7 คำสั่งคือ

JNP (Jump) เป็นคำสั่งที่ให้ซีพียูข้ามไปจัดการข้อมูล หรือคำสั่งในหน่วยความจำในตำแหน่งใด ๆ ที่ถูกกำหนดในโอเปอร์แรนด์ ค่า PC จะเท่ากับแอดเดรสปลายทางที่ระบุให้กระโดดไป

JSP (Jump to Sudroutine) เป็นคำสั่งที่ให้ซีพียูกระโดดไปทำงานยังโรแกรมย่อย เมื่อทำคำสั่งค่าของ PC เดิมจะถูกเก็บไว้ในสแต็ก จากนั้นค่า PC จะถูกเปลี่ยนเป็นค่าแอดเดรสที่ต้องการกระโดดไป จากนั้นซีพียูก็จะทำงานตามที่โปรแกรมย่อยกำหนด จะกลับมายังโปรแกรมหลักได้ก็ต่อเมื่อพบคำสั่ง RTS

BRA (Branch Always) เป็นคำสั่งที่ให้ซีพียูข้ามไปทำงานที่แอดเดรสปลายทางที่ถูกระบุด้วยค่าสัมพัทธ์ (Relative:Rel) ที่อยู่ตามหลังคำสั่ง BRA นี้ มีลักษณะคล้าย ๆ คำสั่ง JR ของ Z80

BRN (Branch Never) เมื่อทำคำสั่งนี้ค่า PC จะเพิ่ม 2 ค่า (PC=PC+2) นั่นคือ เป็นการสั่งให้ซีพียูกระโดดข้ามไปทำงานที่แอดเดรสปลายทางที่ 2 แอดเดรสข้างหน้า

BSR (Branch to Subroutine) เป็นคำสั่งที่ให้ซีพียูกระโดดไปทำงานที่โปรแกรมย่อยโดยค่า PC เดิมจะถูกเก็บไว้ในสแต็กและค่า PC จะเปลี่ยนไปเป็นแอดเดรสที่กำหนดปลายทาง ซึ่งได้มาจากการคำนวณค่าสัมพัทธ์ และจะกลับมาโปรแกรมหลักเมื่อพบคำสั่ง RTS

RTS (Return from Subroutine) เมื่อพบคำสั่งนี้ซีพียูจะกระโดดกลับไปทำงานที่โปรแกรมหลัก โดยเรียกค่า PC เดิมที่ถูกเก็บไว้ในสแต็กออกมา

RTI (Return from Interrupt) เมื่อมีการอินเตอร์รัปต์ ซีพียูจะทำการเก็บค่าของรีจิสเตอร์รหัสเงื่อนไข (CCR) แอกคิวมูเลเตอร์ A และ B ค่า IX , IY และค่า PC ไว้ในสแต็กทั้งหมด และเมื่อกระทำโปแกรมตอบสนองการอินเตอร์รัปต์เรียบร้อยแล้ว พบคำสั่งนี้ซีพียูจะทำการดึงค่าของ CR,A,B,IX,IY และ PC คืนกลับมาจากสแต็กเพื่อทำงานปกติต่อไป

ความแตกต่างของการ JUMP และ BRANCH

เพิ่มเติมอีกนิดจากการใช้คำสั่ง JUMP หรือ BRANCH ซึ่งการใช้งานจะขึ้นอยู่กับผู้เขียนโปรแกรม โดยที่คำสั่ง JUMP อันได้แก่ JMP,JSR จะสามารถกระโดดไปที่ใดในหน่วยความจำก็ได้ โดยกำหนดค่าแอดเดรสลงไปชัดเจน ทำให้ต้องใช้หน่วยความจำสิ้นเปลือง แต่ถ้าเป็นคำสั่ง BRA หรือ BSR จะสามารถกระโดดไปได้ในขอบเขต +127 แบะ -128 ตำแหน่งจากจุดที่กระทำคำสั่ง ทำให้ใช้หน่วยความจำในการเขียนโปรแกรมน้อยกว่า และซีพียูทำงานเร็วกว่า แต่ก็มีข้อด้อยตรงที่ขอบเขตของการกระโดด

คำสั่งเกี่ยวกับการ BRANCH ถ้าจะเทียบกับไมโครโปรเซสเซอร์ Z80 ที่นิยมใช้ในอดีตนั้น ก็คือคำสั่ง Jump Relative หรือ JR นั่งเอง ซึ่งค่าสัมพัทธ์ (relative) ต้องมีการคำนวณด้วยดังสูตร

ค่าสัมพัทธ์ (relative : rr) = PC ใหม่ - PC เก่า - 2 โดย rr จะอยู่ที่ค่า 8 บิตท้ายของผลที่ได้ เช่น PC ใหม่

= $E009, PC เก่า = $E012 ดังนั้นค่า rr = $E009 - $E012 - 2 = $FFF5 ค่า rr จริง ๆ คือ $F5

กลุ่มคำสั่งกระโดดแบบมีเงื่อนไข แบ่งได้ 4 กลุ่มคือ

- กลุ่มคำสั่งกระโดดแบบทดสอบบิตในไบต์ มีด้วยกัน 2 คำสั่ง

BRSET (Branch if Bit Set) เป็นคำสั่งที่ให้ซีพียูกระโดดไปทำงานยังแอดเดรสปลายทางหลังจากที่มีการตรวจสอบใด ๆ ในหน่วยความจำแล้วพบว่าบิตที่ตรวจสอบนั้นเป็น “1” แต่ถ้าบิตที่ตรวจสอบนั้นเป็น “0” ก็จะไม่มีการกระโดด

BRCLR (Branch if Bit Clear) เป็นคำสั่งที่มีลักษณะการทำงานตรงข้ามกับ BRSET คือ ซีพียูจะกระโดดไปยังแอดเดรสปลายทางที่กำหนด เมื่อตรวจสอบบิตที่ต้องการแล้วพบว่าเป็น “0” ถ้าเป็น “1” ก็จะไม่มีการกระโดด

- กลุ่มคำสั่งกระโดดแบบทดสอบแฟล็ก มีด้วยกัน 8 คำสั่ง จากบิตเงื่อนไขของ CCR 4 บิต คือ บิตตัวทด (Carry:C) , บิตศูนย์ (Zero:Z) , บิตลบ (Negative:N) และ บิตเกิน (Overflow:V)

BCC (Branch if Carry Clear) เป็นคำสั่งที่ให้ซีพียู กระโดดเมื่อตรวจสอบบิตตัวทด (Carry : C) แล้วพบค่าเป็น “0”

BCS (Branch if Carry Set) ซีพียูจะกระโดดเมื่อตรวจสอบบิตตัวทดแล้วพบว่าเป็น “1”

BNE (Branch if Not Zero) เป็นคำสั่งให้ซีพียูกระโดดเมื่อตรวจสอบบิตศูนย์ (Zero) แล้วพบว่าเป็น “0”

BEQ (Branch if Zero) เป็นคำสั่งให้ซีพียูกระโดดเมื่อตรวจสอบบิตศูนย์แล้วพบว่าเป็น “1” นั่นคือ เกิดค่าศูนย์ในกระบวนการประมวลผลข้อมูล

BPL (Branch if Plus) เป็นคำสั่งให้ซีพียูกระโดดเมื่อตรวจสอบบิตลบแล้วพบว่าเป็น “0” นั่นคือ ค่าข้อมูลเป็นบวก

BMI (Branch if Minus) เป็นคำสั่งให้ซีพียูกระโดดเมื่อตรวจสอบบิตลบแล้วพบว่าเป็น "1" นั่นคือ เกิดเป็นค่าลบ

BVC (Branch if Overflow Clear) เป็นคำสั่งให้ซีพียูกระโดด เมื่อตรวจสอบบิตเกิน (Overflow) แล้วพบว่าเป็น "0" นั่นคือ ไม่เกิดค่าเกินย่านที่กำหนดในการประมวลผลข้อมูล (ค่าเกินคือ ค่าที่มากหรือน้อยกว่า +127 ถึง −128)

BVS (Branch if Overflow Set) เป็นคำสั่งให้ซีพียูกระโดด เมื่อตรวจสอบบิตเกิน แล้วพบว่าเป็น "1" นั่นคือ เกิดค่าเกินย่าน +127 ถึง −128 ในกระบวนการประมวลผลข้อมูล

กลุ่มคำสั่งกระโดดแบบเปรียบเทียบกับข้อมูลศูนย์ ในกลุ่มคำสั่งนี้จะทำการเปรียบเทียบข้อมูลศูนย์ว่ามากกว่า น้อยกว่าหรือเท่ากัน โดยมีการคำนึงถึงเครื่องหมายของข้อมูลด้วยว่าเป็นลบหรือบวก มีด้วยกันทั้งสิ้น 6 คำสั่ง ดังนี้

BLT (Branch if = zero) เมื่อกระทำคำสั่งนี้ซีพียูจะตรวจสอบการเอ็กซ์คลูซีฟออร์ของบิตลบ (N) กับบิตเกิน (V) ใน CCR ว่าเป็น "1" หรือไม่ ถ้าเป็นนั่นแสดงว่าข้อมูลที่ทำการเปรียบเทียบน้อยกว่า "0" ซีพียูจะทำการกระโดด

BLE (Branch if = zero) เป็นคำสั่งตรวจสิบข้อมูลว่าน้อยกว่าหรือเท่ากับศูนย์หรือไม่ ถ้าใช่ซีพียูก็จะกระโดดไปทำงานตามแอดเดรสที่กำหนดต่อไป

BEQ (Branch if = Zero) นอกจากจะใช้คำสั่งนี้ในการกระโดดแบบทดสอบแฟล็กแล้วยังสามารถใช้โดยกลุ่มคำสั่งกระโดดแบบเปรียบเทียบ โดยคำนึงถึงเครื่องหมายได้ด้วย นั่นคือถ้าเปรียบเทียบข้อมูลแล้วพบว่า เท่ากับ "0" ซีพียูก็จะกระโดดไปทำงานตามแอดเดรสที่กำหนด

BEE (Branch if = zero) เป็นคำสั่งที่ให้ซีพียูกระโดดเมื่อผลการเปรียบเทียบออกมาแล้วมีค่ามากกว่าหรือเท่ากับ "0"

BGT (Branch if > zero) เป็นคำสั่งที่ให้ซีพียูกระโดดเมื่อผลการเปรียบเทียบมีค่ามากกว่า "0"

BNE (Branch if not = zero) เป็นคำสั่งให้ซีพียูกระโดดเมื่อเปรียบเทียบแล้วพบว่า ข้อมูลนั้นไม่ใช่ค่า "0"

กลุ่มคำสั่งแบบเปรียบเทียบขอ้มูล 2 ข้อมูล เป็นกลุ่มคำสั่งที่ใช้กำหนดเงื่อนไขการกระโดด หลังจากเกิดการเปรียบเทียบข้อมูล 2 ข้อมูลว่ามากกว่า น้อยกว่า หรือเท่ากัน ที่เปรียบเทียบกับค่าศูนย์เป็นหลักมีด้วยกัน 6 คำสั่ง ดังนี้

BLO (Branch if Lower) เป็นคำสั่งให้ซีพียูกระโดดเมื่อผลการเปรียบเทียบข้อมูลปรากฏว่าข้อมูลหลักน้อยกว่า โดยบิตทดจะเซตเป็น "1"

BLS (Branch if Lower or Same) เป็นคำสั่งให้ซีพียูกระโดดเมื่อผลการเปรียบเทียบข้อมูล ปรากฏว่าข้อมูลหลักน้อยกว่าหรือเท่ากันข้อมูลที่นำมาเปรียบเทียบ โดยซีพียูจะตรวจสอบผลได้จากการเซตของบิตศูนย์หรือบิตทด

BEQ (Branch if = Zero) เป็นคำสั่งให้ซีพียูกระโดดเมื่อข้อมูลที่นำมาเปรียบเทียบเท่ากันเท่านั้น โดยซีพียูจะตรวจสอบได้จากเซตของบิตศูนย์

BHS (Branch if Higher or Same) เป็นคำสั่งให้ซีพียูกระโดดเมื่อข้อมูลหลักมากกว่าหรือเท่ากับข้อมูลที่นำมาเปรียบเทียบ โดยซีพียูจะตรวจสิบผลจากการเคลียร์ของบิตทด (C=0)

BHI (Branch if Higher) เป็นคำสั่งให้ซีพียูกระโดดเมื่อข้อมูลหลักมากกว่าข้อมูลที่นำมาเปรียบเทียบ โดยซีพียูจะตรวจสอบจากการออร์กันของบิตศูนย์และบิตทดต้องได้ศูนย์ (C+Z=0)

BNE (Branch if Not = Zero) เป็นคำสั่งให้ซีพียูกระโดด หากเปรียบเทียบข้อมูลทั้งสองแล้วไม่เท่ากัน โดยซีพียูตรวจสอบได้จากบิตศูนย์ต้องเป็น "0" (Z=0)

ข้อสังเกต คำสั่ง BNE และ BEG สามารถนำไปใช้ในกลุ่มคำสั่งการกระโดดแบบมีเงื่อนไขได้ทุกแบบ ทั้งที่ผลของการทำงานก็เหมือนกัน ทั้งนี้เนื่องจากขอบเขตการทำงานของมันจะค่อนข้างกว้างมากนั่นเอง คำสั่ง BNE จะให้ซีพียูกระโดด เมื่อผลการเปรียบเทียบแล้วข้อมูลไม่เท่ากัน ไม่เท่ากับศูนย์ ในขณะที่คำสั่ง BEQ จะให้ซีพียูกระโดดเมื่อผลการเปรียบเทียบแล้วขอมูลเท่ากันหรือเท่ากับศูนย์ นี่คือคุณสมบัติใช้คำสั่งอย่างมีประสิทธิภาพของ 68HC11

6. กลุ่มคำสั่งจัดการกับรีจิสเตอร์รหัสเงื่อนไข (CCR Instructions)

ในกลุ่มนี้เป็นคำสั่งที่ทำการเซตหรือเคลียร์บิตต่างๆ ในรีจิสเตอร์รหัสเงื่อนไข มีด้วยกัน 6 คำสั่งคือ

SEC (Set Carry) เป็นคำสั่งที่ใช้ในการเซตบิตทด หรือ Carry ให้เป็น "1"

CLC (Clear Carry) เป็นคำสั่งเคลียร์บิตทดให้เป็น "0"

SEV (Set two's complement overflow bit) เป็นคำสั่งเซตบิตโอเวอร์โฟลว์ให้เป็น "1"

CLV (Clear two's complement overflow bit) เป็นคำสั่งเคลียร์บิตโอเวอร์โฟลว์ให้เป็น "0"

SEI (Set interrupt mask) เป็นคำสั่งเซตบิตอินเตอร์รัปต์มาส์กในก CCR ให้เป็น "1" เพื่อเป็นการดิสเอเบิลการอินเตอร์รัปต์

CLI (Clear interrupt mask) เป็นคำสั่งเคลียร์บิตอินเตอร์รัปต์มาส์กใน CCR ให้เป็น "0" เพื่อเป็นการอีนาเบิลการอินเตอร์รัปต์

7. กลุ่มคำสั่งควบคุม (Control Instruction)

เป็นกลุ่มคำสั่งที่ใช้ควบคุมการทำงานหลักของไมโครคอนโทรลเลอร์ ซึ่งจะเกี่ยวข้องกับอินเตอร์รัปต์ และโหมดการทำงานประหยัดพลังงาน มีด้วยกัน 4 คำสั่งคือ

SWI (Software Interrupt) เป็นคำสั่งอินเตอร์รัปต์ 68HC11 โดยใช้ซอฟต์แวร์ซึ่งตามปกติการอินเตอร์รัปต์มักเกิดจากสัญญาณทางฮาร์ดแวร์ แต่ใน 68HC11 สามารถอินเตอร์รัปต์โดยซอฟต์แวร์ได้ด้วยคำสั่งนี้ เมื่อกระทำคำสั่งซีพียูจะทำการเก็บค่ารีจิสเตอร์ต่างๆ ไว้ในสแต็ก เซตบิต I ใน CCR เพื่อบอกสถานะการอินเตอร์รัปต์

WAI (Wait for Interrupt) เป็นคำสั่งให้ 68HC11 รอรับการอินเตอร์รัปต์โดยเมื่อกระทำคำสั่งนี้ ซีพียูจะเก็บค่ารีจิสเตอร์ทุกตัวไว้ในสแต็ก แล้วจะ halt เพื่อหยุดการทำงาน จากนั้นตอการอินเตอร์รัปต์หรือจะเรียกสภาวะนี้ว่า "Wait state"

STOP (Stop internal clock) เมื่อกระทำคำสั่งนี้ ระบบสัญญาณนาฬิกา ภายในชิปจะหยุดการทำงาน ทำให้ซีพียูอยู่ในสภาวะสแตนด์บาย กินกำลังไฟฟ้าต่ำ คำสั่งนี้จะไม่มีผลต่อข้อมูลภายในรีจิสเตอร์หรือแรมภายในชิป 68HC11 จะตอบสนองคำสั่งนี้หรือไม่ขึ้นอยู่กับการกำหนดบิต S ใน CCR ถ้าหากเป็น "1" จะเป็นการดิสเอเบิลคำสั่ง STOP โดยเมื่อเอ็กซีคิวต์คำสั่งนี้ 68HC11 จะมองเหมือนคำสั่ง NOP คือไม่มีการทำงานอะไรทั้งสิ้น เพียงแต่เพิ่มค่า PC เท่านั้น ในทางตรงข้ามถ้าบิต S เป็น "0" จะเป็นการอีนาเบิลคำสั่งนี้แก่ซีพียู ซึ่งเมื่อเอ็กซีคิวต์แล้วต้องทำงานตามที่กำหนด

NOP (No Operation) เป็นคำสั่งที่ทำให้ค่า PC เพิ่มขึ้น ไม่มีรีจิสเตอร์ตัวใดได้รับผลกระทบจากคำสั่งนี้ มักใช้คำสั่งนี้เมื่อต้องการหน่วงเวลาการทำงานของ 68HC11

2.3 ตัวตั้งเวลาโปรแกรมได้ รีลไทม์อินเตอร์รัพต์ และพัลส์แอกคิวมูเลเตอร์

นี่คือรายละเอียดอีกข้อหนึ่งของไมโครคอนโทรลเลอร์ตัวนี้ ที่ใช้ในการควบคุมอุปกรณ์ภายนอกในตอนนี้จะกล่าวถึง การทำงานของตัวตั้งเวลาโปรแกรมได้ (programmable timer) ขนาด 16 บิต, ส่วนรีลไทม์อินเตอร์รัพต์ และพัลส์แอกคิวมูเลเตอร์ภายในชิพ 68HC11 การทำงานของส่วนนี้ทั้งหมดจะแสดงการทำงานโดยใช้รีจิสเตอร์และผลทางลอจิกที่ขาสัญญาณพอร์ต A(PA0-PA7)

ตัวตั้งเวลาโปรแกรมได้

ในตัวตั้งเวลา (timer) นี้จะประกอบด้วย ตัวนับอิสระขนาด 16 บิต (free-running counter) , วงจรอินพุตแคปเจอร์ (input capture) และส่วนเอาต์พุตเปรียบเทียบ (output compare) ซึ่งมีลักษณะการทำงานตามบล็อกไดอะแกรมในรูปที่ 1 รายละเอียดส่วนต่าง ๆ มีดังนี้

ตัวนับอิสระ 16 บิต

เป็นส่วนสำคัญที่สุดในระบบของตัวตั้งเวลาใน 68HC11 ซึ่งในตัวนับนี้จะมีรีจิสเตอร์ 1 ตัว เป็นตัวแสดงข้อมูลในการทำงานเรียกว่า รีจิสเตอร์ตัวนับเวลา (timer counter register : TCNT) ซึ่งเป็นรีจิสเตอร์ขนาด 16 บิต มีแอดเดรสอยู่ ที่ $100E-$100F ใช้เก็บข้อมูลที่ได้จากการนับของตัวนับหลังจากการรีเซตไมโครคอนโทรลเลอร์จะป้อนสัญญาณนาฬิกา E ให้แก่ตัวนับอิสระนี้ ซึ่งสามารถเลือกอัตราความถี่ของสัญญาณนาฬิกาได้ 4 ระดับ จากปรีสเกลเลอร์ คือ หารด้วย 1, 4, 8 หรือ 16 โดยกำหนดสถานะของบิต PR1 และ PR0 ในรีจิสเตอร์ TMSK2 ดังแสดงในตารางที่ 1

ตารางที่ 1 การกำหนดสถานะที่บิต PR0 และ PR1 เพื่อกำหนดตัวหาร (เลือกปรีสเกลเลอร์)

PR1	PR0	ตัวหาร
0	0	1

0	1	4
1	0	8
1	1	16

ในขณะที่ไมโครคอนโทรลเลอร์กำลังทำงานสามารถที่จะอ่านค่าของการนับได้ตลอดเวลาโดยใช้ซอฟแวร์ ทั้งนี้เนื่องจากจังหวะของการอ่านค่านั้นจะมีเฟสตรงข้ามกับสัญญาณนาฬิกา E ทำให้ในการอ่านค่าของการนับจึงไม่มีผลต่อการทำงานของตัวนับ ค่าของการนับในไบต์ต่ำ ($100F : บิต 0-7) จะถูกนำเข้าไปเก็บในบัฟเฟอร์โดยอัตโนมัติ ถึงแม้จะมีการรีเซตเกิดขึ้นค่านี้จะไม่เปลี่ยนแปลง ส่วนในไบต์สูง ($100E : บิต 8-15) ไมโครคอนโทรลเลอร์จะต้องทำการอ่านเอง นั่งคือจะต้องทำการอ่านข้อมูลในไซเกิลที่ติดต่อกันเมื่อตัวนับนับค่าจาก $0000-$FFFF แล้วจะรีเซตค่าเป็น $0000 จากนั้นจะเซตบิตโอเวอร์โฟลว์ของตัวตั้งเวลาคือบิต TOF ในรีจิสเตอร์ TFLG2 การอินเตอร์รัพต์สามารถเกิดขึ้นได้ในจังหวะนี้ ถ้าหากได้รับการอีนาเบิลไว้ที่บิต TOI ในรีจิสเตอร์ TMSK2

ส่วนอินพุตแคปเจอร์

หน้าที่ของส่วนนี้ คือ ทำการบันทึกหรือเก็บค่าเมื่ออินพุตเกิดการเปลี่ยนแปลงตรงตามเงื่อนไขที่กำหนดไว้ของระบบ โดยตรวจสอบจากขอบขาของสัญญาณอินพุตซึ่งสามารถตรวจจับได้ทั้งขอบขาขึ้น, ขอบขาลง หรือใช้ทั้งสองขอบขาลง หรือใช้ทั้งสองขอบขา (ตรวจจับเป็นพัลส์) จากความสามารถอันนี้เองทำให้นำไปใช้วัดระยะเวลาระหว่างสัญญาณพัสส์ได้และเมื่อนำซอฟต์แวร์มาคำนวณก็จะสามารุนำระบบไมโครคอนโทรลเลอร์นี้มาวัดคาบเวลา,ความถี่ หรือปริมาณอื่น ๆ ที่สัมพันธ์กับช่วงเวลา เช่น วัดความเร็ว,ตรวจจับการเคลื่อนที่ และระยะของการเคลื่อนที่ เป็นต้น

หลักการในการวัดความถี่ของส่วนอินพุตแคปเจอร์คือ จะทำการวัดช่วงเวลาของขอบขาลงของสัญญาณ 2 ช่วง ซึ่งซอฟต์แวร์ของไมโครคอนโทรลเลอร์ที่เขียนไว้จะทำการหาคาบเวลาที่ห่างกันของขอบขาของสัญญาณทั้งสอง โดยการนำช่วงเวลาทั้งสองมาลบกัน แล้วนำไปหาร 1 (จากความสัมพันธ์ ความถี่ = 1/เวลา) ก็จะได้ความถี่ของสัญญาณออกมา แต่ด้วยวิธีการเดียวกันนี้ จึงนำไปใช้วัดความกว้างของพัลส์ได้ด้วย โดยให้เริ่มบันทึกค่าเมื่อพบสัญญาณขอบขาขึ้น และบันทึกค่าเมื่อพบสัญญาณของขาลงแล้วลบกันก็จะได้ค่าของความกว้างของพัลส์ออกมา โดยค่าของเวลานั้น ๆ จะถูกเก็บไว้ในรีจีสเตอร์ TNCT

รีจิสเตอร์ที่เกี่ยวข้องในตัวตั้งเวลาโปรแกรมได้

นอกจากรีจิสเตอร์ TCNT ซึ่งเป็นรีจิสเตอร์เก็บค่าการนับในตัวนับอิสระขนาด 16 บิต ในระบบตัวตั้งเวลาโปรแกรมได้ภายใน 68HC11 ยังมีรีจิสเตอร์ที่เกี่ยวข้องอีกหลายตัว สามารถแบ่งกลุ่มได้ดังนี้

1. รีจิสเตอร์ควบคุม

ประกอบด้วยรีจิสเตอร์ TCTL1, TCTL2, TMSK1 และ TMSK2 ดังมีรายละเอียดของความหมายในแต่ละบิตดังนี้

1.1 รีจิสเตอร์ควบคุมตัวตั้งเวลา 1 (Timer control register 1 : TCTL2) เป็นรีจิสเตอร์ขนาด 8 บิต อยู่ที่แอดเดรส $1020 ใช้ในการควบคุมการทำงานของส่วนเอาต์พุตเปรียบเทียบ ความหมายและหน้าที่ในแต่ละบิตมีดังนี้

7	6	5	4	3	2	1	0	
OM2	OL2	OM3	OL3	OM4	OL4	OM5	OL5	TCTL1

OM2, OM3, OM4, OM5 เป็นบิตกำหนดโหมดเอาต์พุต

OL2, OL3, OL4, OL5 เป็นบิตกำหนดระดับเอาต์พุต

2 กลุ่มของบิตควบคุมนี้ จะถูกเข้ารหัสเพื่อกำหนดการทำงานของเอาต์พุต หลังจากที่ทำการเปรียบเทียบสัญญาณแล้ว

1.2 รีจิสเตอร์ควบคุมตัวตั้งเวลา 2 (Timer control register 2 : TCTL2)เป็นรีจิสเตอร์ขนาด 8 บิต มีแอดเดรสอยู่ที่ $1021 ใช้ควบคุมขอบขาของสัญญาณที่ต้องการตรวจจับที่อินพุตของส่วนอินพุตแคปเจอร์ (ซึ่งจะกล่าวรายละเอียดส่วนนี้ต่อไป) ความหมายของบิตต่าง ๆ ดังนี้

7	6	5	4	3	2	1	0	
0	0	EDG1B	EDG1A	EDG2B	EDG2A	EDG3B	ESG3B	TCTL2

บิต 6 และ 7 : ไม่ใช้งานกำหนดให้เป็น "0"

บิต EDGxA และ EDGxB (Input capture x edge control) : ทั้ง 2 บิตนี้จะถูกเคลียร์เป็น "0" เมื่อเกิดการรีเซต และจะถูกเข้ารหัสเพื่อกำหนดการตรวจจับลอจิกที่อินพุต สำหรับการทำงานของส่วนอินพุตแคปเจอร์

1.3 รีจิสเตอร์อินเตอร์รัพต์มาส์คของตัวตั้งเวลา1(Timer interrup mask register 1 : TMSK 1) เป็นรีจิสเตอร์ขนาด 8 บิต มีแอดเดรสอยู่ที่ $1022 ใช้ควบคุมการอีนาเบิลการอินเตอร์รัพต์ของอินพุตแคปเจอร์ และส่วนเปรียบเทียบเอาต์พุต ความหมายของแต่ละบิตมีดังนี้

7	6	5	4	3	2	1	0	
OC1I	OC2I	CO3I	OC4I	OC5I	IC1I	IC2I	IC3I	TMSK1

OCxI (Output compare x interrupt) : บิตนี้จะเซตเมื่อแฟลก OCxF ถูกเซต เมื่อบิตนี้เซตแสดงว่ามีการร้องขอการอินเตอร์รัพต์ ณ ที่บิตของเอาต์พุตนั้น ๆ

ICxI (Input capture x interrupt) : บิตนี้จะเซตเมื่อแฟลก ICxF ถูกเซต เมื่อบิตนี้เซตแสดงว่ามีการร้องขอการอินเตอร์รัพต์ ณ ที่บิตของอินพุตนั้น ๆ

1.4 รีจิสเตอร์อินเตอร์รัพต์มาส์คของตัวตั้งเวลา2 (Timer interrup mask register 2 : TMSK2) รีจิสเตอร์ TMSK2 นี้ใช้ควบคุมลำดับของการร้องขอการอินเตอร์รัพต์ทางฮาร์ดแวร์ ที่มาจากสถานะของบิตในรีจิสเตอร์ TFLG2 และใน 2 บิตต่ำ (คือบิต 0 และ 1) ยังใช้เป็นบิตที่ทำหน้าที่กำหนดการหารความถี่ของสัญญาณนาฬิกา E หรือเรียกว่า ปรีสเกลเลอร์ด้วย ความหมายของแต่ละบิตมีดังนี้

7	6	5	4	3	2	1	0	
TOI	RTII	PAOVI	PAII	0	0	PR1	PR0	TMSK2

TOI (Timer overflow interrupt enable) : เมื่อบิตนี้เป็น"0"ส่วนอินเตอร์รัพต์ของ TOF จะถูกดิสเอเบิล ถ้าบิตนี้เป็น "1" จะเกิดการร้องขอการอินเตอร์รัพต์ ถ้าหากบิต TOF เป็น "1" ด้วย

RTII (RTI interrupt enable) : เมื่อบิตนี้เป็น "0" การอินเตอร์รัพต์ของ RTIF จะถูกดิสเอเบิล ถ้าหากเป็น "1" จะเกิดการร้องขอการอินเตอร์รัพต์ ถ้าหากบิต RTIF เป็น "1" ด้วย

PAOVI (Pulse accumulator overflow interrupt enable) : เมื่อบิตนี้เป็น "0" การอินเตอร์รัพต์ของ PAOVF จะถูกดิสเอเบิล แต่ถ้าหากเป็น "1" จะเกิดการร้องขอการอินเตอร์รัพต์ถ้าหากบิต PAOVF เป็น "1"

PAII (Pulse accumulator input interrupt inable) : ถ้าเป็น "0" การอินเตอร์รัพต์ของ PAIF จะถูกดิสเอเบิล แต่ถ้าเป็น "1" จะเกิดการร้องขอการอินเตอร์รัพต์เมื่อบิต PAIF ถูกเซต

บิต 3 และ 2 : ไม่ใช้งาน กำหนดให้เป็น "0"

PR1 PR0 (Timer prescaler selects) : ทั้ง 2 บิตนี้ใช้ในการกำหนดตัวหารสัญญาณนาฬิกาตามตารางที่ 1 สามารถที่จะอ่านข้อมูลหรือสถานะของบิตนี้ได้ตลอดเวลา แต่ในการเขียนข้อมูลมาที่ 2 บิตนี้จะถูกดิสเอเบิลไว้หลังจากเขียนข้อมูลในตัวแรกไปแล้ว หรือหลังจากที่ 68HC11 รีเซตไปแล้วนาน 64 ไซเกิลของสัญญาณนาฬิกา E แต่ถ้าหา 68HC11 ทำงานในโหมดทดสอบพิเศษจะสามารถเขียนข้อมูลที่ 2 บิตนี้ได้ตลอดเวลา

2. รีจิสเตอร์แสดงสถานะ

ประกอบด้วยรีจิสเตอร์ TELG1 และ TFLG2 ดังมีรายละเอียดดังต่อไปนี้

2.1 รีจิสเตอร์อินเตอร์รัพต์แฟลกของตัวตั้งเวลา 1 (Timer interrupt flag register 1 : TFLG1)รีจิสเตอร์ TFLG1 ใช้ในการแสดงเหตุการณ์ที่เกิดขึ้นในระบบตั้งเวลานี้ และทำงานร่วมกับรีจิสเตอร์ TMSK1 เพื่อก่อให้เกิดการอินเตอร์รัพต์ขึ้นในระบบ แต่ละบิตของรีจิสเตอร์ TFLG1 จะทำงานสัมพันธ์สอดคล้องกับบิตต่าง ๆ ในรีจิสเตอร์ TMSK1 ในตำแหน่งเดียวกัน ถ้าบิตในรีจิสเตอร์ TMSK1 เซตแล้วในช่วงเวลานั้นพบว่าแฟลกหรือบิตที่สัมพันธ์กัน ณ

ตำแหน่งเดียวกันนั้นเซตอยู่เหมือนกัน ก็จะทำให้เกิดการร้องขอการอินเตอร์รัพต์ขึ้นทันที ความหมายของแต่ละบิตในรีจิสเตอร์ TFLG1 มีดังนี้

7	6	5	4	3	2	1	0	
TOF	RTIF	PAOVF	PAIF	0	0	0	0	TFLG2

TOF (Timer overflow) : บิตนี้สามารถเคลียร์โดยการรีเซต บิตนี้จะเซตเมื่อตัวนับอิสระ 16 บิต ทำการนับครบรอบ แล้วเปลี่ยนค่าจาก $FFFF เป็น $0000 การเคลียร์บิตนี้ทำได้โดยการเขียนข้อมูล "1" มายังบิตนี้

RTIF (Real time interrupt flag) : บิตนี้จะเซตในทุกจังหวะของขอบขาขึ้นของสัญญาณ ณ จุดที่ต้องการตรวจจับ บิตนี้จะเคลียร์โดยการเขียนข้อมูล "1" มายังบิตนี้

PAOVF (Pulse accumulator overflow interrupt flag) : บิตนี้จะเซตเมื่อตัวนับในพัลส์แอกคิวมูเลเตอร์เปลี่ยนค่าจาก $FF เป็น $00

PAIF (Pulse accumulater input edge interrupt flag) : บิตนี้จะเซตเมื่อสามารถตรวจจับขอบขาสัญญาณที่ต้องการได้ที่ขาอินพุต PAI

บิต 3 - บิต 0 : ไม่ได้ใช้งาน กำหนดให้เป็น "0" ทั้งหมด

3. รีจิสเตอร์เก็บข้อมูลในระบบตั้งเวลา

ประกอบด้วยรีจิสเตอร์ขนาด 16 บิต 9 ตัว คือ รีจิสเตอร์ TCNT ซึ่งได้กล่าวในขั้นต้นไปแล้ว , TIC1-TIC3 (3 ตัว) และ TOC1- TOC5 (5ตัว) โดยรีจิสเตอร์ TCNT เก็บข้อมูลของการนับของตัวอิสระขนาด 16 บิต รีจิสเตอร์ TIC1-TIC3 มีขนาด 16 บิต จะทำหน้าที่เก็บข้อมูลของอินพุตแคปเจอร์ มีแอดเดรสอยู่ที่ $1010-$1015 (รีจิสเตอร์ 1 ตัว ใช้แอดเดรส 2 ตำแหน่ง) ส่วนรีจิสเตอร์ TOC1-TOC5 เก็บข้อมูลของส่วนเปรียบเทียบเอาต์พุตมีขนาด 16 บิต มีแอดเดรสอยู่ที่ $1016-$101F

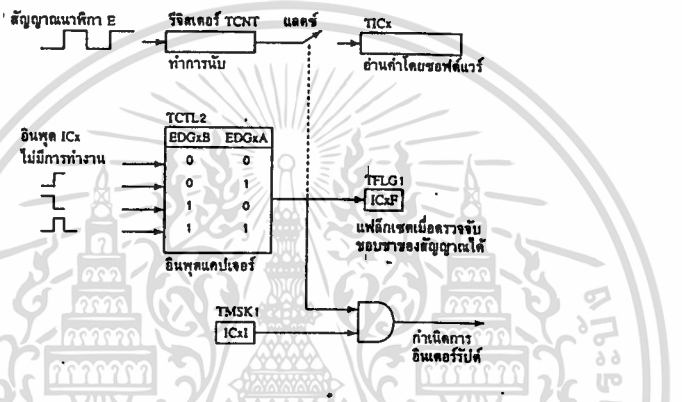
ลำดับการทำงาน

รูปที่ 2.5 แสดงขั้นตอนการทำงานของส่วนอินพุตแคปเจอร์ในช่องใด ๆ (Icx) การกำหนดเงื่อนไขที่ต้องการตรวจจับสามารถกำหนดให้ทำงานที่ขอบขาขึ้น,ขอบขาลงของสัญญาณหรือทั้งคู่ โดยกำหนดสถานะของบิตในรีจีสเตอร์ TCTL2 (บิต ECGxA และEDGxB) เมื่อระบบเริ่มทำงาน วงจรนับอิสระ 16 บิตจะเริ่มต้นนับค่าแล้วเก็บไวในรีจีสเตอร์ TCNT จนกว่าขาอินพุตแคปเจอร์ที่ต้องการได้ค่าของรีจีสเตอร์ TCNT จะถูกแลตช์เก็บไว้ในรีจีสเตอร์ TICx (ขึ้นกับว่าใช้อินพุตตัวใด) จากนั้นบิต ICxI ในรีจีสเตอร์ TFLGI และถ้าหากบิต ICxI รีจีสเตอร์ TMSKI เซตอยู่ด้วย ก็จะทำให้เกิดการอินเตอร์รัพต์

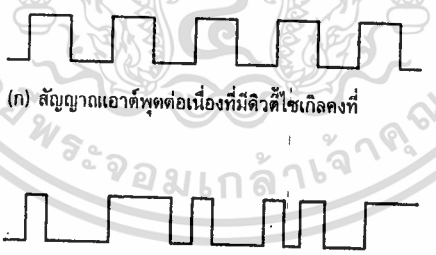
ส่วนเอาต์พุตเปรียบเทียบ

ในส่วนนี้สามารถกำเนิดสัญญาณรูปสี่เหลี่ยมที่มีค่าความถี่และดิวตี้ไซเกิลที่คงที่,รูปสัญญาณที่มีการเปลี่ยนแปลงของดิวตี้ไซเกิล และรูปสัญญาณลักษณะพัลส์แบบโมโนสเตเบิลก็ได้ ในรูปที่ 2.6 แสดงลักษณะรูปสัญญาณที่ส่วนเอาต์พุตเปรียบเทียบนี้กำเนิดออกมาได้

ด้วยการทำงานของส่วนเอาต์พุตเปรียบเทียบนี้ สามารถที่จะกำเนิดควบของการหน่วงเวลาได้ โดยไม่ขึ้นกับโปรแกรมลูปที่ใช้ในการหน่วงเวลาของไมโครคอนโทรลเลอร์ ยกตัวอย่าง เช่น มันสามารกำเนิดคาบเวลา 10 มิลิวินาที ที่ใช้ในการหน่วงเวลาสำหรับการเขียนและลบข้อมูลในอีพรอมกำเนิดพัลส์สำหรับการบันทึกข้อมูลลงในอีอีพรอมสำหรับเครื่องบันทึกข้อมูลอีพรอม เป็นต้น



รูปที่ 2.5 ลำดับการทำงานของส่วนอินพุต



รูปที่ 2.6 แสดงลักษณะรูปสัญญาณที่กำเนิดออกมาจากส่วนเอาต์พุตเปรียบเทียบ

ลำดับการทำงาน

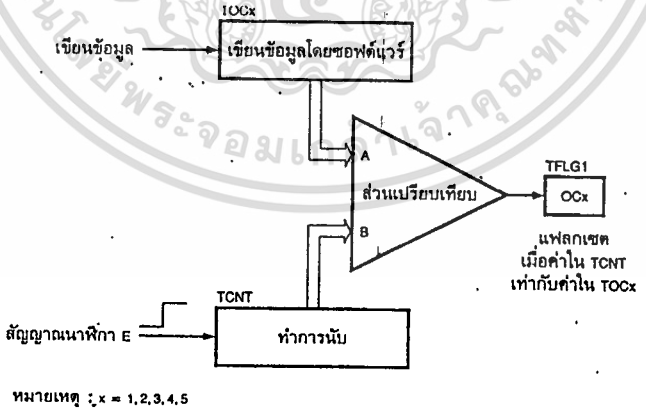
จากรูปที่ 2.7 แสดงการทำงานของส่วนเอาต์พุตเปรียบเทียบ เอาต์พุตเปรียบเทียบของ 68HC11 นี้มีอยู่ทั้งสิ้น 5 ช่อง สัญญาณ คือ ขา OC1-OC5 ข้อมูลที่ต้องการเปรียบเทียบใน

แต่ละเอาต์พุตจะถูกเก็บไว้ในรีจิสเตอร์ TOC1-TOC5 ซึ่งมีขนาด 16 บิต การทำงานจะเริ่มต้นโดยการเขียนข้อมูลที่ต้องการให้เปรียบเทียบหรือเรียกว่าข้อมูลอ้างอิงในรีจิสเตอร์ TOCx ด้วยคำสั่งการเขียนข้อมูล และเมื่อเริ่มต้นให้ระบบการทำงานตัวนับอิสระขนาด 16 บิต ก็จะเริ่มนับเก็บค่าไว้ในรีจิสเตอร์ TCNT ไว้ตลอดเวลา เมื่อค่าในรีจิสเตอร์ TCNT เท่ากับค่าในรีจิสเตอร์ TOCx ไมโครคอนโทรลเลอร์จะทำการเซตบิต OCxF ในรีจิสเตอร์ TFLG1 ภาวะนี้ เรียกว่า *การเปรียบเทียบเท่าสมบูรณ์ (successful compare)* นั่นคือการเปรียบเทียบสมบูรณ์จะเกิดขึ้นในทุก ๆ ค่าที่ข้อมูลในรีจิสเตอร์เอาต์พุตเปรียบเทียบและรีจิสเตอร์เก็บค่าตัวนับมีค่าเท่ากัน

ส่วนเปรียบเทียบข้อมูลจะให้เอาต์พุตเป็น "0" ถ้าหากข้อมูลที่อินพุต A ซึ่งได้มาจากรีจิสเตอร์ TOCx ยังไม่เท่ากับข้อมูลที่อินพุต B ซึ่งได้มาจากรีจิสเตอร์ TCNT เมื่อใดที่ค่าทั้งสองเท่ากันก็จะได้เอาต์พุตเป็น "1" นั่นคือบิต OCxF จะเซต

การเขียนข้อมูลไปยังรีจิสเตอร์ในส่วนเอาต์พุตเปรียบเทียบนี้ให้ใช้คำสั่งดับเบิ้ลไบต์ เช่น STD TOC5 เนื่องจากรีจิสเตอร์ในส่วนนี้เป็นรีจิสเตอร์ขนาด 16 บิตทั้งสิ้น ถ้าหากใช้คำสั่งเขียนทีละไบต์แยกกันอาจทำให้เกิดความผิดพลาดขึ้นได้ ในการเขียนข้อมูลนั้นจะเก็บค่าไบต์สูงก่อนแล้วจึงตามด้วยไบต์ต่ำในไซเกิลถัดมา ส่วนเปรียบเทียบข้อมูลจะเริ่มทำงานหลังจากนั้น 1 ไซเกิล ทั้งนี้เพื่อให้การเก็บข้อมูลลงในรีจิสเตอร์TOCx สมบูรณ์เสียก่อนจะได้ไม่เกิดความผิดพลาดในการทำงาน

ลักษณะของสัญญาณเอาต์พุตจะถูกควบคุมโดยบิต OMx และ OIx ในรีจิสเตอร์ TCTL1 เมื่อบิต OCxF เซตเป็นการบ่งบอกให้ส่งสัญญาณเอาต์พุตออกไปทางขา Ocx ดังแสดงในรูปที่ 2.4 การอินเตอร์รัพต์จะเกิดขึ้นหากบิต OCxL ในรีจิสเตอร์ TMSK1 ถูกเซตอยู่



รูปที่ 2.7 แสดงการทำงานของส่วนเอาต์พุตเปรียบเทียบ

2.4 รีลไทม์อินเตอร์รัพต์

เป็นระบบการอินเตอร์รัพต์ที่มีช่วงเวลาการเกิดคงที่แน่นอน แล้วแต่ว่าจะต้องการให้เกิดการอินเตอร์รัพต์ในช่วงเวลาใด สามารถที่จะกำหนดอัตราเวลาที่ให้เกิดการอินเตอร์รัพต์โดย

กำหนดสถานะของบิต RTR0 และ RTR1 ในรีจิสเตอร์ PACTL จากการกำหนดตัวหารให้แก่สัญญาณนาฬิกา E ซึ่งสรุปความสัมพันธ์ได้ดังตารางที่ 2

ตารางที่ 2 สรุปความสัมพันธ์ของบิต RTR0 และ RTR1 เพื่อกำหนดตัวหารให้แก่สัญญาณนาฬิกาE

บิตใน PACTL		หารด้วย	คาบเวลาที่ได้เมื่อความถี่ของสัญญาณนาฬิกา = 2 เมกะเฮิรตซ์
RTR1	RTR0		
0	0	2^{13}	4.10 มิลลิวินาที
0	1	2^{14}	8.19 มิลลิวินาที
1	0	2^{15}	16.38 มิลลิวินาที
1	1	2^{16}	32.77 มิลลิวินาที

จากตารางที่ 2 ถ้ากำหนดให้บิต RTR0 และ RTR1 เป็น "0" ทั้งคู่ ก็จะเกิดการอินเตอร์รัพต์ในทุก 4.10 มิลลิวินาที แต่อย่างไรก็ตามจะเกิดการอินเตอร์รัพต์ในระบบได้หรือไม่ยังขึ้นกับการกำหนดสถานะที่บิต RTIF ของรีจิส TFLG2 และบิต RTII ของรีจิสเตอร์ TMSK2 ถ้าหากทั้งสองบิตเป็น "1" ทั้งคู่ ก็จะเป็นการอีนาเบิลการอินเตอร์รัพต์ตัวอย่างการนำรีลไทม์ไปใช้งานก็คือ ทำเป็นนาฬิกาบอกเวลา เป็นต้น

2.5 พัลส์แอกคิวมูเลเตอร์

พัลส์แอกคิวมูเลเตอร์ คือตัวนับที่สามารถอ่านและเขียนข้อมูลได้ขนาด 8 บิต สามารถทำงานได้ 2 โหมด คือ โหมดอีเวนต์เคาน์ติ้ง (event counting mode) และโหมดเกตไทม์แอกคิวมูเลชั่น (gate time accumulation) โดยการกำหนดที่บิต PAMOD ในรีจิสเตอร์ควบคุมพัลส์แอกคิวมูเลเตอร์ (Pulse accumulator control register : PACTL)ในโหมดอีเวนต์เคาน์ติ้ง ตัวนับนี้จะเพิ่มค่าเมื่อได้รับสัญญาณนาฬิกาจากภายนอก อัตราสัญญาณนาฬิกาสูงสุดที่ป้อนให้แก่ตัวนับได้มีค่าเท่ากับครึ่งหนึ่งของความถี่ของสัญญาณ E ส่วนในโหมดเกตไทม์แอกคิวมูเลเตอร์จะใช้สัญญาณออสซิลเลเตอร์ความถี่เท่ากับความถี่ของสัญญาณนาฬิกา E หารด้วย 64 (E/64) เป็นตัวขับให้ตัวนับทำงาน

พัลส์แอกคิวมูเลเตอร์ใช้บิต 7 ของ พอร์ต A เป็นอินพุตของมันเรียกว่าขา PAI (pulse accumulator input) แต่อย่างไรก็ตามขานี้ยังต้องใช้ร่วมกับส่วนเอาต์พุตเปรียบเทียบ โดยเป็นขาเอาต์พุต (OC) และใช้เป็นขาของพอร์ตอินพุตเอาต์พุต (PA7) ด้วย โดยปกติขานี้จะถูกกำหนดให้เป็นขาอินพุตเมื่อถูกใช้งานสำหรับพัลส์แอกคิวมูเลเตอร์ และถึงแม้ว่าจะกำหนดให้ขานี้เป็นเอาต์พุตอยู่ แต่เมื่อมาใช้งานในพัลส์แอกคิวมูเลเตอร์ มันจะเปลี่ยนหน้าที่เป็นขาอินพุตโดยอัตโนมัติ

การทำงานของพัลส์แอกคิวมูเลเตอร์

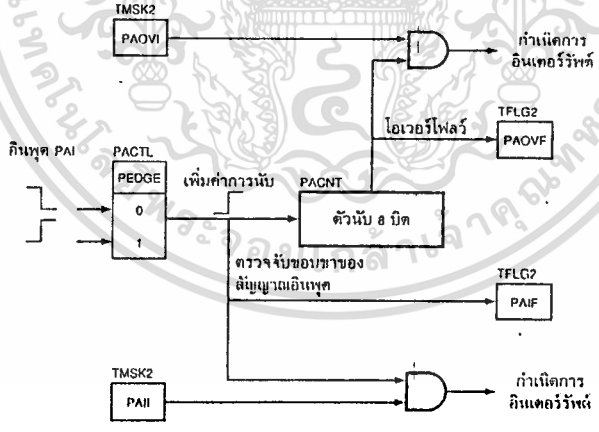
พัลส์แอกคิวมูเลเตอร์มีอินพุตเพียง 1 ช่อง PAI ซึ่งเป็นขา PA7 ของพอร์ต A ซึ่งกำหนดทิศทางข้อมูลของ PA7 (DDRA7) ในรีจิสเตอร์ PACTL ปกติแล้วเมื่อใช้งานสำหรับพัลส์แอกคิวมูเลเตอร์บิตนี้จะเป็น "0"

ในรีจิสเตอร์ควบคุม PACTL จะมีบิตควบคุมการอินาเบิลของระบบพัลส์แอกคิวมูเลเตอร์ คือ บิต PAEN ซึ่งจะต้องเซตบิตนี้ก่อนที่จะให้ระบบทำงานสำหรับการเลือกโหมดการทำงานจะกำหนดได้ที่บิต PAMOD ซึ่งในพัลส์แอกคิวมูเลเตอร์แบ่งการทำงานได้ 2 โหมด คือ

- อีเวนต์เคาน์ติ้ง (event counting)
- เกตไทม์แอกคิวมูเลชั่น (gated time accumulation)

โหมดอีเวนต์เคาน์ติ้ง

ในรูปที่ 2.8 เป็นไดอะแกรมการทำงานในโหมดนี้ของพัลส์แอกคิวมูเลเตอร์เริ่มต้นต้องทำการเซตบิต PAEN ในรีจิสเตอร์ PACTL ก่อนเพื่ออินาเบิลระบบจากนั้นเคลียร์บิต PAMOD เพื่อกำหนดให้ทำงานในโหมดอีเวนต์เคาน์ติ้ง ตามด้วยเลือกขอบของสัญญาณอินพุตที่ต้องการให้เริ่มทำการนับ โดยกำหนดที่บิต PEDGE ในตัวอย่างที่ยกมาอธิบายนี้เลือกขอบขาลงของสัญญาณ ดังนั้นบิต PEDGE จึงเป็น "0"



หมายเหตุ : กำหนดบิต PAEN ให้เป็น "1" และบิต PAMOD ให้เป็น "0" เพื่อให้ทำงานในโหมดอีเวนต์เคาน์ติ้ง

รูปที่ 2.8 ไดอะแกรมการทำงานของพัลซ์แอกคิวมูเลเตอร์ในโหมดอีเวนต์เคาต์ติ้ง

ทุกค่าที่สามารถตรวจจับขอบขาลงของสัญญาณที่ขา PAI ได้ ค่าในรีจิสเตอร์ PACNT ก็จะเพิ่มขึ้น บิต PAIF เซตเพื่อเตรียมพร้อมที่จะให้เกิดการอินเตอร์รัพต์ได้ ถ้าหากบิต PAII เซตอยู่ด้วย ถ้าหากต้องการเคลียร์บิต PAIF ต้องเขียนข้อมูล "1" เหมือนกับแฟลกของตัวตั้งเวลา

เมื่อพัลส์แอกคิวมูเลเตอร์นับจนกระทั่งตั้งค่าในรีจิสเตอร์ PACNT เกินคือ เปลี่ยนจาก $FF เป็น $00 บิต PAOVF จะเซต ซึ่งเกิดจากการอินเตอร์รัพต์ได้ ถ้าหากบิต PAOVI เซตอยู่ด้วย ถ้าต้องการเคลียร์บิต PAOVF ต้องเขียนข้อมูล "1" มายังบิตนี้เช่นเดียวกับบิต PAIF

การทำงานในโหมดอีเวนต์เคาน์ติ้งยังแบ่งลักษณะของการนับได้อีก 2 ลักษณะ คือ

- นับค่าช่วงสั้น (short counts) และ
- นับค่าช่วงยาว (long counts)

พัลส์คิวมูเลเตอร์สามารถที่จะกำหนดช่วงจังหวะการกำเนิดสัญญาณที่จะกำหนดช่วงจังหวะการกำเนิดสัญญาณอินเตอร์รัพต์ได้ โดยกำหนดค่าตัวเลขค่าหนึ่งเพื่อนับไปถึงค่าที่กำหนดนั้นก็ให้เกิดการอินเตอร์รัพต์จะเป็นไปอย่างรวดเร็ว ถ้าค่าของตัวเลขนั้น ๆ น้อยกว่า 256 วิธีการก็คือ เขียนค่า 2's คอมพลีเมนต์ของค่าตัวเลขที่กำหนดนั้นไปเก็บไว้ในรีจิสเตอร์ PACNT และเมื่อตรวจจับ ขอบขาของสัญญาณที่อินพุตได้ พัลส์แอกคิวมูเลเตอร์ก็จะนับต่อไปจากค่าที่เขียนนั้น จนกระทั่งถึงค่า $FF แล้วเพิ่มเป็นค่า $00 ก็จะเกิดโอเวอร์โฟลว์ ทำให้เกิดการอินเตอร์รัพต์ถ้าหากส่วนอินเตอร์รัพต์ได้รับการอีนาเบิลไว้ ยกตัวอย่างถ้าค่าที่ต้องการตั้งเป็น 20 นั่นคือ ค่า$14 ค่า 2's คอมพลีเมนต์จะเป็น $EE ดังนั้นค่าของการนับในระบบพัลส์แอกคิวมูเลเตอร์จะเริ่มต้นที่ $EE เมื่อตรวจจับสัญญาณได้เริ่มนับก็จะเป็นค่า$EF ไปจนถึง $FF แล้วเปลี่ยนเป็น $00 ก็จะเกิดโอเวอร์โฟลว์ทั้งหมดที่กล่าวมาคือ ลักษณะการนับแบบช่วงสั้น

สำหรับการนับช่วงยาวถ้าหากการนับมีค่ามากกว่า 256 จะต้องมีการเก็บค่าของการนับเมื่อครบทุกรอบ หรือทุกครั้งที่เกิดโอเวอร์โฟลว์เพื่อให้ค่าของการนับต่อเนื่อง ถ้าหากตัวนับเริ่มต้นที่ค่าศูนย์ เมื่อนับไปถึง 256 ก็จะเกิดโอเวอร์โฟลว์ ดังนั้นตัวเลขของการเกิดโอเวอร์โฟลว์จึงเท่ากับค่าที่ตั้งไว้หารด้วย 256 เมื่อต้องนับค่ามากกว่า 256 จึงต้องใช้แอกคิวมูเลเตอร์ D ซึ่งมีขนาด 16 บิตมาช่วย เป็นที่ทราบอยู่แล้วว่าแอกคิวมูเลเตอร์ D เกิดจากการรวมกันของแอกคิวมูเลเตอร์ A และ B ในการนับช่วงยาวนี้ แอกคิวมูเลเตอร์ A จะเก็บตัวเลขการเกิดโอเวอร์โฟลว์ถ้าต้องการทราบเป็นจำนวนของการนับต้องคูณด้วย 256 ส่วนแอกคิวมูเลเตอร์ B จะเก็บจำนวนที่เหลือ จากนั้นจะทำการ 2's คอมพลีเมนต์กับจำนวนในแอกคิวมูเลเตอร์ B แล้วเริ่มทำการนับ ยกตัวอย่างถ้าค่าที่ต้องการเป็น 800 ซึ่งก็คือ $320 ค่า $3 ในไบต์สูงจะถูกเก็บไว้ในแอกคิวมูเลเตอร์ A ส่วน $20 ในไบต์ต่ำถูกทำเป็นเลข 2's คอมพลีเมนต์ เป็นค่า $E0 จากนั้นก็ให้ระบบเริ่มทำการนับหลังจากตรวจจับสัญญาณที่อินพุตได้ ซึ่งจะนับได้ 32 ครั้งก่อนที่จะเกิดโอเวอร์โฟลว์แล้วก็จะนับต่อซึ่งก็จะเกิดการโอเวอร์โฟลว์อีก 3 ครั้ง ได้ค่าการนับเป็น 768 (มาจาก 256 x 3) ครั้ง รวมกับ 32 ครั้งแรกก็จะได้ เท่ากับ 800 พอดี ตัวอย่างการนำลักษณะการนับช่วงยาวนี้ได้แก่ นำไปพัฒนาโปรแกรมเพื่อตรวจสอบหน่วยความจำแรมที่มีประมาณมาก ๆ เป็นต้น

รีจิสเตอร์ควบคุมพัลส์แอกคิวมูเลเตอร์ (Pulse accumulator register : PACTL)

เป็นรีจิสเตอร์ขนาด 8 บิต อยู่ที่แอดเดรส $1026 โดย 4 บิตแรกจะถูกใช้ควบคุมระบบพัลส์ แอกคิวมูเลเตอร์ ส่วนอีก 2 บิต ล่างถูกใช้กำหนดอัตราของระบบรีลไทม์อินเตอร์รัพต์ ความหมายและ การทำงานของแต่ละบิตมีดังนี้

7	6	5	4	3	2	1	0	
DDRA7	PAEN	PAMOD	PEDGE	0	0	RTR1	RTRO	PACTL

DDRA7 (Data direction for port A pin 7) : ใช้กำหนดทิศทางของข้อมูลสำหรับบิต 7 ของพอร์ต A โดยถ้าเป็น "0" จะเป็นอินพุต ถ้าเป็น "1" จะเป็นเอาต์พุต

PAEN (Pulse accumulator system inable) : ใช้ในการอีนาเบิลระบบพัลส์แอกคิวมูลเลเตอร์ โดยถ้าเป็น "1" จะเป็นการอีนาเบิล

PAMOD (Pulse accumulator mode) : ใช้กำหนดโหมดการทำงานของพัลส์แอกคิวมูเลเตอร์ โดยถ้าเป็น "0" จะเป็นโหมดอีเวนต์เคาน์ติ้ง ถ้าเป็น "1" เป็นโหมดเกตไทม์แอกคิวมูเลเตอร์

PEDGE (Pulse accmulator edge control) : จะใช้งานร่วมกับบิต PAMOD เพื่อกำหนดลักษณะการทำงานดังตารางข้างล่าง

RTR1 และ RTRO (RTI Interrupt rate selects) : ใช้กำหนดอัตราของคาบเวลาของรีลไทม์อินเตอร์รัพต์ ดังในตารางที่ 2 ถ้าหากมีการรีเซตเกิดขึ้นบิตนี้จะเคลียร์ทันที

รีจิสเตอร์เก็บค่าการนับของพัลส์แอกคิวมูเลเตอร์ (Pulse accumualtor count register : PACNT)

เป็นรีจิสเตอร์ขนาด 8 บิต อยู่ที่แอดเดรส $1027 ใช้สำหรับเก็บค่าของการนับของพัลส์แอกคิวมูเลเตอร์ มีลักษณะเช่นเดียวกับรีจิสเตอร์ TCNT แต่มีจำนวนบิตน้อยกว่าคือ 8 บิต รีจิสเตอร์ตัวนี้จะถูกเขียนและเพิ่มค่าเมื่อพัลส์แอกคิวมูเลเตอร์เริ่มทำงาน

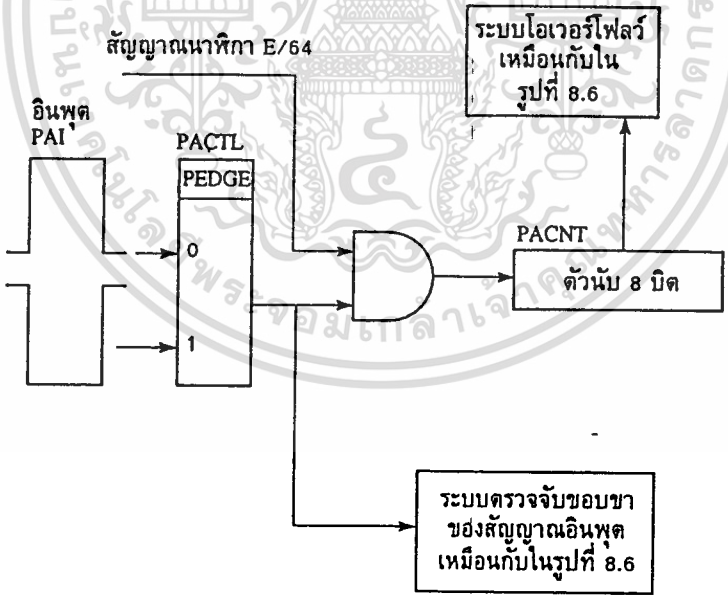
PAMOD	PEDGE	การทำงาน
0	0	ตัวนับทำงานที่ขอบขาขึ้นของสัญญาณที่ขา PA1 (โหมดอีเวนต์เคาน์ติ้ง)
0	1	ตัวนับทำงานที่ขอบขาขึ้นของสัญญาณที่ขา PA1 (โหมดอีเวนต์เคาน์ติ้ง)
1	0	ถ้าที่ขา PA1 เป็น "0" จะไม่ทำการนับ (โหมดเกตไทม์)
1	1	ถ้าที่ขาPA1 เป็น"1" จะไม่ทำการนับ(โหมดเกตไทม์)

โหมดเกตไทม์แอกคิวมูเลชั่น

ในโหมดนี้พัลส์แอกคิวมูเลเตอร์จะเพิ่มค่าของการนับทุก 64 ไซเกิลของสัญญาณนาฬิกา เมื่อตรวจจับระดับลอจิกได้ที่ขา PAI ในโหมดนี้จะไม่สนใจการเปลี่ยนแปลงของขอบขาสัญญาณ แต่จะดูที่ระดับลอกจิกแทน ในรูปที่ 2.9 เป็นไดอะแกรมการทำงานของโหมดนี้

เริ่มต้นด้วยการเซตบิต PAMOD เพื่อกำหนดให้พัลส์แอกคิวมูเลเตอร์ทำงานในโหมดเกตไทม์แอกคิวมูเลชั่น จากนั้นกำหนดระดับลอจิกที่ต้องการบิต PEDGE ถ้าเป็น "0" จะเริ่มทำการนับที่ลอจิก "1" ถ้าบิตนี้เป็น "1" การนับก็จะเริ่มขึ้นเมื่อลอจิกที่อินพุตเป็น "0" การนับจะเริ่มต้นเพิ่มค่าขึ้น เมื่อระดับลอจิกที่อินพุตนั้นมีความยาวของเวลา 64 ไซเกิลของสัญญาณนาฬิกา E และถ้าหากระดับลอจิกที่อินพุตเปลี่ยนไป ค่าของการนับเดิมยังคงเก็บไว้อยู่ และจะนับต่อเมื่อระดับลอจิกที่อินพุตกลับมาเป็นระดับที่ต้องการเช่นเดิม

การทำงานแฟลกและอินเตอร์รัพต์จะมีลักษณะคล้ายกับการทำงานในโหมดอีเวนต์เคาน์ติ้ง กล่าวคือ บิต PAIF จะเซตทุก ๆ ครั้งที่ระดับลอจิกอินพุต PAI เปลี่ยนมาเป็นระดับลอจิกที่ต้องการ และเมื่อบิต PAIF เซตแล้ว บิต PAII เซตอยู่ด้วยก็จะเกิดการอินเตอร์รัพต์ขึ้น ตัวอย่างการนำพัลส์แอกคิวมูเลเตอร์ในโหมดนี้ไปใช้งานก็ได้แก่การวัดความกว้างของพัลส์ทั้งแบบต่อเนื่องและแบบเป็นช่วง ๆ หรือ จะนำไปทำเป็นเครื่องตั้งเวลาควบคุมการ เปิด-ปิดอัตโนมัติก็ได้



หมายเหตุ : กำหนดบิต PAEN และบิต PAMOD ให้เป็น "1"
เพื่อให้ทำงานในโหมดเกตไทม์แอกคิวมูเลชัน

รูปที่ 2.9 ไดอะแกรมการทำงานของพัลซ์แอกคิวมูลเลเตอร์ในโหมดเกตไทม์แอกคิวมูเลเตอร์

บทที่ 3
การเขียนโปรแกรมเบื้องต้น

3.1 เขียนโปรแกรมเพื่ออ้างแอดเดรส

68HC11 มีการอ้างแอดเดรส (addressing) 6 แบบ คือ

- แบบทันทีทันใด (immediate)
- แบบโดยตรง (driect)
- แบบขยาย (extended)
- แบบอินเด็กซ์ (index)
- แบบอินเฮียเรนต์ (inherent)
- แบบสัมพัทธ์ (relative)

การเขียนโปรแกรมเพื่ออ้างแอดเดรสในแต่ละแบบก็จะแตกต่างกันไป

โปรแกรมสำหรับการอ้างแอดเดรสแบบทันทีทันใด

รูปแบบการเขียนโปรแกรมเป็นดังนี้

คำสั่ง #$ค่าที่ต้องการ

$ คือ ตัวอักษรที่ใช้บ่งบอกว่า ค่าที่ตามหลังนี้จะเป็นเลขฐานสิบหก

ตัวอย่างของโปรแกรมมีดังนี้

LDAA #$5C

หมายถึง นำค่า $5C ไปเก็บไว้ในแอกคิวมูเลเตอร์ A

LDX #$3B27

เป็นการนำค่า $3B27 ไปเก็บไว้ในรีจิสเตอร์อินเด็กซ์ X (IX) โดย $3B จะถูกเก็บไว้ ไบต์สูงของ IX ส่วน $27 จะเก็บไว้ในไบต์ต่ำของ IX

โปรแกรมสำหรับการอ้างแอดเดรสแบบโดยตรง

รูปแบบการเขียนโปรแกรมเป็นดังนี้

คำสั่ง $ ค่าแอดเดรสของแรมที่อยู่ภายในชิพ (มีค่าได้ตั้งแต่ $00-$FF)

ตัวอย่างของโปรแกรมมีดังนี้

LDAA $1B

เป็นการนำค่าที่อยู่ในแรมแอดเดรสที่ $1B มาเก็บไว้ในแอกคิวมูเลเตอร์ A ไม่ใช่นำค่า $1B มาเก็บไว้ในแอกคิวมูเลเตอร์ A

โปรแกรมสำหรับการอ้างแอดเดรสแบบขยาย

รูปแบบการเขียนโปรแกรมเป็นดังนี้

คำสั่ง $ ค่าแอดเดรสแรมที่อยู่ภายนอกชิพ

ตัวอย่างของโปรแกรมมีดังนี้

DAB $2000

เป็นการนำข้อมูลที่อยู่ในหน่วยความจำ แอดเดรส $2000 มาเก็บไว้ในแอกคิวมูเลเตอร์ B

โปรแกรมสำหรับการอ้างแอดเดรสแบบอินเด็กซ์

รูปแบบการเขียนโปรแกรมเป็นดังนี้

คำสั่ง $ ค่าออฟเซต, X (หรือ Y)

ตัวอย่างโปรแกรมมีดังนี้

STAA $05, X

สมมติว่า IX มีแอดเดรสก่อนหน้าที่จะเอ็กซิคิวต์คำสั่งนี้อยู่ที่ $2000 เมื่อเอ็กซิควต์คำสั่งในโปรแกรมนี้ จะเป็นการนำค่าที่อยู่ในแอกคิวมูเลเตอร์ A ไปเก็บลงในหน่วยความจำที่มีแอดเดรสเท่ากับ IX + 5 นั่นคือ $20000 + $5 = $2005

โปรแกรมสำหรับการอ้างแอดเดรสแบบอินเฮียเรนต์

เนื่องจากการอ้างแอดเดรสแบบนี้ จะไม่ยุ่งเกี่ยวกับหน่วยความจำแต่อย่างใด ดังนั้นทุก ๆ คำสั่งจะเป็นการกระทำกับรีจิสเตอร์เท่านั้น จึงเป็นการอ้างแอดเดรสแบบอินเฮียเรนต์ ทั้งสิ้น ดังตัวอย่างต่อไปนี้

CLRA : เป็นการเคลียร์ค่าในแอกคิวมูเลเตอร์ A

XGDY : สลับค่าในแอกคิวมูเลเตอร์ D กับรีจิสเตอร์ IY

DEX : ลดค่าของ IX

โปรแกรมสำหรับการอ้างแอดเดรสแบบสัมพัทธ์

รูปแบบการเขียนโปรแกรมเป็นดังนี้

คำสั่ง ค่าออฟเซตสัมพัทธ์ที่ต้องการ

เนื่องจากการอ้างแอดเดรสแบบนี้ มักใช้ในโปรแกรมที่มีการกระโดด ค่าออฟเซตสัมพัทธ์จึงเป็นค่าที่ใช้บอกให้ทราบว่า หลังจากเอ็กซิคิวต์คำสั่งนี้แล้ว ต้องกระโดดไปทำงานที่แอดเดรสใด ดังตัวอย่าง

```
$ 1000                    BEQ JMP
                          .
                          .
                          .
$ 1007          JMP       LDAA #$05
```

ที่บรรทัด $1000 นั่นคือโปรแกรมที่เขียนขึ้นโดยอาศัยการอ้างแอดเดรสแบบสัมพัทธ์ ค่าออฟเซตสัมพัทธ์ที่ต้องการทราบ คำนวณได้จาก *ค่าแอดเดรสใหม่ - ค่าแอดเดรส*

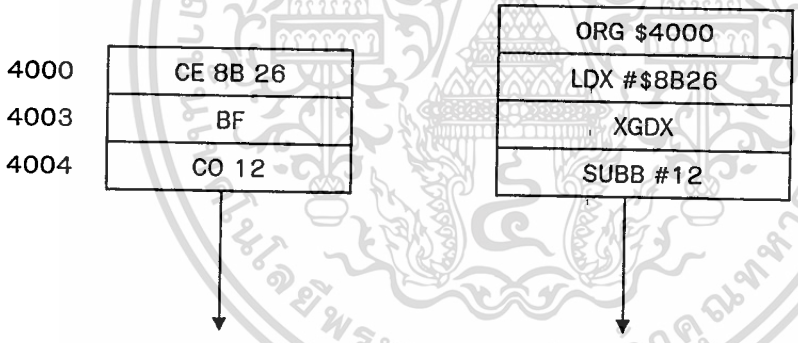
เก่า - *T* โดยที่ T จะเท่ากับ 2 สำหรับคำสั่งปกติ จะเป็น 4 เมื่อใช้คำสั่ง BRSET และ BRCLR สำหรับ IX และจะเท่ากับ 5 เมื่อใช้คำสั่ง BREST และ BRCLR สำหรับ IY จากตัวอย่างจะได้ค่าออฟเซตสัมพัทธ์ เท่ากับ $1007 - $1000 - $2 = $05

3.2 รูปแบบการเขียนโปรแกรมทั่วไป

ทุกครั้งที่เริ่มต้นเขียนโปรแกรมภาษาแอสเซมบลีของ 68HC11 จะต้องเริ่มต้นด้วย ORG แล้วตามด้วยแอดเดรสเริ่มต้นของโปรแกรมทุกครั้ง และเมื่อเขียนโปรแกรมจบแล้ว ต้องปิดท้ายด้วยคำสั่ง END เสมอ ดังตัวอย่าง

```
ORG    $1000
LDAA   #$5C
STAA   $1007
END
```

ในการเขียนโปรแกรมด้วยภาษาแอสเซมเบิลทุกครั้ง จะต้องนำโปรแกรมไปทำการคอมไพล์ให้เป็นออปโค้ดเสียก่อน เพื่อให้ซีพียูภายใน 68HC11 เข้าใจ และนี่คือตัวอย่างของโปรแกรมที่ผ่านการคอมไพล์แล้ว



ออปโค้ดที่ผ่านการคอมไพล์แล้ว โปรแกรมภาษาแอสเซมบลี

โปรแกรมที่ใช้ในการคอมไพล์เรียกว่า ครอสแอสเซมเบลอร์ (cross assembler) ซึ่งนอกจากจะใช้แปลง ภาษาแอสเซมบลียังสามารถแปลงโปรแกรมที่เป็นภาษาซีได้ด้วยทำให้การพัฒนา 68HC11 ทำได้ง่ายและทรงประสิทธิภาพมากขึ้น เพราะการเขียนโปรแกรมด้วยภาษาซีจะทำได้ง่ายกว่า และประหยัดเวลาได้มากกว่าการเขียนโปรแกรมด้วยภาษาแอสเซมบลี
ตัวอย่าง

```
time ( )
/*  function time ( ) increments sec,min,hour
based onthe 24-hour clock format */
{
sec+1 = 1
```

```
if (sec< = 60)
        {sec = 0;
        min+ = 1;
        if (min> = 60)
                {min = 0;
                hour+ = 1;
                if (hour> = 24)
                {hour = 0}
                }
            }
        }
```

นั่นคือโปรแกรมนาฬิกา จะเห็นว่ามีเพียงไม่กี่บรรทัดเท่านั้น แต่ถ้าเป็นการเขียนด้วยภาษาแอสเซมบลี จะยาวกว่านี้มากทีเดียว

3.3 โปรแกรมคณิตศาสตร์

โปรแกรมบวกลบเลข

ตัวอย่างของโปรแกรมทางคณิตศาสตร์อย่างง่ายๆที่ยกมาเป็นตัวอย่างนี้เป็นโปรแกรมบวกลบเลขฐาน โดยตัวตั้งมีค่าเท่ากับ $05 เก็บไว้ในหน่วยความจำแอดเดรส $3000 ตัวบวกมีค่าเท่ากับ $03 อยู่ในแอดเดรส $3001 ตัวลบมีค่า $02 อยู่ในแอดเดรส $3002 ผลลัพธ์จากการคำนวณให้มาเก็บไว้ที่ $3003

```
ORG   $2000
LDAA  $3000; A =$05
ADCA  $3001; A = $05 + $03 = $08
SBCA  $3002; A = $08 - $02 = $06
STAA  $3003; A → $3003
END
```

โปรแกรมคูณเลข

ต้องคูณเลขฐานสิบหกระหว่าง $20 กับ $35

```
ORG $2000
LDD #$2035          ; $20 → A , $35 → B
MUL                 ; A * B = $06A0 → D
ADCA #$00           ; adjust to 8 bit
```

```
END
```

โปรแกรมหารเลข

ต้องการหาร 6 ด้วย 4 ซึ่งคำตอบก็จะได้ 1 เศษ 2 สามารถเขียนโปรแกรมได้ดังนี้

```
ORG   .$2000
LDD   #$0006
LDX   #$0004
IDIV
STX   $00
LDX   #$0004
FDIV
END
```

หลังจากกระทำคำสั่ง IDIV จะได้ผลหารออกมาเป็น 1 เก็บไว้ในรีจิสเตอร์ IX ส่วนเศษ 2 จะถูกเก็บไว้ในรีจิสเตอร์ จากนั้นจะทำการหารเศษ 2 ด้วย 4 จึงต้องเก็บค่าของผลลัพธ์ในช่วงแรกไว้ใน IX จากนั้นก็จะมาทำงานด้วยคำสั่ง FDIV

หลังจากที่เอ็กซิคิวต์คำสั่ง FDIV จะได้ผลหารเท่ากับ 0.5 ถ้าแปลงเป็นฐานสิบหก จะได้ค่า $8000 เก็บในรีจิสเตอร์ IX เหตุผลที่เป็นค่า $8000 เนื่องจากเป็นค่าครึ่งหนึ่งจาก $0000 ถึง $FFFF พอดี ดังนั้นผลหารจึงมีค่าเท่ากับ $1.8 หรือ 1.5 ในเลขฐานสิบ

บทที่ 4
การเชื่อมต่อ 68HC11 กับอุปกรณ์ภายนอก

4.1 การเชื่อมต่อกับหน่วยความจำ

การต่อหน่วยความจำให้แก่ 68HC11จะใช้ขาสัญญาณพอร์ต B และ C โดยพอร์ต B เป็นบัสแอดเดรสไบต์สูงส่วนพอร์ต C เป็นบัสแอดเดรสไบต์ต่ำ และบัสข้อมูลซึ่งจะสลับกันทำงานโดยการมัลติเพล็กซ์

ในการเชื่อมต่อกับหน่วยความจำต้องมีวงจรที่เรียกว่าวงจรถอดรหัสแอดเดรสหน่วยความจำประกอบด้วย เพื่อให้ 68HC11สามารถเขียนอ่านข้อมูลกับหน่วยความจำได้อย่างถูกต้องและยังช่วยในการขยายหน่วยความจำให้ระบบอีกด้วย

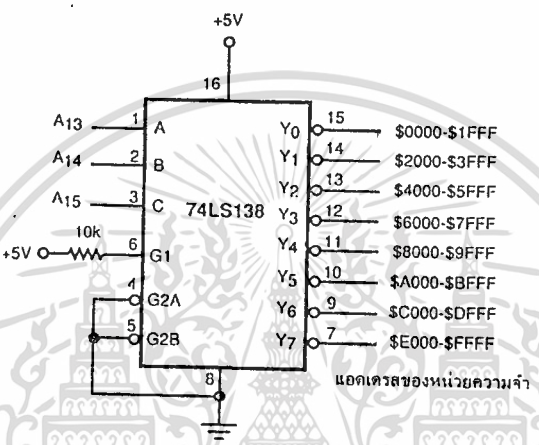
วงจรถอดรหัสแอดเดรสนี้มักใช้ไอซีเบอร์ 74LS138 ซึ่งเป็นไอซีถอดรหัส 3 ออก 8 การใช้ 74LS138 เป็นวงจรถอดรหัสแอดเดรสแสดงดังในรูปที่ 4.1 จากรูปจะใช้ขาแอดเดรส A_{13}- A_{15} ป้อนเข้าที่อินพุต A, B, C แล้วทำการกำหนดสถานะลอจิกที่ขาควบคุม G1, G2A และ G2B ให้ตัวไอซีสามารถทำงานได้ จากการต่อวงจรในลักษณะดังกล่าวทำให้ 68HC11 สามารถต่อกับหน่วยความจำได้ 8 ช่วงแอดเดรส ดังตารางที่ 1

ตารางที่ 1 แสดงการจัดแอดเดรสของหน่วยความจำและการทำงานของ 74LS138

A_{15}	A_{14}	A_{13}	เอาต์พุตของ 74LS138 ที่อีนาเบิล	ช่วงแอดเดรสของหน่วยความจำ
0	0	0	Y_0	$0000 - $1FFF
0	0	1	Y_1	$2000 - $3FFF
0	1	0	Y_2	$4000 - $5FFF
0	1	1	Y_3	$6000 - $7FFF
1	0	0	Y_4	$8000 - $9FFF
1	0	1	Y_5	$A000 - $BFFF
1	1	0	Y_6	$C000 - $DFFF
1	1	1	Y_7	$E000 - $FFFF

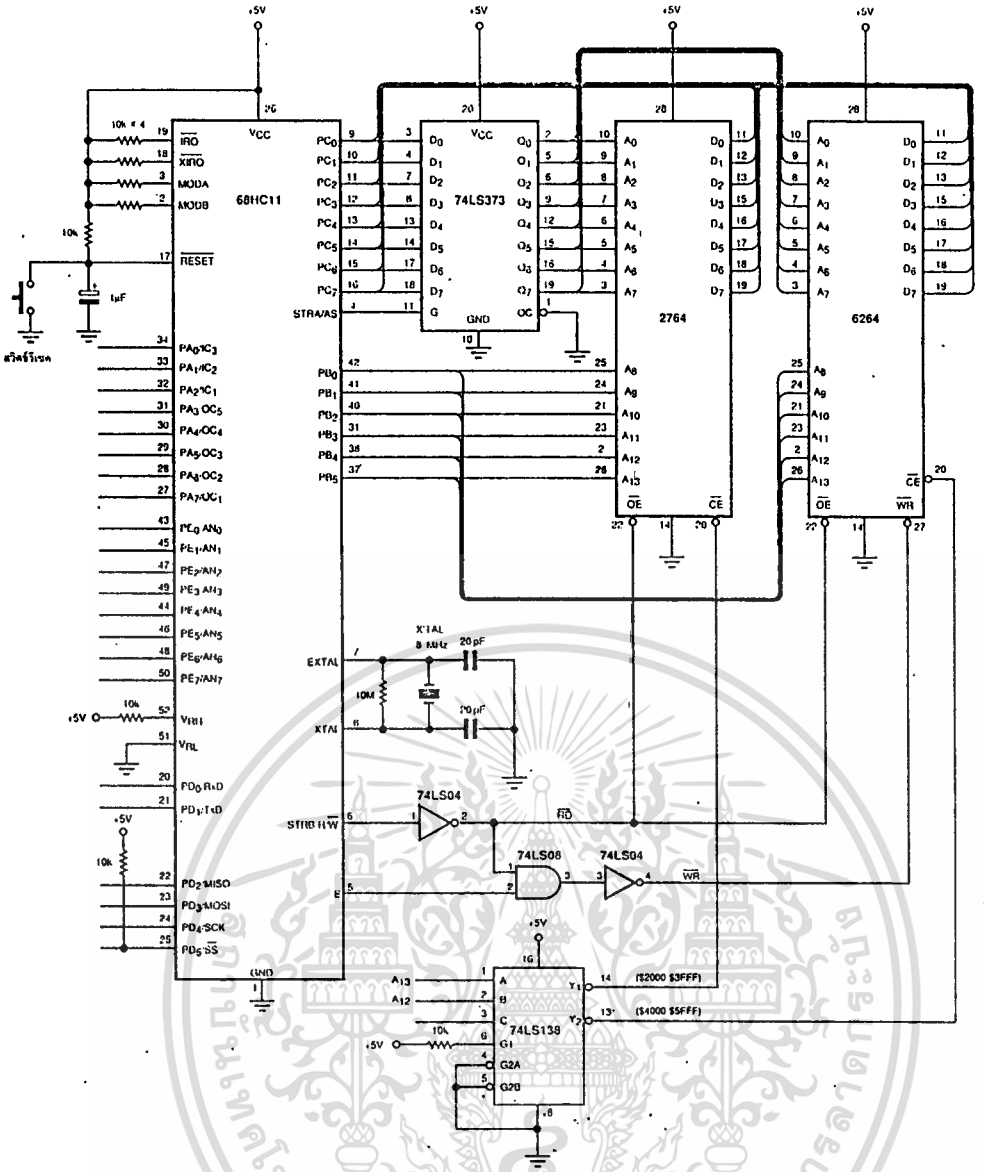
หน่วยความจำในแต่ละช่วงแอดเดรสจะมีขนาด 8 กิโลไบต์ ดังนั้นถ้าพิจารณาจากรูปที่ 4.1 68HC11 สามารถติดต่อกับหน่วยความจำขนาด 8 กิโลไบต์ได้ถึง 8 ตัว แต่ในความเป็นจริงแล้ว บางช่วงแอดเดรสจะถูกสงวนไว้เพื่อเป็นพื้นที่ของหน่วยความจำภายในชิพ 68HC11 เอง เช่น แอดเดรส $0000 - $00FF เป็นช่วงแอดเดรสของแรมภายในชิพ , แอดเดรส

$1000 – $103F เป็นช่วงแอดเดรสของรีจิสเตอร์ควบคุม, ช่วงแอดเดรส $B000 - $B7FF เป็นช่วงแอดเดรสของอีอีพรอมภายในชิพ และถ้าเป็นชิพรุ่น 68HC11A8 ซึ่งมีรอมภายในแอดเดรส$E000 - $FFFF จะถูกจัดสรรไว้เป็นพื้นที่ของรอมภายใน ดังนั้นการเลือกช่วงแอดเดรสเพื่อเชื่อมต่อกับหน่วยความจำภายนอก พึงระวังตรงจุดนี้ด้วย



รูปที่ 4.1 วงจรถอดรหัสแอดเดรสของหน่วยความจำ โดยใช้ 74LS138

อย่างไรก็ตาม ช่วงแอดเดรสของแรมและรีจิสเตอร์สามารถที่จะโยกย้ายแอดเดรส (ตามข้อจำกัดคือ ภายใน 4 กิโลไบต์) ถ้าหากมีความต้องการที่จะใช้งานหน่วยความจำภายนอกที่แอดเดรสของแรม และรีจิสเตอร์นั้น



รูปที่ 4.2 วงจรการเชื่อมต่อ 68HC11 เข้ากับหน่วยความจำอีพรอมและแรม

ในรูปที่ 4.2 เป็นวงจรสมบูรณ์ของการเชื่อมต่อ 68HC11 เข้ากับหน่วยความจำอีพรอมและแรมภายนอกขนาด 8 กิโลไบต์ โดยกำหนดให้แอดเดรสของอีพรอมภายนอกเบอร์ 2764 อยู่ที่ $2000 - $3FFF และแรมเบอร์ 3234 อยู่ที่ $4000 - $5FFF ทั้งนี้ก็เพื่อหลบเลี่ยงไม่ใช้พื้นที่ของแรมและรีจิสเตอร์ภายในชิพ

สัญญาณจาก 74LS138 จะถูกต่อเข้ากับขา $\overline{CS}$ (หรือ $\overline{CE}$) ของหน่วยความจำเพื่ออีนาเบิลให้ติดต่อกับ 68HC11 ได้ และเมื่อต้องการอ่านข้อมูลจากอีพรอม ก็ต้องมีสัญญาณ $\overline{RD}$ ติดต่อกับ อีพรอม สัญญาณ ที่ได้มาจาก 68HC11จะแอกตีฟด้วยลอจิก "1" แต่อีพรอมแอกตีฟที่ลอจิก "0" จึงต้องมีการกลับสถานะลอจิกก่อน ด้วยเกตอินเวอร์เตอร์ 74LS04ได้สัญญาณ RD ไปต่อเข้ากับขา $\overline{OE}$ ของหน่วยความจำทั้งอีพรอมและแรม

สำหรับในกรณีที่ต้องการเขียนข้อมูลลงในแรมภายนอกต้องมีการสร้างสัญญาณ $\overline{WR}$ จากวงจรสัญญาณ $\overline{WR}$ ที่ได้มาจาก 68HC11 จะถูกสลับสถานะลอจิกด้วยอินเวอร์เตอร์ แล้วไปแอนด์กับสัญญาณจากขา E ของ 68HC11 ทั้งนี้ก็เพื่อกำหนดจังหวะการเขียนข้อมูลลงในแรม

ภายนอกให้สอดคล้องกับจังหวะของสัญญาณนาฬิกา E อันเป็นสัญญาณนาฬิกาหลักของระบบนั่นเอง สัญญาณที่ผ่านการแอน$\overline{WR}$ด์จะผ่านอินเวอร์เตอร์ได้เป็นสัญญาณ $\overline{WR}$ ต่อเข้ากับขา $\overline{WR}$ ของแรม

สิ่งสำคัญอีกประการหนึ่งที่ต้องไม่มองข้ามคือ การที่จะให้ 68HC11 สามารถเชื่อมต่อกับหน่วยความจำภายนอกได้ต้องกำหนดให้ 68HC11 ทำงานในโหมดมัลติเพล็กซ์ขยาย ดังนั้นจึงต้องกำหนดสถานะลอจิกที่ขา MODA และ MODB เป็น "1" ทั้งคู่ ซึ่งจากวงจรจะต่อตัวต้านทาน 10 กิโลโอห์มถูกพูลอัพไว้ (การพูลอัพ คือ การทำให้ขาสัญญาณนั้น ๆ มีสถานะลอจิกเป็น "1" โดยไม่จำเป็นต้องต่อไฟเลี้ยงเข้าโดยตรง แต่ต่อผ่านตัวต้านทานแทน)

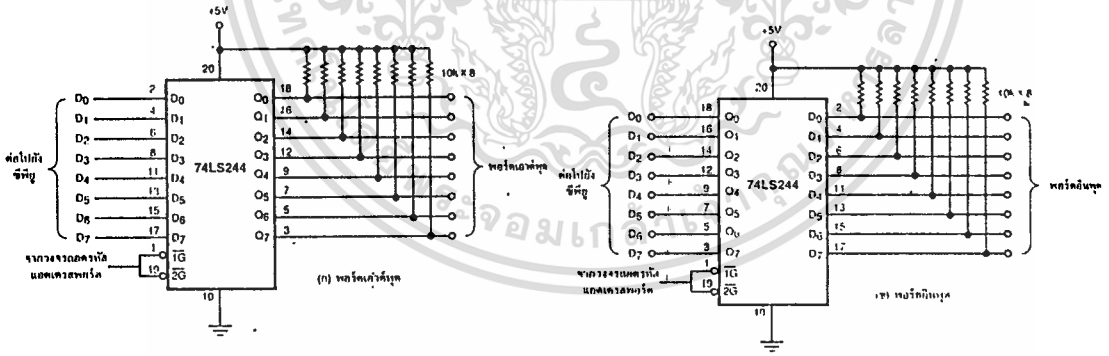
ส่วนสัญญาณนาฬิกาโดยปกติมักจะให้ 68HC11 ทำงานที่สัญญาณนาฬิกา E ความถี่ 2 เมกะเฮิรตซ์ ดังนั้นจึงต้องต่อคริสตอลความถี่ 8 เมกะเฮิรต์ เข้าที่ขา EXTAL และ XTAL พร้อมกับต่อตัวต้านทานและตัวเก็บประจุบายพาส ดังในรูปที่ 4.2 ด้วย

การเชื่อมต่อกับพอร์ต

พอร์ตจะแบ่งได้เป็นพอร์ตอินพุตและพอร์ตเอาต์พุต

-พอร์ตอินพุต คือ พอร์ตที่ใช้รับสัญญาณจากภายนอกเข้ามายังไมโครคอนโทรลเลอร์ เพื่อทำการประมวลผล

- พอร์ตเอาต์พุต คือ พอร์ตที่ใช้ส่งสัญญาณข้อมูลออกไปยังอุปกรณ์ภายนอก

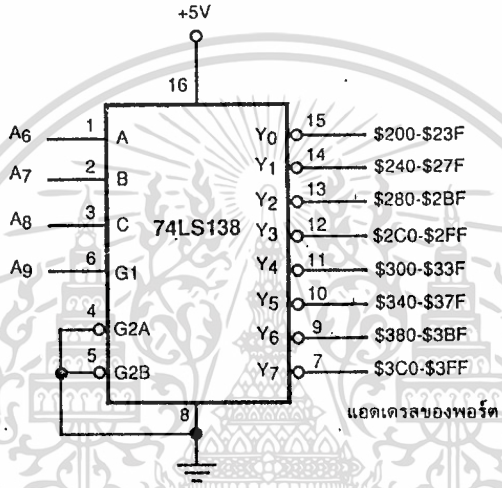


รูปที่ 4.3 การใช้ 74LS244 เป็นพอร์ตอินพุตและเอาต์พุต

ในรูปที่ 4.3 เป็นการใช้ไอซีเบอร์ 74LS244 ในการทำเป็นพอร์ตอินพุต และเอาต์พุต จากรูปที่ขา 1G และ 2G จะเห็นว่า ต้องต่อสัญญาณจากวงจรถอดรหัสแอดเดรสพอร์ตซึ่งวงจรนี้มีลักษณะเหมือนกับวงจรถอดรหัสแอดเดรสหน่วยความจำ หากแต่จะใช้ขาสัญญาณไม่เหมือนกัน (หรือจะเหมือนกันก็ได้ แต่ปกติไม่นิยม) ในที่นี้จะใช้ขา A6 - A9 เป็นตัวกำหนดช่วงแอดเด

รสของพอร์ต ซึ่งก็ได้ 8 ช่วง ดังในรูปที่ 4.4 การกำหนดแอดเดรสของพอร์ตนั้นจะขึ้นอยู่กับผู้ใช้งานเป็นหลัก

นอกเหนือไปจากการใช้ชิพ เช่น 74LS244, 74LS245, 74LS373, 74LS374, 74LS573 หรือ 74LS574 เป็นพอร์ตแล้ว ยังสามารถใช้ชิพที่เป็นพอร์ตอเนกประสงค์โดยเฉพาะ เช่น เบอร์ 8255 ได้ด้วย 8255 เป็นชิพที่มีพอร์ตขนาด 8 บิต ทั้งสิ้น 3 พอร์ต ในตัวเดียวกัน คือ พอร์ต A , B และ C โดยสามารถกำหนดแต่ละพอร์ตให้เป็นพอร์ตอินพุตหรือเอาต์พุตก็ได้



รูปที่ 4.4 วงจรถอดรหัสแอดเดรสของพอร์ต โดยใช้74LS244

4.2 การเชื่อมต่อกับพอร์ต 8255

ขั้นแรกต้องกำหนดเสียก่อนว่า ต้องการให้แต่ละพอร์ตทำหน้าที่เป็นพอร์ตอินพุตหรือเอาต์พุต จากนั้นก็ส่งข้อมูลควบคุมไปยัง 8255 เพื่อให้ทำงานตามที่กำหนดไว้ การกำหนดข้อมูลควบคุมในแต่ละบิตแสดงดังรูปที่ 4.5

โหมดการทำงานของ 8255 สามารถเลือกได้ 3 โหมด คือ โหมด 0, 1 และ 2

โหมด 0: เป็นการกำหนดให้ 8255 ทำงานเป็นพอร์ตอินพุตและเอาต์พุตแบบพื้นฐาน

โหมด 1: เป็นการกำหนดให้ 8255 ทำการรับส่งข้อมูลแบบมีการตรวจสอบสัญญาณความพร้อมหรือที่เรียกว่า *การแฮนด์เชค (hand shake)* โดยพอร์ต A และ B เป็นพอร์ตข้อมูล ส่วนพอร์ต C จะถูกใช้เป็นสัญญาณแฮนด์เชคโดยแบ่ง 4 บิตบนของพอร์ต C เป็นสัญญาณแฮนด์เชคของพอร์ต A และ 4 บิตล่างเป็นสัญญาณแฮนด์เชคของพอร์ต B นั่นคือ มีลักษณะการทำงานเหมือนกับระบบ SCI ของ 68HC11

โหมด 2: โหมดนี้จะทำให้สามารถใช้งานในการถ่ายเทข้อมูลได้เฉพาะพอร์ต A เท่านั้น โดยพอร์ต A จะทำงานเป็นพอร์ตอินพุตและเอาต์พุต หรือเรียกว่าเป็นพอร์ต 2 ทิศทางก็ได้ และ

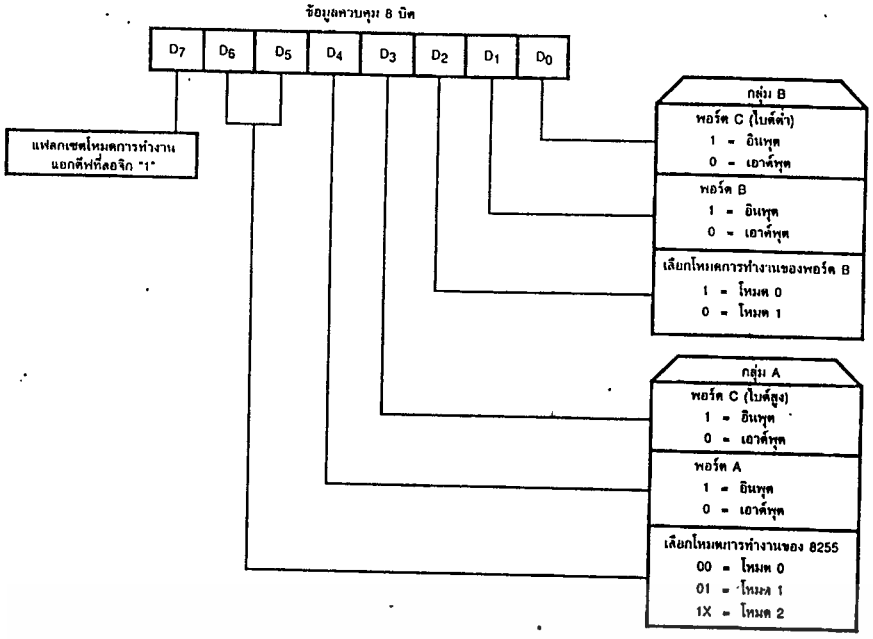
ในโหมดนี้ก็ยังคงมีการแฮนด์เชคเช่นเดียวกัน โดยใช้ขาพอร์ต C ทำหน้าที่เป็นขาสัญญาณแฮนด์เชคแสดงลักษณะการทำงาน

แต่โดยปกติการใช้งานทั่วไปแล้ว มักใช้ควบคุมพอร์ต 8255 ทำงานในโหมด 0 เพราะต้องการใช้งานในพอร์ต 8255 ให้เต็มที่ และยิ่ง 68HC11 มีระบบ SCI ซึ่งก็มีวงจรแฮนด์เชคอยู่ด้วยแล้ว ทำให้ควรใช้งาน 8255 ในโหมด 0 จะเหมาะสมที่สุด

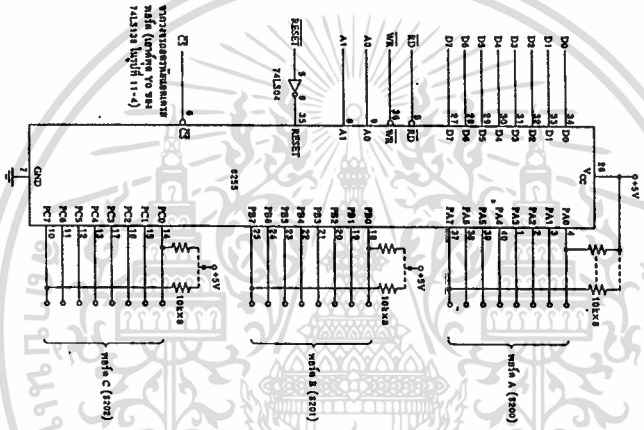
การเชื่อมต่อ 68HC11 เข้ากับ 8255 แสดงดังในรูปที่ 4.6 ขา D_0-D_7 และ A_0-A_1 ของ 68HC11 จะต่อเข้ากับขา D_0-D_7 และ A_0-A_1 ของ 8255, ขา $\overline{WR}$ และ $\overline{RD}$ ของ 8255 จะต่อกับสัญญาณ $\overline{WR}$ และ $\overline{RD}$ ที่ได้จากวงจรในรูปที่ 4.2 ส่วนขา RESET จะต้องทำการกลับสถานะของสัญญาณ RESET จาก 68HC11 ด้วย อินเวอร์เตอร์ก่อน และขาCS จะถูกต่อเข้ากับวงจรถอดรหัสแอดเดรสซึ่งไอซี74LS138ในวงจรจะกำหนดให้พอร์ต 8255 มีแอดเดรสอยู่ในช่วง $200 - $203โดย $200 เป็นแอดเดรสของพอร์ต A , $201เป็นแอดเดรสของพอร์ตB,$202เป็นแอดเดรสของพอร์ต Cและ$203 เป็นแอดเดรสของพอร์ตควบคุม 8255

จากรูปที่ 4.5 ข้อมูลควบคุมเป็น%10011001(%ที่นำหน้ามีมีความหมายว่าข้อมูลตัวเลขที่ตามหลังเป็นเลขฐานสอง)หรือเท่ากับ$99จะหมายความว่ากำหนดให้ 8255 ทำงานในโหมด 0 คือ เป็นพอร์ตอินพุตเอาต์พุตแบบพื้นฐาน โดยพอร์ต A เป็นพอร์ตอินพุต พอร์ต B เป็นพอร์ต C เป็นพอร์ตอินพุต ในการอินนีเชียลพอร์ต8255จะต้องมีการหน่วงเวลาเล็กน้อยเพื่อให้ตัว8255 พร้อมทำงานเสียก่อนหลังจากที่จ่ายไฟเลี้ยงให้ตัวมันจากนั้นจึงค่อยส่งข้อมูลควบคุมต่อไปดังนั้นการเขียนโปรแกรมเพื่ออินิเชียลพอร์ต 8255 จึงต้องเริ่มด้วยโปรแกรมหน่วงเวลา (delay) ก่อน แล้วค่อยส่งข้อมูลควบคุมดังนี้

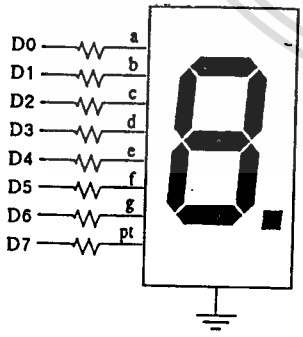
```
PORT_CRTL   EQU   $203
            LDAA  #$00
SYS_DLY     DECA
            BNE   SYS_DLY
            LDAA  #$98
            STAA  PORT_CTRL
```



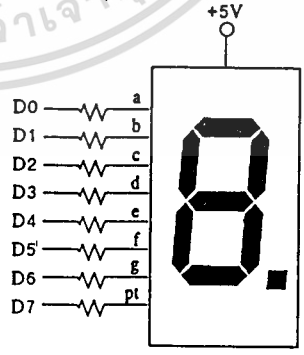
รูปที่ 4.5 การกำหนดข้อมูลควบคุมเพื่อเลือกโหมดการทำงานของ 8255



รูปที่ 4.6 วงจรการเชื่อมต่อ 68HC11 กับ 8255



(ก) LED แบบแคโทดร่วม



(ข) LED แบบแอโนดร่วม

รูปที่ 4.7 วงจรการเชื่อมต่อ 68HC11 กับ LED 7 ส่วน

4.3 การเชื่อมต่อกับ LED 7 ส่วน

LED 7 มีด้วยกัน 2 แบบคือแอโนดร่วมและแคโทดร่วม การกำหนดบิตข้อมูลของ 68HC11 เพื่อควบคุมติดดับของ LED 7 ส่วนทั้ง 2 แบบแสดงข้อมูลในรูปที่ 7.7 (ผู้ใช้สามารถกำหนดแตกต่างไปจากนี้ก็ได้) การเชื่อมต่อกับ LED 7 ส่วนโดยปกติแล้วควรผ่านพอร์ตเอาต์พุต จะโดยการใช้ไอซีอย่างเช่น 74LS373 หรือ 74LS244 ฯลฯ หรือใช้พอร์ต 8255 ก็ได้จากรูปที่ 6 อาจให้LED 7 ส่วนนี้ต่อเข้ากับพอร์ต B ซึ่งเป็นพอร์ตเอาต์พุต โดยขา PB_0 ทำหน้าที่เป็นขาข้อมูล D_0 ไล่ไปเรื่อย ๆ จนถึง PB_7 เป็นขาข้อมูล D_7 เป็นต้น

การควบคุมให้ LED 7 ส่วน แสดงข้อมูลตามที่ต้องการนั้น จะขึ้นอยู่กับข้อมูลที่ส่งไปขับ LED ในแต่ละส่วน จากรูปที่ 7 ถ้าหากต้องการให้ LED แสดงเลข 0 จะต้องส่งข้อมูล $3F ในกรณีที่ใช้ LED แบบแคโทดร่วมหรือ $C0 ในกรณีที่ใช้ LED แบบแคโทดร่วม ในตารางที่ 2 เป็นการสรุปข้อมูลที่ส่งออกไปควบคุมให้ LED 7 ส่วน แสดงตัวเลขและตัวอักษรต่าง ๆ

ตารางที่ 2 ข้อมูลของตัวเลขและตัวอักษรที่ใช้ในการแสดงผลของ LED 7 ส่วน

7 SEGMENT	LED แบบแคโทดร่วม	LED แบบแอโนดร่วม
0	$3F	$C0
1	$06	$F9
2	$5B	$A4
3	$4F	$B0
4	$66	$99
5	$6D	$92
6	$7D	$82
7	$07	$F8
8	$7F	$80
9	$6F	$90
A	$77	$88
B	$7C	$83
C	$39	$C6
D	$5E	$A1
E	$79	$86
F	$71	$8E
H	$76	$89
P	$73	$8C

4.4 การเชื่อมต่อกับ LCD

ตัวแสดงผล LCD (Liqiud Crystal Dispaly) เป็นอุปกรณ์แสดงผลแบบหนึ่งที่นิยมใช้มากและพบเห็นกันบ่อย ๆ LCD สามารถแบ่งประเภทตามลักษณะของการแสดงผลได้ 3 แบบ คือ

- แบบอักขระ LCD (character LCD module)
- แบบกราฟิก LCD (graphic LCD module)
- แบบเซกเมนต์ LCD (segment LCD module)

LCD แบบอักขระ

เป็นโมดูล LCD ที่สามารถแสดงอักษร, ตัวเลข และเครื่องหมายต่าง ๆ โดยสร้างจากจุดเล็ก ๆ เรียกว่า ดอตเมตริกซ์ ซึ่งก็จะมีขนาดความกว้างและสูงของอักขรแต่ละตัว โดยทั่ว ๆ ไปมี 2 ขนาด คือ 5x7 จุด และ 5x10 จุด นอกจากนั้น LCD แบบนี้สามารถแสดงข้อความได้ 1 บรรทัด มากกว่าก็ได้ ขึ้นอยู่กับรุ่นของ LCD นั้น ๆ

LCD แบบกราฟิก

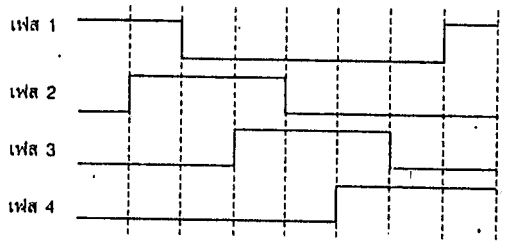
สามารถแสดงข้อมูลเป็นทั้งตัวอักษรตัวเลขเครื่องหมายและรูปภาพได้ความละเอียดของภาพจะเกิดขึ้นอยู่กับความละเอียดของดอตเมตริกซ์ของCD นั้น ๆ ขนาดของLCDแบบนี้มีหลายขนาดให้เลือกใช้ในปัจจุบันมีการพัฒนาเป็นสี่สีแล้วด้วย

LCD แบบเซกเมนต์

เป็นโมดูล LCD แบบเล็กสุด มีลักษณะการแสดงผลคล้ายกับ LED 7 ส่วน โดยปกติมักจะมีมากกว่า 1 หลักพบเห็นทั่วไปในดิจิตตอลมัลติมิเตอร์ขาสัญญาณของโมดูล LCD แบบอักขระ

โมดูล LCD แบบอักขระที่นำมาเป็นตัวอย่างนี้เป็นแบบ 1 บรรทัด 16 ตัวอักษร มีขาต่อใช้งานได้ทั้งสิ้น 14 ขา ได้แก่

V_{SS} (ขา 1)	: ต่อกราวด์
V_{DD}	: ต่อไฟเลี้ยง
V_o	: เป็นขาอินพุตสำหรับป้อนแรงดันเพื่อปรับความเข้มของการ แสดงผล
RS (ขา 4)	: เป็นขาอินพุต ใช้เลือกว่าข้อมูลที่ทำการส่งในขณะนั้นเป็นข้อมูลคำสั่งสำหรับรีจิสเตอร์คำสั่งของLCDหรือเป็นข้อมูลสำหรับรีจิสเตอร์ข้อมูลแสดงผลของLCDโดยถ้าขานี้เป็น “0” ข้อมูลที่ส่งมาจะเป็นข้อมูลคำสั่ง แต่ถ้าเป็น “1” ข้อมูลที่ส่งมาจะเป็นข้อมูลสำหรับการแสดงผล



รูปที่ 4.11 ไทมิ่งไดอะแกรมของการป้อนพัลซ์กระตุ้นแบบฮาล์ฟสเต็ป

การควบคุมการหมุนของมอเตอร์ด้วยข้อมูลดิจิตอล

ถ้าพิจารณาจากไทมิ่งไดอะแกรมของการป้อนพัลส์กระตุ้นจะเห็นได้ว่าพัลส์ที่ส่งออกไปกระตุ้นนั้นจะมีการเคลื่อนที่ไปทีละขั้น ซึ่งมีลักษณะเหมือนกับการเลื่อนข้อมูลในระบบไมโครโปรเซสเซอร์

ดังนั้นถ้าหากส่งข้อมูลดิจิตอลผ่านออกไปทางพอร์ตแล้วนำข้อมูลเหล่านี้เข้าวงจรขับกระแสก่อน เอาต์พุตที่ได้จากวงจรขับจะส่งไปควบคุมให้มอเตอร์หมุนได้ เมื่อเปลี่ยนแปลงข้อมูลดิจิตอลก็จะทำให้ทิศทางการหมุนของมอเตอร์เปลี่ยนไป

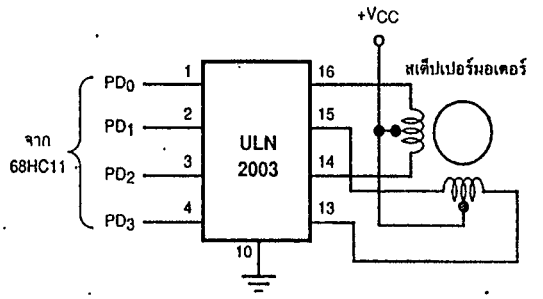
ถ้าหากต้องการขับมอเตอร์ให้หมุนด้วยการป้อนพัลส์กระตุ้นแบบเฟสเดี่ยว กลับไปพิจารณาไทมิ่งไดอะแกรมรูปที่ 4.9 จะได้ข้อมูลดิจิตอลเพื่อนำไปขับมอเตอร์ดังนี้

ข้อมูลเริ่มต้น	0001
ข้อมูลเฟส 1	1000
ข้อมูลเฟส 2	0100
ข้อมูลเฟส 3	0010
ข้อมูลเฟส 4	0001

แล้ววนเช่นนี้เรื่อยไป มอเตอร์ก็จะหมุนไปในทิศทางเดียวกันอย่างต่อเนื่อง

การเชื่อมต่อ 68HC11 กับสเต็ปเปอร์มอเตอร์

รูปที่ 4.12 เป็นตัวอย่างวงจรเชื่อมต่อ 68HC11 กับสเต็ปเปอร์มอเตอร์ จากวงจรจะใช้พอร์ต D เป็นพอร์ตที่ส่งข้อมูลที่ต้องการควบคุมให้มอเตอร์หมุนผ่านไอซี ULN2003 เพื่อขับกระแสให้มอเตอร์หมุนได้

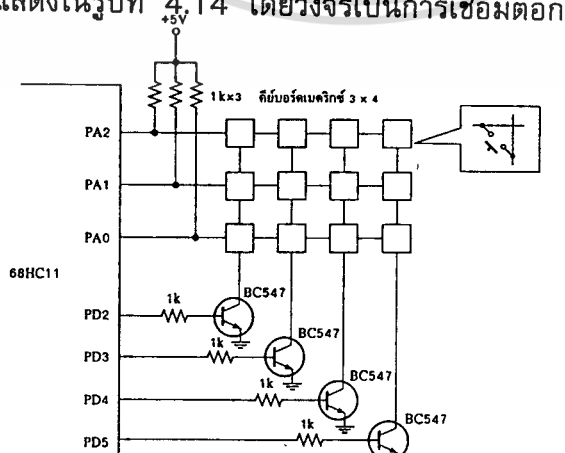


รูปที่ 4.12 การใช้ ULN2003 เพื่อควบคุมสเต็ปเปอร์มอเตอร์โดยได้รับการควบคุมจาก 68HC11

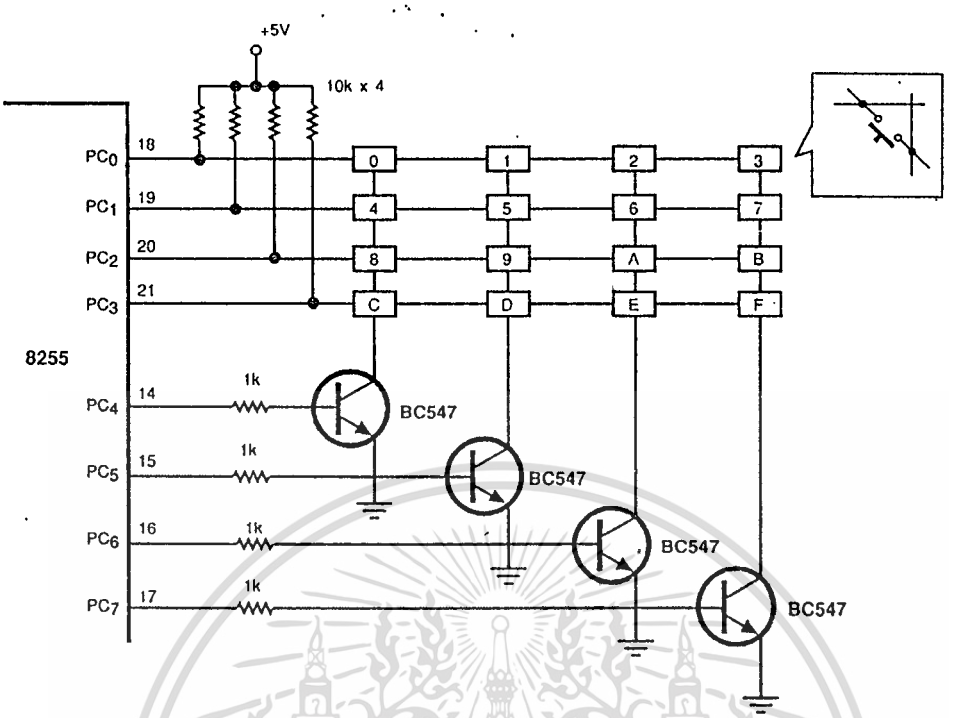
4.6 การเชื่อมต่อ 68HC11 กับคีย์บอร์ด

ในการเชื่อมต่อ 68HC11กับคีย์บอร์ด อาจใช้พอร์ตที่มีอยู่แล้วของตัว 68HC11เองดังแสดงในรูปที่4.13อันเป็นวงจรเชื่อมต่อ 68HC11กับคีย์บอร์ดเมตริกซ์ 3x4 โดยให้พอร์ต PA_0 - PA_2 เป็นพอร์ตอินพุตรับข้อมูลจากการกดคีย์ และพอร์ต PD_2 - PD_5 เป็นพอร์ตเอาต์พุตใช้ส่งข้อมูลเพื่อเลือกหลักของคีย์บอร์ดดังนั้นจึงต้องมีการเขียนโปรแกรมเพื่อกำหนดให้พอร์ต A และ D ถูกใช้งานเป็นพอร์ตอินพุตเอาต์พุตธรรมดา

แต่ในการเชื่อมต่อคีย์บอร์ด โดยปกติควรจะต่อผ่านพอร์ตภายนอก เช่น 8255 เพราะพอร์ต A นั้นเหมาะสมที่จะใช้ในการเป็นขาสัญญาณอินพุตเอาต์พุตของตัวตั้งเวลามากกว่าในขณะพอร์ตDจะใช้ในการสื่อสารข้อมูลอนุกรม ตัวอย่างวงจรของการเชื่อมต่อ 68HC11 กับ 8255 และคีย์บอร์ดแสดงในรูปที่ 4.14 โดยวงจรเป็นการเชื่อมต่อกับคีย์บอร์ดเมตริกซ์ 4x4



รูปที่ 4.13 การต่อ 68HC11 เข้ากับคีย์บอร์ด โดยใช้พอร์ตที่มีอยู่ภายในชิพเอง

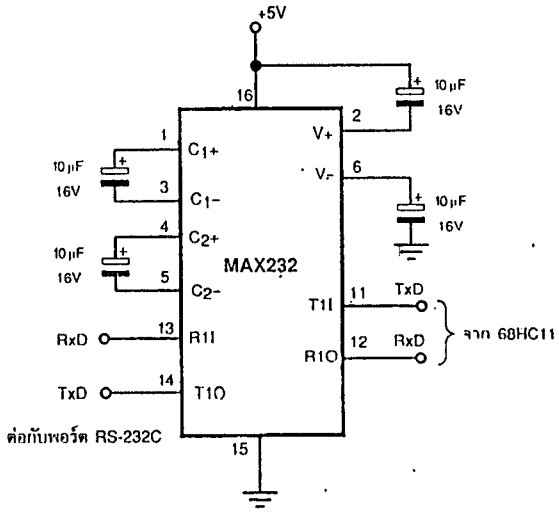


รูปที่ 4.14 วงจรการเชื่อมต่อ 68HC11 เข้ากับคีย์บอร์ด โดยผ่านพอร์ต 8255

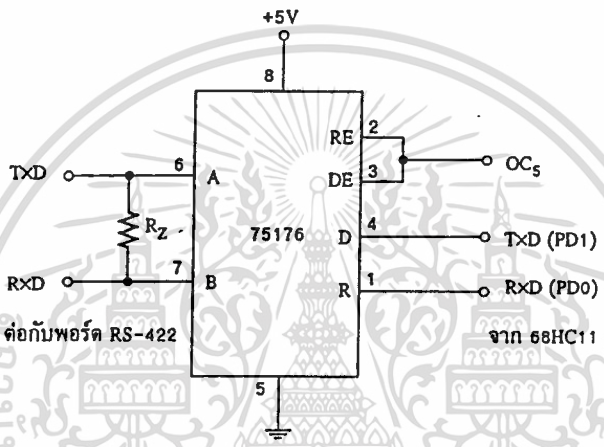
4.7 การเชื่อมต่อกับพอร์ตอนุกรม RS-232 และ RS-422

การเชื่อมต่อ 68HC11 เข้ากับพอร์ต RS-232 และ RS-422 ของเครื่องคอมพิวเตอร์ เพื่อรับ-ส่งข้อมูลระหว่างกันและกัน ขาสัญญาณของ 68HC11 ที่ถูกใช้งานคือขา RxD และ TxD ซึ่งก็คือ ขา PD_0 และ PD_1 นั่นเอง ชิพที่ทำหน้าที่ปรับสัญญาณข้อมูลให้เหมาะสมสำหรับพอร์ต RS-232 หรือที่เรียกว่า ไดรเวอร์ (driver) ได้แก่เบอร์ MAX232 จะเชื่อมต่อเข้ากับ 68HC11 ดังวงจรในรูปที่ 4.15 โดยก่อนที่จะใช้งานต้องเขียนโปรแกรมเพื่ออีนาเบิลระบบ SCI เสียก่อน

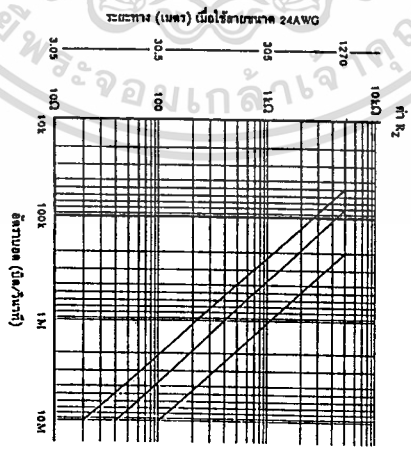
เนื่องจากข้อจำกัดของระยะทางในการสื่อสารข้อมูลของ RS-232 ซึ่งมีระยะทางเพียง 50 ฟุต ทำให้ถ้าต้องการใช้งานในระยะทางที่ไกลกว่านั้น จำเป็นต้องใช้พอร์ตอนุกรมอีกพอร์ตหนึ่งคือ RS-422 โดยนำ
ชิพไดรเวอร์เบอร์ 75176 มาต่อใช้งานแทน MAX232 ดังมีวงจรการเชื่อมต่อดังรูปที่ 4.16 จะเห็นว่าต้องต่อตัวต้านทาน R_z เข้าไประหว่างขา A และ B ของ 75176 หรือ ขา TxD และ RxD ที่จะต่อกับพอร์ต RS-422 ค่าของ R_z หาได้จากกราฟในรูปที่ 4.17 ยกตัวอย่างกำหนดอัตราบอดเท่ากับ 9,600 บิตต่อวินาที ระยะทางที่ต้องการรับส่งเท่ากับ 30 เมตร จะได้ค่า R_z ประมาณ 100 โอห์ม



รูปที่ 4.15 การต่อ 68HC11 กับพอร์ต RS-232C โดยใช้ชิพ MAX232



รูปที่ 4.16 การต่อ 68HC11 เข้ากับพอร์ต RS-422 โดยใช้ชิพเบอร์ 74176



รูปที่ 4.17 กราฟเพื่อใช้ช่วยในการเลือกค่า R_z ในกรณีต่อ 68HC11 เข้ากับ 75176

บทที่ 5

เอกสารประกอบการทดลอง MICROCONTROLLER 68HC11

การทดลองที่ 1 : ภาษาเครื่องและภาษาแอสเซมบลี้
การทดลองที่ 2 : การใช้งานรีจิสเตอร์ ของ 68HC11
การทดลองที่ 3 : การอ้างตำแหน่งของ 68HC11
การทดลองที่ 4 : การใช้คำสั่งในการโอนย้ายข้อมูลของ 68HC11
การทดลองที่ 5 : การใช้คำสั่งทางคณิตศาสตร์ของ 68HC11
การทดลองที่ 6 : การใช้คำสั่งทางตรรกศาสตร์ หรือคำสั่งทางลอจิก ของ 68HC11
การทดลองที่ 7 : การใช้คำสั่งในการกระโดดของ 68HC11
การทดลองที่ 8 : การใช้งาน LCD MODULE
การทดลองที่ 9 : การใช้งาน สเต็ปปิ้งมอเตอร์ (STEPPING MOTOR)
การทดลองที่10: การใช้งาน SERIAL PORT RS-232

การทดลองที่ 1
ภาษาเครื่องและภาษาแอสเซมบลี้

วัตถุประสงค์

1. เข้าใจความหมายของภาษาเครื่อง
2. เข้าใจความหมายเชงภาษาแอสเซมบลี้
3. สามารถป้อนโปรแกรมภาษาเครื่องบนไมโครคอมพิวเตอร์แผ่นพิมพ์เดียว
4. สามารถแปลโปรแกรมภาษาแอสเซมบลี้เป็นโปรแกรมภาษาเครื่องได้

ทฤษฎี

ภาษาของคอมพิวเตอร์ก็คือสัญญาณไฟฟ้า มีลักษณะเป็นสัญญาณสองระดับเช่น 0 โวลท์ กับ 5 โวลท์ และเพื่อให้ง่ายต่อการใช้ เราจะเขียนแทนสัญญาณนี้ด้วยตัวเลขฐานสอง เช่น เลข 0 แทนค่า 0 โวลท์ และเลข 1 แทนด้วย 5 โวลท์ การทำงานหนึ่งๆ อาจจะประกอบด้วยสัญญาณ 0 กับ 1 หลายๆ ตัว กลุ่มของสัญญาณแต่ละกลุ่มนี้เราเรียกว่า "คำสั่ง" เมื่อนำคำสั่งหลายๆ คำสั่งมาเขียนเรียงกันเพื่อให้ทำงานอย่างหนึ่งเราเรียกว่า "โปรแกรม" โปรแกรมที่เขียนอยู่ในรูปของเลขฐานสอง เราเรียกว่า "โปรแกรมภาษาเครื่อง" แต่โดยทั่วไปมักจะเขียนภาษาเครื่องในรูปของเลขฐานสิบหก เพราะง่ายต่อการจำ

ลักษณะของภาษาเครื่อง

ภาษาเครื่อง คือภาษาที่สั่งให้ CPU ทำงาน ดังนั้นถ้าจะเขียนโปรแกรมภาษาเครื่องกับเครื่องคอมพิวเตอร์ใดต้องรู้ว่าเครื่องนั้นใช้ CPU เบอร์ใด เมื่อรู้เบอร์ CPU นั้น เพื่อเขียนโปรแกรมรหัสภาษาเครื่องให้ตรงกับเบอร์นั้นได้ ฟิลด์ของภาษาเครื่องมีเพียง 2 ฟิลด์ คือฟิลด์ตำแหน่ง (Address) และฟิลด์ข้อมูล (Data)

การโปรแกรมหรือการป้อนโปรแกรมภาษาเครื่องหมายถึงการนำข้อมูลลงไปใส่ในหน่วยความจำของไมโครคอมพิวเตอร์ ตั้งแต่ตำแหน่งแรกสุดถึงตำแหน่งสุดท้าย รูปที่ 1 แสดงการเก็บโปรแกรมของโปรแกรมที่ 1 ลงหน่วยความจำตามตำแหน่งของข้อมูล

ลักษณะของภาษาแอสเซมบลี้

ภาษาแอสเซมบลี้เป็นภาษาที่สูงกว่าภาษาเครื่อง มีความสะดวกในการใช้งานมากกว่าภาษาเครื่อง โดยคำสั่งจะเขียนอยู่ในรูปคำย่อภาษาอังกฤษ เช่น LDAA ย่อมาจาก LODA Accumulator A หมายถึงเป็นคำสั่งที่ใช้โหลดข้อมูลที่อ้างอิงมาเก็บไว้ใน A หรือคำสั่ง STAA = Store Accumulator A เป็นคำสั่งที่ใช้ในการอ่านค่า แอคคิวมูเลเตอร์ A มาเก็บไว้ในหน่วยความจำ จะมีลักษณะการทำงานตรงข้ามกับ LDAA จะเห็นว่าแต่ละคำสั่งนั้นมีความหมายในตัวเอง

และจำได้ง่ายกว่าคำสั่งในรูปรหัสฐานสิบหกของภาษาเครื่อง ฟิลด์ของภาษาแอสเซมบลี้จะแบ่งแยกจากกัน แต่ละฟิลด์จะถูกแบ่งด้วยช่องว่าง แต่ละฟิลด์มีข้อกำหนดดังนี้

1. เลเบลฟิลด์ (Label field) เป็นส่วนใช้สำหรับอ้างอิงถึงในโปรแกรม หรือเป็นตำแหน่ง Address นั่นเอง ข้อกำหนดทั่วไปของการตั้งชื่อเลเบลมีดังนี้
   1. ชื่อเลเบล ควรมีความหมายตรงกับงานที่ทำ
   2. เป็นตัวอักษร A ถึง Z (เป็นตัวพิมพ์ใหญ่หรือเล็กก็ได้)
   3. เป็นตัวเลข 0 ถึง 9
   4. เป็นสัญญาณลักษณ์พิเศษบางตัวเช่น ขีดล่าง (_) เป็นต้น
   5. ประกอบไปด้วยตัวอักษรจากข้อ 1 ถึง 3 อย่างไรก็ได้ ส่วนความยาวขึ้นอยู่กับตัวแปลภาษาที่ใช้ (6-10 ตัว)
   6. ขึ้นต้นด้วยตัวอักษรในข้อ 2 เสมอ
   7. เลเบลต้องตามหลังด้วยโคลอน (:) เสมอ ยกเว้นคำสั่งที่ใช้คู่กับคำสั่งเทียม EQU
   8. ห้ามตั้งชื่อซ้ำกับรหัสนีโมนิคของ CPU หรือคำสั่งเทียม

2. นีโมนิคฟิลด์ (Nemonic Field) หรือออปโค้ดฟิลด์ (Operation-Code) เป็นฟิลด์คำสั่งซึ่งมีอยู่ 2 ประเภท คือ
   1. คำสั่งของ CPU (รหัสนีโมนิค) คำสั่งประเภทนี้เราต้องดูจากคู่มือของ CPU เช่นคำสั่งเกี่ยวกับโหลด ข้อมูลของไมโครคอนโทลเลอร์ 68HC11 จะใช้คำสั่ง LDAA
   2. คำสั่งเทียม (Pseudo) เป็นคำสั่งที่สร้างขึ้นมาเพื่ออำนวยความสะดวกในการเขียนโปรแกรม มีอยู่ด้วยกันหลายคำสั่ง ขึ้นอยู่กับตัวแปลโปรแกรม

      ORG มาจาก ORGIN เป็นตัวกำหนด Address ของคำสั่งที่อยู่ถัดไป

      EQU มาจาก EQUAL เป็นการกำหนดค่าของตัวแปล

      DB มาจาก Define Byte เป็นการกำหนดค่าในหน่วยความจำครั้งละ 1 byte หรือหลาย Byte

      DW มาจาก Define Word เป็นการกำหนดค่าใจหน่วยความจำครั้งละสอง byte ลักษณะการใช้คล้ายกับ D8

      DS มาจาก Define Storge เป็นการจองพื้นที่ในหน่วยความจำ มีค่าเป็นจำนวน byte เช่นจองพื้นที่ในหน่วยความจำตั้งแต่ address ที่ชื่อ LIGHT เป็นจำนวน 9 byte (โดยปกติพื้นที่ที่จองตัวแปลจะใส่ค่า 00 ไว้ให้)

      END เป็นคำสั่งบอกจุดสิ้นสุดของโปรแกรม ต้องมีทุกครั้ง

โอเปอร์แรนด์ฟิลด์ (Operand field) เป็นฟิลด์ที่บอกถึงตัวกระทำ ซึ่งอาจจะเป็น register data address หรือตัวแปร ขึ้นอยู่กับคำสั่งนั้นๆ

ฟิลด์หมายเหตุ (Operand field) เป็นส่วนที่ใช้อธิบายโปรแกรม ฟิลด์นี้ต้องนำหน้าด้วยเครื่องหมาย เซมิ โคลอน (:) เสมอ และอาจให้เป็นหมายเหตุทั้งบร

```
OP-CODE

                        ORG       $2000
7F 10 07                CLR       $1007
96 02                   LDAA      #$02
B7 10 02                STAA      $1002
18 FE 20 0A             LDY       #POINTER
13 10 02          LOOP  BRCLR     $1002,RET
B6 10 05                LDAA      $1005
18 08                   INY
18 BC                   CPY       #POINTER+10
26                      BNE  LOOP
CF                      STOP
7E 20 00                RET       JMP LOOP
                        END
```

<u>การทดลอง</u>

```
OP-CODE

                        ORG       $5000
________                CLRA
________                CLRB
________                LDAA      #$55
________                LDAB      #$2A
________                TAB
________                LDAB      $3000
________                CLRA
________                STAA      $1007
________                LDD       #$5678
________                LDY       #$5A07
                        END
```

1. ให้เปิดตาราง OP-CODE ภาษาเครื่อง แล้วใส่ลงในช่องว่างฟิลด์ Data
2. จากตาราง อธิบายว่าเป็นการอ้างแอดเดรสแบบใด
3. สรุปผลการทดลอง

<u>คำถาม</u>

1. คำสั่งเทียมมีคำสั่งอะไรบ้าง ?
2. บอกความแตกต่างของคำสั่งเทียม และคำสั่งของ CPU ?
3. ภาษาเครื่องและภาษาแอสเซมบลี้แตกต่างกันอย่างไร ?

การทดลองที่ 2
การใช้งานรีจิสเตอร์ ของ 68HC11

วัตถุประสงค์

1. เพื่อศึกษาความเข้าใจของการใช้งานรีจิสเตอร์
2. เพื่อสามรถนำเอาหลักการการใช้งานรีจิสเตอร์มาเขียนโปรแกรมได้

ทฤษฎี

การจัดสรรหน่วยความจำ หรือที่เรียกว่า Memory maps จะขึ้นอยู่กับว่า MC68HC11A8 นี้ทำงานอยู่ในโหมดใด จากรูปที่ 1 ส่วนที่แรเงาจะถูกแสดงรายละเอียดเอาไว้ในรูปด้านขวามือเรียบร้อยแล้ว เพื่อบอกให้กับผู้ใช้งานทราบว่า พื้นที่ในส่วนใดถูกจัดสรรไว้เพื่อรองรับงานอย่างใดบ้าง แรมขนาด 256 ไบต์ถูกจัดสรรไว้ในแอดเดรส $0000-$00FF ที่แอดเดรส $1000-$103F เป็นพื้นที่สำหรับรีจิสเตอร์ควบคุมต่าง ๆ ขนาด 64 ไบต์ ทั้งแรมและรีจิสเตอร์สามารถที่จะมีการจัดสรรตำแหน่งกันใหม่ ถ้าหากมีการใช้งานในลักษณะ 4 เพจ ซึ่งสามารถกำหนดได้โดยรีจิสเตอร์ INIT อันจะได้กล่าวถึงต่อไป

ที่แอดเดรส $B600-4B7FF ขนาด 512 ไบต์เป็นพื้นที่ของหน่วยความจำอีอีพรอม ที่แอดเดรส 4BF40-4BFFF ถ้า 68 HC11 ทำงานในโหมดบูตสแตร็ปพิเศษพื้นที่นี้ จะสงวนไว้เป็นรอมที่เก็บข้อมูลสำหรับเริ่มต้นการทำงานของระบบ (Boot ROM) ถ้าหาก 68HC11 ทำงานในโหมดทดสอบพิเศษที่แอดเดรส $BFC0-$BFFF ซึ่งมีขนาดของหน่วยความจำเป็น 192 ไบต์ จะถูกจัดสรรไว้สำหรับเก็บอินเตอร์รัปต์เวกเตอร์ของการทำงานโหมดบูตสแตร็ปพิเศษ (Special modes interrupt vectors)

สำหรับแอดเดรส $E000-$FFFF จะถูกจัดสรรไว้สำหรับเป็นพื้นที่ของรอมขนาด 8 กิโลไบต์ แต่ถ้าหาก 68HC11 ทำงานในโหมดชิงเกิลชิปและมัลติเพล็กซ์ขยายที่แอดเดรส $FFC0-$FFFF จะถูกจัดสรรไว้เพื่อเก็บค่าของอินเตอร์รัปต์เวกเตอร์ปกติ (Normal interrupt vectors)

ส่วนบริเวณที่เขียนว่า Ext เป็นพื้นที่ที่จัดสรรไว้สำหรับต่อหน่วยความจำภายนอกเพิ่มขึ้นจะเห็นว่ามีด้วยกันเพียง 2 โหมดมัลติเพล็กซ์ขยายและโหมดการทดสอบพิเศษ และทั้ง 2 โหมดนั้นก็ได้กำหนดแอดเดรสสำหรับหน่วยความจำไว้ 3 ช่วงคือ$0100-$0FFF มีขนาด 3,840ไบต์,$1040-$B5FF ขนาด 42,432 ไบต์ (41.4375 กิโลไบต์) และ $B800-$DFFF ขนาด 10 กิโลไบต์

รีจิสเตอร์ควบคุมภายใน 68 HC11

MC68HC11A8 จะมีรีจิสเตอร์ควบคุมอยู่ภายใน อยู่ที่แอดเดรส $1000-$103F รวม 64 ไบต์ด้วยกัน รีจิสเตอร์แต่ละตัวจะมีหน้าที่ควบคุมการทำงานของไมโครคอนดทรลเลอร์แตก

ต่างกัน และขนาดของรีจิสเตอร์แต่ละตัวก็ไม่เท่ากัน ในแต่ละบิตของรีจิสเตอร์แต่ละตัวก็จะมีความหมายและฟังก์ชันการทำงานที่แตกต่างกัน

ส่วนรีจิสเตอร์ควบคุมหลัก ๆ พอจะแบ่งกลุ่มได้ดังนี้

1. รีจิสเตอร์เกี่ยวกับพอร์ตมีด้วยกัน 9 ตัว ประกอบด้วย รีจิสเตอร์ PORTA,PORTB,PORTC,PORTD,PORTE,PORTCL,DDRC,DDRD แต่ละตัวมีขนาด 8 บิต
2. รีจิสเตอร์เกี่ยวกับตัวตั้งเวลาและตัวนับมีด้วยกัน 8 ตัว ประกอบด้วย TCNT,TCTL1,TCTL2,TMSK1,TMSK2,TFLG1,TFLG2 และ COPRST แต่ละตัวมีขนาด 8 บิต ยกเว้น TCNT ที่มีขนาด 16 บิต
3. รีจิสเตอร์เกี่ยวกับการเปรียบเทียบมี 6 ตัว ประกอบด้วย TOC1-TOC5 และ CFORC แต่ละตัวมีขนาด 16 บิต ยกเว้น CFORC จะมีขนาด 8 บิต
4. รีจิสเตอร์อินพุตแคปเจอร์ มี 3 ตัวคือ TIC1-TIC3 แต่บะตัวมีขนาด 16 บิต
5. รีจิสเตอร์ควบคุมพัลส์แอกคิวมูเลเตอร์ มี 2 ตัวคือ PACTL และ PACNT แต่ละตัวมีขนาด 8 บิต
6. รีจิสเตอร์เกี่ยวกับการควบคุมพอร์ตอนุกรม มี 8 ตัว ประกอบด้วย SPCR,SPSR,SPDR,BAUD,SCCR1,SCCR2,SCSR และ SCDR แต่ละตัวมีขนาด 8 บิต
7. รีจิสเตอร์ควบคุมการแปลงสัญญาณแอนาล็อกเป็นดิจิตอลมี 5 ตัวคือ ADCTL และ ADR4 แต่ละตัวมีขนาด 8บิต
8. รีจิสเตอร์ควบคุมระบบและหน่วยความจำ มี 6 ตัว ประกอบด้วย OPTION,PPROG,HPRIO,INIT,TEST1 และ CONFIG แต่ละตัวมีขนาด 8 บิต

<u>รีจิสเตอร์</u>

68HC11 มีรีจิสเตอร์ที่ใช้ในการเขียนโปรแกรมต่าง ๆ ทั้ง 8 บิต และ 16 บิต อยู่ด้วยกัน 7 ตัว ได้แก่

แอกคิวมูเลเตอร์ A และ B เป็นรีจิสเตอร์ขนาด 8 บิต ที่ใช้ในการคำนวณทางด้านคณิตศาสตร์ และตรรกศาสตร์ นอกจากนั้นยังรวมกันเป็นรีจิสเตอร์ D ขนาด 16 บิต โดยแอกคิวมูเลเตอร์ A เป็น 8 บิตสูงและแอกคิวมูเลเตอร์ B เป็น 8 บิตต่ำ

อินเด็กซ์รีจิสเตอร์ IX เป็นรีจิสเตอร์ขนาด 16 บิต ใช้เป็นตัวชี้ตำแหน่งของข้อมูลและเป็นตัวตั้งในการบวกแบบ 16 บิตกับ แอกคิวมูเลเตอร์ B

อินเด็กซ์รีจิสเตอร์ IY เป็นรีจิสเตอร์ขนาด 16 บิต มีลักษณะคล้ายกับรีจิสเตอร์ IX แต่คำสั่งที่ใช้จะมีออปโค้ดขนาดสองไบต์

สแต็คพอยเตอร์ SP เป็นรีจิสเตอร์ขนาด16 บิต ชี้ตำแหน่งของสแต็คในการเก็บข้อมูลต่าง ๆ จากการเรียกโปรแกรมย่อย หรือการอินเตอร์รัพต์ การเก็บข้อมูลลงสแต็คมีลักษณะแบบเข้าก่อนออกทีหลัง (FIFO)

โปรแกรมเคาน์เตอร์ PC เป็นรีจิสเตอร์ขนาอ 16 บิต ใช้เป็นตัวชี้ตำแหน่งของคำสั่งต่อไปจะประมวลผล

คอนดิชั่นโค้ดรีจิสเตอร์ CCR เป็นรีจิสเตอร์ขนาด 8 บิต เป็นตัวบ่งบอกลักษณะการทำงานต่าง ๆ ของซีพียู และกำหนดการทำงานของซีพียู เช่น การกำหนดให้อีนาเบิลหรือดิสเอเบิลอินเตอร์รัพต์และการทำงานใน STOP โหมด

ตัวอย่างโปรแกรมการใช้รีจิสเตอร์ TIC1

```
RSETIME         EQU     $3000
PULSEWIDTH      EQU     $3002
                LDAA    #$10
                STAA    TCTL2
                LDAA    #$04
                STAA    TFLG1
WAITRISE        BRLCR   TFLG1,$04,WAITRISE
                LDD     TIC1,$1010
                STD     RISETIME
                LDAA    #$20
                STAA    TCTL2
                LDAA    #$04
                STAA    TFLG1
WAITFALL        BRCLR   TFLG1,WAITFALL
                LDD     TIC1
                SUBD    RISETIME
                STD     PULSEWIDTH
                BRA     $
```

เมื่อวงจรสามารถตรวจจับขอบขาขึ้นของสัญญาณได้ จะได้ค่าของการนับเป็น $000A อยู่ในรีจิสเตอร์ TCNT ซึ่งเมื่อเทียบในโปรแกรมจะใช้ตัวแปรชื่อ RISETIME ถ้าหากความกว้างของสัญญาณเป็น 65,528 ไซเกิลของสัญญาณนาฬิกา E ๖ปรีสเกลเลอรีด้วยหนึ่ง) รีจิสเตอร์

TIC1 จะเก็บค่าได้ $0002 เมื่อตรวจจับขอบขาลงของสัญญาณได้ กรณีนี้จะเกิดการโอเวอร์โฟลว์ขึ้น

การทดลอง

```
        ORG     $2000
        LDAA    #$8A
LSLA
        CLC
        SEI
        SEV
        SEC
        CLI
        CLV
        END
```

1. ทดลองป้อนโปรแกรม
2. แล้วทำการเช็คค่าต่อไปนี้ทีละ Step แล้วเติมในตาราง

	Carry Flag	Overflow	บิต Interrupt
Step 1			
Step 2			
Step 3			
Step 4			
Step 5			
Step 6			
Step 7			
Step 8			

<u>คำถาม</u>

1. รีจิสเตอร์ที่ในการขียนโปรแกรมมีอะไรบ้าง

การทดลองที่ 3
การอ้างตำแหน่งของ 68 HC11

<u>วัตถุประสงค์</u>

1. เพื่อศึกษาความเข้าใจในการอ้างตำแหน่งของ 68 HC11
2. เพื่อนำเอาหลักการของการอ้างตำแหน่งไปใช้ในการเขียนโปรแกรม

<u>ทฤษฎี</u>

คำสั่งแอสเซลบลี้ของ 68 HC11 จะประกอบด้วยนิวมอนิกรหัสการทำงาน 1 ไบต์ (op-code) และตัวโอเปอร์แรนด์ (operand) 0 - 3 ไบต์

68 HC11 มีกระบวนการอ้างแอดเดรส 6 โหมด

การอ้างแอดเดรสแบบทันทีทันใด (Immediate addressing) เป็นการติดต่อเพื่อจัดการข้อมูลซึ่งเป็นค่าใด ๆ โดยตรงทันทีทันใด มีขนาดตั้งแต่ 2 - 4 ไบต์ ขึ้นอยู่กับขนาดของรีจิสเตอร์

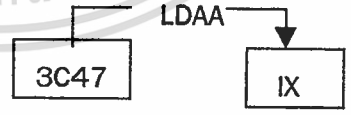
$ คือ ตัวอักษรที่ใช้บ่งบอกว่า ค่าที่ตามหลังนี้จะเป็นเลขฐาน 16

ตัวอย่าง

```
LDAA    #$5B
```

เป็นการนำค่าที่อยู่ในแรมแอดเดรสที่ $5C มาเก็บไว้ในแอกคิวมูเลเตอร์ A

```
LDAA    #$3C47
```



เป็นการนำค่าที่อยู่ในแรมแอดเดรสที่ $3C47 มาเก็บไว้ในรีจิสเตอร์อินเด็กซ์ X (IX) โดย $3C จะถูกเก็บไว้ในไบต์สูงของ IX ส่วน $47 จะเก็บไว้ในไบต์ต่ำของ IX

การอ้างแบบโดยตรง (Direct addressing) เป็นการติดต่อข้อมูลขนาด 8 บิตที่อยู่ในแรม โดยมีแอดเดรสตั้งแต่ $00-$FF ดังนั้นค่าโอเปอร์แรนด์ที่ตามหลังออปโค้ดคือ ค่าแอดเดรสของแรมนั่นเอง

ตัวอย่าง

```
LDAA    $6B
```

เป็นการนำค่าที่อยู่ในแรมแอดเดรสที่ $6B มาเก็บไว้ในแอกคิวมูเลเตอร์ A ไม่ใช่นำค่า $6B มาเก็บไว้ในแอกคิวมูเลเตอร์ A

การสร้างแอดเดรสแบบขยาย (Extended addressing)ในการสร้างแอดเดรสแบบนี้ ข้อมูลในไบต์ที่ 2 และ 3 ที่ตามหลังออปโค้ต จะเก็บค่าแอดเดรสจริงของหน่วยความจำที่ต้องการนำข้อมูลในหน่วยความจำออกมาประมวลผลหรือจัดเก็บข้อมูลลงไปใหม่ เมื่อใช้การอ้างแอดเดรสแบบนี้คำสั่งจะมีขนาด 3-4 ไบต์ โดยเป็นออปโค้ตไบต์ที่ 1 หรือ 2 (กรณีถ้ามีขนาด 4 ไบต์) ส่วน 2 ไบต์หลังจะเป็นค่าแอดเดรสที่อ้างถึงในหน่วยความจำ

ตัวอย่าง

DAB $000

เป็นการนำข้อมูลที่อยู่ในหน่วยความจำ แอดเดรส $4000 มาเก็บไว้ในแอกคิวมูเลเตอร์ B

การอ้างแอดเดรสแบบอินเด็กซ์ (Indexed addressing) การอ้างแอดเดรสแบบนี้จะมีการนำรีจิสเตอร์ชี้ข้อมูล (Index registert) ทั้ง 2 ตัว คือ IX และ IY มาใช้ในการคำนวณค่าแอดเดรสที่ต้องการเรียกใช้ข้อมูล ดังนั้นค่าแอดเดรสที่ต้องการใช้งานจะเปลี่ยนแปลง หรือขึ้นอยู่กับองค์ประกอบหลัก 2 ประการคือ

1. ค่าแอดเดรสที่ถูกกำหนดอยู่ในปัจจุบัน
2. ค่าออฟเซต 8 บิต ที่บรรจุอยู่ในคำสั่ง (หรือค่าโอเปอแรนด์)

ตัวอย่าง

STAA $05, X

สมมติว่า IX มีแอดเดรสก่อนหน้าที่จะเอ็กซิคิวต์คำสั่งนี้อยู่ที่ $2000 เมื่อเอ็กซิคิวต์คำสั่งในโปรแกรมนี้ จะเป็นการนำค่าที่อยู่ในแอกคิวมูเลเตอร์ A ไปเก็บลงในหน่วยความจำที่มีแอดเดรสเท่ากับ IX + 5 นั่นคือ $20000 + $5 = $2005

การอ้างแอดเดรสแบบอินเฮียเรนต์ (Inherent addressing) การอ้างแอดเดรสแบบนี้จะไม่ยุ่งเกี่ยวกับข้อมูลในหน่วยความจำแต่อย่างใด แต่จะเข้าในจัดการในรีจิสเตอร์แทน ดังนั้นขนาดของคำสั่งในการอ้างแอดเดรสแบบนี้จะมีเพียง 1-2 ไบต์ โดยเป็นออปโค้ตทั้งสิ้นไม่มีโอเปอแรนด์

ตัวอย่าง

CLRA : เป็นการเคลียร์ค่าในแอกคิวมูเลเตอร์ A

XGDY : สลับค่าในแอกคิวมูเลเตอร์ D กับรีจิสเตอร์ IY

DEX : ลดค่าของ IX

การอ้างแอดเดรสแบบสัมพันธ์ (Relative addressing) การอ้างแอดเดรสแบบนี้จะใช้ในชุดคำสั่งกระโดด (Jump and branch instructions) ถ้าหากเงื่อนไขในการกระโดดเป็นจริง ค่าออฟเซตขนาด 8 บิต ที่อยู่ตามหลังออปโค้ต ก็จะถูกบวกเข้าไปในรีจิสเตอร์โปรแกรมเคาน์เตอร์ (PC) เพื่อที่กำหนดแอดเดรสต่อไปที่จะข้ามไปทำงานของซีพียู ปกติแล้วขนาดของ

คำสั่งในการอ้างแอดเดรสแบบนี้จะเท่ากับ 2 ไบต์โดยไบต์แรกเป็นออปโค้ต ส่วนไบต์ที่สองเป็นค่าออฟเซตเพื่อบอกจำนวนที่จะเข้าไปเพิ่มใน PC

<u>การทดลอง</u>

1. ทำการป้อนโปรแกรมเข้าเครื่อง
2. ทำการทดลองโปรแกรม
3. แล้วทำการใส่ค่า OP-CODE ลงในตาราง
4. ให้อธิบายว่าแต่ละคำสั่งเป็นการอ้างแอดเดรสแบบใด

OP-CODE			
_____		LDAB	$2000
_____		LDAB	#$5A23
_____		LDAA	$50
_____		STAA	$05,X
_____		BEQ	JMP
_____	JMP	LDAA	#$05

<u>คำถาม</u>

1. จงเขียนโปรแกรมที่อ้างตำแหน่งโดยตรง ของข้อมูล
2. การอ้างตำแหน่งมีอะไรบ้าง จงอธิบาย

การทดลองที่ 4
การใช้คำสั่งในการโอนย้ายข้อมูลของ 68HC11

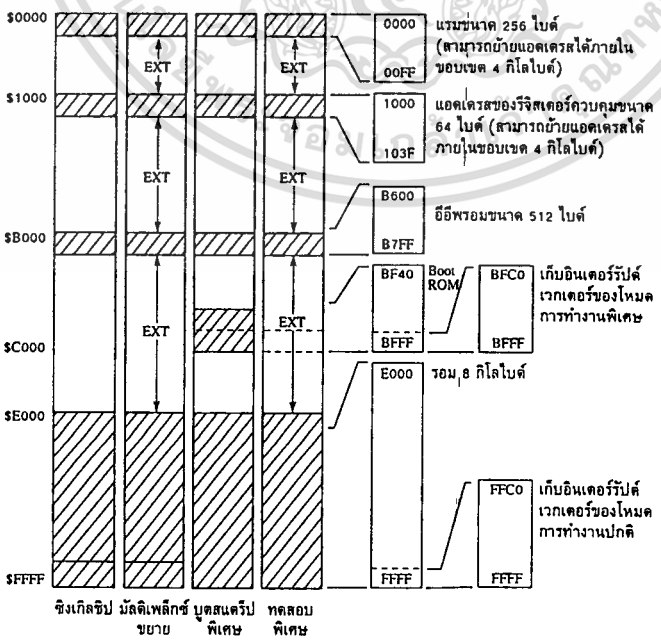
วัตถุประสงค์

1. เพื่อให้เข้าใจการโอนย้ายข้อมูลของ 68HC11
2. ให้เข้าใจลักษณะการใช้คำสั่งการโอนย้ายข้อมูลซึ่งมีอยู่หลายตำแหน่งแบบคำสั่ง
3. สามารถนำเอาคำสั่ง การโอนย้ายขอ้มูลของ 68HC11 ใช้งานในการเขียนโปรแกรม

<u>ทฤษฎี</u>

คำสั่งโอนย้ายข้อมูลเป็นคำสั่งสำคัญคำสั่งหนึ่งซึ่งใน 68HC11 ได้แบ่งคำสั่งในการโอนย้ายด้วยข้อมูลออกตามหน้าที่ที่จะทำการโอนย้ายข้อมูลระหว่าง Destination กับ Source และการโอนย้ายข้อมูลกับหน่วยความจำ

หน่วยความจำภายในของ 68HC11 จะมีเป็นชนิดใดขนาดเท่าใดขึ้นอยู่กับเวอร์ชั่นของ 68HC11 ในที่นี้จะกล่าวถึงเบอร์ MC68HC11A8 ซึ่งมีหน่วยความจำภายในค่อนข้างสมบูรณ์ คือมีหน่วยความจำทุกแบบ และมีขนาดที่แตกต่างกันออกไป คือมีรอมขนาด 8 กิโลไบต์ แรมขนาด 256 ไบต์ และอีอีพรอมขนาด 512 ไบต์ โดยมีการจัดหน่วยความจำดังรูป



รูปแสดงการจัดสรรหน่วยความจำของ MC68HC11A8 แบ่งตามโหมดการทำงาน

กลุ่มคำสั่งจัดการเกี่ยวกับข้อมูล (data handling instructions) แบ่งออกได้ 3 กลุ่มคือ

กลุ่มเคลื่อนย้ายข้อมูล กลุ่มคำสั่งชุดนี้แยกออกได้อีก 3 กลุ่ม คือ

- กลุ่มเคลื่อนย้ายข้อมูลระดับบิต มีคำสั่งดังนี้

BSET (Set bit) เป็นกลุ่มคำสั่งที่ใช้ในการทำให้บิตใด ๆ ของข้อมูลที่อ้างถึงเป็น "1" แล้วนำข้อมูลที่ทำการเซตบิตแล้วกลับเข้าไปเก็บในตำแหน่งเดิม รูปแบบการทำงานคือ M + mm -> M (M คือค่าของข้อมูลที่อ้างถึงในหน่วยความจำ) mm คือ บิตใด ๆ ที่ต้องการเซตส่วน เครื่องหมาย + หมายถึงการออร์ (OR)

BCLR (Clear bit) มีลักษณะตรงข้ามกับ BSET คือ ทำให้บิตใด ๆของข้อมูลที่อ้างถึงเป็น "0" มีรูปแบบการทำงานดังนี้ $M.\overline{mm}$ -> M (. คือการแอนด์ (AND))

- กลุ่มคำสั่งเคลื่อนย้ายข้อมูลระดับไบต์ มี 15 คำสั่ง

LDAA (Load Accumulator A) เป็นคำสั่งที่ใช้โหลดข้อมูลที่อ้างถึงมาเก็บไว้ในแอกคิวมูเลเตอร์ A

LDAB (Load Accumulator B) เป็นคำสั่งที่ใช้โหลดข้อมูลที่อ้างถึงมาเก็บไว้ในแอกคิวมูเลเตอร์ B

STAA (Store Accumulator A) เป็นคำสั่งที่ใช้ในการอ่านค่าจากแอกคิวมูเลเตอร์ A มาเก็บไว้ในหน่วยความจำ จะมีลักษณะการทำงานตรงข้ามกับ LDAA

STAB (Store Accumulator B) เหมือนกับ STAA แต่จะอ่านค่าจากแอกคิวมูเลเตอร์ B

TAB (Transfer B to A) เป็นคำสั่งในการถ่ายเทข้อมูลจากแอกคิวมูเลเตอร์ A ไป B

TBA (Transfer B to B) เป็นคำสั่งในการถ่ายเทข้อมูลจากแอกคิวมูเลเตอร์ B ไป A

TAP (Transfer A to CCR) เป็นคำสั่งในการถ่ายเทข้อมูลจากแอกคิวมูเลเตอร์ A ไปยังรีจิสเตอร์รหัสเงื่อนไข (CCR)

CCLRA และ CLRB (Clear Accumulator A & Clear Accumulator B) เป็นคำสั่งที่ใช้ในการเคลียร์ข้อมูลในแอกคิวมูเลเตอร์ A และ B

CLR (Clear memoryu byte) เป็นคำสั่งในการเคลียร์ข้อมูลในหน่วยความจำ 1 ไบต์ นั่นคือนำข้อมูล "0" เขียนลงในหน่วยความจำ

PSHA และ PSHB (Push A onto Stack & Push B onto Stack) เป็นคำสั่งที่ใช้เก็บค่าที่ข้อมูลในแอกคิวมูเลเตอร์ A หรือ B ไปไว้ในสแต็ก เมื่อทำคำสั่งนี้แล้วค่าของรีจิสเตอร์ชี้สแต็ก (SP) จะลดลง

PULA และ PULB (Pull A from stack & Pull B from stack) เป็นคำสั่งที่ใช้เก็บค่าที่ข้อมูลในแอกคิวมูเลเตอร์ A หรือ B ไปไว้ในสแต็ก กลับมาเก็บไว้ในแอกคิวมูเลเตอร์ A และ B อย่างเดิม ค่าของ SP จะเพิ่มขึ้นเมื่อทำคำสั่งนี้แล้ว

- กลุ่มคำสั่งเคลื่อนย้ายข้อมูลระดับเวิร์ด มีกลุ่มคำสั่ง 18 คำสั่งคือ

LDD (Load Double Accumulator D) เป็นการโหลดข้อมูล 16 บิต มาเก็บไว้ในแอกคิวมูเลเตอร์ D โดยไบต์ต่ำจะเก็บไว้ในแอกคิวมูเลเตอร์ A ส่วนไบต์สูงจะเก็บไว้ในแอกคิวมูเลเตอร์ B (แอกคิวมูเลเตอร์ D เกิดจากการรวมกันของ A และ B)

STD (Store Accumulator D) เป็นคำสั่งในการเก็บข้อมูลจาก D ไปไว้ในหน่วยความจำ โดยมีการแยกเก็บคือ ข้อมูลใน A จะถูกเก็บไว้ในหน่วยความจำแอดเดรสต่ำ ส่วนข้อมูลใน B จะเก็บไว้ในหน่วยความจำแอดเดรสถัดไป

XGDX และ XGDY (Exchange D with X & Exchange D with Y) เป็นคำสั่งในการเปลี่ยนข้อมูลระหว่างรีจิสเตอร์ข้อมูล IX และ IY กับแอกคิวมูเลเตอร์ D โดยXGDX เป็นการเปลี่ยนระหว่าง IX กับ D และ XGDY เป็นการเปลี่ยนระหว่าง IY กับ D

LDX และ LDY (Load Index Register X & Load Index Register Y) เป็นคำสั่งในการใช้โหลดข้อมูลขนาด 16 บิต จากหน่วยความจำ 2 แอดเดรส (แอดเดรสละ 8 บิต) มาเก็บไว้ใน IX และ IY

STX STY (Store Index Register X & Store Index Register Y) เป็นคำสั่งในการเก็บข้อมูลจาก IX และ IY ลงในหน่วยความจำ โดยไบต์ต่ำเก็บในแอดเดรสต่ำ ส่วนไบต์สูงเก็บในแอดเดรสถัดไป

LDS (Load Stack Pointer) เป็นคำสั่งใช้ในการกำหนดค่าแอดเดรสของสแต็กให้แก่รีจิสเตอร์ตัวชี้สแต็ก (SP)

STS (Store Stack Pointer) เป็นคำสั่งใช้ในการนำค่าของ SP เก็บลงในหน่วยความจำ

PSHX และ PSHY (Push X onto Stack & Push Y onto Stack) เป็นการเก็บค่า IX และ IY กลับมาคืนจากสแต็ก โดยข้อมูลไบต์สูงของ IX และ IY จะออกมาก่อน SP จะเพิ่มขึ้น 2 ค่า (SP=SP+2)

TSX และ TYS (Transfer Stack pointer to X & Transfer Stack pointer to Y) เป็นการนำค่าของตัวชี้สแต็กที่เพิ่มขึ้น 1ค่า (SP+1) มาเก็บไว้ใน IX หรือ IY

TXS และ TYS (Transfer X to Stack pointer & Transfer Y to Stack pointer) เป็นการนำค่าของ IX หรือ IY ที่ลดลง 1 แอดเดรส (IX-1 หรือ IY-1) ไปเก็บใน SP

<u>การทดลอง</u>

1. แปลงภาษาแอสเซมบลี้ให้เป็น op-code
2. ป้อนโปรแกรมเข้าเครื่อง ซึ่งเป็นโปรแกรมการทดลองใช้งาน

ADRESS	OP-CODE	ASSEMBLY
		LDD #$567A
		PSHA
		LDY #$05FC
		PSHY
		X6DX

3. สั่งให้เครื่องทำงาน โดยให้สังเกตุการทำงานของโปรแกรม แล้วบันทึกผลของโปรแกรม

B	A	IX	IY

<u>คำถาม</u>

1. คำสั่งการโอนย้ายข้อมูลระหว่างหน่วยความจำมีอะไรบ้าง

การทดลองที่ 5
การใช้คำสั่งทางคณิตศาสตร์ของ 68HC11

วัตถุประสงค์

1. เพื่อให้เข้าใจทางคณิตศาสตร์ของ 68HC11
2. ให้เข้าใจลักษณะการใช้คำสั่งทางคณิตศาสตร์ของ 68HC11
3. สามารถนำเอาคำสั่งทางคณิตศาสตร์ของ 68HC11 ใช้งานในการเขียนโปรแกรม

ทฤษฎี

ใน 68HC11 ได้มีคำสั่งเกี่ยวกับการทำงานทางคณิตศาสตร์ทางพื้นฐานสิ่งานด้วยกัน และจะใช้ ขนาดของข้อมูลในการคำนวณ 8 BIT ที่ไม่คิดเครื่องหมายเป็นตัวคำนวณโดยตรง อย่างไรก็ตามการใช้ แฟลก OV(overflow)ยังคงใช้งานในการบวก และลบเพื่อทำให้ข้อมูลที่เป็นตัวทางบวก และลบได้อยู่ทางคณิตศาสตร์ซึ่งคำสั่งทางคณิตศาสตร์เป็นคำสั่งที่ใช้ในการคำนวณ ได้แบ่งตามพื้นฐานการคำนวณออกเป็น 4 แบบด้วยกันคือ

1. พื้นฐานของการบวก

เป็นการนำเอาค่าของตัวตั้งไปบวกเข้ากับตัวบวกโดยในการบวกในของไมโครโปรเซสเซอร์ แล้วจะเป็นการบวกตัวเลขฐานสองเท่านั้น ซึ่งต่างจากการบวกแบบตัวเลขฐานสิบ คือในการบวกเลขฐานสอง จะบวกได้เฉพาะ"0"กับ"1"เท่านั้น ส่วนการบวกเลขฐานสิบนี้จะมีค่าที่จะทำการบวกได้ตั้งแต่ 0ถึง9 ดังตัวอย่างที่เปรียบเทียบการบวกเลขทั้งสองดังนี้

การบวกเลขฐานสิบ	การบวกเลขฐานสอง
81	1010
+31	+1011
112	1 0101

เราจะสังเกตดูความแตกต่างกันระหว่างการบวกเลขฐานสองจะเป็น1+1=10ซึ่งกฎของการบวกเลขฐานสองมีดังนี้

กฎของการบวกเลขฐานสอง

0+0=0

0+1=1

1+0=1

1+1=10

คำสั่งที่ใช้ในการบวกของ 68HC11จะใช้รีจิสเตอร์Aเป็นตัวตั้งเสมอ และ เมื่อทำการบวก ผลลัพธ์จะเก็บไว้ที่ รีจิสเตอร์ A เสมอ และเมื่อมีตัวทดก็จะไปเก็บไว้ที่ CY (carry) ซึ่งคำสั่งที่เกี่ยวกับการบวกมีดังนี้

กลุ่มคำสั่งการบวก มี 9 คำสั่ง

ADDA และ ADDB (Add memory to A & Add memory to B) เป็นคำสั่งใช้ในการบวกค่าของแอกคิวมูเลเตอร์ A หรือ B เข้ากับค่าข้อมูลในหน่วยความจำแล้วนำผลลัพธ์มาเก็บไว้ในแอกคิวมูเลเตอร์ A หรือ B การบวกข้อมูลทั้ง 2 คำสั่งนี้เป็นการบวกข้อมูลขนาด 8 บิต

ABA (Add Accumulators) คำสั่งนี้เป็นการบวกค่าของแอกคิวมูเลเตอร์ทั้ง 2 ตัวคือ A และ B เข้าด้วยกัน ผลลัพธ์จะเก็บไว้ที่แอกคิวมูเลเตอร์ A

ADCA และ ADCB (Add with Carry To A & Add with Carry To B) เป็นคำสั่งใช้บวกค่าของแอกคิวมูเลเตอร์กับค่าข้อมูลขนาด 8 บิต โดยมีการคิดทดด้วยผลลัพธ์ถูกนำมาเก็บไว้ที่แอกคิวมูเลเตอร์ A หรือ B

ADDD (Add 16 bit to D) เป็นคำสั่งใช้ในการบวกข้อมูลขนาด 16 บิต ระหว่างแอกคิวมูเลเตอร์ D กับหน่วยความจำ ผลลัพธ์เก็บไว้ที่ D

ADX และ ADY (Add B to X & Add B to Y) เป็นคำสั่งที่ใช้รวมค่าของแอกคิวมูเลเตอร์ B กับรีจิสเตอร์ IX และ IY โดย 8 บิตแรกของ IX และ IY จะถูกบวกด้วย 00 ส่วน 8 บิตหลังจะถูกรวมเข้ากับค่าของ B ผลลัพธ์เก็บไว้ที่ IX หรือ IY

DAA (Decimal Adjust A) เป็นคำสั่งใช้ในการปรับค่าในแอกคิวมูเลเตอร์ A ซึ่งไบนารี 8 บิตปกติเป็นเลขฐานสิบหรือรหัส BCD (Binary Code Decimal)

กลุ่มคำสั่งลบ มี 9 คำสั่ง

SUB A และ SSUB B (Subtractor memory from A & Subtractor memory from B) เป็นคำสั่งใช้ในการลบค่าในแอกคิวมูเลเตอร์ A หรือ B ด้วยค่าข้อมูลในหน่วยความจำ ผลลบถูกเก็บไว้ใน A หรือ B การลบในคำสั่งนี้เป็นการลบข้อมูลขนาด 8 บิต

SBA (Subtractor B from A) ใช้ในการลบค่าใน A ด้วย B ผลลัพธ์เก็บไว้ใน A

SBCA และ SBCB (Subtractor with Carry from A & Subtractor with Carry from B) เป็นการลบค่าใน A หรือ B ด้วยค่าข้อมูลในหน่วยความจำ โดยมีการคิดตัวทด (ตัวยืม)ด้วย ผลลัพธ์ถูกเก็บไว้ใน A หรือ B

SUBD (Subtractor memory from D) เป็นคำสั่งลบข้อมูล 16 บิต ระหว่างค่าในแอกคิวมูเลเตอร์ D กับในหน่วยความจำ ผลลัพธ์เก็บไว้ที่ D

NEGA และ NEGB (2's complement A & 2's complement B) เป็นคำสั่งที่ใช้การจัดการข้อมูลในแอกคิวมูเลเตอร์ A และ B ให้เป็นค่า 2's complement โดยนำค่า 00 ตั้งลบด้วยค่าใน A หรือ B แล้วผลลัพธ์ที่ได้จะเก็บไว้ใน A หรือ B

NEG (2's complement memory byte) เป็นการทำให้ค่าข้อมูลในหน่วยความจำเป็นตัวเลข 2's complement ด้วยการนำข้อมูลลบออกจาก 00 แล้วผลลัพธ์จะถูกนำมาเก็บไว้ในหน่วยความจำ

กลุ่มคำสั่งการคูณและการหาร มี 3 คำสั่ง

MUL (Multiply 8 by 8) เป็นคำสั่งคูณข้อมูลขนาด 8 บิต ระหว่างข้อมูลในแอกคิวมูเลเตอร์ A กับ B ผลคูณจะถูกเก็บไว้ในแอกคิวมูเลเตอร์ D

IDIV (Integer Divide 16 by 16) เป็นคำสั่งการหารเลขจำนวนเต็มขนาด 16 บิต โดยตัวตั้งจะถูกเก็บไว้ใน D ส่วนตัวหารจะถูกเก็บไว้ใน IX เศษหารจะถูกเก็บไว้ใน IX เศษของการหารเก็บไว้ใน D

FDIV (Fractional Divide 16 by 16) เป็นคำสั่งการหารเลขเศษส่วนขนาด 16 บิต โดยตัวตั้งเก็บไว้ใน D ส่วนตัวหารเก็บไว้ใน IX ผลหารจะถูกเก็บไว้ใน IX

คำสั่ง IDIV และ FDIV จะใช้งานร่วมกันเมื่อการหารเกิดเศษและต้องการหารค่าของเศษ

EORA EORB (Exclusive OR A with memory & Exclusive OR B Exclusive OR) เป็นคำสั่งในการเอ็กซ์คลูซีฟออร์ข้อมูลขนาด 8 บิต ระหว่างแอกคิวมูเลเตอร์ A หรือ B กับข้อมูลในหน่วยความจำ ผลนำมาเก็บไว้ใน A หรือ B

การทดลอง

```
ORG     $2000
LDAA    #$30
ADCA    #$50
SBCA    #$7F
STAA    $3500
END
```

โปแกรมที่ 1

```
ORG     $2100
LDD     #$2534
MUK
ADCA    #$05
END
```

โปแกรมที่ 2

```
ORG     $2000
LDD     #$1111
LDX     #$2222
IDIV
STX     $00
LDX     #$5555
FIDV
END
```

โปแกรมที่ 3

1. ทำการแปลงเป็น PO-CODE ของแต่ละโปรแกรม
2. ทำการทดลองแต่ละโปรแกรม
3. แล้วสังเกตุผลของแต่ละโปแกรม

การทดลองที่ 6
การใช้คำสั่งทางตรรกศาสตร์หรือคำสั่งทางลอจิก ของ 68HC11

วัตถุประสงค์

1. เพื่อให้เข้าใจคำสั่งทางตรรกศาสตร์หรือคำสั่งที่กระทำทางลอจิก ของ 68HC11
2. ให้เข้าใจลักษณะการใช้คำสั่งทางตรรกศาสตร์หรือคำสั่งทางลอจิก ของ 68HC11
3. สามารถนำเอาคำสั่งทางคณิตศาสตร์ของ 68HC11 ใช้งานในการเขียนโปรแกรม

<u>ทฤษฎี</u>

คำสั่งทางโลจิก หรือ คำสั่งทางตรรกศาสตร์ เป็นคำสั่งที่ทำงานทางพื้นฐานของ68 HC11หรือ ไมโครคอมพิวเตอร์ทั่วไป จะเอาข้อมูลมากระทำทางโลจิกโดยสามารถจะทำคำสั่งได้ทั้งไบต์และบิต โดยคำสั่งดังกล่าวมีดังต่อไปนี้

กลุ่มคำสั่งทางลอจิก (Logic Instruction) มีด้วยกัน 3 กระบวนการ คือ แอนด์ ออร์และเอ็กซ์คลูซีฟออร์ มีคำสั่งรวมทั้งสิ้น 6 คำสั่งดังนี้

ANDA และ ANDB (AND A with memory & AND B with memory)เป็นคำสั่งการแอนด์ข้อมูลขนาด 8 บิต ระหว่างแอกคิวมูเลเตอร์ A หรือ B กับค่าข้อมูลใด ๆ ผลของการแอนด์เก็บไว้ใน A หรือ B

ORAA และ ORAB (OR Accumulator A & OR Accumulator B) เป็นคำสั่งการออร์ข้อมูลขนาด 8 บิต ระหว่างแอกคิวมูเลเตอร์ A หรือ B กับข้อมูลในหน่วยความจำ ผลของการออร์ถูกนำมาเก็บไว้ใน A หรือ B

EORA EORB (Exclusive OR A with memory & Exclusive OR B Exclusive OR) เป็นคำสั่งในการเอ็กซ์คลูซีฟออร์ข้อมูลขนาด 8 บิต ระหว่างแอกคิวมูเลเตอร์ A หรือ B กับข้อมูลในหน่วยความจำ ผลนำมาเก็บไว้ใน A หรือ B

<u>การทดลอง</u>

แอดเดรส	OP-CODE	Assembly	ค่า
		ANDA #$33	A=
		ORAA #$0F	A=
		EORB A	B=
		ANDB #$50	B=
		LDD	D=

แอดเดรส	OP-CODE	Assembly	ค่า (อธิบาย 8 bit)
2000		LDAA #$82	A=
		LSLA	A=
		LDAB #$7A	B=
		LSRD	A=
			B=

1. ทำการเขียน OP-CODE และ แอดเดรส แล้วใส่ลงในตาราง
2. หาค่าของแต่ละแอกคิวมูเลเตอร์แล้วใส่ในตาราง

การทดลองที่ 7
การใช้คำสั่งการกระโดด ของ 68 HC11

วัตถุประสงค์

1. เพื่อเข้าใจในการใช้คำสั่งในการกระโดด และ การเก็บค่าแสตค
2. สามารถสร้างเงื่อนไขในการกระโดดได้
3. สามารถใช้คำสั่งในการเขียนโปรแกรมได้

ทฤษฎี

คำสั่งในการกระโดด เป็นคำสั่งที่จะทำการย้ายตำแหน่งการทำงาน หรือ ตำแหน่งของโปรแกรม โดยการเปลี่ยนค่าใน PC (Program Counter) ไปที่ตำหน่งของโปรแกรมที่จะย้ายไปทำงานในคำสั่งต่อไป คำสั่งในการกระโดดนี้จะแบ่งออกตามการใช้งานดังนี้

1. การกระโดดแบบไม่มีเงื่อนไข

JNP (Jump) เป็นคำสั่งที่ให้ซีพียูข้ามไปจัดการข้อมูล หรือคำสั่งในหน่วยความจำในตำแหน่งใด ๆ ที่ถูกกำหนดในโอเปอร์แรนด์ ค่า PC จะเท่ากับแอดเดรสปลายทางที่ระบุให้กระโดดไป

JSP (Jump to Sudroutine) เป็นคำสั่งที่ให้ซีพียูกระโดดไปทำงานยังโรแกรมย่อย เมื่อทำคำสั่งค่าของ PC เดิมจะถูกเก็บไว้ในสแต็ก จากนั้นค่า PC จะถูกเปลี่ยนเป็นค่าแอดเดรสที่ต้องการกระโดดไป จากนั้นซีพียูก็จะทำงานตามที่โปรแกร่มย่อยกำหนด จะกลับมายังโปรแกรมหลักได้ก็ต่อเมื่อพบคำสั่ง RTS

BRA (Branch Always) เป็นคำสั่งที่ให้ซีพียูข้ามไปทำงานที่แอดเดรสปลายทางที่ถูกระบุด้วยค่าสัมพัทธ์ (Relative:Rel) ที่อยู่ตามหลังคำสั่ง BRA นี้ มีลักษณะคล้าย ๆ คำสั่ง JR ของ Z80

BRN (Branch Never) เมื่อทำคำสั่งนี้ค่า PC จะเพิ่ม 2 ค่า (PC=PC+2) นั่นคือ เป็นการสั่งให้ซีพียูกระโดดข้ามไปทำงานที่แอดเดรสปลายทางที่ 2 แอดเดรสข้างหน้า

BSR (Branch to Subroutine) เป็นคำสั่งที่ให้ซีพียูกระโดดไปทำงานที่โปรแกรมย่อย โดยค่า PC เดิมจะถูกเก็บไว้ในสแต็กและค่า PC จะเปลี่ยนไปเป็นแอดเดรสที่กำหนดปลายทาง ซึ่งได้มาจากการคำนวณค่าสัมพัทธ์ และจะกลับมาโปรแกรมหลักเมื่อพบคำสั่ง RTS

RTS (Return from Subroutine) เมื่อพบคำสั่งนี้ซีพียูจะกระโดดกลับไปทำงานที่โปรแกรมหลัก โดยเรียกค่า PC เดิมที่ถูกเก็บไว้ในสแต็กออกมา

RTI (Return from Interrupt) เมื่อมีการอินเตอร์รัปต์ ซีพียูจะทำการเก็บค่าของรีจิสเตอร์รหัสเงื่อนไข (CCR) แอกคิวมูเลเตอร์ A และ B ค่า IX , IY และค่า PC ไว้ในสแต็กทั้งหมด

และเมื่อกระทำโปแกรมตอบสนองการอินเตอร์รัปต์เรียบร้อยแล้ว พบคำสั่งนี้ซีพียูจะทำการดึงค่าของCR,A,B,IX,IY และ PC คืนกลับมาจากสแต็กเพื่อทำงานปกติต่อไป

2. การกระโดดแบบมีเงื่อนไข สามารถแบ่งแยกย่อยได้อีก 4 กลุ่มย่อยดังนี้

- กลุ่มคำสั่งกระโดดแบบทดสอบบิตในไบต์ มีด้วยกัน 2 คำสั่งคือ BRSET และ BRCLR

BRSET (Branch if bit Set) เป็นคำสั่งที่ให้ซีพียูกระโดดไปทำงานยังแอดเดรสปลายทาง หลังจากที่มี

การตรวจสอบข้อมูลบิตใด ๆ ในหน่วยความจำแล้วพบว่าบิตที่ตรวจสอบนั้นเป็น"1" แต่ถ้าบิตที่ตรวจสอบนั้นเป็น "0" ก็จะไม่มีการกระโดด

BRCLR (Branch if bit Clear) เป็นคำสั่งที่มีลักษณะการทำงานตรงข้ามกับ BRSET คือ ซีพียูกระโดดไป

แอดเดรสปลายทางที่กำหนด เมื่อตรวจสอบบิตที่ต้องการแล้วพบว่าเป็น "0" ถ้าเป็น "1" ก็จะไม่มีการกระโดด

- กลุ่มคำสั่งกระโดดแบบทดสอบแฟล็ก มีด้วยกัน 8 คำสั่ง จากบิตเงื่อนไขของ CCR 4 บิต คือบิตตัวทด(Carry:C),บิตศูนย์ (Zero:Z), บิตลบ (Negative:N) และบิตเกิน (Overflow:V) มีรายละเอียดดังนี้

BCC (Branch if Carry Clear) เป็นคำสั่งที่ให้ซีพียูกระโดด เมื่อตรวจสอบบิตตัวทด (Carry:C) แล้วพบค่าเป็น "0"

BCS (Branch if Carry Set) ซีพียูจะกระโดดเมื่อตรวจสอบบิตตัวทดแล้วพบว่าเป็น "1"

BNE (Branch if Not Zero) เป็นคำสั่งให้ซีพียูกระโดดเมื่อตรวจสอบบิตศูนย์ (Zero) แล้วพบว่าเป็น"0"

BEQ (Branch if Zero) เป็นคำสั่งให้ซีพียูกระโดดเมื่อตรวจสอบบิตศูนย์แล้วพบว่าเป็น "1" นั่นคือ เกิดค่าศูนย์ในกระบวนการประมวบผลข้อมูล

BPL (Branch if Plus) เป็นคำสั่งให้ซีพียูกระโดดเมื่อตรวจสอบบิตลบแล้วพบว่าเป็น "0"นั่นคือ ค่าข้อมูลเป็นบวก

BMI (Branch if Minus) เป็นคำสั่งให้ซีพียูกระโดดเมื่อตรวจสอบบิตลบแล้วพบว่าเป็น "1" นั่นคือ เกิดเป็นค่าลบ

BVC (Branch if Overflow Clear) เป็นคำสั่งให้ซีพียูกระโดด เมื่อตรวจสอบบิตเกิน (Overflow) แล้วพบว่าเป็น "0" นั่นคือ ไม่เกิดค่าเกินย่านที่กำหนดในการประมวลผลข้อมูล (ค่าเกินคือ ค่าที่มากหรือน้อยกว่า +127 ถึง -128)

BVS (Branch if Overflow Set) เป็นคำสั่งให้ซีพียูกระโดด เมื่อตรวจสอบบิตเกิน แล้วพบว่าเป็น "1" นั่นคือ เกิดค่าเกินย่าน +127 ถึง -128 ในกระบวนการประมวลข้อมูล

- กลุ่มคำสั่งกระโดดแบบเปรียบเทียบกับข้อมูลศูนย์ ในกลุ่มคำสั่งนี้จะทำการเปรียบเทียบข้อมูลกับค่าศูนย์ว่ามากกว่า น้อยกว่าหรือเท่ากัน โดยมีการคำนึงถึงเครื่องหมายของข้อมูลด้วยว่าเป็นลบหรือบวก มีด้วยกันทั้งสิ้น 6 คำสั่ง ดังนี้

BLT (Branch if < zero)เมื่อกระทำคำสั่งนี้ซีพียูจะตรวจสอบการเอ็กซ์คลูซีฟออร์ของบิตลบ (N) กับบิตเกิน (V) ใน CCR ว่าเป็น "1" หรือไม่ ถ้าเป็นนั่นแสดงว่าข้อมูลที่ทำการเปรียบเทียบน้อยกว่า "0" ซีพียูจะทำการกระโดด

BLE (Branch if < zero) เป็นคำสั่งตรวจสอบข้อมูลว่าน้อยกว่าหรือเท่ากับศูนย์หรือไม่ ถ้าใช่ซีพียูก็จะกระโดดไปทำงานตามแอดเดรสที่กำหนดต่อไป

BEQ (Branch if = zero)นอกจากจะใช้คำสั่งนี้ในการกระโดดแบบทดสอบแฟล็กแล้วยังสามารถใช้โดยกลุ่มคำสั่งกระโดดแบบเปรียบเทียบ โดยคำนึงถึงเครื่องหมายได้ด้วย นั่นคือถ้าเปรียบเทียบข้อมูลแล้วพบว่า เท่ากับ "0" ซีพียูก็จะกระโดดไปทำงานตามแอดเดรสที่กำหนด

BEE (Branch if > zero)เป็นคำสั่งที่ให้ซีพียูกระโดดเมื่อผลการเปรียบเทียบออกมาแล้วมีค่ามากกว่าหรือเท่ากับ "0"

BGT (Branch if > zero) เป็นคำสั่งที่ให้ซีพียูกระโดดเมื่อผลการเปรียบเทียบมีค่ามากกว่า "0"

BNE (Branch if not = zero)เป็นคำสั่งให้ซีพียูกระโดดเมื่อเปรียบเทียบแล้วพบว่า ข้อมูลนั้นไม่ใช่ค่า "0"

- กลุ่มคำสั่งแบบเปรียบเทียบข้อมูบ 2 ข้อมูล เป็นกลุ่มคำสั่งที่ใช้กำหนดเงื่อนไขการกระโดด หลังจากเกิดการเปรียบเทียบข้อมูล 2 ข้อมูลว่ามากกว่า น้อยกว่า หรือเท่ากัน

BLO (Branch if Lower) เป็นคำสั่งให้ซีพียูกระโดดเมื่อผลการเปรียบเทียบข้อมูลปรากฏว่าข้อมูลหลักน้อยกว่า โดยบิตทดจะเซตเป็น "1"

BLS (Branch if Lower or Same) เป็นคำสั่งให้ซีพียูกระโดดเมื่อผลการเปรียบเทียบข้อมูล ปรากฏว่าข้อมูลหลักน้อยกว่าหรือเท่ากับข้อมูลที่นำมาเปรียบเทียบ โดยซีพียูจะตรวจสอบผลได้จากการเซตของบิตศูนย์หรือบิตทด

BEQ (Branch if = zero)เป็นคำสั่งให้ซีพียูกระโดดเมื่อข้อมูลที่นำมาเปรียบเทียบเท่ากันเท่านั้น โดยซีพียูจะตรวจสอบได้จากเซตของบิตศูนย์

BHS (Branch if Higher or Same)เป็นคำสั่งให้ซีพียูกระโดดเมื่อข้อมูลหลักมากกว่าหรือเท่ากับข้อมูลที่นำมาเปรียบเทียบ โดยซีพียูจะตรวจสอบผลจากการเคลียร์ของบิตทด (C=0)

BHI (Branch if Higher) เป็นคำสั่งให้ซีพียูกระโดดเมื่อข้อมูลหลักมากกว่าข้อมูลที่นำมาเปรียบเทียบโดยซีพียูจะตรวจสอบจากการออร์กันของบิตศูนย์และบิตทดต้องได้ศูนย์ (C+Z=0)

BNE (Branch if Not = zero)เป็นคำสั่งให้ซีพียูกระโดด หากเปรียบเทียบข้อมูลทั้งสองแล้วไม่เท่ากัน โดยซีพียูตรวจสอบได้จากบิตศูนย์ต้องเป็น "0" (Z=0)

ข้อสังเกตุ คำสั่ง BNE และ BEG สามารถนำไปใช้ในกลุ่มคำสั่งการกระโดดแบบมีเงื่อนไขได้ทุกแบบ ทั้งที่ผลของการทำงานก็เหมือนกัน ทั้งนี้เนื่องจากขอบเขตการทำงานของมันจะค่อนข้างกว้างมากนั่นเอง คำสั่ง BNE จะให้ซีพียูกระโดด เมื่อผลการเปรียบเทียบแล้วข้อมูลไม่เท่ากัน ไม่เท่ากับศูนย์ ในขณะที่คำสั่ง BEQ จะให้ซีพียูกระโดดเมื่อผลการเปรียบเทียบแล้วข้อมูลเท่ากับศูนย์ นี่คือคุณสมบัติใช้คำสั่งอย่างมีประสิทธิภาพของ 68HC11

<u>การทดลอง</u>

ADDRESS	OP-CODE	ASSEMBLY		ผล PC หลังRUN
		ORG $2000		
		LDAA #$50	Step 1	PC=
		LSLA	Step 2	PC=
		BCC XXX	Step 3	PC=
		HHH:LDX #$0011	Step 4	PC=
		DECA	Step 5	PC=
		RTS	Step 6	PC=
		XXX:ABA	Step 7	PC=
		ROLA	Step 8	PC=
		INCA	Step 9	PC=
		BSR HHH	Step 10	PC=
		END	Step 11	PC=

1. ทำการทดลองป้อนโปรแกรม
2. หา OP-CODE แล้วใส่ลงในตาราง
3. หาผลของ PC หลังจากการ RUN ทีละ Step

การทดลองที่ 8
การใช้งาน LCD MODULE

วัตถุประสงค์

1. เพื่อที่จะศึกษาอุปกรณ์แสดงผลแบบ LCD MODULE
2. สามารถเขียนโปรแกรมในการควบคุมการทำงานของ LCD MODULE
2. สามารถประยุกต์นำเอา LCD MODULE มาใช้งานในการแสดงผลของไมโครคอนโทรลเลอร์ 68HC11

ทฤษฎี

อุปกรณ์ในปัจจุบันในส่วนแสดงผลนั้นจะใช้ LCD เสียส่วนใหญ่ไม่ว่าจะเป็นเครื่องคิดเลข เครื่องถ่ายเอกสาร เครื่องมือวัดต่างๆ ไปจนถึง คอมพิวเตอร์สามารถแบ่งการแสดงผลแบบ Dot Matrix LCD ได้ 3 แบบด้วยกันคือ

1. Character LCD MODULE
2. Graphic LCD MODULE
3. Segment Display ชนิด LCD MODULE

โดยในแต่ละแบบนี้ก็จะมีส่วนประกอบใหญ่ๆ แบ่งได้เป็น

1. DOT MATRIX LCD เป็นตัวแสดงผลให้เราได้มองเห็นในลักษณะ การปิด และเปิดตัวเอง กับ แสงภายนอก ส่วนของที่เป็นตัวกระจกบรรจุผลึก

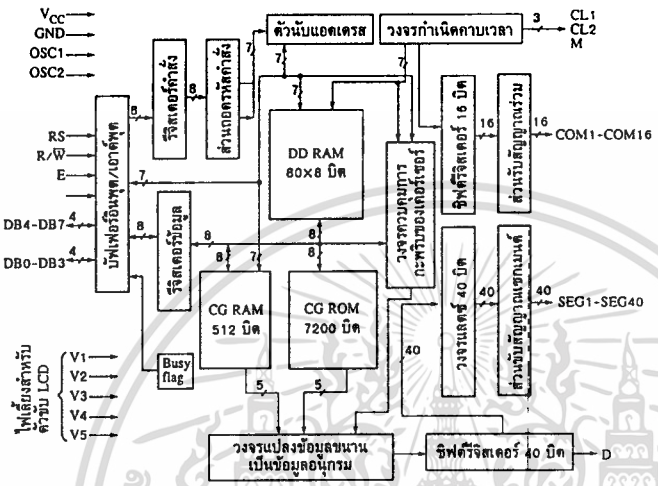
2. DRIVER เป็นตัวรับสัญญาณจากตัวควบคุมมาขับผลึก LCD อีกทีหนึ่ง โดยมีเบอร์ที่นิยมใช้ใน LCD Module เช่น HD44100H, MSM5259

3. CONTROLLER เป็นตัวรับข้อมูลจากตัวอุปกรณ์ภายนอกมาและจัดการควบคุม LCD Module ให้ทำงานแสดงผลต่างๆ เช่น การลบจอภาพ, การเกิดตัวอักษร เป็นต้น โดยมีเบอร์ที่นิยมใช้กันคือ HD44780 ซึ่งจะใช้ในแบบ Character Lcd Module เป็นส่วนใหญ่ และเบอร์ HD61830 จะใช้ในแบบ Graphic LCD Module

การควบคุมตัว Controller ของ LCD นั้นต้องเข้าใจหลักการทำงานของ Controller ก่อน ซึ่ง LCD MODULE ของแต่ละบริษัทจะใช้ตัว Controller ที่มีหลักการทำงานเหมือนๆ กันเป็นส่วนใหญ่และเป็น LCD MODULE แต่ละขนาดจำนวนตัวอักษรหรือจำนวนบรรทัดก็มีหลักการทำงานของแบบเดียวกันทั้งหมด IC ที่นิยมมากที่สุดตัวหนึ่งที่เป็น Controller LCD ก็คือเบอร์ HD44780 โดยรูปแบบการทำงานของมันได้เป็นมาตรฐานให้กับ Controller LCD ตัวอื่นๆ ด้วย

HD44780 เป็นไอซี LSI ตัวหนึ่งใช้ควบคุม LCD โดยแสดงผลในรูปตัวอักษรหรือสัญลักษณ์ต่างๆ ด้วยกันเองสามารถต่อใช้งานได้ทั้งแบบ 4 Bit และ 8 Bit โดยถ้าเราต่อแบบ 4

Bit จะต่อใช้งานที่ DB7-DB4 เท่านั้นโดยข้อมูลครั้งแรกที่ส่งนั้น HD44780 จะถือเป็นข้อมูล 4 Bit ล่าง จากรูปที่1 จะแสดงโครงสร้าง และการทำงานภายในตัว Controller และ Timing Diagram ในการรับข้อมูลของตัว Controller



รูปที่ 1 บล็อกไดอะแกรมการทำงานของชิปควบคุม LCD เบอร์ HD44780

จากรูปที่ 1 จะเห็นว่าในโครงสร้างจะเห็นว่าโครงสร้างภายในของ Controller จะประกอบหลายส่วนด้วยกัน ได้แก่

I/O Buffer เป็นวงจรในการติดต่อข้อมูลจากภายนอกเพื่อที่จะส่งข้อมูลเข้าไปยังภายในตัว Controller

Instruction Register (IR) เป็นรีจิสเตอร์ที่จะรับคำสั่งจากภายนอกเพื่อที่จะควบคุมการแสดงผลเช่น การจัดการแสดงผล, การ Cler จอ, กำหนดตำแหน่งของการแสดงผล เป็นต้น

Data Register (DR) เป็นรีจิสเตอร์ที่จะใช้รับข้อมูลจากภายนอกส่งไปให้ DD RAM หรือสร้างตัวอักษรเพิ่มเติมให้กับ CG RAM

Display Data RAM (DD RAM) เป็นหน่วยความจำที่สามารถเขียนข้อมูลเข้าไปได้โดยผ่าน Data Register (DR) ซึ่งตัว Controller จะนำข้อมูลที่อยู่ใน DD RAM ไปชี้ค่าที่ตารางหรือตำแหน่งของ CG ROM หรือ CG RAM เพื่อจะนำไปแสดงที่หน้าจอ LCD

Character Generater ROM (CG RAM) เป็นหน่วยความจำที่ถาวรใช้เก็บ Character ของตัวอักษรหรือสัญลักษณ์ที่จะนำไปแสดงที่หน้าจอ LCD และยังสามารถสร้าง Font ขึ้นมาใช้เองได้โดยการเขียนข้อมูลลงไปใน CG Ram และชี้ Font ได้โดยการกำหนดค่าที่ DD RAM

Busy Flag เป็นตัวบอกสถานะการทำงานของ Controller ให้อุปกรณ์ภายนอกทราบว่า Controller พร้อมที่จะรับข้อมูล หรือ คำสั่งที่จะทำการส่งไปให้ตัว Controller หรือยัง โดยให้อุปกรณ์ภายนอกทำการอ่านสัญญาณ ตัวนี้งานเสร็จหรือยัง

คำสั่งของชิปควบคุม LCD เบอร์ HD44780 มีทั้งหมด 9 คำสั่ง

1. Display Clear จะทำการลบข้อมูล
2. Function Set โดยจะ SET ค่าภายใน

   DL = 1 เป็นการ Set ให้การติดต่อเป็นแบบ 8 Bit

   N = 0 Set เป็น 1 บรรทัดการแสดงผล

   F = 0 5*7 Dot ต่อหนึ่งตัวอักษร
3. Display ON/OFF

   D = Display OFF

   C = 0 Cursor OFF

   B = 0 Blink OFF
4. Entry Mode Set

   I/D = 1 (เพิ่มค่า Counter ขึ้น 1)

   S = 0 NO Shift
5. Return home ต้องการให้บิต1 ของข้อมูลเป็น "1"(ให้เคอร์เซอร์เคลื่อนที่กลับไปยังตำแหน่งซ้ายสุดของจอแสดงผล
6. คำสั่งควบคุมการเลื่อนเคอร์เซอร์และข้อมูลตัวอักษร

บิต S/C	บิต R/L	
0	0	เลื่อนเคอร์เซอร์จากตำแหน่งเดิมไปทางซ้าย 1 ตำแหน่ง
0	1	เลื่อนเคอร์เซอร์จากตำแหน่งเดิมไปทางขวา 1 ตำแหน่ง
1	0	เลื่อนตัวอักษรที่เกิดขึ้นใหม่ไปทางซ้าย
0	1	เลื่อนตัวอักษรที่เกิดขึ้นใหม่ไปทางขวา

7. คำสั่งเลือกแอดเดรสของ CGRAM

บิต7 เป็น "0"

บิต6 เป็น"1" ส่วนอีก 6 บิตจะแทนด้วยค่าของแอดเดรสของ CGRAM

8. การเลือกแอดเดรสของ DDRAM

บิต7 เป็น "1"

ส่วนอีกส่วนที่เหลือจะแทนด้วยค่าของแอดเดรสของ CGRAM

9. คำสั่งอ่านแฟล็ก busy และแอดเดรส

บิต BF เป็น "0"พร้อมรับข้อมูล

บิต BF เป็น "1"ยังไม่พร้อมรับข้อมูล

รายละเอียดของคำสั่ง HD44780

Clear Display

บิต 7	บิต 6	บิต 5	---	---	---	---	---	---	บิต 0
0	0	0	0	0	0	0	0	0	1

เป็นคำสั่งที่จะเขียนช่องว่าง หรือ Space (20h) เข้าไปใน DD RAM ทั้งหมด และทำการ Set DD RAM Address เป็น 0 ตัว Cursor จะกลับไปอยู่ที่ตำแหน่งบนสุดทางซ้ายมือของจอภาพ และทำการ Set I/O = 1, S ไม่มีการเปลี่ยนแปลง

Return Home

บิต 7	บิต 6	บิต 5	---	---	---	---	---	---	บิต 0
0	0	0	0	0	0	0	0	1	*

เป็นคำสั่งที่จะทำการ Set DD RAM Address เป็น 0 ตัว Cursor จะกลับไปอยู่ตำแหน่งบนสุดซ้ายมือของจอภาพ ข้อมูลในจอภาพไม่เปลี่ยนแปลง

Entry Mode Set

บิต 7	บิต 6	บิต 5	---	---	---	---	---	---	บิต 0
0	0	0	0	0	0	0	0	I/D	S

BIT I/D โดยเป็นตัวกำหนดให้ว่าเมื่อเขียนหรืออ่านข้อมูลแล้วจะทำให้ DD RAM Address เพิ่มขึ้นหนึ่งลดลงหนึ่งโดย 1 = เพิ่ม, 0 = ลดลงหนึ่ง

Bit S เป็นตัวกำหนดการแสดงผลโดยถ้า S = 1 จะเป็นการใส่ข้อมูลแล้วตัว Cursor อยู่กับที่ข้อมูลจะถูกดันไปทางซ้าย ถ้า S = 0 ข้อมูลจะอยู่ที่ตัว Cursor จะถูกดันไปทางขวามือ

Display on/off Control

บิต 7	บิต 6	บิต 5	---	---	---	---	---	---	บิต 0
0	0	0	0	0	0	0	D	C	S

D เป็น Bit ให้เปิดปิดหน้าจอภายในโดยถ้า D =1 จะ ON และ D = 0 จะ OFF

C จะให้แสดง Cursor ให้ Bit C = 1 และถ้าไม่ต้องการแสดง Cursor จะอยู่ที่ Line ที่ 8 ในแบบ 5*7 Dot และจะอยู่ Line ที่ 11 ในแบบ 5*10 Dot

B เป็น Bit Set การกระพริบของ Cursor โดย B = 1 มีการกระพริบ B = o ไม่มีการกระพริบ โดยมีระยะเวลาการกระพริบประมาณ 379.2 mS

Cursor or Display Shift

บิต 7	บิต 6	บิต 5	---	---	---	---	---	---	บิต 0
0	0	0	0	0	0	S/C	R/L	*	*

S/C	R/L	
0	0	ทำการย้าย Cursor ไปจากตำแหน่งเดินไปซ้ายมือ 1 ตำแหน่ง
0	1	ทำการย้าย Cursor ไปจากตำแหน่งเดินไปขวามือ 1 ตำแหน่ง
1	0	เป็นการดันตัวอักษรที่เกิดไปทางซ้าย
1	1	เป็นการดันต้ออักษรที่เกิดไปทางขวา

Function Set

บิต 7	บิต 6	บิต 5	---	---	---	---	---	---	บิต 0
0	0	0	0	1	DL	N	F	*	*

DL เป็นการ SET การติดต่อว่าจะให้เป็นแบบ 8 Bit หรือ 4 Bit โดยถ้าต้องการติดต่อ 4 Bit DL = 0 และ 8 Bit DL = 1

N เป็นการ SET บรรทัดของการแสดงผล N =0 แสดง 1 บรรทัด N =1 แสดง 2 บรรทัด ในกรณีมากกว่าก็ให้ SET N =1

F เป็นการ SET ขนาด DOT การแสดงผล 5*7 หรือ 5*10 โดย F = 0 เป็น

Set CG RAM Address

บิต 7	บิต 6	บิต 5	---	---	---	---	---	---	บิต 0
0	0	0	1	A	A	A	A	A	A

<--High order bit Low order bit-->

ใน HD44750 นั้นจะมีหน่วยความจำอยู่ 2 ชุด คือ Display DATA RAM (DO RAM) จำนวน 80*80 Bit และ Character Generater ROM CG RAM จำนวน 512 Bit และ 7200 Bit คำสั่งนี้เป็นการ Set Address ใน CG RAM โดยต้องทำการ Set Address ก่อน เขียนหรืออ่านข้อมูล CG RAM ด้วย

Set DD RAM Address

บิต 7	บิต 6	บิต 5	---	---	---	---	---	---	บิต 0
0	0	1	A	A	A	A	A	A	A

<--High order bit Low order bit-->

เป็นคำสั่ง SET ค่า Address ใน DD RAM ในการเขียนหรืออ่านค่าจาก DD RAM (DD RAM คือส่วนที่จะแสดงผลหน้าจอ LCD) โดยจำนวน Address ที่จะเกิดขึ้นบนจอ LCD จะอยู่กับ SET ค่า N ด้วย

ถ้า N = 0 (1 บรรทัด) Address จะอยู่ 00H-4FH

ถ้า N = 1 (2 บรรทัด) Address จะอยู่ 00H-27H สำหรับบรรทัดที่ 1 และ 40H-67H สำหรับบรรทัดที่ 2

แบบการจัด Address ของ DD RAM หน้าจอ LCD แบบ 16 ตัวอักษร 1 บรรทัด

0	1	2	3	4	5	6	7	40	41	42	43	44	45	46	47

Read Busy Falh and Address

บิต 7	บิต 6	บิต 5	---	---	---	---	---	---	บิต 0
BF	A	A	A	A	A	A	A	A	A

<--High order bit Low order bit-->

เป็นคำสั่งอ่านค่า Busy Flag ซึ่งจะเป็นตัวบอกว่าตัว HD44780 นี้อยู่ในขบวนการทำงานภายในอยู่ในสภาพพร้อมจะรับข้อมูลโดย

BF = 1 อยู่ในขบวนการทำงานภายในไม่พร้อมจะรับข้อมูลหรือคำสั่ง

BF = 0 พร้อมจะรับข้อมูลหรือคำสั่งได้

และนอกจากนี้ยังเป็นคำสั่งอ่านค่าข้อมูล Address ของ CG RAM หรือ DD RAM ด้วย

Write Data To CG หรือ DD RAM

บิต 7	บิต 6	บิต 5	---	---	---	---	---	---	บิต 0
1	0	D	D	D	D	D	D	D	D

<--High order bit Low order bit-->

เป็นคำสั่งเขียนข้อมูลเข้าไปใน CG หรือ DD RAM โดยเมื่อเขียนข้อมูล และ Address จะเพิ่มหรือ ลด โดยอัตโนมัติตามคำสั่งที่ SET ใน Entry Mode ข้อกำหนดที่จะรู้ว่าเป็นการเขียนข้อมูลของ CG RAM หรือ DD Ram ทำได้โดยการ SET Address ของ CG RAM หรือ DD RAM ขึ้นมาก่อนจะเขียนข้อมูล

Read Data From CG หรือ DD RAM

บิต 7	บิต 6	บิต 5	---	---	---	---	---	---	บิต 0
1	1	D	D	D	D	D	D	D	D

<--High order bit Low order bit-->

เป็นคำสั่งอ่านข้อมูลจาก CG RAM หรือ SET Address ก่อนเพื่อให้รู้ว่าข้อมูลที่อ่านได้นั้น DD RAM หรือ CG RAM

การต่อ Dot Matrix LCD Module เข้ากับ Micro Controller MCS-51 นั้น จำเป็นต้องผ่าน Port ก่อนโดยทำการจำลองสัญญาณในการส่งข้อมูลที่จะไปควบคุมการทำงาน Conrtoller ของ LCD Module ใน Dot Matrix LCD Module มีขาสัญญาณในการต่อใช้งานดังนี้

RS (Registor Selection) จะเป็นขาเลือกรีจิสเตอร์ภายในตัวไอซีซึ่งมีอยู่ 2 ตัวด้วยกัน คือ Instruction Registor (IR) และ Data Registor (DR) โดยถ้าเป็น "0" เป็นการเลือกรีจิสเตอร์ของคำสั่ง "1" เป็นการเลือกรีจิสเตอร์ของข้อมูล

R/W (Read/Write) เป็นตัวเลือกว่าจะเขียนข้อมูลหรืออ่านข้อมูลจากตัวไอซีโดยถ้าเป็น "1" จะอ่านข้อมูล และ "0" เป็นการเขียนข้อมูลเข้าตัวไอซี

E (Enable Signal) เป็นขากำหนดสภาพการรับการเขียน และอ่านข้อมูล

DB0-DB7 เป็นขาที่ใช้รับข้อมูล และคำสั่งจากภายนอก

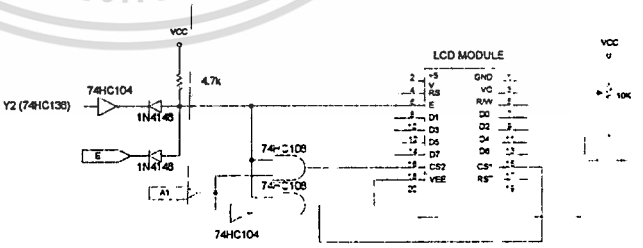
VDD เป็นขาไปเลี้ยงตัววงจร

VSS เป็นขา GND

VO เป็นขาที่ใช้ปรับแรงดันขับในการแสดงของ LCD สว่างหรือมืด

<u>อุปกรณ์</u>

1. บอร์ด 68HC11



<u>การทดลอง</u>

1. ป้อนโปรแกรมเข้าเครื่อง แล้วสั่งให้ทำงานโปรแกรม
2. แล้วอธิบายโปรแกรม

```
;TEST LCD PROGRAM

                ORG     $2000
CTRL_DATA       EQU     $240
CHA_DATA        EQU     $242
```

```
CHA_DATA            EQU     $242
BUSY_DATA           EQU     $244
**INTIAL SYSTEM**
START               LDD     #$00
START1              DECA
                    CMPA    #$00
                    BNE     START1
STAR2               DECB
                    CMPB    #$00
                    BNE     STAR2
                    LDS     #$5000
                    JSR     INNIT_LCCD
                    LDX     #TABLE
                    JSR     WRITE_PORT
STTOP               BRA     STOP
TABLE               FCC     "HC11_CONTROLLER"
;**INITIAL LCD**
INIT_LCD            LDAA    #$38
                    STAA    CTRL_DATA
                    JSR     DELAY
                    JSR     DELAY
                    LDAA    #$0F
                    STAA    CTRL_DATA
                    JSR     DELAY
                    LDAA    #$06
                    STAA    CTRL_DATA
                    JSR     READ_BUSY
                    RTS
;** WRITTE DATA  1 LINE**
WRITE_PORT          LDDA    #$00
                    JSR     GOTO_ADDR
                    JSR     WRUTE_OUT
                    LDAA    #$40
```

```
                    JSR         GOTO_ADDR
                    JSR         WRUTE_OUT
                    RTS
;**WRUTE DATA OUT TO LCD**
WRITE_OUT     LDAB      #$08
TEST1               LDAA        $0,,X
                    JSR         WRITE_BBYTE
                    INX
                    DECB
                    BNE         TEST1
                    RTS
 ;**WRUTE DATA BYTE**
WRITE_BYTE          STAA        CHA_DATA
                    JSR         READ_BBUSY
                    RTS
;**READ BUSY BIT**
READ_BUSY           LDAA        BURY_DATA
                    BRSET       BUSY_DATA,$80READ_BUSY
                    RTS
;**DELAY SUBROUTINE**
DELAY               PSHB
                    PSHA
                    LDAB        #$00
DELAY1              DECB
                    BNE         DELAY1
                    PULA
                    PULB
                    RTS
;**GOTO ADDRESS LCD**
GOTO_ADDR           STAA        DATA_BUF
                    BSET        7,DATA_BUF
                    LDAA        DATA_BUF
                    LDAA        DATA_BUF
```

```
JSR         READ_BUSY
RTS
END
```

<u>คำถาม</u>

1. ให้เขียนคำสั่งของชิปควบคุม LCD เบอร์ HD4478 มีคำสั่งอะไรบ้าง จงอธิบาย ?

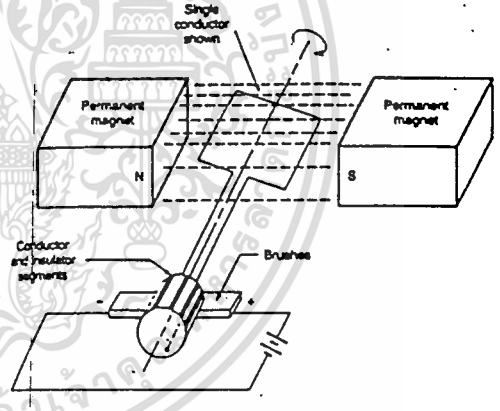
การทดลองที่ 9
การใช้งาน สเต็ปปิ้งมอเตอร์ (STEPPING MORTOR)

วัตถุประสงค์

1. เพื่อศึกษาการทำงานของ สเต็ปปิ้งมอเตอร์ (Stepping Motor)
2. สามารถเขียนโปรแกรมในการควบคุมการทำงานของ สเต็ปปิ้งมอเตอร์ (STEPPING MOTOR)

ทฤษฎี

Stepping Motor คือ Motor ชนิดหนึ่งแต่สามารถควบคุมการหมุนเป็น Step โดยใช้ Pulse เป็นตัวควบคุมการหมุน ภายในจะประกอบด้วย โรเตอร์เป็นเหล็กอ่อน ซึ่งมีคุณสมบัติพยายามปรับตัวเองให้อยู่ในแนวที่เส้นแรงแม่เหล็กผ่านมากที่สุด โดยการหมุนตัวเองทำให้เกิดมุมของการหมุนตัวเองทำให้เกิดมุมของการหมุนขึ้น และมอเตอร์จะหยุดหมุนเมื่อเส้นแรงแม่เหล็กที่ตัดผ่านถึงจุดที่มากที่สุด ซึ่งจะแสดงในรูป



รูปเส้นแรงแม่เหล็กที่ทำให้เกิดแรงบิด

การทำให้ Stepping Motor หมุนได้โดยอาศัยการทำให้เกิดเส้นแรงแม่เหล็กเกิดขึ้นเป็นช่วงต่อกันไปเรื่อย การหมุนของ Stepping Motor จะขึ้นอยู่กับการขับกระแสเข้าขดลวดว่าจะให้ไปในทางไหน และเมื่อต้องการให้มอเตอร์หยุดก็หยุดการจ่ายกระแสให้กับขดลวด Stepping Motor ก็จะหยุดหมุน เราก็สามารถรู้ตำแหน่งของมอเตอร์ได้ โดยการนับจำนวน Piuse ที่ป้อนให้กับตัว Stepping Motor โดยคำนวณหาตำแหน่งของ Stepping Motor ตามสูตร

มุมที่เปลี่ยนไป = ค่ามุมต่อสเต็บ * จำนวน Pulse ที่ป้อนให้

Stepping Motor โดยทั่วไปมีอยู่ 3 ชนิดคือ

1. Variable reluctance(VR) เป็น Stepping Motor ที่ถูกกล่าวถึงและนำไปใช้งานมากที่สุด โดยเฉพาะแบบ 4 เฟส โรเตอร์และสเตเตอร์จะทำมาจากเปลือกผสมซิลิคอน เพลาของ

มอเตอร์จะหมุนไปเป็นค่ามุมคงที่ หรือเรียกว่า มุมสเต็ป มักจะมีมุมที่ต่างกันคือ 0.72, 0.9, 1.8, 2.0, 3.6, 7.5, 15 สำหรับ Stepping Motor ชนิดนี้ส่วนใหญ่จะใช้วงจรขับแบบ unipolar ซึ่งลักษณะการทำงานของวงจรจะต้องป้อนค่า voltage เข้าไปในแต่ละขั้วตามลักษณะของ Pulse ที่ใช้ขับชุด driver ค่า voltage ที่จ่ายให้มอเตอร์จะใช้อ้างอิงกับกราวด์เสมอ

2. *Permanent Megnet (PM)* เป็นแบบที่ตัวโรเตอร์ทำด้วยแม่เหล็กถาวร มีลักษณะการหมุนแบบ bipolar ต้องจ่ายไฟบวกลบเข้าที่แต่ละเฟสของมอร์เตอร์โดยตรง

3. *Hybrid Stepping Motor (HSM)* เป็นแบบผสมระหว่าง (PM) และ (VR) จำนวนซี่ฟันของโรเตอร์และสเตเตอร์จะไม่เท่ากัน มีการทำงานที่ซับซ้อน จึงไม่นิยมใช้กันมากนัก

การกระตุ้นเฟสของขดลวด Stepping Motor

ดังที่รู้กันว่า ในการทำให้ Stepping Motor หมุนได้นั้น จะต้องกระตุ้นเฟสของขดลวด Stepping Motor ให้เรียงกันไปเรื่อยๆ ทางใดทางหนึ่ง ถ้าต้องการให้หมุนกลับก็กระตุ้น เฟสในทิศทางกลับกัน ซึ่งการกระตุ้นเฟสของ Stepping Motor มีอยู่ด้วยกัน 3 แบบคือ

1.การกระตุ้นเฟสเดียวเรียกว่า Single Phase Excitation เป็นการป้อนสัญญาณให้กับ Stepping Motor ที่จะขดจะแสดงในรูป

PHASE	PULSE							
A	1	0	0	0	1	0	0	0
B	0	1	0	0	0	1	0	0
C	0	0	1	0	0	0	1	0
D	0	0	0	1	0	0	0	1

3. การกระตุ้นสองเฟสเรียกว่าแบบ Two Phase Excitation เป็นการป้อนสัญญาณให้กับ Sstepping Motor ที่จะขดจะแสดงในรูป

PHASE	PULSE							
A	1	0	0	1	1	0	0	1
B	1	1	0	0	1	1	0	0
C	0	1	1	0	0	1	1	0
D	0	0	1	1	0	0	1	1

3. การกระตุ้นแบบ 1 และ 2 สลับกันเรียกว่า One- Two- Phase Excitation หรือแบบ Half Step Operation เป็นการป้อนสัญญาณให้กับ Stepping Motor ที่จะขดจะแสดงในรูป

PHASE	PULSE							
A	1	1	0	0	0	0	0	1
B	0	1	1	1	0	0	0	0
C	0	0	1	1	1	1	0	0
D	0	0	0	0	0	1	1	1

การกระตุ้นเฟสของแต่ละแบบจะเห็นว่าเป็นการเลื่อนข้อมูลในระบบไมโครนี้เอง โดยที่เราส่งข้อมูลที่เลื่อนนั้นออกผ่านทาง พอร์ต เอาท์พุต แล้วผ่านวงจรในการขับกระแสก่อน สัญญาณที่ได้จากการเลื่อนข้อมูลนั้นจะทำให้เกิดการหมุนของ Stepping Motor โดยการเลื่อนของแต่ละแบบนั้นจะเลื่อนข้อมูลต่างกัน เท่านั้นเราสามารถเปลี่ยนทิศทางในการหมุนของ Stepping Motor ได้โดยการเปลี่ยนทิศทางในการหมุนของข้อมูล

จากรูปที่ 3,4,5 เราสามารถนำมาเขียนโปรแกรมได้ โดยการนำเอาข้อมูลมาทำการ Shift โดยเราเปลี่ยนทิศทางในการหมุนของ Stepping Motor โดยการเปลี่ยนทิศทางในการ Shift ของข้อมูลในโปรแกรม เมื่อนำข้อมูลมาทำการ Shift แล้วก็ส่งข้อมูลนี้ออกพอร์ต เพื่อไปส่งสัญญาณให้วงจร Driver กระตุ้นการทำงานของ Stepping Motor

ตัวอย่างของการ Shift เพื่อควบคุมการทำงานของ Stepping Motor ในการหมุนแบบการกระตุ้นแบบเฟสเดียว Single Phase Excitation

DATA = 0001 0001
RR1 DATA = 1000 1000
RR2 DATA = 0100 0100
RR3 DATA = 0010 0010
RR4 DATA = 0001 0001

<u>อุปกรณ์</u>

1. Board 68HC11
2. แผงวงจรอินเตอร์เฟสกับ Stepping Motor 1
3. Stepping Motor 1
4. สายต่อวงจร และ แหล่งจ่ายไฟ

<u>การทดลอง</u>

1. ทดลองป้อนโปรแกรมการควบคุมสเต็ปปิ้งมอเตอร์
2. ทำการป้อนโปรแกรมเข้าเครื่อง

3. สังเกตุการทดลอง

```
;**CTRL STEPPER MOTOR
                ORG     $2000
PORTD           EQU     $1008
D_DIRECT        EQU     $1009
                LDAA    #$FF
                STAA    D_DIRECT
                LDAA    #$11
START           ROLA
                STAA    PORTD
                JSR     DELAY
                JMP     START
;**DELAY
DELAY           PSHA
                LDAA    #$00
                LDAB    #$00
DELAY1          DECA
                CMPA    #$00
                BNE     DELAY1
DELAY2          DECB
                BNE     DELAY2
                PULA
                RTS
                END
```

<u>คำถาม</u>

1. Stepping Motor โดยทั่วไปมีกี่ชนิด อะไรบ้าง จงอธิบาย ?
2. การกระตุ้นเฟสมีกี่แบบ อะไรบ้าง จงอธิบาย ?

การทดลองที่ 10
การใช้งาน SERIAL PORT RS-232

<u>วัตถุประสงค์</u>

1. เพื่อเข้าใจในการใช้งาน SERIAL PORT RS-232

<u>ทฤษฎี</u>

การเชื่อมต่อ 68 HC11 เข้ากับพอร์ต RS-232 และเครื่องคอมพิวเตอร์ เพื่อรับ-ส่งข้อมูลระหว่างกันและกัน ขาสัญญาณของ 68HC11 ที่ถูกใช้งาน คือ ขา RxD และ TxD ซึ่งก็คือ ขา PD0 และ PD1 นั่นเอง ชิปที่ทำหน้าที่ปรับสัญญาณข้อมูลให้เหมาะสมสำหรับพอร์ต RS-232 หรือที่เรียกว่า ไดรเวอร์ (Driver) ได้แก่ เบอร์ MAX232 จะเชื่อมต่อเข้ากับ 68HC11 ดังวงจรในรูปที่ 1 โดยก่อนที่จะใช้งานต้องเขียนโปรแกรมเพื่อ อีนาเบิลระบบ SCI เสียก่อน

เนื่องจากข้อจำกัดของระยะทางในการสื่อสารข้อมูลของ RS-232 ซึ่งมีระยะทางเพียง 50 ฟุต ทำให้ถ้าต้องการใช้งานในระยะทางที่ไกลกว่านั้น จำเป็นต้องใช้พอร์ตอนุกรมอีกพอร์ตหนึ่งคือ RS-422 โดยนำชิพไดรเวอร์เบอร์ 75176 มาต่อใช้งานแทน MAX232 ดังมีวงจรการเชื่อมต่อดังรูปที่ 2 จะเห็นว่าต้องต่อตัวต้านทาน R_z เข้าไประหว่างขา A และ B ของ 75176 หรือ ขา TxD และ RxD ที่จะต่อกับพอร์ต RS-422 ค่าของ R_z หาได้จากกราฟในรูปที่ 3 ยกตัวอย่าง กำหนดอัตราบอดเท่ากับ 9,600 บิตต่อวินาที ระยะทางที่ต้องการรับส่งเท่ากับ 30 เมตร จะได้ค่า R_z ประมาณ 100 โอห์ม

<u>การทดลอง</u>

1. ทำการป้อนโปรแกรมดังนี้

```
        ORG     $3000
        LDAA    #$30
        STAA    $102B
        LDAA    #$00
        STAA    $102C
        LDAA    #$0C
        STAA    $102D
UFO     LDAA    #$99
        STAA    $102F
TX      BRCLR   $102E,TX
RX      BRCLR   $102E,TX
```

```
LDAA    $102F
STAA    $50
BRA     UFO
END
```

2. ทำการป้อนโปรแกรมพร้อมกับตรวจสอบค่าดังต่อไปนี้

รีจิสเตอร์ BAU =
รีจิสเตอร์ SCCR1 =
รีจิสเตอร์ SCCR2 =
รีจิสเตอร์ $102F =
รีจิสเตอร์ SCSR =
รีจิสเตอร์ SCSR =
รีจิสเตอร์ $102F =

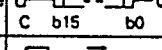
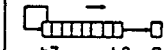
3. ตรวจสอบว่าเมื่อรีจิสเตอร์ควบคุมการติดต่ออนุกรม แต่ละตัวเปลี่ยนไป การปฏิบัติจะเป็นไปตามทฤษฎีหรือไม่ ถ้าเป็นให้สรุปโปรแกรมนี้มาคร่าว ๆ

บรรณานุกรม

1. ผศ. พิพัฒน์ เลาหสงคราม, "ไมโครคอนโทรลเลอร์ MCS-48 & MCS-51" ,ของภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง พฤษภาคม 2537.
2. ชัยวัฒน์ ลิ้มพรจิตรวิไล, "การเรียนรู้และการใช้งานไมโครคอนโทรลเลอร์ 68HC11" , บริษัท ซีเอ็ด ยูเคชั่น จำกัด (มหาชน).
3. Technical Data, "HC11 MC68hc11A8" ,MOTOROLA INC.,1991.

ภาคผนวก

INSTRUCTION SET

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal) Opcode	Operand(s)	Bytes	Cycle	Cycle by Cycle*	Condition Codes S	X	H	I	N	Z	V	C
ABA	Add Accumulators	A + B → A	INH	18		1	2	2-1	-	-	↕	-	↕	↕	↕	↕
ABX	Add B to X	IX + 00:B → IX	INH	3A		1	3	2-2	-	-	-	-	-	-	-	-
ABY	Add B to Y	IY + 00:B → IY	INH	18 3A		2	4	2-4	-	-	-	-	-	-	-	-
ADCA (opr)	Add with Carry to A	A + M + C → A	A IMM	89	ii	2	2	3-1	-	-	↕	-	↕	↕	↕	↕
			A DIR	99	dd	2	3	4-1								
			A EXT	B9	hh ll	3	4	5-2								
			A IND,X	A9	ff	2	4	6-2								
			A IND,Y	18 A9	ff	3	5	7-2								
ADCB (opr)	Add with Carry to B	B + M + C → B	B IMM	C9	ii	2	2	3-1	-	-	↕	-	↕	↕	↕	↕
			B DIR	D9	dd	2	3	4-1								
			B EXT	F9	hh ll	3	4	5-2								
			B IND,X	E9	ff	2	4	6-2								
			B IND,Y	18 E9	ff	3	5	7-2								
ADDA (opr)	Add Memory to A	A + M → A	A IMM	8B	ii	2	2	3-1	-	-	↕	-	↕	↕	↕	↕
			A DIR	9B	dd	2	3	4-1								
			A EXT	BB	hh ll	3	4	5-2								
			A IND,X	AB	ff	2	4	6-2								
			A IND,Y	18 AB	ff	3	5	7-2								
ADDB (opr)	Add Memory to B	B + M → B	B IMM	CB	ii	2	2	3-1	-	-	↕	-	↕	↕	↕	↕
			B DIR	DB	dd	2	3	4-1								
			B EXT	FB	hh ll	3	4	5-2								
			B IND,X	EB	ff	2	4	6-2								
			B IND,Y	18 EB	ff	3	5	7-2								
ADDD (opr)	Add 16-Bit to D	D + M:M + 1 → D	IMM	C3	jj kk	3	4	3-3	-	-	-	-	↕	↕	↕	↕
			DIR	D3	dd	2	5	4-7								
			EXT	F3	hh ll	3	6	5-10								
			IND,X	E3	ff	2	6	6-10								
			IND, Y	18 E3	ff	3	7	7-8								
ANDA (opr)	AND A with Memory	A•M → A	A IMM	84	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			A DIR	94	dd	2	3	4-1								
			A EXT	B4	hh ll	3	4	5-2								
			A IND,X	A4	ff	2	4	6-2								
			A IND,Y	18 A4	ff	3	5	7-2								
ANDB (opr)	AND B with Memory	B•M → B	B IMM	C4	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			B DIR	D4	dd	2	3	4-1								
			B EXT	F4	hh ll	3	4	5-2								
			B IND,X	E4	ff	2	4	6-2								
			B IND,Y	18 E4	ff	3	5	7-2								
ASL (opr)	Arithmetic Shift Left	 C b7 b0	EXT	78	hh ll	3	6	5-8	-	-	-	-	↕	↕	↕	↕
			IND,X	68	ff	2	6	6-3								
			IND,Y	18 68	ff	3	7	7-3								
ASLA			A INH	48		1	2	2-1								
ASLB			B INH	58		1	2	2-1								
ASLD	Arithmetic Shift Left Double	 C b15 b0	INH	05		1	3	2-2	-	-	-	-	↕	↕	↕	↕
ASR (opr)	Arithmetic Shift Right	 b7 b0 C	EXT	77	hh ll	3	6	5-8	-	-	-	-	↕	↕	↕	↕
			IND,X	67	ff	2	6	6-3								
			IND,Y	18 67	ff	3	7	7-3								
ASRA			A INH	47		1	2	2-1								
ASRB			B INH	57		1	2	2-1								
BCC (rel)	Branch if Carry Clear	? C = 0	REL	24	rr	2	3	8-1	-	-	-	-	-	-	-	-
BCLR (opr) (msk)	Clear Bit(s)	M•($\overline{mm}$) → M	DIR	15	dd mm	3	6	4-10	-	-	-	-	↕	↕	0	-
			IND,X	1D	ff mm	3	7	6-13								
			IND,Y	18 1D	ff mm	4	8	7-10								
BCS (rel)	Branch if Carry Set	? C = 1	REL	25	rr	2	3	8-1	-	-	-	-	-	-	-	-
BEQ (rel)	Branch if = Zero	? Z = 1	REL	27	rr	2	3	8-1	-	-	-	-	-	-	-	-

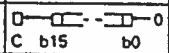
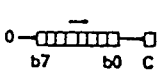
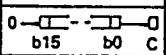
*Cycle-by-cycle number provides a reference to Tables 10-2 through 10-8 which detail cycle-by-cycle operation.
Example: Table 10-1 Cycle-by-Cycle column reference number 2-4 equals Table 10-2 line item 2-4.

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal)		Bytes	Cycle	Cycle by Cycle*	Condition Codes							
				Opcode	Operand(s)				S	X	H	I	N	Z	V	C
BGE (rel)	Branch if ≥ Zero	? N ⊕ V = 0	REL	2C	rr	2	3	8-1	-	-	-	-	-	-	-	-
BGT (rel)	Branch if > Zero	? Z + (N ⊕ V) = 0	REL	2E	rr	2	3	8-1	-	-	-	-	-	-	-	-
BHI (rel)	Branch if Higher	? C + Z = 0	REL	22	rr	2	3	8-1	-	-	-	-	-	-	-	-
BHS (rel)	Branch if Higher or Same	? C = 0	REL	24	rr	2	3	8-1	-	-	-	-	-	-	-	-
BITA (opr)	Bit(s) Test A with Memory	A•M	A IMM	85	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			A DIR	95	dd	2	3	4-1								
			A EXT	B5	hh ll	3	4	5-2								
			A IND,X	A5	ff	2	4	6-2								
			A IND,Y	18 A5	ff	3	5	7-2								
BITB (opr)	Bit(s) Test B with Memory	B•M	B IMM	C5	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			B DIR	D5	dd	2	3	4-1								
			B EXT	F5	hh ll	3	4	5-2								
			B IND,X	E5	ff	2	4	6-2								
			B IND,Y	18 E5	ff	3	5	7-2								
BLE (rel)	Branch if ≤ Zero	? Z + (N ⊕ V) = 1	REL	2F	rr	2	3	8-1	-	-	-	-	-	-	-	-
BLO (rel)	Branch if Lower	? C = 1	REL	25	rr	2	3	8-1	-	-	-	-	-	-	-	-
BLS (rel)	Branch if Lower or Same	? C + Z = 1	REL	23	rr	2	3	8-1	-	-	-	-	-	-	-	-
BLT (rel)	Branch if < Zero	? N ⊕ V = 1	REL	2D	rr	2	3	8-1	-	-	-	-	-	-	-	-
BMI (rel)	Branch if Minus	? N = 1	REL	2B	rr	2	3	8-1	-	-	-	-	-	-	-	-
BNE (rel)	Branch if Not = Zero	? Z = 0	REL	26	rr	2	3	8-1	-	-	-	-	-	-	-	-
BPL (rel)	Branch if Plus	? N = 0	REL	2A	rr	2	3	8-1	-	-	-	-	-	-	-	-
BRA (rel)	Branch Always	? 1 = 1	REL	20	rr	2	3	8-1	-	-	-	-	-	-	-	-
BRCLR(opr) (msk) (rel)	Branch if Bit(s) Clear	? M• mm = 0	DIR	13	dd mm rr	4	6	4-11	-	-	-	-	-	-	-	-
			IND,X	1F	ff mm rr	4	7	6-14								
			IND,Y	18 1F	ff mm rr	5	8	7-11								
BRN (rel)	Branch Never	? 1 = 0	REL	21	rr	2	3	8-1	-	-	-	-	-	-	-	-
BRSET(opr) (msk) (rel)	Branch if Bit(s) Set	? (M)•mm = 0	DIR	12	dd mm rr	4	6	4-11	-	-	-	-	-	-	-	-
			IND,X	1E	ff mm rr	4	7	6-14								
			IND,Y	18 1E	ff mm rr	5	8	7-11								
BSET(opr) (msk)	Set Bit(s)	M + mm → M	DIR	14	dd mm	3	6	4-10	-	-	-	-	↕	↕	0	-
			IND,X	1C	ff mm	3	7	6-13								
			IND,Y	18 1C	ff mm	4	8	7-10								
BSR (rel)	Branch to Subroutine	See Special Ops	REL	8D	rr	2	6	8-2	-	-	-	-	-	-	-	-
BVC (rel)	Branch if Overflow Clear	? V = 0	REL	28	rr	2	3	8-1	-	-	-	-	-	-	-	-
BVS (rel)	Branch if Overflow Set	? V = 1	REL	29	rr	2	3	8-1	-	-	-	-	-	-	-	-
CBA	Compare A to B	A - B	INH	11		1	2	2-1	-	-	-	-	↕	↕	↕	↕
CLC	Clear Carry Bit	0 → C	INH	0C		1	2	2-1	-	-	-	-	-	-	-	0
CLI	Clear Interrupt Mask	0 → I	INH	0E		1	2	2-1	-	-	-	0	-	-	-	-
CLR (opr)	Clear Memory Byte	0 → M	EXT	7F	hh ll	3	6	5-8	-	-	-	-	0	1	0	0
			IND,X	6F	ff	2	6	6-3								
			IND,Y	18 6F	ff	3	7	7-3								
CLRA	Clear Accumulator A	0 → A	A INH	4F		1	2	2-1	-	-	-	-	0	1	0	0
CLRB	Clear Accumulator B	0 → B	B INH	5F		1	2	2-1	-	-	-	-	0	1	0	0
CLV	Clear Overflow Flag	0 → V	INH	0A		1	2	2-1	-	-	-	-	-	-	0	-
CMPA (opr)	Compare A to Memory	A - M	A IMM	81	ii	2	2	3-1	-	-	-	-	↕	↕	↕	↕
			A DIR	91	dd	2	3	4-1								
			A EXT	B1	hh ll	3	4	5-2								
			A IND,X	A1	ff	2	4	6-2								
			A IND,Y	18 A1	ff	3	5	7-2								

*Cycle-by-cycle number provides a reference to Tables 10-2 through 10-8 which detail cycle-by-cycle operation.
Example: Table 10-1 Cycle-by-Cycle column reference number 2-4 equals Table 10-2 line item 2-4.

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal) Opcode	Operand(s)	Bytes	Cycle	Cycle by Cycle*	Condition Codes S	X	H	I	N	Z	V	C
CMPB (opr)	Compare B to Memory	B – M	B IMM	C1	ii	2	2	3-1	-	-	-	-	↕	↕	↕	↕
			B DIR	D1	dd	2	3	4-1								
			B EXT	F1	hh ll	3	4	5-2								
			B IND,X	E1	ff	2	4	6-2								
			B IND,Y	18 E1	ff	3	5	7-2								
COM (opr)	1's Complement Memory Byte	$FF – M → M	EXT	73	hh ll	3	6	5-8	-	-	-	-	↕	↕	0	1
			IND,X	63	ff	2	6	6-3								
			IND,Y	18 63	ff	3	7	7-3								
COMA	1's Complement A	$FF – A → A	A INH	43		1	2	2-1	-	-	-	-	↕	↕	0	1
COMB	1's Complement B	$FF – B → B	B INH	53		1	2	2-1	-	-	-	-	↕	↕	0	1
CPD (opr)	Compare D to Memory 16-Bit	D – M:M + 1	IMM	1A 83	jj kk	4	5	3-5	-	-	-	-	↕	↕	↕	↕
			DIR	1A 93	dd	3	6	4-9								
			EXT	1A B3	hh ll	4	7	5-11								
			IND,X	1A A3	ff	3	7	6-11								
			IND,Y	CD A3	ff	3	7	7-8								
CPX (opr)	Compare X to Memory 16-Bit	IX – M:M + 1	IMM	8C	jj kk	3	4	3-3	-	-	-	-	↕	↕	↕	↕
			DIR	9C	dd	2	5	4-7								
			EXT	BC	hh ll	3	6	5-10								
			IND,X	AC	ff	2	6	6-10								
			IND,Y	CD AC	ff	3	7	7-8								
CPY (opr)	Compare Y to Memory 16-Bit	IY – M:M + 1	IMM	18 8C	jj kk	4	5	3-5	-	-	-	-	↕	↕	↕	↕
			DIR	18 9C	dd	3	6	4-9								
			EXT	18 BC	hh ll	4	7	5-11								
			IND,X	1A AC	ff	3	7	6-11								
			IND,Y	18 AC	ff	3	7	7-8								
DAA	Decimal Adjust A	Adjust Sum to BCD	INH	19		1	2	2-1	-	-	-	-	↕	↕	↕	↕
DEC (opr)	Decrement Memory Byte	M – 1 → M	EXT	7A	hh ll	3	6	5-8	-	-	-	-	↕	↕	↕	-
			IND,X	6A	ff	2	6	6-3								
			IND,Y	18 6A	ff	3	7	7-3								
DECA	Decrement Accumulator A	A – 1 → A	A INH	4A		1	2	2-1	-	-	-	-	↕	↕	↕	-
DECB	Decrement Accumulator B	B – 1 → B	B INH	5A		1	2	2-1	-	-	-	-	↕	↕	↕	-
DES	Decrement Stack Pointer	SP – 1 → SP	INH	34		1	3	2-3	-	-	-	-	-	-	-	-
DEX	Decrement Index Register X	IX – 1 → IX	INH	09		1	3	2-2	-	-	-	-	-	↕	-	-
DEY	Decrement Index Register Y	IY – 1 → IY	INH	18 09		2	4	2-4	-	-	-	-	-	↕	-	-
EORA (opr)	Exclusive OR A with Memory	A ⊕ M → A	A IMM	88	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			A DIR	98	dd	2	3	4-1								
			A EXT	B8	hh ll	3	4	5-2								
			A IND,X	A8	ff	2	4	6-2								
			A IND,Y	18 A8	ff	3	5	7-2								
EORB (opr)	Exclusive OR B with Memory	B ⊕ M → B	B IMM	C8	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			B DIR	D8	dd	2	3	4-1								
			B EXT	F8	hh ll	3	4	5-2								
			B IND,X	E8	ff	2	4	6-2								
			B IND,Y	18 E8	ff	3	5	7-2								
FDIV	Fractional Divide 16 by 16	D/IX → IX; r → D	INH	03		1	41	2-17	-	-	-	-	-	↕	↕	↕
IDIV	Integer Divide 16 by 16	D/IX → IX; r → D	INH	02		1	41	2-17	-	-	-	-	-	↕	0	↕
INC (opr)	Increment Memory Byte	M + 1 → M	EXT	7C	hh ll	3	6	5-8	-	-	-	-	↕	↕	↕	-
			IND,X	6C	ff	2	6	6-3								
			IND,Y	18 6C	ff	3	7	7-3								
INCA	Increment Accumulator A	A + 1 → A	A INH	4C		1	2	2-1	-	-	-	-	↕	↕	↕	-
INCB	Increment Accumulator B	B + 1 → B	B INH	5C		1	2	2-1	-	-	-	-	↕	↕	↕	-
INS	Increment Stack Pointer	SP + 1 → SP	INH	31		1	3	2-3	-	-	-	-	-	-	-	-

*Cycle-by-cycle number provides a reference to Tables 10-2 through 10-8 which detail cycle-by-cycle operation.
Example: Table 10-1 Cycle-by-Cycle column reference number 2-4 equals Table 10-2 line item 2-4.

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal) Opcode	Operand(s)	Bytes	Cycle	Cycle by Cycle*	Condition Codes S	X	H	I	N	Z	V	C
INX	Increment Index Register X	IX + 1 ⟶ IX	INH	08		1	3	2-2	-	-	-	-	-	↕	-	-
INY	Increment Index Register Y	IY + 1 ⟶ IY	INH	18 08		2	4	2-4	-	-	-	-	-	↕	-	-
JMP (opr)	Jump	See Special Ops	EXT	7E	hh ll	3	3	5-1	-	-	-	-	-	-	-	-
			IND,X	6E	ff	2	3	6-1								
			IND,Y	18 6E	ff	3	4	7-1								
JSR (opr)	Jump to Subroutine	See Special Ops	DIR	9D	dd	2	5	4-8	-	-	-	-	-	-	-	-
			EXT	BD	hh ll	3	6	5-12								
			IND,X	AD	ff	2	6	6-12								
			IND,Y	18 AD	ff	3	7	7-9								
LDAA (opr)	Load Accumulator A	M ⟶ A	A IMM	86	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			A DIR	96	dd	2	3	4-1								
			A EXT	B6	hh ll	3	4	5-2								
			A IND,X	A6	ff	2	4	6-2								
			A IND,Y	18 A6	ff	3	5	7-2								
LDAB (opr)	Load Accumulator B	M ⟶ B	B IMM	C6	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			B DIR	D6	dd	2	3	4-1								
			B EXT	F6	hh ll	3	4	5-2								
			B IND,X	E6	ff	2	4	6-2								
			B IND,Y	18 E6	ff	3	5	7-2								
LDD (opr)	Load Double Accumulator D	M ⟶ A, M + 1 ⟶ B	IMM	CC	jj kk	3	3	3-2	-	-	-	-	↕	↕	0	-
			DIR	DC	dd	2	4	4-3								
			EXT	FC	hh ll	3	5	5-4								
			IND,X	EC	ff	2	5	6-6								
			IND,Y	18 EC	ff	3	6	7-6								
LDS (opr)	Load Stack Pointer	M:M + 1 ⟶ SP	IMM	8E	jj kk	3	3	3-2	-	-	-	-	↕	↕	0	-
			DIR	9E	dd	2	4	4-3								
			EXT	BE	hh ll	3	5	5-4								
			IND,X	AE	ff	2	5	6-6								
			IND,Y	18 AE	ff	3	6	7-6								
LDX (opr)	Load Index Register X	M:M + 1 ⟶ IX	IMM	CE	jj kk	3	3	3-2	-	-	-	-	↕	↕	0	-
			DIR	DE	dd	2	4	4-3								
			EXT	FE	hh ll	3	5	5-4								
			IND,X	EE	ff	2	5	6-6								
			IND,Y	CD EE	ff	3	6	7-6								
LDY (opr)	Load Index Register Y	M:M + 1 ⟶ IY	IMM	18 CE	jj kk	4	4	3-4	-	-	-	-	↕	↕	0	-
			DIR	18 DE	dd	3	5	4-5								
			EXT	18 FE	hh ll	4	6	5-6								
			IND,X	1A EE	ff	3	6	6-7								
			IND,Y	18 EE	ff	3	6	7-6								
LSL (opr)	Logical Shift Left	 C b7 b0	EXT	78	hh ll	3	6	5-8	-	-	-	-	↕	↕	↕	↕
			IND,X	68	ff	2	6	6-3								
			IND,Y	18 68	ff	3	7	7-3								
LSLA			A INH	48		1	2	2-1								
LSLB			B INH	58		1	2	2-1								
LSLD	Logical Shift Left Double	 C b15 b0	INH	05		1	3	2-2	-	-	-	-	↕	↕	↕	↕
LSR (opr)	Logical Shift Right	 b7 b0 C	EXT	74	hh ll	3	6	5-8	-	-	-	-	0	↕	↕	↕
			IND,X	64	ff	2	6	6-3								
			IND,Y	18 64	ff	3	7	7-3								
LSRA			A INH	44		1	2	2-1								
LSRB			B INH	54		1	2	2-1								
LSRD	Logical Shift Right Double	 b15 b0 C	INH	04		1	3	2-2	-	-	-	-	0	↕	↕	↕
MUL	Multiply B by A	A x B ⟶ D	INH	3D		1	10	2-13	-	-	-	-	-	-	-	↕

*Cycle-by-cycle number provides a reference to Tables 10-2 through 10-8 which detail cycle-by-cycle operation.
Example: Table 10-1 Cycle-by-Cycle column reference number 2-4 equals Table 10-2 line item 2-4.

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal) Opcode	Operand(s)	Bytes	Cycle	Cycle by Cycle*	Condition Codes S	X	H	I	N	Z	V	C
NEG (opr)	2's Complement Memory Byte	0 − M → M	EXT	70	hh ll	3	6	5-8	-	-	-	-	↕	↕	↕	↕
			IND,X	60	ff	2	6	6-3								
			IND,Y	18 60	ff	3	7	7-3								
NEGA	2's Complement A	0 − A → A	A INH	40		1	2	2-1	-	-	-	-	↕	↕	↕	↕
NEGB	2's Complement B	0 − B → B	B INH	50		1	2	2-1	-	-	-	-	↕	↕	↕	↕
NOP	No Operation	No Operation	INH	01		1	2	2-1	-	-	-	-	-	-	-	-
ORAA (opr)	OR Accumulator A (Inclusive)	A + M → A	A IMM	8A	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			A DIR	9A	dd	2	3	4-1								
			A EXT	BA	hh ll	3	4	5-2								
			A IND,X	AA	ff	2	4	6-2								
			A IND,Y	18 AA	ff	3	5	7-2								
ORAB (opr)	OR Accumulator B (Inclusive)	B + M → B	B IMM	CA	ii	2	2	3-1	-	-	-	-	↕	↕	0	-
			B DIR	DA	dd	2	3	4-1								
			B EXT	FA	hh ll	3	4	5-2								
			B IND,X	EA	ff	2	4	6-2								
			B IND,Y	18 EA	ff	3	5	7-2								
PSHA	Push A onto Stack	A → Stk, SP = SP − 1	A INH	36		1	3	2-6	-	-	-	-	-	-	-	-
PSHB	Push B onto Stack	B → Stk, SP = SP − 1	B INH	37		1	3	2-6	-	-	-	-	-	-	-	-
PSHX	Push X onto Stack (Lo First)	IX → Stk, SP = SP − 2	INH	3C		1	4	2-7	-	-	-	-	-	-	-	-
PSHY	Push Y onto Stack (Lo First)	IY → Stk, SP = SP − 2	INH	18 3C		2	5	2-8	-	-	-	-	-	-	-	-
PULA	Pull A from Stack	SP = SP + 1, A ← Stk	A INH	32		1	4	2-9	-	-	-	-	-	-	-	-
PULB	Pull B from Stack	SP = SP + 1, B ← Stk	B INH	33		1	4	2-9	-	-	-	-	-	-	-	-
PULX	Pull X from Stack (Hi First)	SP = SP + 2, IX ← Stk	INH	38		1	5	2-10	-	-	-	-	-	-	-	-
PULY	Pull Y from Stack (Hi First)	SP = SP + 2, IY ← Stk	INH	18 38		2	6	2-11	-	-	-	-	-	-	-	-
ROL (opr)	Rotate Left	 C b7 — b0 C	EXT	79	hh ll	3	6	5-8	-	-	-	-	↕	↕	↕	↕
			IND,X	69	ff	2	6	6-3								
			IND,Y	18 69	ff	3	7	7-3								
ROLA			A INH	49		1	2	2-1								
ROLB			B INH	59		1	2	2-1								
ROR (opr)	Rotate Right	 C b7 — b0 C	EXT	76	hh ll	3	6	5-8	-	-	-	-	↕	↕	↕	↕
			IND,X	66	ff	2	6	6-3								
			IND,Y	18 66	ff	3	7	7-3								
RORA			A INH	46		1	2	2-1								
RORB			B INH	56		1	2	2-1								
RTI	Return from Interrupt	See Special Ops	INH	3B		1	12	2-14	↕	↕	↕	↕	↕	↕	↕	↕
RTS	Return from Subroutine	See Special Ops	INH	39		1	5	2-12	-	-	-	-	-	-	-	-
SBA	Subtract B from A	A − B → A	INH	10		1	2	2-1	-	-	-	-	↕	↕	↕	↕
SBCA (opr)	Subtract with Carry from A	A − M − C → A	A IMM	82	ii	2	2	3-1	-	-	-	-	↕	↕	↕	↕
			A DIR	92	dd	2	3	4-1								
			A EXT	B2	hh ll	3	4	5-2								
			A IND,X	A2	ff	2	4	6-2								
			A IND,Y	18 A2	ff	3	5	7-2								
SBCB (opr)	Subtract with Carry from B	B − M − C → B	B IMM	C2	ii	2	2	3-1	-	-	-	-	↕	↕	↕	↕
			B DIR	D2	dd	2	3	4-1								
			B EXT	F2	hh ll	3	4	5-2								
			B IND,X	E2	ff	2	4	6-2								
			B IND,Y	18 E2	ff	3	5	7-2								
SEC	Set Carry	1 → C	INH	0D		1	2	2-1	-	-	-	-	-	-	-	1
SEI	Set Interrupt Mask	1 → I	INH	0F		1	2	2-1	-	-	-	1	-	-	-	-
SEV	Set Overflow Flag	1 → V	INH	0B		1	2	2-1	-	-	-	-	-	-	1	-

*Cycle-by-cycle number provides a reference to Tables 10-2 through 10-8 which detail cycle-by-cycle operation.
Example: Table 10-1 Cycle-by-Cycle column reference number 2-4 equals Table 10-2 line item 2-4.

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal) Opcode	Operand(s)	Bytes	Cycles	Cycle by Cycle*	Condition Codes S	X	H	I	N	Z	V	C
STAA (opr)	Store Accumulator A	A → M	A DIR	97	dd	2	3	4-2	-	-	-	-	↕	↕	0	-
			A EXT	B7	hh ll	3	4	5-3								
			A IND,X	A7	ff	2	4	6-5								
			A IND,Y	18 A7	ff	3	5	7-5								
STAB (opr)	Store Accumulator B	B → M	B DIR	D7	dd	2	3	4-2	-	-	-	-	↕	↕	0	-
			B EXT	F7	hh ll	3	4	5-3								
			B IND,X	E7	ff	2	4	6-5								
			B IND,Y	18 E7	ff	3	5	7-5								
STD (opr)	Store Accumulator D	A → M, B → M + 1	DIR	DD	dd	2	4	4-4	-	-	-	-	↕	↕	0	-
			EXT	FD	hh ll	3	5	5-5								
			IND,X	ED	ff	2	5	6-8								
			IND,Y	18 ED	ff	3	6	7-7								
STOP	Stop Internal Clocks		INH	CF		1	2	2-1	-	-	-	-	-	-	-	-
STS (opr)	Store Stack Pointer	SP → M:M + 1	DIR	9F	dd	2	4	4-4	-	-	-	-	↕	↕	0	-
			EXT	BF	hh ll	3	5	5-5								
			IND,X	AF	ff	2	5	6-8								
			IND,Y	18 AF	ff	3	6	7-7								
STX (opr)	Store Index Register X	IX → M:M + 1	DIR	DF	dd	2	4	4-4	-	-	-	-	↕	↕	0	-
			EXT	FF	hh ll	3	5	5-5								
			IND,X	EF	ff	2	5	6-8								
			IND,Y	CD EF	ff	3	6	7-7								
STY (opr)	Store Index Register Y	IY → M:M + 1	DIR	18 DF	dd	3	5	4-6	-	-	-	-	↕	↕	0	-
			EXT	18 FF	hh ll	4	6	5-7								
			IND,X	1A EF	ff	3	6	6-9								
			IND,Y	18 EF	ff	3	6	7-7								
SUBA (opr)	Subtract Memory from A	A - M → A	A IMM	80	ii	2	2	3-1	-	-	-	-	↕	↕	↕	↕
			A DIR	90	dd	2	3	4-1								
			A EXT	B0	hh ll	3	4	5-2								
			A IND,X	A0	ff	2	4	6-2								
			A IND,Y	18 A0	ff	3	5	7-2								
SUBB (opr)	Subtract Memory from B	B - M → B	B IMM	C0	ii	2	2	3-1	-	-	-	-	↕	↕	↕	↕
			B DIR	D0	dd	2	3	4-1								
			B EXT	F0	hh ll	3	4	5-2								
			B IND,X	E0	ff	2	4	6-2								
			B IND,Y	18 E0	ff	3	5	7-2								
SUBD (opr)	Subtract Memory from D	D - M:M + 1 → D	IMM	83	jj kk	3	4	3-3	-	-	-	-	↕	↕	↕	↕
			DIR	93	dd	2	5	4-7								
			EXT	B3	hh ll	3	6	5-10								
			IND,X	A3	ff	2	6	6-10								
			IND,Y	18 A3	ff	3	7	7-8								
SWI	Software Interrupt	See Special Ops	INH	3F		1	14	2-15	-	-	-	↕	-	-	-	-
TAB	Transfer A to B	A → B	INH	16		1	2	2-1	-	-	-	-	↕	↕	0	-
TAP	Transfer A to CC Register	A → CCR	INH	06		1	2	2-1	↕	↕	↕	↕	↕	↕	↕	↕
TBA	Transfer B to A	B → A	INH	17		1	2	2-1	-	-	-	-	↕	↕	0	-
TEST	TEST (Only in Test Modes)	Address Bus Counts	INH	00		1	**	2-20	-	-	-	-	-	-	-	-
TPA	Transfer CC Register to A	CCR → A	INH	07		1	2	2-1	-	-	-	-	-	-	-	-
TST (opr)	Test for Zero or Minus	M - 0	EXT	7D	hh ll	3	6	5-9	-	-	-	-	↕	↕	0	0
			IND,X	6D	ff	2	6	6-4								
			IND,Y	18 6D	ff	3	7	7-4								
TSTA		A - 0	A INH	4D		1	2	2-1	-	-	-	-	↕	↕	0	0
TSTB		B - 0	B INH	5D		1	2	2-1	-	-	-	-	↕	↕	0	0
TSX	Transfer Stack Pointer to X	SP + 1 → IX	INH	30		1	3	2-3	-	-	-	-	-	-	-	-
TSY	Transfer Stack Pointer to Y	SP + 1 → IY	INH	18 30		2	4	2-5	-	-	-	-	-	-	-	-

* Cycle-by-cycle number provides a reference to Tables 10-2 through 10-8 which detail cycle-by-cycle operation.
Example: Table 10-1 Cycle-by-Cycle column reference number 2-4 equals Table 10-2 line item 2-4.

Source Form(s)	Operation	Boolean Expression	Addressing Mode for Operand	Machine Coding (Hexadecimal)		Bytes	Cycle	Cycle by Cycle*	Condition Codes							
				Opcode	Operand(s)				S	X	H	I	N	Z	V	C
TXS	Transfer X to Stack Pointer	IX – 1 → SP	INH	35		1	3	2-2	-	-	-	-	-	-	-	-
TYS	Transfer Y to Stack Pointer	IY – 1 → SP	INH	18 35		2	4	2-4	-	-	-	-	-	-	-	-
WAI	Wait for Interrupt	Stack Regs & WAIT	INH	3E		1	***	2-16	-	-	-	-	-	-	-	-
XGDX	Exchange D with X	IX → D, D → IX	INH	8F		1	3	2-2	-	-	-	-	-	-	-	-
XGDY	Exchange D with Y	IY → D, D → IY	INH	18 8F		2	4	2-4	-	-	-	-	-	-	-	-

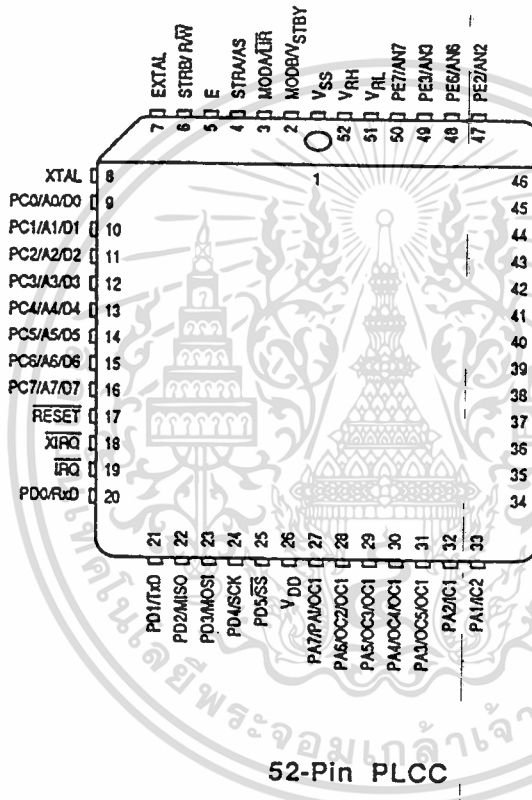
*Cycle-by-cycle number provides a reference to Tables 10-2 through 10-8 which detail cycle-by-cycle operation.
Example: Table 10-1 Cycle-by-Cycle column reference number 2-4 equals Table 10-2 line item 2-4.

**Infinity or Until Reset Occurs

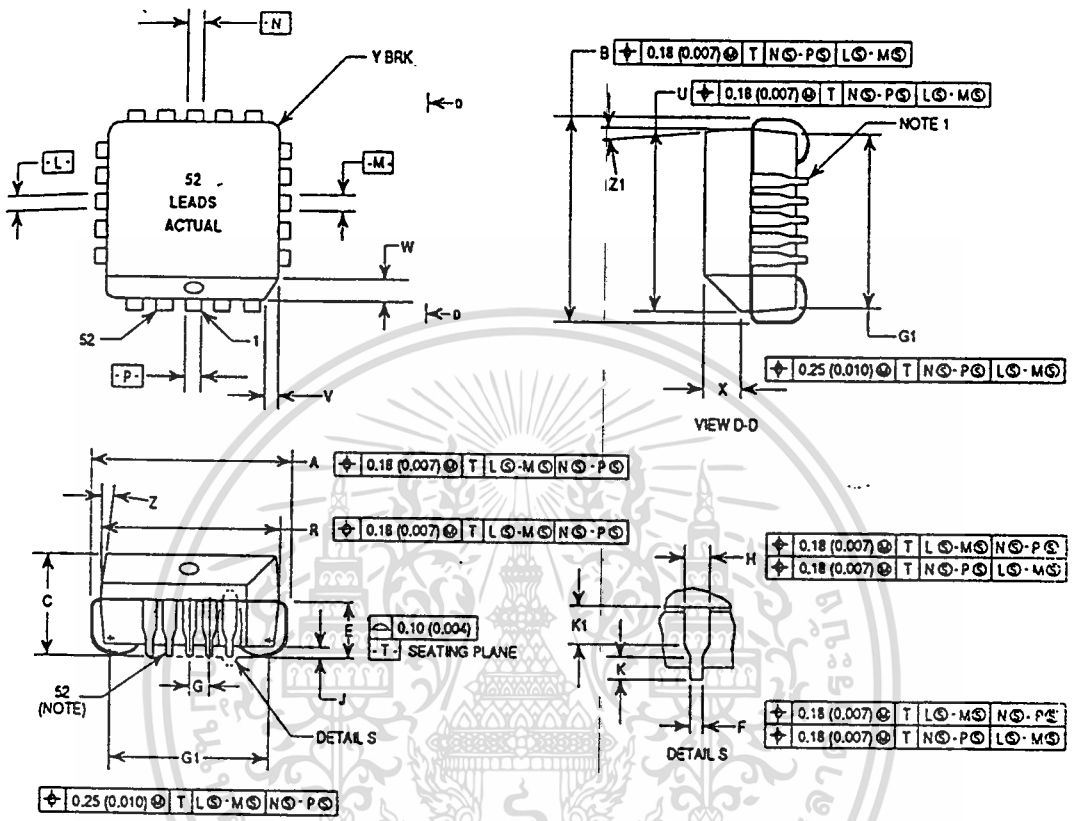
***12 Cycles are used beginning with the opcode fetch. A wait state is entered which remains in effect for an integer number of MPU E-clock cycles (n) until an interrupt is recognized. Finally, two additional cycles are used to fetch the appropriate interrupt vector (14 + n total).

dd = 8-Bit Direct Address ($0000 – $00FF) (High Byte Assumed to be $00)
ff = 8-Bit Positive Offset $00 (0) to $FF (255) (Is Added to Index)
hh = High Order Byte of 16-Bit Extended Address
ii = One Byte of Immediate Data
jj = High Order Byte of 16-Bit Immediate Data
kk = Low Order Byte of 16-Bit Immediate Data
ll = Low Order Byte of 16-Bit Extended Address
mm = 8-Bit Bit Mask (Set Bits to be Affected)
rr = Signed Relative Offset $80 (– 128) to $7F (+ 127)
(Offset Relative to the Address Following the Machine Code Offset Byte)

MECHANICAL DATA



52-Pin PLCC

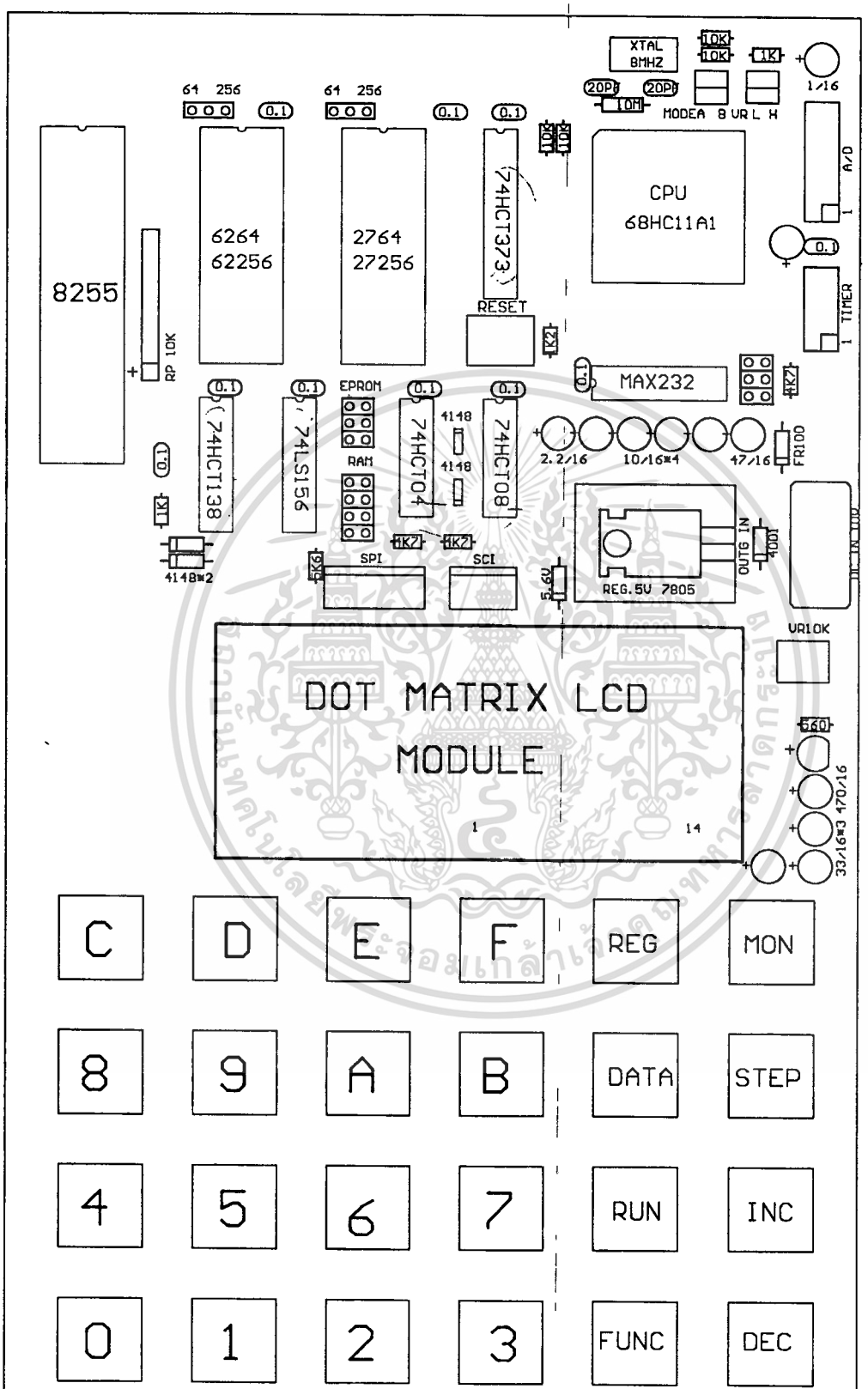


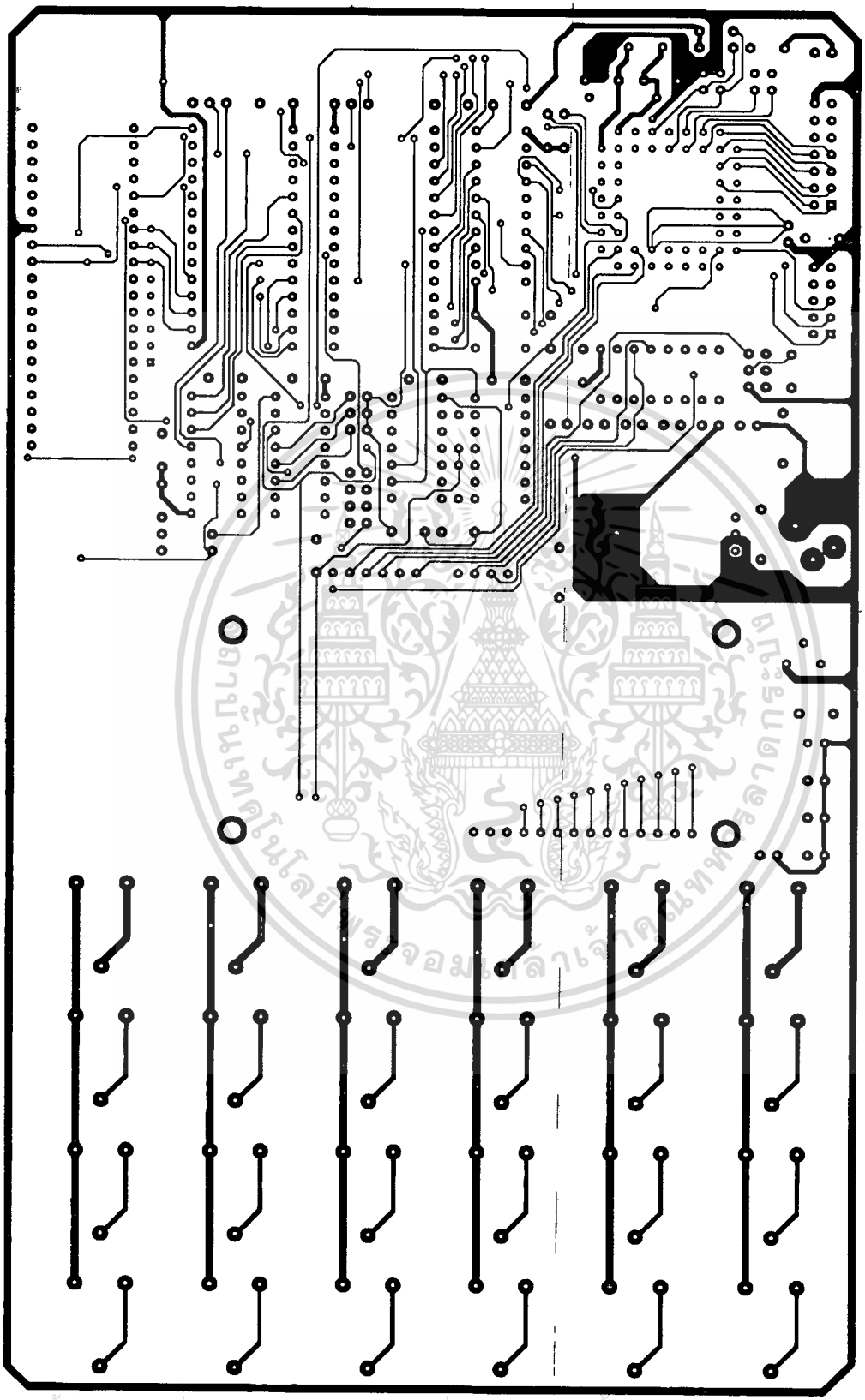
DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	19.94	20.19	0.785	0.795
B	19.94	20.19	0.785	0.795
C	4.20	4.57	0.165	0.180
E	2.29	2.79	0.090	0.110
F	0.33	0.48	0.013	0.019
G	1.27 BSC		0.050 BSC	
H	0.66	0.81	0.026	0.032
J	0.51	-	0.020	-
K	0.64	-	0.025	-
R	19.05	19.20	0.750	0.756
U	19.05	19.20	0.750	0.756
V	1.07	1.21	0.042	0.048
W	1.07	1.21	0.042	0.048
X	1.07	1.42	0.042	0.056
Y	-	0.50	-	0.020
Z	2°	10°	2°	10°
G1	18.04	18.54	0.710	0.730
K1	1.02	-	0.040	-
Z1	2°	10°	2°	10°

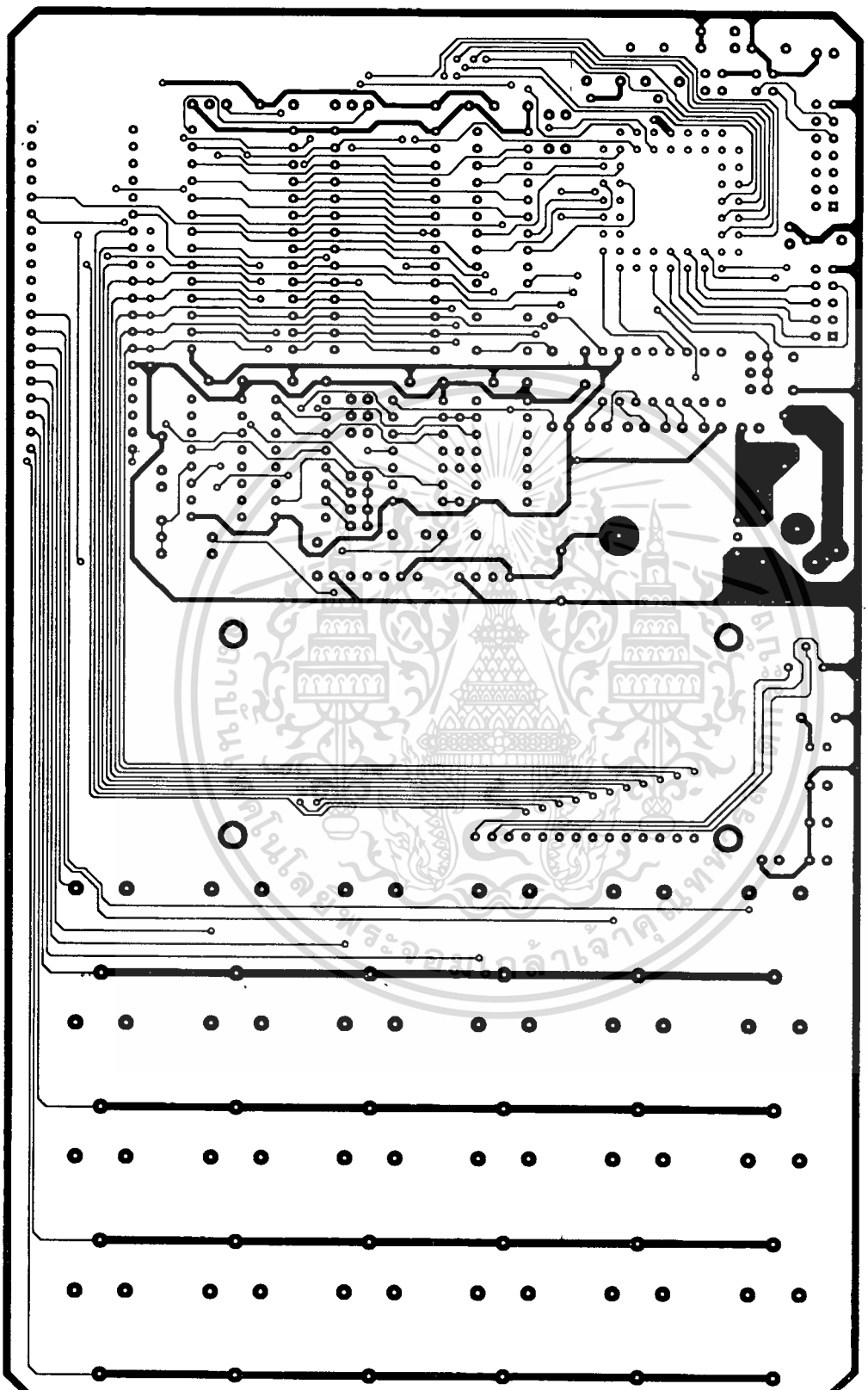
NOTES:
1. DUE TO SPACE LIMITATION, CASE 778-02 SHALL BE REPRESENTED BY A GENERAL (SMALLER) CASE OUTLINE DRAWING RATHER THAN SHOWING ALL 52 LEADS.
2. DATUMS -L-, -M-, -N-, AND -P- DETERMINED WHERE TOP OF LEAD SHOULDER EXIT PLASTIC BODY AT MOLD PARTING LINE.
3. DIM G1, TRUE POSITION TO BE MEASURED AT DATUM -T-, SEATING PLANE.
4. DIM R AND U DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE MOLD PROTRUSION IS 0.25 (0.010) PER SIDE.
5. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
6. CONTROLLING DIMENSION: INCH.

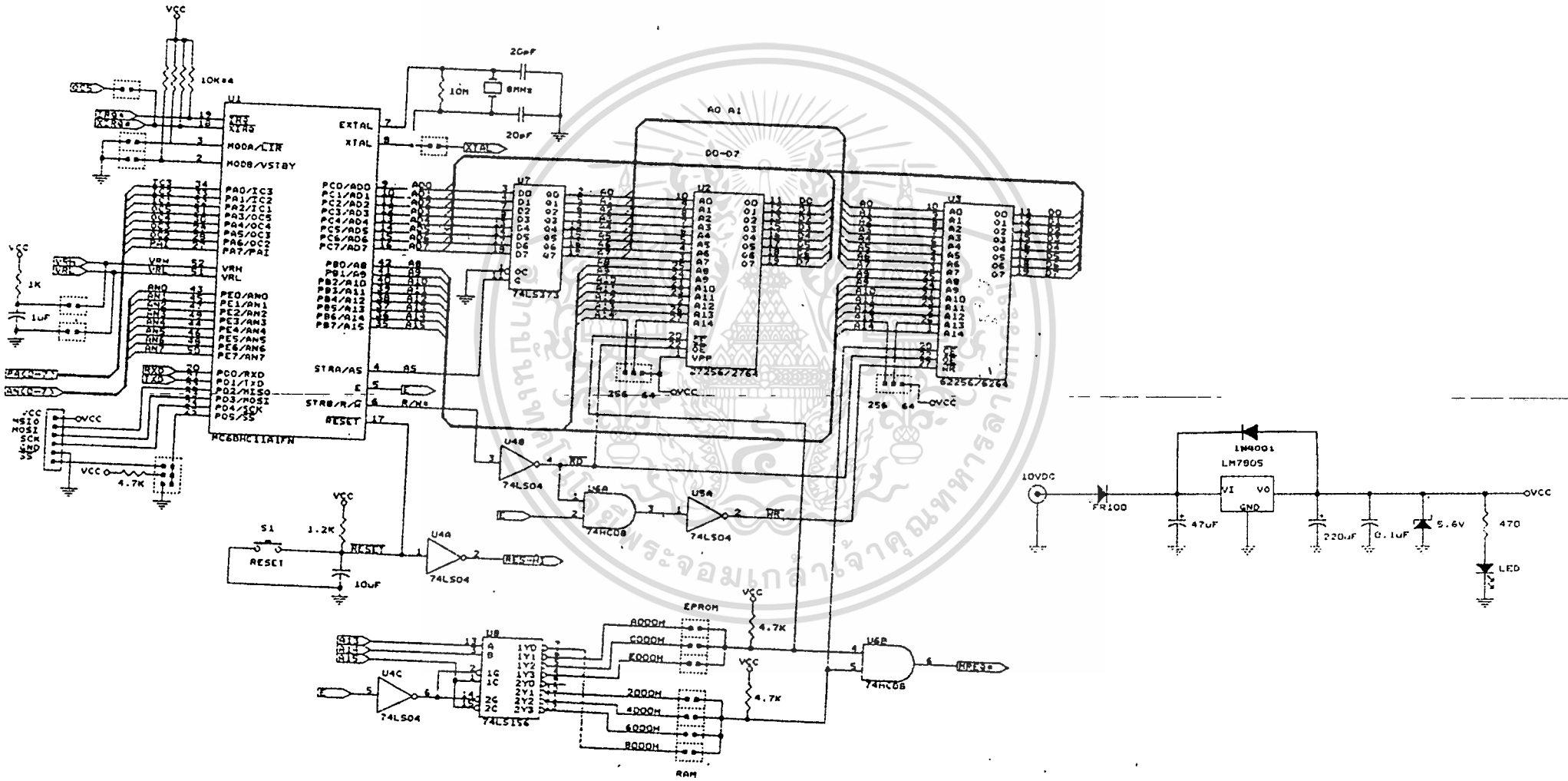
ORDERING INFORMATION

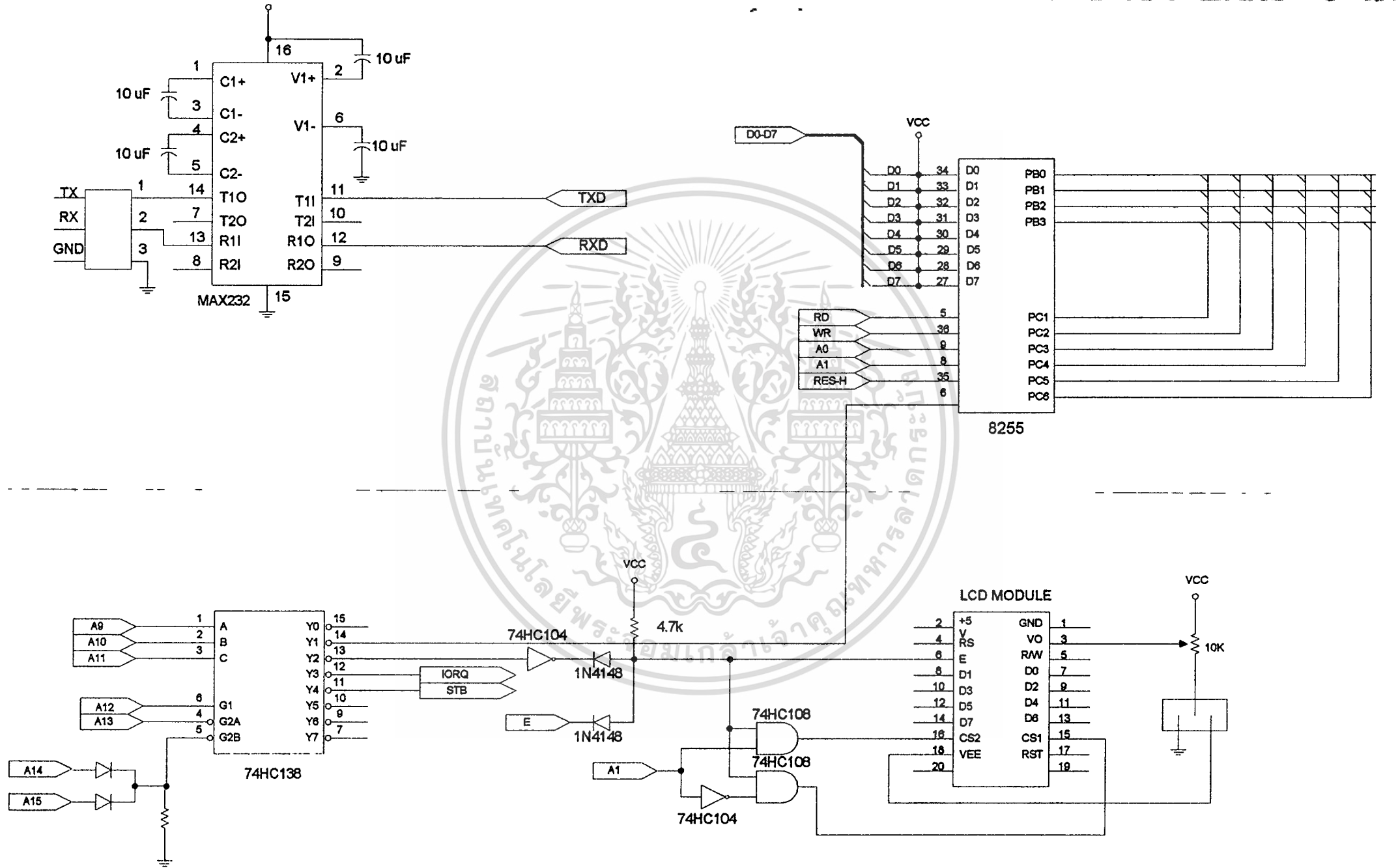
Package	Temperature	CONFIG	Description	MC Order Number
52-Pin PLCC	– 40° to + 85°C	$0C	No ROM, No EEPROM	MC68HC11A0CFN
	– 40° to + 85°C	$0C	No ROM, No EEPROM, 3 MHz	MC68HC11A0CFN3
	– 40° to + 85°C	$0D	No ROM	MC68HC11A1CFN
	– 40° to + 85°C	$0D	No ROM, 3 MHz	MC68HC11A1CFN3
	– 40° to + 105°C	$0D	No ROM	MC68HC11A1VFN
	– 40° to + 125°C	$0D	No ROM	MC68HC11A1MFN
	– 40° to + 85°C	$09	No ROM, COP On	MC68HCP11A1CFN
	– 40° to + 85°C	$09	No ROM, COP On, 3 MHz	MC68HCP11A1CFN3
	– 40° to + 105°C	$09	No ROM, COP On	MC68HCP11A1VFN
	– 40° to + 125°C	$09	No ROM, COP On	MC68HCP11A1MFN
	– 40° to + 85°C	$0F	BUFFALO ROM	MC68HC11A8CFN1
	– 40° to + 105°C	$0F	BUFFALO ROM	MC68HC11A8VFN1
	– 40° to + 125°C	$0F	BUFFALO ROM	MC68HC11A8MFN1
48-Pin DIP	– 40° to + 85°C	$0C	No ROM, No EEPROM	MC68HC11A0CP
	– 40° to + 85°C	$0C	No ROM, No EEPROM, 3 MHz	MC68HC11A0CP3
	– 40° to + 85°C	$0D	No ROM	MC68HC11A1CP
	– 40° to + 85°C	$0D	No ROM, 3 MHz	MC68HC11A1CP3
	– 40° to + 105°C	$0D	No ROM	MC68HC11A1VP
	– 40° to + 125°C	$0D	No ROM	MC68HC11A1MP
	– 40° to + 85°C	$09	No ROM, COP On	MC68HCP11A1CP
	– 40° to + 85°C	$09	No ROM, COP On, 3 MHz	MC68HCP11A1CP3
	– 40° to + 105°C	$09	No ROM, COP On	MC68HCP11A1VP
	– 40° to + 125°C	$09	No ROM, COP On	MC68HCP11A1MP
	– 40° to + 85°C	$0F	BUFFALO ROM	MC68HC11A8CP1
	– 40° to + 105°C	$0F	BUFFALO ROM	MC68HC11A8VP1
	– 40° to + 125°C	$0F	BUFFALO ROM	MC68HC11A8MP1
64-Pin QFP	– 40° to + 85°C	$0C	No ROM, No EEPROM	MC68HC11A0CFU
	– 40° to + 85°C	$0C	No ROM, No EEPROM, 3 MHz	MC68HC11A0CFU3
	– 40° to + 85°C	$0D	No ROM	MC68HC11A1CFU
	– 40° to + 85°C	$0D	No ROM, 3 MHz	MC68HC11A1CFU3
	– 40° to + 105°C	$0D	No ROM	MC68HC11A1VFU
	– 40° to + 125°C	$0D	No ROM	MC68HC11A1MFU
	– 40° to + 85°C	$0F	BUFFALO ROM	MC68HC11A8CFU1
	– 40° to + 105°C	$0F	BUFFALO ROM	MC68HC11A8VFU1








VCC
10K*4
U1
MC68HC11A1FN
MODA/LIR
MODB/VSTBY
PA0/IC3
PA1/IC2
PA2/IC1
PA3/OC5
PA4/OC4
PA5/OC3
PA6/OC2
PA7/PAI
VRH
VRL
PE0/AN0
PE1/AN1
PE2/AN2
PE3/AN3
PE4/AN4
PE5/AN5
PE6/AN6
PE7/AN7
PC0/RXD
PD1/TXD
PD2/MISO
PD3/MOSI
PD4/SCK
PD5/SS
EXTAL
XTAL
PC0/AD0
PC1/AD1
PC2/AD2
PC3/AD3
PC4/AD4
PC5/AD5
PC6/AD6
PC7/AD7
PB0/A8
PB1/A9
PB2/A10
PB3/A11
PB4/A12
PB5/A13
PB6/A14
PB7/A15
STRA/AS
STRB/R/W
RESET
2OpF
10M
8MHz
20pF
XTAL
A0 A1
D0-D7
U7
74LS373
U2
27256/2764
U3
62256/6264
256 64
OVCC
1K
1uF
4.7K
U4B
74LS04
U6A
74HC08
U5A
74LS04
S1
RESET
1.2K
10uF
U4A
74LS04
EPROM
U8
74LS156
U4C
74LS04
A000H
C000H
E000H
2000H
4000H
6000H
8000H
RAM
U6B
74HC08
10VDC
FR100
1N4001
LM7805
VI
VO
GND
47uF
220uF
0.1uF
5.6V
470
LED


MAX232
C1+
C1-
C2+
C2-
V1+
V1-
T1O
T2O
R1I
R2I
T1I
T2I
R1O
R2O
10 uF
TX
RX
GND
TXD
RXD
D0-D7
VCC
8255
PB0
PB1
PB2
PB3
PC1
PC2
PC3
PC4
PC5
PC6
RD
WR
A0
A1
RES-H
74HC138
A9
A10
A11
A12
A13
A14
A15
IORQ
STB
74HC104
1N4148
4.7k
E
74HC108
LCD MODULE
10K
GND
VO
R/W
RS
CS1
CS2
RST
VEE